

# 目录

<b>1 金鱼Scheme简介</b>	7
1.1 背景	7
1.2 文学编程	7
1.3 许可证	7
1.4 相关机构	8
<b>2 boot.scm</b>	9
<b>3 (liii base)</b>	13
3.1 许可证	13
3.2 接口	14
3.3 测试	15
3.4 表达式	16
3.4.1 原语表达式	16
3.4.2 派生表达式	16
3.4.3 宏	19
3.5 程序结构	19
3.6 相等性判断	21
3.7 数	21
3.7.1 数的类型	21
3.7.2 数的准确性	21
3.7.3 数的运算	22
3.7.4 数的转换	27
3.8 布尔值	27
3.9 序对和列表	28
3.10 符号	28
3.11 字符	29
3.12 字符串	30
3.13 向量	30
3.14 字节向量	30
3.15 控制流	35
3.16 异常处理	37
3.17 输入和输出	38
3.17.1 端口	38
3.17.2 输入	39
3.17.3 输出	40
3.18 系统接口	40
3.19 三鲤扩展函数	40
3.20 结尾	43
<b>4 (liii error)</b>	45
4.1 许可证	45
4.2 接口	46
4.3 测试	46
4.4 实现	46
4.5 结尾	48
<b>5 (liii check)</b>	49
5.1 协议	49
5.2 接口	51
5.3 实现	51
5.4 结尾	56

<b>6 (liii case)</b>	57
6.1 测试	57
6.2 结尾	58
<b>7 (liii list)</b>	59
7.1 许可证	59
7.2 接口	60
7.3 测试	61
7.4 SRFI-1	61
7.4.1 构造器	61
7.4.2 谓词	63
7.4.3 选择器	65
7.4.4 常用函数	71
7.4.5 折叠和映射	72
7.4.6 过滤和分组	76
7.4.7 搜索	77
7.4.8 删除	80
7.4.9 关联列表	82
7.4.10 循环列表	83
7.5 三鲤扩展函数	83
7.6 结尾	89
<b>8 (liii bitwise)</b>	91
8.1 概述	91
8.2 许可证	91
8.3 接口	92
8.4 测试	93
8.5 实现	93
8.5.1 基础运算	93
8.5.2 整数运算	97
8.5.3 单位运算	97
8.5.4 按位运算	97
8.5.5 按位转换	97
8.5.6 高阶函数	97
8.6 结束	97
<b>9 (liii string)</b>	99
9.1 许可证	99
9.2 接口	100
9.3 测试	101
9.4 SRFI 13	101
9.4.1 内部公共子函数	101
9.4.2 谓词	102
9.4.3 列表-字符串转换	102
9.4.4 谓词	104
9.4.5 选择器	107
9.4.6 前缀和后缀	113
9.4.7 搜索	114
9.4.8 大写小写转换	116
9.4.9 翻转和追加	117
9.4.10 高阶函数	117
9.4.11 插入和解析	118
9.5 三鲤扩展函数	119
9.6 结尾	120
<b>10 (liii vector)</b>	121
10.1 许可证	121
10.2 接口	122
10.3 测试	123

10.4 实现	123
10.4.1 构造器	123
10.4.2 谓词	125
10.4.3 选择器	126
10.4.4 迭代	126
10.4.5 搜索	129
10.4.6 修改器	130
10.4.7 转换	133
10.5 结尾	135
<b>11 (liii stack)</b>	137
11.1 许可证	137
11.2 接口	137
11.3 测试	138
11.4 实现	138
11.5 结尾	139
<b>12 (liii queue)</b>	141
12.1 许可证	141
12.2 接口	142
12.3 测试	142
12.4 实现	143
12.5 结尾	144
<b>13 (liii comparator)</b>	145
13.1 许可证	145
13.2 接口	146
13.3 测试	148
13.4 内部函数	148
13.5 构造器	149
13.6 标准函数	153
13.7 比较谓词	156
13.8 结尾	156
<b>14 (liii hash-table)</b>	157
14.1 许可证	157
14.2 测试	158
14.3 接口	158
14.3.1 访问哈希表中的元素	159
14.4 实现	159
14.4.1 子函数	159
14.4.2 构造器	159
14.4.3 谓词	160
14.4.4 选择器	161
14.4.5 修改器	162
14.4.6 哈希表整体	164
14.4.7 映射和折叠	166
14.4.8 复制和转换	167
14.5 结尾	167
<b>15 (liii set)</b>	169
15.1 许可证	169
15.2 接口	170
15.3 结尾	170
<b>16 三鯉扩展库说明</b>	171
16.1 结尾	171
<b>17 (liii base64)</b>	173
17.1 协议	173
17.2 接口	173
17.3 测试	174

17.4	Base64编解码介绍	174
17.5	实现	175
17.6	结尾	178
<b>18</b>	<b>(liii os)</b>	179
18.1	协议	179
18.2	接口	179
18.3	测试	179
18.4	实现	180
18.5	结尾	181
<b>19</b>	<b>(scheme case-lambda)</b>	183
19.1	协议	183
19.2	接口	183
19.3	实现	184
<b>20</b>	<b>(scheme char)</b>	185
20.1	许可证	185
20.2	接口	185
20.3	实现	186
20.4	结尾	187
<b>21</b>	<b>(scheme file)</b>	189
21.1	许可证	189
21.2	接口	189
21.3	实现	189
21.4	结尾	190
<b>22</b>	<b>(srfi sicp)</b>	191
22.1	许可证	191
22.2	接口	192
22.3	测试	192
22.4	实现	193
22.5	结尾	193
<b>23</b>	<b>C++部分</b>	195
23.1	许可证	195
23.2	入口	195
23.3	实现	196
23.3.1	胶水代码	197
23.3.1.1	公共辅助函数	197
23.3.1.2	Goldfish基础胶水代码	197
23.3.1.3	(scheme time)的胶水代码	198
23.3.1.4	(scheme process-context)的胶水代码	198
23.3.1.5	(liiii os)的胶水代码	200
23.3.1.6	(liiii uuid)的胶水代码	206
23.3.1.7	胶水代码汇总接口	207
23.3.2	命令行	207
<b>24</b>	<b>基础设施</b>	213
24.1	持续集成	213
24.1.1	Github平台macOS系统	213
24.1.2	Github平台Windows系统	214
24.1.3	Github平台Debian系统	216
<b>索引</b>		219

# 第 1 章

## 金鱼 Scheme 简介

金鱼 Scheme，又称 Goldfish Scheme，是三鲤网络发起的 Scheme 解释器实现。金鱼 Scheme 解释器具有以下特性：

- 兼容 R7RS small 标准
- 提供类似 Python 的标准库
- 小巧且快速

### 1.1 背景

金鱼 Scheme 创建之初的目的是为了服务墨干理工套件及其商业版 Liii STEM。

墨干一开始使用的是 S7 Scheme，然而 S7 Scheme 并不严格遵循 R7RS small 标准，也没有提供常见的 SRFI 实现。故而我们决定在 S7 Scheme 的基础上，发布金鱼 Scheme，更加严格地遵循 R7RS small 标准，并提供常见的 SRFI 实现作为金鱼 Scheme 的内置标准库。另外，无论是 R7RS small 还是 SRFI 都没有提供同时兼容 macOS、GNU/Linux 和 Windows 的类似于 Python 的 os 模块的标准库，故而我们模仿了 Python 的 os 模块，实现金鱼的内置标准库 (liii os)，以提供 `listdir`、`mkdir` 等常见功能。

### 1.2 文学编程

金鱼 Scheme 最初的三位作者是沈达、刘念和 Yansong Li，项目立项时就是采用文学编程的方式实现金鱼 Scheme 的 Scheme 部分。

一开始，我们的文学编程文档 `Goldfish.tmu` 是不开源的，我们只开源了金鱼 Scheme 的代码，其中代码是几乎没有任何注释的，因为我们在文学编程文档中的代码块之外会撰写注释。后来来自 Emacs China 社区的 StarSugar 开始向金鱼 Scheme 项目贡献代码，于是我们决定将金鱼 Scheme 的文学编程文档 `Goldfish.tmu` 也开源出来，整个项目以开源的方式推进。

从金鱼 Scheme V17.10.8 开始，我们确定了金鱼 Scheme 的文学编程文档的许可证为 **知识共享署名-相同方式共享 4.0 国际许可协议**。

本文档是采用中文编写的。在 2024 年，开源软件项目的文档采用中文编写并不是主流。首先，金鱼 Scheme 项目的前三位作者的母语都是中文，采用母语作为文档的语言是自然而然的；其次，在大模型时代，计算机辅助翻译已经非常成熟且廉价，我们相信中文文档不会是金鱼 Scheme 走向世界的阻碍。

### 1.3 许可证

本文档采用文学编程方式编写，金鱼 Scheme 相关的所有代码几乎都包含在本文档中。本文档的许可证分为代码和文档两部分，其中代码部分采用 Apache 许可证，相关信息在每一个源代码文件的头部都会显式声明，未显式声明的源代码默认采用 Apache 许可证；其中文档部分采用 **知识共享署名-相同方式共享 4.0 国际许可协议** 授权。您可以自由地分享、修改本作品，只要您遵循以下条件：

**署名。** 您必须给出适当的署名，提供作品的链接，并指明是否有任何变更。您可以以任何合理的方式做这些事，但不得以任何方式暗示许可人赞同您或您的使用。

**相同方式共享.** 如果您再混合、转换或基于本作品创作, 您必须以相同的许可协议发布其成果。

金鱼Scheme是浙江三鲤网络科技有限公司发起的开源项目, 我们在分发源代码时, 版权信息统一采用The Goldfish Scheme Authors作为金鱼Scheme的作者。那么谁是The Goldfish Scheme Authors呢? 这些信息我们会在金鱼Scheme的仓库的根目录下的AUTHORS文件中维护:

#### AUTHORS

---

```
# The original authors of the SRFI reference IMPL are kept in indivisual files
# This AUTHORS file is for The Goldfish Scheme Authors.
#
# Names should be added to this file like so:
# Name or Organization <email address>
```

```
Liii Network Inc. <*@liii.pro>
```

```
沈达 <da@liii.pro>
```

```
刘念 <nian@liii.pro>
```

```
Yansong Li <haggittli@gmail.com>
```

```
Leiyu He <heleiyu1231@gmail.com>
```

```
Duolei Wang <duolei.wang@gmail.com>
```

```
Yingyao Zhou <yingyaozhou51@gmail.com>
```

---

我们会联系金鱼Scheme的贡献者, 更新AUTHORS文件。金鱼Scheme的贡献者并不一定在这个文件中, 有可能我们没有及时更新, 也有可能相关贡献者希望保持匿名。

## 1.4 相关机构

支持金鱼Scheme的相关机构如下所示:

- 浙江三鲤网络科技有限公司
- 中国科学技术大学-德清阿尔法创新研究院

## 第 2 章

### boot.scm

S7 Scheme默认并不遵循R7RS，在启动S7 Scheme之后，我们需要做的第一件事就是加载boot.scm，实现R7RS的define-library和import。

[R7RS](#) [索引](#)  
file-exists?

goldfish/scheme/boot.scm

1 ▾

---

```
(define (file-exists? path)
  (if (string? path)
      (if (not (g_access path 0)) ; F_OK
          #f
          (if (g_access path 4) ; R_OK
              #t
              (error 'permission-error (string-append "No_\u" path))))
      (error 'type-error "(file-exists?_\u" path):_\u" path_\u" should_\u" be_\u" string))))
```

---

[R7RS](#) [索引](#)  
delete-file

goldfish/scheme/boot.scm

△ 2 ▾

```
(define (delete-file path)
  (if (not (string? path))
      (error 'type-error "(delete-file path): path should be string")
      (if (not (file-exists? path))
          (error 'read-error (string-append path " does not exist"))
          (g_delete-file path))))
```

R7RS

索引

define-library

goldfish/scheme/boot.scm

△ 3 ▾

```
; 0-clause BSD
; Adapted from S7 Scheme's r7rs.scm
(define-macro (define-library libname . body) ; |(lib name)| -> environment
  '(define ,(symbol (object->string libname))
    (with-let (sublet (unlet)
      (cons 'import import)
      (cons '*export* ())
      (cons 'export (define-macro (, (gensym) . names)
        '(set! *export* (append ',names *export*))))
      ,@body
      (apply inlet
        (map (lambda (entry)
              (if (or (member (car entry) '(*export* export import))
                    (and (pair? *export*)
                        (not (member (car entry) *export*))))
                  (values)
                  entry))
            (curlet))))))

(unless (defined? 'r7rs-import-library-filename)
  (define (r7rs-import-library-filename libs)
    (when (pair? libs)
      (let ((lib-filename (let loop ((lib (if (memq (caar libs) '(only except prefix rename)
                                                (cadar libs)
                                                (car libs)))
              (name ""))
            (set! name (string-append name (symbol->string (car lib))))
            (if (null? (cdr lib))
                (string-append name ".scm")
                (begin
                  (set! name (string-append name "/" ))
                  (loop (cdr lib) name))))))
        (unless (member lib-filename (*s7* 'file-names))
          (load lib-filename)))
      (r7rs-import-library-filename (cdr libs)))))
```

R7RS

索引

import



goldfish/scheme/boot.scm

△ 4

---

```

(define-macro (import . libs)
  '(begin
    (r7rs-import-library-filename ',libs)
    (varlet (curlet)
      ,@(map (lambda (lib)
        (case (car lib)
          ((only)
            '((lambda (e names)
              (apply inlet
                (map (lambda (name)
                  (cons name (e name)))
                  names)))
              (symbol->value (symbol (object->string (cadr ',lib))))
              (caddr ',lib)))
            ((except)
            '((lambda (e names)
              (apply inlet
                (map (lambda (entry)
                  (if (member (car entry) names)
                    (values)
                    entry))
                  e)))
              (symbol->value (symbol (object->string (cadr ',lib))))
              (caddr ',lib)))
            ((prefix)
            '((lambda (e prefix)
              (apply inlet
                (map (lambda (entry)
                  (cons (string->symbol
                    (string-append (symbol->string prefix)
                    (symbol->string (car entry))))
                    (cdr entry)))
                  e)))
              (symbol->value (symbol (object->string (cadr ',lib))))
              (caddr ',lib)))
            ((rename)
            '((lambda (e names)
              (apply inlet
                (map (lambda (entry)
                  (let ((info (assoc (car entry) names)))
                    (if info
                      (cons (cadr info) (cdr entry))
                      entry)))
                  e)))
              (symbol->value (symbol (object->string (cadr ',lib))))
              (caddr ',lib)))
            (else
            '(let ((sym (symbol (object->string ',lib))))
              (if (not (defined? sym))
                (format () "~A not loaded~%" sym)
                (symbol->value sym))))))
    libs))))

```

---



# 第 3 章

## (liii base)

Goldfish Scheme解释器默认会加载(liiii base)。如果不想默认加载(liiii base)，请使用s7、r7rs或者sicp模式。

(liiii base)由以下函数库组成：

(scheme base). 由R7RS定义的Scheme基础函数库

(srfi srfi-2). 由SRFI-2定义的and-let\*

(srfi srfi-8). 由SRFI-8定义的receive

(liiii base). 三鯉扩展函数和S7内置的非R7RS函数，例如display\*

### 3.1 许可证

goldfish/scheme/base.scm

1 ▾

```
;  
; Copyright (C) 2024 The Goldfish Scheme Authors  
;  
; Licensed under the Apache License, Version 2.0 (the "License");  
; you may not use this file except in compliance with the License.  
; You may obtain a copy of the License at  
;  
; http://www.apache.org/licenses/LICENSE-2.0  
;  
; Unless required by applicable law or agreed to in writing, software  
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT  
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the  
; License for the specific language governing permissions and limitations  
; under the License.  
;
```

goldfish/liiii/base.scm

1 ▾

```
;  
; Copyright (C) 2024 The Goldfish Scheme Authors  
;  
; Licensed under the Apache License, Version 2.0 (the "License");  
; you may not use this file except in compliance with the License.  
; You may obtain a copy of the License at  
;  
; http://www.apache.org/licenses/LICENSE-2.0  
;  
; Unless required by applicable law or agreed to in writing, software  
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT  
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the  
; License for the specific language governing permissions and limitations  
; under the License.  
;
```

tests/goldfish/liii/base-test.scm

1 ▾

---

```

;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;

```

---

## 3.2 接口

goldfish/scheme/base.scm

△ 2 ▾

---

```

(define-library (scheme base)
  (export
    let-values
    ; R7RS 5: Program Structure
    define-values define-record-type
    ; R7RS 6.2: Numbers
    square exact inexact floor s7-floor ceiling s7-ceiling truncate s7-truncate
    round s7-round floor-quotient gcd lcm s7-lcm boolean=?
    ; R7RS 6.4: list
    pair? cons car cdr set-car! set-cdr! caar cadr cdar cddr
    null? list? make-list list length append reverse list-tail
    list-ref list-set! memq memv member assq assv assoc list-copy
    ; R7RS 6.5: Symbol
    symbol? symbol=? string->symbol symbol->string
    ; R7RS 6.6: Characters
    digit-value
    ; R7RS 6.7: String
    string-copy
    ; R7RS 6.8: Vector
    vector->string string->vector vector-copy vector-copy! vector-fill!
    ; R7RS 6.9: Bytevectors
    bytevector? make-bytevector bytevector bytevector-length bytevector-u8-ref
    bytevector-u8-set! bytevector-append utf8->string string->utf8 u8-string-length
    ; Input and Output
    call-with-port port? binary-port? textual-port? input-port-open? output-port-open?
    open-binary-input-file open-binary-output-file close-port eof-object
    ; Control flow
    string-map vector-map string-for-each vector-for-each
    ; Exception
    raise guard read-error? file-error?)
  (begin

```

---

goldfish/liii/base.scm

△ 2 ▽

---

```

(define-library (liii base)
  (import (scheme base)
          (srfi srfi-2)
          (srfi srfi-8))
  (export
   ; (scheme base) defined by R7RS
   let-values
   ; R7RS 5: Program Structure
   define-values define-record-type
   ; R7RS 6.2: Numbers
   square exact inexact floor s7-floor ceiling s7-ceiling truncate s7-truncate
   round s7-round floor-quotient gcd lcm s7-lcm
   ; R7RS 6.3: Booleans
   boolean=?
   ; R7RS 6.4: list
   pair? cons car cdr set-car! set-cdr! caar cadr cdar cddr
   null? list? make-list list length append reverse list-tail
   list-ref list-set! memq memv member assq assv assoc list-copy
   ; R7RS 6.5: Symbol
   symbol=? string->symbol symbol->string
   ; R7RS 6.6: Characters
   digit-value
   ; R7RS 6.7: String
   string-copy
   ; R7RS 6.8 Vector
   vector->string string->vector vector-copy vector-copy! vector-fill!
   ; R7RS 6.9 Bytevectors
   bytevector? make-bytevector bytevector bytevector-length bytevector-u8-ref
   bytevector-u8-set! bytevector-append utf8->string string->utf8 u8-string-length
   u8-substring
   ; Input and Output
   call-with-port port? binary-port? textual-port? input-port-open? output-port-open?
   open-binary-input-file open-binary-output-file close-port eof-object
   ; Control flow
   string-map vector-map string-for-each vector-for-each
   ; Exception
   raise guard read-error? file-error?
   ; SRFI-2
   and-let*
   ; SRFI-8
   receive
   ; Extra routines for (liii base)
   == != display* in? let1 compose identity typed-lambda
  )
  (begin

```

---

### 3.3 测试

tests/goldfish/liii/base-test.scm

△ 2 ▽

---

```

(import (liii check)
        (liii base)
        (liii list))

(check-set-mode! 'report-failed)

```

---

## 3.4 表达式

本节对应R7RS的第四节：表达式。

### 3.4.1 原语表达式

原语表达式可以分为：变量、字面量、函数的应用、匿名函数、分支表达式、赋值表达式、文件包含表达式。

**lambda**

**quote**

**if**

**set!**

**include**

**include-ci**

### 3.4.2 派生表达式

**cond**

**case**

tests/goldfish/liii/base-test.scm

△ 3 ▾

```
(check (case '+
          ((+ -) 'p0)
          ((* /) 'p1))
  => 'p0)
```

```
(check (case '-
          ((+ -) 'p0)
          ((* /) 'p1))
  => 'p0)
```

```
(check (case '*
          ((+ -) 'p0)
          ((* /) 'p1))
  => 'p1)
```

```
(check (case '@
          ((+ -) 'p0)
          ((* /) 'p1))
  => #<unspecified>)
```

```
(check (case '&
          ((+ -) 'p0)
          ((* /) 'p1))
  => #<unspecified>)
```

**and**

检查 **and** 是否正确处理多个布尔表达式。

tests/goldfish/liii/base-test.scm

△ 4 ▾

```
(check-true (and #t #t #t))
(check-false (and #t #f #t))
(check-false (and #f #t #f))
(check-false (and #f #f #f))
```

验证当 `and` 没有参数时的行为。

```
tests/goldfish/liii/base-test.scm
```

△ 5 ▾

```
(check-true (and))
```

测试 `and` 与混合类型参数的组合。

```
tests/goldfish/liii/base-test.scm
```

△ 6 ▾

```
(check-true (and 1 '() "non-empty" #t))
(check-false (and #f '() "non-empty" #t))
(check-false (and 1 '() "non-empty" #f))
```

检查 `and` 在复合表达式中的行为。

```
tests/goldfish/liii/base-test.scm
```

△ 7 ▾

```
(check-true (and (> 5 3) (< 5 10)))
(check-false (and (> 5 3) (> 5 10)))
```

验证 `and` 的短路行为。

```
tests/goldfish/liii/base-test.scm
```

△ 8 ▾

```
(check-catch 'error-name
  (and (error 'error-name "This should not be evaluated") #f))
(check-false (and #f (error "This should not be evaluated")))
```

验证 `and` 返回值非布尔值的情况。

```
tests/goldfish/liii/base-test.scm
```

△ 9 ▾

```
(check (and #t 1) => 1)
```

**letrec** 索引

**or** R7RS

**when** R7RS 索引

**unless** R7RS 索引

**let** R7RS 索引

**let\*** R7RS 索引

**letrec**

```
tests/goldfish/liii/base-test.scm
```

△ 10 ▾

```
(define (test-letrec)
  (letrec ((even?
            (lambda (n)
              (if (= n 0)
                  #t
                  (odd? (- n 1)))))
          (odd?
            (lambda (n)
              (if (= n 0)
                  #f
                  (even? (- n 1)))))
          (list (even? 10) (odd? 10))))

  (check (test-letrec) => (list #t #f))

  (check-catch 'wrong-type-arg
    (letrec ((a 1) (b (+ a 1))) (list a b)))
```

**letrec\*** R7RS 索引

tests/goldfish/liii/base-test.scm

△ 11 ▾

```
(check
  (letrec* ((a 1) (b (+ a 1))) (list a b))
  => (list 1 2))
```

---

R7RS

索引

let-values

goldfish/scheme/base.scm

△ 3 ▾

```
; 0-clause BSD
; Bill Schottstaedt
; from S7 source repo: r7rs.scm
(define-macro (let-values vars . body)
  (if (and (pair? vars)
          (pair? (car vars))
          (null? (cдар vars)))
      ‘((lambda ,(caar vars)
          ,@body)
        ,(cadar vars))
      ‘(with-let
        (apply sublet (curlet)
              (list
                ,@(map
                  (lambda (v)
                    ‘((lambda ,(car v)
                      (values ,@(map (lambda (name)
                                     (values (symbol->keyword name) name))
                                     (let args->proper-list ((args (car v)))
                                       (cond ((symbol? args)
                                             (list args))
                                             ((not (pair? args))
                                              args)
                                             ((pair? (car args))
                                              (cons (caar args)
                                                    (args->proper-list (cdr args))))
                                             (else
                                              (cons (car args)
                                                    (args->proper-list (cdr args))))))))
                  (lambda (v)
                    (lambda (v)
                      (values ,@(map (lambda (name)
                                     (values (symbol->keyword name) name))
                                     (let args->proper-list ((args (car v)))
                                       (cond ((symbol? args)
                                             (list args))
                                             ((not (pair? args))
                                              args)
                                             ((pair? (car args))
                                              (cons (caar args)
                                                    (args->proper-list (cdr args))))
                                             (else
                                              (cons (car args)
                                                    (args->proper-list (cdr args))))))))
                    ,(cadr v)))
                vars)))
        ,@body)))
```

---

tests/goldfish/liii/base-test.scm

△ 12 ▾

```
(check (let-values (((ret) (+ 1 2))) (+ ret 4)) => 7)
(check (let-values (((a b) (values 3 4))) (+ a b)) => 7)
```

---

R7RS

索引

let\*-values

and-let\*



tests/goldfish/liii/base-test.scm

△ 13 ▾

```
(check (and-let* ((hi 3) (ho #f)) (+ hi 1)) => #f)
(check (and-let* ((hi 3) (ho #t)) (+ hi 1)) => 4)
```

## SRFI

goldfish/srfi/srfi-2.scm

; 0-clause BSD by Bill Schottstaedt from S7 source repo: s7test.scm

```
(define-library (srfi srfi-2)
  (export and-let*)
  (begin

    (define-macro (and-let* vars . body)
      '(let () (and ,@map (lambda (v) '(define ,@v)) vars) (begin ,@body))))

  ) ; end of begin
  ) ; end of define-library
```

### 3.4.3 宏

金鱼Scheme基于S7 Scheme，不支持`let-syntax`，`letrec-syntax`，`syntax-rules`和`syntax-error`等R7RS定义的宏的基础设施。

#### define-macro

金鱼Scheme支持`define-macro`来定义宏。宏采用正则序求值。

## 3.5 程序结构

本节对应R7RS的第5节：程序结构。

R7RS 索引  
**define**

R7RS 索引  
**define-values**

goldfish/scheme/base.scm

△ 4 ▾

```
; 0-clause BSD by Bill Schottstaedt from S7 source repo: s7test.scm
(define-macro (define-values vars expression)
  '(if (not (null? ',vars))
      (varlet (curlet) ((lambda ,vars (curlet)) ,expression))))
```

tests/goldfish/liii/base-test.scm

△ 14 ▾

```
(let ()
  (define-values (value1 value2) (values 1 2))
  (check value1 => 1)
  (check value2 => 2))
```

R7RS 索引  
**define-record-type**

实现

goldfish/scheme/base.scm

△ 5 ▽

---

```
; 0-clause BSD by Bill Schottstaedt from S7 source repo: r7rs.scm
(define-macro (define-record-type type make ? . fields)
  (let ((obj (gensym))
        (typ (gensym)) ; this means each call on this macro makes a new type
        (args (map (lambda (field)
                     (values (list 'quote (car field))
                                (let ((par (memq (car field) (cdr make))))
                                  (and (pair? par) (car par))))))
                   fields)))
    '(begin
      (define (,? ,obj)
        (and (let? ,obj)
              (eq? (let-ref ,obj ',typ) ',type)))

      (define ,make
        (inlet ',typ ',type ,@args))

      ,@(map
          (lambda (field)
            (when (pair? field)
              (if (null? (cdr field))
                  (values)
                  (if (null? (cddr field))
                      '(define ,(cadr field) ,obj)
                      (let-ref ,obj ',(car field)))
                  '(begin
                     (define ,(cadr field) ,obj)
                     (let-ref ,obj ',(car field)))
                     (define ,(caddr field) ,obj val)
                     (let-set! ,obj ',(car field) val))))))
          fields)
      ',type)))
```

---

## 测试

通过 `define-record-type`，定义了一种名为 `pare` 的记录类型，其中 `kons` 是这种记录类型的构造器，`pare?` 是谓词，`kar` 和 `kdr` 是选择器，`set-kar!` 是修改器。

tests/goldfish/liii/base-test.scm

△ 15 ▽

---

```
(define-record-type :pare
  (kons x y)
  pare?
  (x kar set-kar!)
  (y kdr))

(check (pare? (kons 1 2)) => #t)
(check (pare? (cons 1 2)) => #f)
(check (kar (kons 1 2)) => 1)
(check (kdr (kons 1 2)) => 2)

(check
  (let ((k (kons 1 2)))
    (set-kar! k 3)
    (kar k))
  => 3)
```

---

上面那个例子比较难懂，提供一个更易懂的例子：

tests/goldfish/liii/base-test.scm

△ 16 ▾

```
(define-record-type :person
  (make-person name age)
  person?
  (name get-name set-name!)
  (age get-age))

(check (person? (make-person "Da" 3)) => #t)
(check (get-age (make-person "Da" 3)) => 3)
(check (get-name (make-person "Da" 3)) => "Da")
(check
  (let ((da (make-person "Da" 3)))
    (set-name! da "Darcy")
    (get-name da))
    => "Darcy")
```

---

## SRFI

goldfish/srfi/srfi-9.scm

```
(define-library (srfi srfi-9)
  (import (scheme base))
  (export define-record-type)
  (begin
    ) ; end of begin
  ) ; end of define-library
```

---

## 3.6 相等性判断

### 3.7 数

本节对应R7RS的6.2节。

#### 3.7.1 数的类型

R7RS 索引  
**number?** (number? obj) => boolean

判断一个对象是否是数。

**complex?**

判断一个对象是否是复数。

**real?**

**rational?**

**integer?**

#### 3.7.2 数的准确性

R7RS 索引  
**exact?**

[#e3.0需要支持一下]

```
tests/goldfish/liii/base-test.scm △ 17 ▾
(check-true (exact? 1))
(check-true (exact? 1/2))
(check-false (exact? 0.3))
; (check-true (exact? #e3.0))
```

---

**inexact?**

**exact-integer?**

**exact**

```
goldfish/scheme/base.scm △ 6 ▾
(define exact inexact->exact)
```

---

**inexact**

```
goldfish/scheme/base.scm △ 7 ▾
(define inexact exact->inexact)
```

---

### 3.7.3 数的运算

R7RS 索引  
**zero?** (x) => boolean?

判断一个数是否为0，注意该判断为真的情况有两种，其一是整数的0，其二是浮点数的0.0。如有需要，可以使用**exact?**和**inexact?**准确区分两者。

真

```
tests/goldfish/liii/base-test.scm △ 18 ▾
(let1 zero-int 0
  (check-true (and (integer? zero-int) (zero? zero-int))))
(let1 zero-exact (- 1/2 1/2)
  (check-true (and (exact? zero-exact) (zero? zero-exact))))
(let1 zero-inexact 0.0
  (check-true (and (inexact? zero-inexact) (zero? zero-inexact))))
```

---

假

```
tests/goldfish/liii/base-test.scm △ 19 ▾
(check-false (zero? 1+1i))
(check-false (zero? #b11))
```

---

类型错误

```
tests/goldfish/liii/base-test.scm △ 20 ▾
(check-catch 'wrong-type-arg (zero? #\A))
(check-catch 'wrong-type-arg (zero? #t))
(check-catch 'wrong-type-arg (zero? #f))
```

---

R7RS 索引  
**positive?** (x) => boolean?

判断一个数是否为正数。

真

```
tests/goldfish/liii/base-test.scm △ 21 ▾
(check-true (positive? 1))
(check-true (positive? 0.1))
(check-true (positive? 1/2))
```

---

## 假

tests/goldfish/liii/base-test.scm △ 22 ▾

```
(check-false (positive? 0))
(check-false (positive? -1))
(check-false (positive? -1.1))
(check-false (positive? -1/2))
```

## 类型错误

tests/goldfish/liii/base-test.scm △ 23 ▾

```
(check-catch 'wrong-type-arg (positive? #\A))
(check-catch 'wrong-type-arg (positive? #t))
(check-catch 'wrong-type-arg (positive? #f))
```

R7RS 索引  
**negative?** (x) => boolean?

判断一个数是否为负数。

## 真

tests/goldfish/liii/base-test.scm △ 24 ▾

```
(check-true (negative? -1))
(check-true (negative? -0.1))
(check-true (negative? -1/2))
```

## 假

tests/goldfish/liii/base-test.scm △ 25 ▾

```
(check-false (negative? 0))
(check-false (negative? 1))
(check-false (negative? 1.1))
(check-false (negative? 1/2))
```

## 类型错误

tests/goldfish/liii/base-test.scm △ 26 ▾

```
(check-catch 'wrong-type-arg (negative? #\A))
(check-catch 'wrong-type-arg (negative? #t))
(check-catch 'wrong-type-arg (negative? #f))
```

R7RS 索引  
**odd?**  
R7RS 索引  
**even?**  
R7RS 索引  
**floor** (x) => integer

返回最大的不大于 $x$ 的整数。如果 $x$ 是准确值，那么返回值也是准确值，如果 $x$ 不是准确值，那么返回值也不是准确值。

goldfish/scheme/base.scm △ 8 ▾

```
(define s7-floor floor)

(define (floor x)
  (if (inexact? x)
      (inexact (s7-floor x))
      (s7-floor x)))
```

tests/goldfish/liii/base-test.scm

△ 27 ▽

---

```
(check (floor 1.1) => 1.0)
(check (floor 1) => 1)
(check (floor 1/2) => 0)
(check (floor 0) => 0)
(check (floor -1) => -1)
(check (floor -1.2) => -2.0)

(check (s7-floor 1.1) => 1)
(check (s7-floor -1.2) => -2)
```

---

## ceiling

goldfish/scheme/base.scm

△ 9 ▽

---

```
(define s7-ceiling ceiling)

(define (ceiling x)
  (if (inexact? x)
      (inexact (s7-ceiling x))
      (s7-ceiling x)))
```

---

tests/goldfish/liii/base-test.scm

△ 28 ▽

---

```
(check (ceiling 1.1) => 2.0)
(check (ceiling 1) => 1)
(check (ceiling 1/2) => 1)
(check (ceiling 0) => 0)
(check (ceiling -1) => -1)
(check (ceiling -1.2) => -1.0)

(check (s7-ceiling 1.1) => 2)
(check (s7-ceiling -1.2) => -1)
```

---

## truncate

goldfish/scheme/base.scm

△ 10 ▽

---

```
(define s7-truncate truncate)

(define (truncate x)
  (if (inexact? x)
      (inexact (s7-truncate x))
      (s7-truncate x)))
```

---

tests/goldfish/liii/base-test.scm

△ 29 ▽

---

```
(check (truncate 1.1) => 1.0)
(check (truncate 1) => 1)
(check (truncate 1/2) => 0)
(check (truncate 0) => 0)
(check (truncate -1) => -1)
(check (truncate -1.2) => -1.0)

(check (s7-truncate 1.1) => 1)
(check (s7-truncate -1.2) => -1)
```

---

**round**

goldfish/scheme/base.scm

△ 11 ▾

```
(define s7-round round)

(define (round x)
  (if (inexact? x)
      (inexact (s7-round x))
      (s7-round x)))
```

tests/goldfish/liii/base-test.scm

△ 30 ▾

```
(check (round 1.1) => 1.0)
(check (round 1.5) => 2.0)
(check (round 1) => 1)
(check (round 1/2) => 0)
(check (round 0) => 0)
(check (round -1) => -1)
(check (round -1.2) => -1.0)
(check (round -1.5) => -2.0)
```

R7RS

索引

**floor-quotient** ((x real?) (y real?)) => integer

$$\text{floor\_quotient}(y, x) = \lfloor y/x \rfloor$$

goldfish/scheme/base.scm

△ 12 ▾

```
(define (floor-quotient x y) (floor (/ x y)))
```

**测试**

## 正常情况测试

tests/goldfish/liii/base-test.scm

△ 31 ▾

```
(check (floor-quotient 11 2) => 5)
(check (floor-quotient 11 -2) => -6)
(check (floor-quotient -11 2) => -6)
(check (floor-quotient -11 -2) => 5)

(check (floor-quotient 10 2) => 5)
(check (floor-quotient 10 -2) => -5)
(check (floor-quotient -10 2) => -5)
(check (floor-quotient -10 -2) => 5)
```

## 特殊情况测试

tests/goldfish/liii/base-test.scm

△ 32 ▾

```
(check-catch 'division-by-zero (floor-quotient 11 0))
(check-catch 'division-by-zero (floor-quotient 0 0))

(check (floor-quotient 0 2) => 0)
(check (floor-quotient 0 -2) => 0)
```

R7RS

索引

**quotient** (quotient (y real?) (x real?)) => integer

求两个数的商。

$$\text{quotient}(y, x) = \text{truncate}(y/x)$$





tests/goldfish/liii/base-test.scm

△ 35 ▾

```
(check (lcm) => 1)
(check (lcm 1) => 1)
(check (lcm 0) => 0)
(check (lcm 32 -36) => 288)
(check (lcm 32 -36.0) => 288.0)
(check (lcm 2 4) => 4)
(check (lcm 2 4.0) => 4.0)
(check (lcm 2.0 4.0) => 4.0)
(check (lcm 2.0 4) => 4.0)
```

**square**

索引

求一个数的平方。

goldfish/scheme/base.scm

△ 14 ▾

```
(define (square x) (* x x))
```

tests/goldfish/liii/base-test.scm

△ 36 ▾

```
(check (square 2) => 4)
```

sqrt

exact-integer-sqrt

expt

### 3.7.4 数的转换

## 3.8 布尔值

**boolean=?**

(obj1 obj2 ...) => boolean

索引

goldfish/scheme/base.scm

△ 15 ▾

```
(define (boolean=? obj1 obj2 . rest)
  (define (same-boolean obj rest)
    (if (null? rest)
        #t
        (and (equal? obj (car rest))
              (same-boolean obj (cdr rest))))))
(cond ((not (boolean? obj1)) #f)
      ((not (boolean? obj2)) #f)
      ((not (equal? obj1 obj2)) #f)
      (else (same-boolean obj1 rest))))
```

测试

tests/goldfish/liii/base-test.scm

△ 37 ▾

```
(check-true (boolean=? #t #t))
(check-true (boolean=? #f #f))
(check-true (boolean=? #t #t #t))
(check-false (boolean=? #t #f))
(check-false (boolean=? #f #t))
```

## 3.9 序对和列表

见章 7

### 3.10 符号

R7RS `symbol?` (obj) => bool 索引

tests/goldfish/liii/base-test.scm △ 38 ▾

```
(check-true (symbol? 'foo))
(check-true (symbol? (car '(foo bar))))
(check-true (symbol? 'nil))
```

```
(check-false (symbol? "bar"))
(check-false (symbol? #f))
(check-false (symbol? '()))
(check-false (symbol? '123))
```

R7RS `symbol=?` 索引

如果所有参数都是符号并且它们的名称通过string=? 测试是相同的，返回#t（真），否则返回#f（假）。

goldfish/scheme/base.scm △ 16 ▾

```
(define (symbol=? sym1 sym2 . rest)
  (define (same-symbol sym rest)
    (if (null? rest)
        #t
        (and (eq? sym (car rest))
              (same-symbol sym (cdr rest)))))
  (cond ((not (symbol? sym1)) #f)
        ((not (symbol? sym2)) #f)
        ((not (eq? sym1 sym2)) #f)
        (else (same-symbol sym1 rest))))
```

#### 测试

tests/goldfish/liii/base-test.scm △ 39 ▾

```
(check-catch 'wrong-number-of-args (symbol=? 'a))
(check-catch 'wrong-number-of-args (symbol=? 1))
```

```
(check-true (symbol=? 'a 'a))
(check-true (symbol=? 'foo 'foo))
(check-false (symbol=? 'a 'b))
(check-false (symbol=? 'foo 'bar))
```

```
(check-true (symbol=? 'bar 'bar 'bar))
```

#### 测试动态生成的符号

tests/goldfish/liii/base-test.scm △ 40 ▾

```
(check-true (symbol=? (string->symbol "foo") (string->symbol "foo")))
(check-false (symbol=? (string->symbol "foo") (string->symbol "bar")))
```

#### 测试混合类型

```
tests/goldfish/liii/base-test.scm △ 41 ▾
(check-false (symbol=? 1 1))
(check-false (symbol=? 'a 1))
(check-false (symbol=? (string->symbol "foo") 1))
```

### 测试空列表

```
tests/goldfish/liii/base-test.scm △ 42 ▾
(check-false (symbol=? 'a 'b '()))
```

R7RS 索引  
**symbol->string**

symbol->string 是一个 S7 内置的函数，用于将符号转换为字符串。

#### 测试

```
tests/goldfish/liii/base-test.scm △ 43 ▾
(check (symbol->string 'MathAgape) => "MathAgape")
(check (symbol->string 'goldfish-scheme) => "goldfish-scheme")

(check (symbol->string (string->symbol "MathApage")) => "MathApage")
(check (symbol->string (string->symbol "Hello_World")) => "Hello_World")
```

R7RS 索引  
**string->symbol**

string->symbol 是一个 S7 内置的函数，用于将字符串转换为符号。

#### 测试

Goldfish Scheme 17.10.6 Community Edition by LiiiLabs  
 implemented on S7 Scheme (10.12, 16-Aug-2024)

```
> (string->symbol "123")
```

```
(symbol "123")
```

```
> '123
```

```
123
```

```
> '123
```

```
123
```

```
>
```

```
tests/goldfish/liii/base-test.scm △ 44 ▾
(check (string->symbol "MathAgape") => 'MathAgape)
(check-false (equal? (string->symbol "123") '123))
(check (string->symbol "+") => '+)
```

```
(check (string->symbol (symbol->string 'MathAgape)) => 'MathAgape)
```

## 3.11 字符

R7RS 索引  
**char?**

判断一个对象 x 是否是字符类型。

```
tests/goldfish/liii/base-test.scm △ 45 ▾
(check (char? #\A) => #t)
(check (char? 1) => #f)
```

[R7RS](#) [索引](#)  
**char=?**

判断两个及以上字符对象是否相等。

tests/goldfish/liii/base-test.scm △ 46 ▾

```
(check (char=? #\A #\A) => #t)
(check (char=? #\A #\A #\A) => #t)
(check (char=? #\A #\a) => #f)
```

[R7RS](#) [索引](#)  
**char->integer**

将字符转换为其对应的Unicode标量值。

tests/goldfish/liii/base-test.scm △ 47 ▾

```
(check (char->integer #\A) => 65)
(check (char->integer #\a) => 97)
(check (char->integer #\newline) => 10)
(check (char->integer #\space) => 32)
(check (char->integer #\tab) => 9)
```

## 3.12 字符串

见章 9

## 3.13 向量

见章 10

## 3.14 字节向量

字节向量是固定长度的单字节序列，每一个单字节的取值范围是[0, 255]。字节向量的每一个元素都是类型一致的单字节，其空间利用率会比向量高一些。

类型一致的向量在S7 Scheme中有**int-vector**和**byte-vector**等特殊形式，这是S7 Scheme里面扩展的功能。R7RS定义的**bytevector**可以使用**byte-vector**作为实现。

[R7RS](#) [索引](#)  
**bytevector** (byte ...) => **bytevector**

**byte**. 一个或多个介于 0 到 255 之间的整数（代表字节的值）。

**bytevector**过程返回一个新分配的字节向量，其元素包含传递给过程的所有参数。每个参数都必须是一个介于 0 到 255 之间的整数，表示字节向量中的一个字节。如果没有提供任何参数，将创建一个空的字节向量。

goldfish/scheme/base.scm △ 17 ▾

```
(define bytevector byte-vector)
```

tests/goldfish/liii/base-test.scm △ 48 ▾

```
(check (bytevector 1) => #u8(1))
(check (bytevector) => #u8())
(check (bytevector 1 2 3) => #u8(1 2 3))

(check (bytevector 255) => #u8(255))
(check-catch 'wrong-type-arg (bytevector 256))
(check-catch 'wrong-type-arg (bytevector -1))
```

[R7RS](#) [索引](#)  
**bytevector?**

[goldfish/scheme/base.scm](#)

△ 18 ▾

```
(define bytevector? byte-vector?)
```

[tests/goldfish/liii/base-test.scm](#)

△ 49 ▾

```
(check-true (bytevector? #u8(0)))
(check-true (bytevector? #u8()))
```

**make-bytevector**

索引

[goldfish/scheme/base.scm](#)

△ 19 ▾

```
(define make-bytevector make-byte-vector)
```

[tests/goldfish/liii/base-test.scm](#)

△ 50 ▾

```
(check (make-bytevector 3 0) => #u8(0 0 0))
(check (make-bytevector 3 3) => #u8(3 3 3))
```

**bytevector-length**

索引

[goldfish/scheme/base.scm](#)

△ 20 ▾

```
(define bytevector-length length)
```

**bytevector-u8-ref**

索引

[goldfish/scheme/base.scm](#)

△ 21 ▾

```
(define bytevector-u8-ref byte-vector-ref)
```

**bytevector-u8-set!**

索引

[goldfish/scheme/base.scm](#)

△ 22 ▾

```
(define bytevector-u8-set! byte-vector-set!)
```

**bytevector-append**

索引

实现

[goldfish/scheme/base.scm](#)

△ 23 ▾

```
(define bytevector-append append)
```

测试

[tests/goldfish/liii/base-test.scm](#)

△ 51 ▾

```
(check (bytevector-append #u8() #u8()) => #u8())
(check (bytevector-append #u8() #u8(1)) => #u8(1))
(check (bytevector-append #u8(1) #u8()) => #u8(1))
```

**u8-string-length**

索引

```

R7RS utf8->string 索引
R7RS string->utf8 索引
u8-substring 索引

```

这三个函数依赖于一样的公共子函数，故而放在一起实现。

### 公共子函数

首先定义公共子函数 `bytevector-advance-u8`: `(bv index end) => index`。

1. 当前位置超过指定的开区间的结尾时，直接返回当前位置
2. 当前位置是非法位置，直接返回当前位置
3. 当前位置是合法位置，返回下一个位置

[goldfish/scheme/base.scm](http://goldfish/scheme/base.scm)

△ 24 ▽

```

(define* (bytevector-advance-u8 bv index (end (length bv)))
  (if (>= index end)
      index ; Reached the end without errors, sequence is valid
      (let ((byte (bv index)))
        (cond
         ;; 1-byte sequence (0xxxxxxx)
         ((< byte #x80)
          (+ index 1))

         ;; 2-byte sequence (110xxxxx 10xxxxxx)
         ((< byte #xe0)
          (if (>= (+ index 1) end)
              index ; Incomplete sequence
              (let ((next-byte (bv (+ index 1))))
                (if (not (= (logand next-byte #xc0) #x80))
                    index ; Invalid continuation byte
                    (+ index 2))))))

         ;; 3-byte sequence (1110xxxx 10xxxxxx 10xxxxxx)
         ((< byte #xf0)
          (if (>= (+ index 2) end)
              index ; Incomplete sequence
              (let ((next-byte1 (bv (+ index 1)))
                    (next-byte2 (bv (+ index 2))))
                (if (or (not (= (logand next-byte1 #xc0) #x80))
                        (not (= (logand next-byte2 #xc0) #x80)))
                    index ; Invalid continuation byte(s)
                    (+ index 3))))))

         ;; 4-byte sequence (11110xxx 10xxxxxx 10xxxxxx 10xxxxxx)
         ((< byte #xf8)
          (if (>= (+ index 3) end)
              index ; Incomplete sequence
              (let ((next-byte1 (bv (+ index 1)))
                    (next-byte2 (bv (+ index 2)))
                    (next-byte3 (bv (+ index 3))))
                (if (or (not (= (logand next-byte1 #xc0) #x80))
                        (not (= (logand next-byte2 #xc0) #x80))
                        (not (= (logand next-byte3 #xc0) #x80)))
                    index ; Invalid continuation byte(s)
                    (+ index 4))))))
          (else index)))) ; Invalid leading byte

```

## 实现u8-string-length

goldfish/scheme/base.scm

△ 25 ▽

---

```
(define (u8-string-length str)
  (let ((bv (string->byte-vector str))
        (N (string-length str)))
    (let loop ((pos 0) (cnt 0))
      (let ((next-pos (bytevector-advance-u8 bv pos N)))
        (cond
         ((= next-pos N)
          (+ cnt 1))
         ((= next-pos pos)
          (error 'value-error "Invalid UTF-8 sequence at index:~a" pos))
         (else (loop next-pos (+ cnt 1))))))))))
```

---

tests/goldfish/liii/base-test.scm

△ 52 ▽

---

```
(check (u8-string-length "中文") => 2)
```

---

## 实现utf8-&gt;string

goldfish/scheme/base.scm

△ 26 ▽

---

```
(define* (utf8->string bv (start 0) (end (bytevector-length bv)))
  (if (or (< start 0) (> end (bytevector-length bv)) (> start end))
      (error 'out-of-range start end)
      (let loop ((pos start))
        (let ((next-pos (bytevector-advance-u8 bv pos end)))
          (cond
           ((= next-pos end)
            (copy bv (make-string (- end start)) start end))
           ((= next-pos pos)
            (error 'value-error "Invalid UTF-8 sequence at index:~a" pos))
           (else
            (loop next-pos))))))))))
```

---

tests/goldfish/liii/base-test.scm

△ 53 ▽

---

```
(check (utf8->string (bytevector #x48 #x65 #x6C #x6C #x6F)) => "Hello")
(check (utf8->string #u8(#xC3 #xA4)) => "Ã")
(check (utf8->string #u8(#xE4 #xB8 #xAD)) => "中")
(check (utf8->string #u8(#xF0 #x9F #x91 #x8D)) => "<#1F44D>")

(check-catch 'value-error (utf8->string (bytevector #xFF #x65 #x6C #x6C #x6F)))
```

---

## 实现string-&gt;utf8

goldfish/scheme/base.scm

△ 27 ▽

```

(define* (string->utf8 str (start 0) (end #t))
  ; start < end in this case
  (define (string->utf8-sub str start end)
    (let ((bv (string->byte-vector str))
          (N (string-length str)))
      (let loop ((pos 0) (cnt 0) (start-pos 0))
        (let ((next-pos (bytevector-advance-u8 bv pos N)))
          (cond
            ((and (not (zero? start)) (zero? start-pos) (= cnt start))
             (loop next-pos (+ cnt 1) pos))
            ((and (integer? end) (= cnt end))
             (copy bv (make-byte-vector (- pos start-pos)) start-pos pos))
            ((and end (= next-pos N))
             (copy bv (make-byte-vector (- N start-pos)) start-pos N))
            ( (= next-pos pos)
              (error 'value-error "Invalid_UTF-8_sequence_at_index:" pos))
            (else
             (loop next-pos (+ cnt 1) start-pos))))))

    (when (not (string? str))
      (error 'type-error "str_must_be_string"))
    (let ((N (u8-string-length str)))
      (when (or (< start 0) (>= start N))
        (error 'out-of-range
                (string-append "start_must_>=0_and_<" (number->string N))))
      (when (and (integer? end) (or (< end 0) (>= end (+ N 1))))
        (error 'out-of-range
                (string-append "end_must_>=0_and_<" (number->string (+ N 1))))
      (when (and (integer? end) (> start end))
        (error 'out-of-range "start_<=end_failed" start end))

      (if (and (integer? end) (= start end))
          (byte-vector)
          (string->utf8-sub str start end))))

```

tests/goldfish/liii/base-test.scm

△ 54 ▽

```

(check (string->utf8 "Hello") => (bytevector #x48 #x65 #x6C #x6C #x6F))
(check (utf8->string (string->utf8 "Hello" 1 2)) => "e")
(check (utf8->string (string->utf8 "Hello" 0 2)) => "He")
(check (utf8->string (string->utf8 "Hello" 2)) => "llo")
(check (utf8->string (string->utf8 "Hello" 2 5)) => "llo")

(check-catch 'out-of-range (string->utf8 "Hello" 2 6))

(check (utf8->string (string->utf8 "汉字书写")) => "汉字书写")
(check (utf8->string (string->utf8 "汉字书写" 1)) => "字书写")
(check (utf8->string (string->utf8 "汉字书写" 2)) => "书写")
(check (utf8->string (string->utf8 "汉字书写" 3)) => "写")

(check-catch 'out-of-range (string->utf8 "汉字书写" 4))

(check (string->utf8 "ä") => #u8(#xC3 #xA4))
(check (string->utf8 "中") => #u8(#xE4 #xB8 #xAD))
(check (string->utf8 "<#1F44D>") => #u8(#xF0 #x9F #x91 #x8D))

```



## 实现u8-substring

goldfish/liii/base.scm

△ 3 ▾

```
(define* (u8-substring str (start 0) (end #t))
  (utf8->string (string->utf8 str start end)))
```

tests/goldfish/liii/base-test.scm

△ 55 ▾

```
(check (u8-substring "汉字书写" 0 1) => "汉")
(check (u8-substring "汉字书写" 0 4) => "汉字书写")
(check (u8-substring "汉字书写" 0) => "汉字书写")
```

## 3.15 控制流

procedure?

索引

apply

索引

apply是R7RS定义的函数。

tests/goldfish/liii/base-test.scm

△ 56 ▾

```
(check (apply + (list 3 4)) => 7)
(check (apply + (list 2 3 4)) => 9)
```

在这个例子中，(apply + (list 3 4))实际被展开为(+ 3 4)。+这个函数接受两个参数，但是无法接受一个列表，利用apply就可以把列表展开并作为+的两个参数。

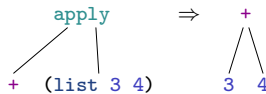


图 3.1. apply的原理可视化

## call-with-current-continuation

call/cc

values

索引

values是一个过程 (procedure)，它用于返回多个值。在 Scheme 中，通常的过程只能返回一个值，但是values允许你从一个过程返回多个值，这些值可以被另一个过程接收并处理。

当我们只传一个值给 values 时 (比如 (values 4))，它就简单地返回这个值，这时 values 的行为就像一个恒等函数。这是最简单的使用场景。

如果调用 values 时不带任何参数 (即 (values))，它会返回一个空值。这种情况在实践中比较少见，但在某些特殊场合可能会用到。

tests/goldfish/liii/base-test.scm

△ 57 ▾

```
(check (values 4) => 4)
(check (values) => #<unspecified>)
```

在金鱼Scheme中，`values`的用法有一些特别之处，它允许将多个值直接插入到调用者的参数列表中。这种特性使得金鱼Scheme在处理多个返回值时非常灵活和强大。以下是`values`在金鱼Scheme中的一些特殊用法（这些特殊用法源自于S7 Scheme）：

**直接插入到调用者参数列表.** 使用 `values` 生成的多个值会被直接插入到调用者的参数列表中。这意味着，如果你有一个接受多个参数的函数，你可以直接从`values`获得的多个值中解构这些参数。

```
tests/goldfish/liii/base-test.scm △ 58 ▽
(check (+ (values 1 2 3) 4) => 10)
```

---

这里，`values`生成的三个值(1 2 3)被直接插入到 `+` 函数的参数列表中，相当于`(+ 1 2 3 4)`。

**与`lambda`配合使用.** 当使用 `lambda` 表达式时，`values` 也可以直接将多个值插入到参数列表中：

```
tests/goldfish/liii/base-test.scm △ 59 ▽
(check (string-ref ((lambda () (values "abcd" 2)))) => #\c)
```

---

这里，`values` 生成的字符串 "abcd" 和整数 2 被直接插入到 `string-ref` 函数的参数列表中，相当于`(string-ref "abcd" 2)`。

**`values`与`call/cc`的隐式使用.** `call/cc` (call with current continuation) 是一个特殊的函数，它接受一个返回多个值的过程，并将其值作为参数传递给一个消费者。在金鱼Scheme中，`call/cc` 有一个隐式的 `values`，这意味着你可以直接返回多个值：

```
tests/goldfish/liii/base-test.scm △ 60 ▽
(check (+ (call/cc (lambda (ret) (ret 1 2 3))) 4) => 10)
```

---

这里，`ret` 宏生成的多个值 (1 2 3) 被 `+` 函数接收并求和。

这些特性使得金鱼Scheme在处理多个返回值时非常灵活，允许程序员以一种直观和高效的方式编写代码。

**R7RS**

**call-with-values** (producer consumer)

```
tests/goldfish/liii/base-test.scm △ 61 ▽
(check (call-with-values (lambda () (values 4 5))
                        (lambda (x y) x))
      => 4)
```

---

由于`*`这个函数在没有参数时返回1，而`-`在只有一个参数的时候，返回该值的相反数。故而下面这个测试用例的返回值是`-1`。

```
tests/goldfish/liii/base-test.scm △ 62 ▽
(check (*) => 1)
(check (call-with-values * -) => -1)
```

---

**receive** 索引

官网：<https://srfi.schemers.org/srfi-8/srfi-8.html>

官网的参考实现是：

```
(define-syntax receive
  (syntax-rules ()
    ((receive formals expression body ...)
```

```
(call-with-values (lambda () expression)
                  (lambda formals body ...))))
```

但是S7 Scheme里面没有define-syntax和syntax-rule, 在墨干中的实现是这样的:

```
> (define-macro (receive vars vals . body)
    '((lambda ,vars ,@body) ,vals)) ; GPL
    receive
> (receive (a b) (values 1 2) (+ a b))
> ((lambda (a b) (+ a b)) 1 2)
>
```

S7里面有call-with-values, 使用define-macro就可以实现SRFI-8.

goldfish/srfi/srfi-8.scm

---

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;

(define-library (srfi srfi-8)
  (export receive)
  (begin

    (define-macro (receive formals expression . body)
      '(call-with-values
         (lambda () (values ,expression))
         (lambda ,formals ,@body)))

    ) ; end of begin
  ) ; end of define-library
```

tests/goldfish/liii/base-test.scm

---


△ 63 ▽

```
(check
  (receive (a b) (values 1 2) (+ a b))
  => 3)
```

---

dynamic-wind

## 3.16 异常处理

guard 

goldfish/scheme/base.scm

△ 28 ▾

```
(define (raise . args)
  (apply throw #t args))

(define-macro (guard results . body)
  '(let ((,(car results)
        (catch #t
          (lambda ()
            ,@body)
          (lambda (type info)
            (if (pair? (*s7* 'catches))
                (lambda () (apply throw type info))
                (car info))))))
    (cond ,(cdr results)
          (else
           (if (procedure? ,(car results))
               ,(car results)
               ,(car results))))))
```


### 测试用例

tests/goldfish/liii/base-test.scm

△ 64 ▾

```
(guard (condition
  (else
   (display "condition:␣")
   (write condition)
   (newline)
   'exception))
  (+ 1 (raise 'an-error)))
; PRINTS: condition: an-error


(guard (condition
  (else
   (display "something␣went␣wrong")
   (newline)
   'dont-care))
  (+ 1 (raise 'an-error)))
; PRINTS: something went wrong
```

read-error? 

goldfish/scheme/base.scm

△ 29 ▾

```
(define (read-error? obj) (eq? (car obj) 'read-error))
```

file-error? 


goldfish/scheme/base.scm

△ 30 ▾

```
(define (file-error? obj) (eq? (car obj) 'io-error))
```

## 3.17 输入和输出

### 3.17.1 端口

call-with-port 

[goldfish/scheme/base.scm](#)

△ 31 ▾

---

```
(define (call-with-port port proc)
  (let ((res (proc port)))
    (if res (close-port port)
        res)))
```

---

**input-port?**

S7内置函数。

**output-port?**

S7内置函数。

**port?**[goldfish/scheme/base.scm](#)

△ 32 ▾

---

```
(define (port? p) (or (input-port? p) (output-port? p)))
```

---

**textual-port?**[goldfish/scheme/base.scm](#)

△ 33 ▾

---

```
(define textual-port? port?)
```

---

**binary-port?**[goldfish/scheme/base.scm](#)

△ 34 ▾

---

```
(define binary-port? port?)
```

---

**input-port-open?**[goldfish/scheme/base.scm](#)

△ 35 ▾

---

```
(define (input-port-open? p) (not (port-closed? p)))
```

---

**output-port-open?**[goldfish/scheme/base.scm](#)

△ 36 ▾

---

```
(define (output-port-open? p) (not (port-closed? p)))
```

---

**close-port**[goldfish/scheme/base.scm](#)

△ 37 ▾

---

```
(define (close-port p)
  (if (input-port? p)
      (close-input-port p)
      (close-output-port p)))
```

---

**close-input-port**

S7内置函数。

**close-output-port**

S7内置函数。

### 3.17.2 输入

**read**[tests/goldfish/liii/base-test.scm](#)

△ 65 ▾

---

```
(with-input-from-string "(+ 1 2)"
  (lambda ()
    (let ((datum (read)))
      (check-true (list? datum))
      (check datum => '(+ 1 2)))))
```

---

`read-char`

`peek-char`

`read-line`

`eof-object?`

`eof-object`

[索引](#)

实现

[goldfish/scheme/base.scm](#)

[△ 38 ▾](#)

---

```
(define (eof-object) #<eof>)
```

---

测试

[tests/goldfish/liii/base-test.scm](#)

[△ 66 ▾](#)

---

```
(check (eof-object) => #<eof>)
```

---

`char-ready?`

`read-string`

`read-u8`

`peek-u8`

`u8-ready?`

`read-bytevector`

`read-bytevector!`

### 3.17.3 输出

## 3.18 系统接口

### 3.19 三鲤扩展函数

`==` [索引](#) `(x y) => bool`

`equal?`的语法糖。

[goldfish/liii/base.scm](#)

[△ 4 ▾](#)

---

```
(define == equal?)
```

---

[tests/goldfish/liii/base-test.scm](#)

[△ 67 ▾](#)

---

```
(check (== (list 1 2) (list 1 2)) => #t)
```

```
(check (!= (list 1 2) (list 1 2)) => #f)
```

---

`!=` [索引](#) `(x y) => bool`

语法糖。

[goldfish/liii/base.scm](#)

[△ 5 ▾](#)

---

```
(define (!= left right)
```

```
  (not (equal? left right)))
```

---

tests/goldfish/liii/base-test.scm

△ 68 ▾

```
(check (== (list 1 2) (list 1 2)) => #t)
(check (!= (list 1 2) (list 1 2)) => #f)
```

**display\*** <sup>索引</sup> (x y z ...) => <#unspecified>  
 display\*可以输入多个参数，是display的加强版。

goldfish/liii/base.scm

△ 6 ▾

```
(define (display* . params)
  (for-each display params))
```

tests/goldfish/liii/base-test.scm

△ 69 ▾

```
(check
  (with-output-to-string
    (lambda ()
      (display* "hello_world" "\n"))))
=> "hello_world\n")
```

**in?** <sup>索引</sup> (x sequence) => bool  
 判断一个元素是否在对象中：

1. 一个元素是否在列表中
2. 一个元素是否在向量中
3. 一个字符是否在字符串中

其中的相等性判断使用的是==，也就是R7RS定义的equal?。

tests/goldfish/liii/base-test.scm

△ 70 ▾

```
(check (in? 1 (list )) => #f)
(check (in? 1 (list 3 2 1)) => #t)
(check (in? #\x "texmacs") => #t)
(check (in? 1 (vector )) => #f)
(check (in? 1 (vector 3 2 1)) => #t)
(check-catch 'type-error (in? 1 "123"))
```

goldfish/liii/base.scm

△ 7 ▾

```
(define (in? elem l)
  (cond ((list? l) (not (not (member elem l))))
        ((vector? l)
         (let loop ((i (- (vector-length l) 1))
                   (if (< i 0)
                       #f
                       (if (== elem (vector-ref l i))
                           #t
                           (loop (- i 1))))))
          ((and (char? elem) (string? l))
           (in? elem (string->list l)))
          (else (error 'type-error "type_mismatch")))))
```

**let1** <sup>索引</sup>

let的语法嵌套层次太多了，故而引入let1，作为let的单参数版本，简化语法。

goldfish/liii/base.scm

△ 8 ▾

```
(define-macro (let1 name1 value1 . body)
  `(let ((,name1 ,value1))
     ,@body))
```

---

```
tests/goldfish/liii/base-test.scm △ 71 ▽
```

```
(check (let1 x 1 x) => 1)
(check (let1 x 1 (+ x 1)) => 2)
```

---

**identity** 索引 (x) => x

```
goldfish/liii/base.scm △ 9 ▽
```

```
(define identity (lambda (x) x))
```

---

**compose** 索引 (f g ...) => f

```
goldfish/liii/base.scm △ 10 ▽
```

```
(define (compose . fs)
  (if (null? fs)
      (lambda (x) x)
      (lambda (x)
        ((car fs) ((apply compose (cdr fs)) x))))))
```

---

```
tests/goldfish/liii/base-test.scm △ 72 ▽
```

```
(check-true ((compose not zero?) 1))
(check-false ((compose not zero?) 0))
```

---

**lambda\*** 索引

S7 Scheme 内置宏。

```
tests/goldfish/liii/base-test.scm △ 73 ▽
```

```
(let1 add1/add (lambda* (x (y 1)) (+ x y))
  (check (add1/add 1) => 2)
  (check (add1/add 0) => 1)
  (check (add1/add 1 2) => 3))
```

---

**typed-lambda** 索引

实现

```
goldfish/liii/base.scm △ 11 ▽
```

```
; 0 clause BSD, from S7 repo stuff.scm
(define-macro (typed-lambda args . body)
  ; (typed-lambda ((var [type])...) ...)
  (if (symbol? args)
      (apply lambda args body)
      (let ((new-args (copy args)))
        (do ((p new-args (cdr p)))
            ((not (pair? p)))
            (if (pair? (car p))
                (set-car! p (caar p))))
        `(lambda ,new-args
           ,@(map (lambda (arg)
                    (if (pair? arg)
                        `(unless ,(cadr arg) ,(car arg))
                        (error 'type-error
                              "~S is not ~S" ',(car arg) ',(cadr arg)))
                  (values)))
           args)
        ,@body))))
```

---



---

`tests/goldfish/liii/base-test.scm``△ 74 ▾`

```
(define add3
  (typed-lambda
    ((i integer?) (x real?) z)
    (+ i x z)))

(check (add3 1 2 3) => 6)
(check-catch 'type-error (add3 1.2 2 3))
```

---

## 3.20 结尾

---

`goldfish/liii/base.scm``△ 12`

```
) ; end of begin
) ; end of define-library
```

---

---

`tests/goldfish/liii/base-test.scm``△ 75 ▾`

```
(check-report)
```

---



# 第 4 章

## (liii error)

异常的命名参考Python标准库的内置异常。

### 4.1 许可证

`goldfish/liiii/error.scm`

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

`tests/goldfish/liiii/error-test.scm`

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

## 4.2 接口

```
goldfish/liii/error.scm △ 2 ▾
; see https://docs.python.org/3/library/exceptions.html#exception-hierarchy
(define-library (liii error)
  (export ???
    os-error file-not-found-error not-a-directory-error file-exists-error
    timeout-error
    type-error type-error? value-error)
  (begin
```

---

## 4.3 测试

```
tests/goldfish/liii/error-test.scm △ 2 ▾
(import (liii check)
        (liii error)
        (liii base))

(check-set-mode! 'report-failed)
```

---

## 4.4 实现

**os-error** 索引  
系统级别的错误。

```
goldfish/liii/error.scm △ 3 ▾
(define (os-error . args)
  (apply error (cons 'os-error args)))
```

---

```
tests/goldfish/liii/error-test.scm △ 3 ▾
(check-catch 'os-error (os-error))
```

---

**file-not-found-error** 索引  
文件未找到。

```
goldfish/liii/error.scm △ 4 ▾
(define (file-not-found-error . args)
  (apply error (cons 'file-not-found-error args)))
```

---

```
tests/goldfish/liii/error-test.scm △ 4 ▾
(check-catch 'file-not-found-error (file-not-found-error))
```

---

**not-a-directory-error** 索引  
不是一个目录。

```
goldfish/liii/error.scm △ 5 ▾
(define (not-a-directory-error . args)
  (apply error (cons 'not-a-directory-error args)))
```

---

tests/goldfish/liii/error-test.scm

△ 5 ▾

```
(check-catch 'not-a-directory-error (not-a-directory-error))
```

### file-exists-error

文件已存在。

goldfish/liii/error.scm

△ 6 ▾

```
(define (file-exists-error . args)
  (apply error (cons 'file-exists-error args)))
```

tests/goldfish/liii/error-test.scm

△ 6 ▾

```
(check-catch 'file-exists-error (file-exists-error))
```

### timeout-error

超时错误。

goldfish/liii/error.scm

△ 7 ▾

```
(define (timeout-error . args)
  (apply error (cons 'timeout-error args)))
```

tests/goldfish/liii/error-test.scm

△ 7 ▾

```
(check-catch 'timeout-error (timeout-error))
```

### type-error

如果类型不匹配，直接报错。

goldfish/liii/error.scm

△ 8 ▾

```
(define (type-error . args)
  (apply error (cons 'type-error args)))
```

tests/goldfish/liii/error-test.scm

△ 8 ▾

```
(check-catch 'type-error (type-error))
(check-catch 'type-error (type-error "msg"))
(check-catch 'type-error (type-error "msg" "msg2"))
```

### type-error?

goldfish/liii/error.scm

△ 9 ▾

```
(define (type-error? err)
  (in? err '(type-error wrong-type-arg)))
```

tests/goldfish/liii/error-test.scm

△ 9 ▾

```
(check-true (type-error? 'type-error))
(check-true (type-error? 'wrong-type-arg))
```

### value-error

值不正确

goldfish/liii/error.scm

△ 10 ▾

```
(define (value-error . args)
  (apply error (cons 'value-error args)))
```

---

```
tests/goldfish/liii/error-test.scm △ 10 ▾
(check-catch 'value-error (value-error))
```

---

```
???
```

Scala风格未实现错误，一般用于标记为实现的接口。

```
goldfish/liii/error.scm △ 11 ▾
(define (???)
  (error 'not-implemented-error "???"))
```

---

```
tests/goldfish/liii/error-test.scm △ 11 ▾
(check-catch 'not-implemented-error (???)
```

---

## 4.5 结尾

```
goldfish/liii/error.scm △ 12
) ; begin
) ; define-library
```

---

```
tests/goldfish/liii/error-test.scm △ 12
(check-report)
```

---

# 第 5 章

## (liii check)

### 5.1 协议

[goldfish/liii/check.scm](#)

1 ▾

---

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

---

[goldfish/srfi/srfi-78.scm](#)

1 ▾

---

```

; <PLAINTEXT>
; Copyright (c) 2005-2006 Sebastian Egner.
;
; Permission is hereby granted, free of charge, to any person obtaining
; a copy of this software and associated documentation files (the
; 'Software'), to deal in the Software without restriction, including
; without limitation the rights to use, copy, modify, merge, publish,
; distribute, sublicense, and/or sell copies of the Software, and to
; permit persons to whom the Software is furnished to do so, subject to
; the following conditions:
;
; The above copyright notice and this permission notice shall be
; included in all copies or substantial portions of the Software.
;
; THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND,
; EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
; MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
; NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
; LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
; OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
; WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
;
; -----
; Lightweight testing (reference implementation)
; =====
;
; Sebastian.Egner@philips.com
; in R5RS + SRFI 23 (error) + SRFI 42 (comprehensions)
;
; history of this file:
; SE, 25-Oct-2004: first version based on code used in SRFIs 42 and 67
; SE, 19-Jan-2006: (arg ...) made optional in check-ec
;
; Naming convention "check:<identifier>" is used only internally.
;
; Copyright (c) 2024 The Goldfish Scheme Authors
; Follow the same License as the original one

```

---



tests/goldfish/srfi/srfi-78-test.scm

1 ▾

```

;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;

```

---

## 5.2 接口

将 `check:proc` 导出的原因是 `check` 是使用宏实现的，而宏里面用到了 `check:proc`，不导出无法生效。

goldfish/liii/check.scm

△ 2 ▾

```

(define-library (liii check)
  (export test check check-set-mode! check:proc
          check-catch check-report check-failed?
          check-true check-false)
  (import (srfi srfi-78)
          (rename (srfi srfi-78)
                  (check-report srfi-78-check-report)))
  (begin

```

---

goldfish/srfi/srfi-78.scm

△ 2 ▾

```

(define-library (srfi srfi-78)
  (export check check-set-mode! check-report check-reset!
          check-passed? check-failed?
          check:proc)
  (begin

```

---

## 5.3 实现

使用 `display` 作为测试结果的展示函数，比 `write` 好，因为 `display` 可以正常显示文本中的汉字。

goldfish/srfi/srfi-78.scm

△ 3 ▾

```

(define check:write display)

```

---

check-set-mode!

[索引](#)

goldfish/srfi/srfi-78.scm △ 4 ▾

```
(define check:mode #f)

(define (check-set-mode! mode)
  (set! check:mode
    (case mode
      ((off)          0)
      ((summary)     1)
      ((report-failed) 10)
      ((report)      100)
      (else (error "unrecognized_mode" mode))))))
```

将检查模式初始化为 'report':

goldfish/srfi/srfi-78.scm △ 5 ▾

```
(check-set-mode! 'report)
```

以 `check` 函数为例，不同的检查模式下，得到的结果不同。可以使用单元测试来查看这四种测试用例具体使用方法：

tests/goldfish/srfi/srfi-78-test.scm △ 2 ▾

```
(import (srfi srfi-78))
```

**report.** 默认检查模式，无论正确还是错误，都会展示详细的信息。

tests/goldfish/srfi/srfi-78-test.scm △ 3 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;

(display "-----\n")
(display "check_mode:report\n")

(check-set-mode! 'report)

(check (+ 1 2) => 3)

(check (+ 1 2) => 4)

(check-reset!)
```

**off.** 设置为off的时候，禁用后续的测试用例。

```
tests/goldfish/srfi/srfi-78-test.scm △ 4 ▽


---


(display "\n-----\n")
(display "check_mode: off\n")

(check-set-mode! 'off)

(check (+ 1 2) => 3)

(check (+ 1 2) => 4)

(check-reset!)
```

---

**report-failed.** 设置为report-failed的时候，在check正确时只返回正确的测试数量，在check错误时，显示错误。

```
tests/goldfish/srfi/srfi-78-test.scm △ 5 ▽


---


(display "\n-----\n")
(display "check_mode: report-failed\n")

(check-set-mode! 'report-failed)

(check (+ 1 2) => 3)

(check (+ 1 2) => 4)

(check-reset!)
```

---

**summary.** 设置为summary的时候，不汇报错误，需要显示调用check-report才能显示的测试汇总结果，显式调用check:failed显式查看错误的例子。

```
tests/goldfish/srfi/srfi-78-test.scm △ 6 ▽


---


(display "\n-----\n")
(display "check_mode: summary\n")

(check-set-mode! 'summary)

(check (+ 1 2) => 3)

(check (+ 1 2) => 4)

(check-report)
```

---

这个时候如果需要查看错误，那么可以重新设置检查模式，重新检查报告：

```
tests/goldfish/srfi/srfi-78-test.scm △ 7


---


(check-set-mode! 'report)

(check-report)
```

---

goldfish/srfi/srfi-78.scm

△ 6 ▽

---

```

(define check:correct 0)
(define check:failed '())

(define (check-reset!)
  (set! check:correct 0)
  (set! check:failed '()))

(define (check:add-correct!)
  (set! check:correct (+ check:correct 1)))

(define (check:add-failed! expression actual-result expected-result)
  (set! check:failed
    (cons (list expression actual-result expected-result)
          check:failed)))

```

---

goldfish/srfi/srfi-78.scm

△ 7 ▽

---

```

(define (check:report-expression expression)
  (newline)
  (check:write expression)
  (display "␣=>␣"))

(define (check:report-actual-result actual-result)
  (check:write actual-result)
  (display "␣;␣"))

(define (check:report-correct cases)
  (display "correct")
  (if (not (= cases 1))
      (begin (display "␣(")
              (display cases)
              (display "␣cases␣checked"))))
  (newline))

(define (check:report-failed expected-result)
  (display "***␣failed␣***")
  (newline)
  (display ";␣expected␣result:␣")
  (check:write expected-result)
  (newline))

(define (check-passed? expected-total-count)
  (and (= (length check:failed) 0)
       (= check:correct expected-total-count)))

(define (check-failed?)
  (>= (length check:failed) 1))

```

---

## check

索引

check的具体过程，依据不同的检查模式，做不同的处理：

[goldfish/srfi/srfi-78.scm](#)[△ 8 ▾](#)

```
(define (check:proc expression thunk equal expected-result)
  (case check:mode
    ((0) #f)
    ((1)
     (let ((actual-result (thunk)))
       (if (equal actual-result expected-result)
           (check:add-correct!)
           (check:add-failed! expression actual-result expected-result))))
    ((10)
     (let ((actual-result (thunk)))
       (if (equal actual-result expected-result)
           (check:add-correct!)
           (begin
              (check:report-expression expression)
              (check:report-actual-result actual-result)
              (check:report-failed expected-result)
              (check:add-failed! expression actual-result expected-result))))))
    ((100)
     (check:report-expression expression)
     (let ((actual-result (thunk)))
       (check:report-actual-result actual-result)
       (if (equal actual-result expected-result)
           (begin (check:report-correct 1)
                  (check:add-correct!))
           (begin (check:report-failed expected-result)
                  (check:add-failed! expression
                                     actual-result
                                     expected-result))))))
    (else (error "unrecognized_␣check:mode" check:mode))))
```

check的接口实现，使用S7 Scheme的define-macro实现：

[goldfish/srfi/srfi-78.scm](#)[△ 9 ▾](#)

```
(define-macro (check expr => expected)
  `(check:proc ',expr (lambda () ,expr) equal? ,expected))
```

### check-true

[goldfish/liii/check.scm](#)[△ 3 ▾](#)

```
(define-macro (check-true body)
  `(check ,body => #t))
```

### check-false

[goldfish/liii/check.scm](#)[△ 4 ▾](#)

```
(define-macro (check-false body)
  `(check ,body => #f))
```

### check-catch

[goldfish/liii/check.scm](#)[△ 5 ▾](#)

```
(define-macro (check-catch error-id body)
  `(check
   (catch ,error-id
    (lambda () ,body)
    (lambda args ,error-id))
   => ,error-id))
```

`test` 索引 ((obj1 obj2) => boolean)

`test`函数是为了兼容S7 Scheme仓库里面的测试用例。

[goldfish/liii/check.scm](#)

△ 6 ▾

```
(define-macro (test left right)
  `(check ,left => ,right))
```

`check-report` 索引

[goldfish/liii/check.scm](#)

△ 7 ▾

```
(define (check-report . msg)
  (if (not (null? msg))
      (begin
        (display (car msg))))
      (srfi-78-check-report)
      (if (check-failed?) (exit -1)))
```

[goldfish/srfi/srfi-78.scm](#)

△ 10 ▾

```
(define (check-report)
  (if (>= check:mode 1)
      (begin
        (newline)
        (display ";_***_checks_***:_:_")
        (display check:correct)
        (display "_correct,_" )
        (display (length check:failed))
        (display "_failed.")
        (if (or (null? check:failed) (<= check:mode 1))
            (newline)
            (let* ((w (car (reverse check:failed)))
                  (expression (car w))
                  (actual-result (cadr w))
                  (expected-result (caddr w)))
              (display "_First_failed_example:")
              (newline)
              (check:report-expression expression)
              (check:report-actual-result actual-result)
              (check:report-failed expected-result))))))
```

## 5.4 结尾

[goldfish/srfi/srfi-78.scm](#)

△ 11

```
) ; end of begin
) ; end of define-library
```

[goldfish/liii/check.scm](#)

△ 8

```
) ; end of begin
) ; end of define-library
```

# 第 6 章

## (liii case)

case\* 克服了 R7RS 中 case 无法处理字符串等的缺点。

### 6.1 测试

tests/goldfish/liii/case-test.scm

1 ▾

---

```
(import (liii check)
        (liii case))
```

```
(check-set-mode! 'report-failed)
```

---

case\*

索引

tests/goldfish/liii/case-test.scm

△ 2 ▽

```
; 0 clause BSD, from S7 repo s7test.scm
(define (scase x)
  (case* x
    ((a b) 'a-or-b)
    ((1 2/3 3.0) => (lambda (a) (* a 2)))
    ((pi) 1 123)
    (("string1" "string2"))
    ((#<symbol?>) 'symbol!)
    (((+ x #<symbol?>)) 'got-list)
    ((#(1 x 3)) 'got-vector)
    (((+ #<>)) 'empty)
    (((#<x:symbol?> #<x>)) 'got-label)
    (((#<> #<x:> #<x>)) 'repeated)
    (((#<symbol?> #<symbol?>)) 'two)
    (((#<x:> #<x>)) 'pair)
    (((#<x:> #<x>)) 'vector)
    (((#<symbol?> #<...> #<number?>)) 'vectsn)
    (((#<...> #<number?>)) 'vectstart)
    (((#<string?> #<char-whitespace?> #<...>)) 'vectstr)
    (else 'oops)))

(test (scase 3.0) 6.0)
(test (scase 'pi) 123)
(test (scase "string1") "string1")
(test (scase "string3") 'oops)
(test (scase 'a) 'a-or-b)
(test (scase 'abc) 'symbol!)
(test (scase #()) 'oops)
(test (scase '(+ x z)) 'got-list)
(test (scase #(1 x 3)) 'got-vector)
(test (scase '(+ x 3)) 'oops)
(test (scase '(+ x)) 'empty)
(test (scase '(* z z)) 'got-label)
(test (scase '(* z x)) 'oops)
(test (scase '(+ (abs x) (abs x))) 'repeated)
(test (scase '(+ (abs x) (abs y))) 'oops)
(test (scase '(a b)) 'two)
(test (scase '(1 1)) 'pair)
(test (scase '(1 1 2)) 'oops)
(test (scase #(1 1)) 'vector)
(test (scase #(a b c 3)) 'vectsn)
(test (scase #(1 b 2)) 'vectstart)
(test (scase #( "asdf" #\space +nan.0 #<eof>)) 'vectstr)
(test (scase #(a 3)) 'vectsn)
(test (scase #(1)) 'vectstart)
(test (scase #( "asdf" #\space)) 'vectstr)
(test (scase #( "asdf")) 'oops)
```

## 6.2 结尾

tests/goldfish/liii/case-test.scm

△ 3

(check-report)



# 第 7 章

## (liii list) chapter:liii\_list

### 7.1 许可证

goldfish/liiii/list.scm

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

tests/goldfish/liiii/list-test.scm

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

goldfish/srfi/srfi-1.scm

1 ▾

---

```

;;; SRFI-1 list-processing library          -*- Scheme -*-
;;; Reference implementation
;;;
;;; SPDX-License-Identifier: MIT
;;;
;;; Copyright (c) 1998, 1999 by Olin Shivers. You may do as you please with
;;; this code as long as you do not remove this copyright notice or
;;; hold me liable for its use. Please send bug reports to shivers@ai.mit.edu.
;;;      -Olin
;;;
;;;
;;; Copyright (c) 2024 The Goldfish Scheme Authors
;;; Follow the same License as the original one

```

---

## 7.2 接口

Liii List 函数库是金鱼标准库中的 List 函数库，其中的大部分函数来自函数库 (`srfi srfi-1`)，小部分是三鲤自己设计的函数。来自 SRFI 1 的，我们只是在 Liii List 中导出相关函数名，相关实现和单元测试都在 SRFI 1 中维护。

goldfish/liii/list.scm

△ 2 ▾

---

```

(define-library (liii list)
  (export
    ; S7 built-in
    cons car cdr map for-each
    ; SRFI 1: Constructors
    circular-list iota list-copy
    ; SRFI 1: Predicates
    null-list? circular-list? proper-list? dotted-list?
    ; SRFI 1: Selectors
    first second third fourth fifth sixth seventh eighth ninth tenth
    take drop take-right drop-right split-at
    last-pair last
    ; SRFI 1: fold, unfold & map
    count fold fold-right reduce reduce-right
    filter partition remove append-map
    ; SRFI 1: Searching
    find any every list-index
    take-while drop-while
    ; SRFI 1: Deleting
    delete
    ; SRFI 1: Association List
    assoc assq assv alist-cons
    ; Liii List extensions
    list-view flatmap
    list-null? list-not-null? not-null-list?
    length=? length>? length>=? flatten
  )
  (import (srfi srfi-1)
          (liii error))
  (begin

```

---

goldfish/srfi/srfi-1.scm

△ 2 ▽

---

```
(define-library (srfi srfi-1)
  (import (liii error)
          (liii base))
  (export
   ; SRFI 1: Constructors
   circular-list iota list-copy
   ; SRFI 1: Predicates
   circular-list? null-list? proper-list? dotted-list?
   ; SRFI 1: Selectors
   first second third fourth fifth
   sixth seventh eighth ninth tenth
   take drop take-right drop-right count fold fold-right split-at
   reduce reduce-right append-map filter partition remove find
   delete delete-duplicates
   ; SRFI 1: Association List
   assoc assq assv alist-cons
   take-while drop-while list-index any every
   last-pair last)
  (begin
```

---

## 7.3 测试

在金鱼Scheme中的SRFI 1实现需要遵循最小依赖原则，目前`delete-duplicates`是一个复杂度比较高的实现，在SRFI 1中保留，但并不在`(liii list)`导出，故而在本测试文件的开头需要从`(srfi srfi-1)`单独导入。

tests/goldfish/liii/list-test.scm

△ 2 ▽

---

```
(import (liii list)
        (liii check)
        (only (srfi srfi-1) delete-duplicates))

(check-set-mode! 'report-failed)
```

---

## 7.4 SRFI-1

SRFI-1中有一部分函数已经在R7RS的(`scheme base`)库里面了。本节包含了R7RS定义的(`scheme base`)里面和列表相关的函数，这些函数的实现和测试均在`base.scm`和`base-test.scm`中维护。用户可以通过`(import (liii base))`导入这些函数，或者使用`(import (liii list))`导入这些函数。

### 7.4.1 构造器

R7RS 索引  
**cons**

S7 Scheme内置的R7RS中定义的函数。

R7RS 索引  
**list**

S7 Scheme内置的R7RS中定义的函数。

R7RS 索引  
**make-list** `(make-list k [fill]) => list`

S7 Scheme内置的R7RS中定义的函数。

tests/goldfish/liii/base-test.scm

△ 76 ▽

---

```
(check (make-list 3 #\a) => (list #\a #\a #\a))
(check (make-list 3) => (list #f #f #f))

(check (make-list 0) => (list ))
```

---

**iota**[goldfish/srfi/srfi-1.scm](#)

△ 3 ▾

---

```
; 0 clause BSD, from S7 repo stuff.scm
(define* (iota n (start 0) (incr 1))
  (when (not (integer? n))
    (type-error "iota:~n must be a integer"))
  (when (< n 0)
    (value-error "iota:~n must be positive but received ~d" n))
  (let ((lst (make-list n)))
    (do ((p lst (cdr p))
        (i start (+ i incr)))
        ((null? p) lst)
      (set! (car p) i))))
```

---

## 测试用例

[tests/goldfish/liii/list-test.scm](#)

△ 3 ▾

---

```
(check (iota 3) => (list 0 1 2))
(check (iota 3 7) => (list 7 8 9))
(check (iota 2 7 2) => (list 7 9))

(check-catch 'value-error (iota -1))
(check-catch 'type-error (iota 'a))
```

---

**list-copy**[goldfish/scheme/base.scm](#)

△ 39 ▾

---

```
; 0 clause BSD, from S7 repo r7rs.scm
(define list-copy copy)
```

---

## 测试用例

[tests/goldfish/liii/list-test.scm](#)

△ 4 ▾

---

```
;; list-copy tests

;; Check that copying an empty list works as expected
(check (list-copy '()) => '())

;; Check that copying a list of numbers works correctly
(check (list-copy '(1 2 3 4 5)) => '(1 2 3 4 5))

;; Check that copying a list of symbols works correctly
(check (list-copy '(a b c d)) => '(a b c d))

;; Check that copying nested lists works correctly
(check (list-copy '((1 2) (3 4) (5 6))) => '((1 2) (3 4) (5 6)))

;; Check that copying the list does not result in the same object
(check-false (eq? (list-copy '(1 2 3)) '(1 2 3)))

;; Check if list-copy is a deep copy or not
(let ((obj1 '(1 2 3 4))
      (obj2 (list-copy '(1 2 3 4))))
  (check obj1 => obj2)
  (set-car! obj1 3)
  (check-false (eq? obj1 obj2)))
```

---

## 7.4.2 谓词

**pair?** (obj) => bool

**pair?**是S7 Scheme内置的函数：当且仅当obj是序对时，返回真。

tests/goldfish/liii/base-test.scm

△ 77 ▾

```
(check-true (pair? '(a . b)))
(check-true (pair? '(a b c)))

(check-false (pair? '()))
(check-false (pair? '#(a b)))
```

**list?** (obj) => bool

基本列表测试：检查空列表和包含元素的列表是否被识别为列表。

tests/goldfish/liii/base-test.scm

△ 78 ▾

```
(check-true (list? '()))
(check-true (list? '(a)))
(check-true (list? '(a b c)))
(check-true (list? '(1 . 2)))
(check-true (list? '(1 2 . 3)))
```

嵌套列表测试：检查包含其他列表作为元素的列表是否被正确识别。

tests/goldfish/liii/base-test.scm

△ 79 ▾

```
(check-true (list? '((a) (b) (c))))
(check-true (list? '(a (b) c)))
```

循环列表测试：检查函数是否能够正确处理循环列表结构。

tests/goldfish/liii/base-test.scm

△ 80 ▾

```
(check-true (list? (let ((x '(1 2 3))) (set-cdr! (caddr x) x) x)))
```

非列表测试：检查基本数据类型以及向量是否被正确地识别为非列表。

tests/goldfish/liii/base-test.scm

△ 81 ▾

```
(check-false (list? #t))
(check-false (list? #f))
(check-false (list? 123))
(check-false (list? "Hello"))
(check-false (list? '#(1 2 3)))
(check-false (list? '#()))
(check-false (list? '#12345))
```

**null?** (obj) => bool

**null?**是S7 Scheme内置的函数：当且仅当obj是空列表的时候，返回值为真。

tests/goldfish/liii/base-test.scm

△ 82 ▾

```
(check (null? '()) => #t)
(check (null? '(1)) => #f)
(check (null? '(1 2)) => #f)
```

### proper-list?

正规列表 (proper list) 是指满足以下条件的列表：

1. **空列表**：也写作 `()`，它是唯一的一个既是正规列表也是空列表的数据结构。空列表在 Scheme 中被认为是正规列表。

2. **非空列表**: 如果一个列表的每个元素都是通过 `cons` 过程构造的, 并且该列表的 `cdr` 部分是正规列表, 那么这个列表就是正规列表。

上述递归定义中, 该列表通过 `cons` 过程构造保证了列表不会是循环列表。在判断一个列表是否是正规列表时, 我们需要考虑该列表是循环列表的这种情况。

goldfish/srfi/srfi-1.scm

△ 4 ▽

```
(define (proper-list? x)
  (let loop ((x x) (lag x))
    (if (pair? x)
        (let ((x (cdr x))
              (lag (cdr lag)))
          (and (not (eq? x lag)) (loop x lag)))
        (null? x))))
```

tests/goldfish/liii/list-test.scm

△ 5 ▽

```
(check-true (proper-list? (list 1 2)))
(check-true (proper-list? '()))
(check-true (proper-list? '(1 2 3)))

(check-false (proper-list? '(a . b)))
(check-false (proper-list? '(a b . c)))
(check-false (proper-list? (circular-list 1 2 3)))
```

### dotted-list?

点状列表 (dotted list) 是一种列表:

**空的点状列表**. 不是空列表、不是序对的任意 Scheme 对象都是空的点状列表

**非空点状列表**. 其中除了最后一个元素之外, 其他元素的 `cdr` 指向另一个元素, 最后一个元素的 `cdr` 指向一个空的点状列表。

goldfish/srfi/srfi-1.scm

△ 5 ▽

```
(define (dotted-list? x)
  (let loop ((x x) (lag x))
    (if (pair? x)
        (let ((x (cdr x))
              (lag (cdr lag)))
          (and (not (eq? x lag)) (loop x lag)))
        (not (null? x))))
```

tests/goldfish/liii/list-test.scm

△ 6 ▽

```
(check-true (dotted-list? 1))
(check-true (dotted-list? '(1 . 2)))
(check-true (dotted-list? '(1 2 . 3)))

(check-false (dotted-list? (circular-list 1 2 3)))
(check-false (dotted-list? '()))
(check-false (dotted-list? '(a)))
```

**null-list?**

索引

`null-list?` 是一个函数，返回 `#t` 当且仅当参数是空列表。当参数为空列表，返回 `#t`；否则报错。

goldfish/srfi/srfi-1.scm △ 6 ▾

```
(define (null-list? l)
  (cond ((pair? l) #f)
        ((null? l) #t)
        (else
         (error 'wrong-type-arg "null-list?:_argument_out_of_domain" l))))
```

tests/goldfish/liii/list-test.scm △ 7 ▾

```
(check (null-list? '()) => #t)
```

当参数为序对，返回 `#f`。

tests/goldfish/liii/list-test.scm △ 8 ▾

```
(check (null-list? '(1 . 2)) => #f)
```

当参数为非空列表，返回 `#f`。

tests/goldfish/liii/list-test.scm △ 9 ▾

```
(check (null-list? '(1 2)) => #f)
```

当参数既不是序对也不是列表，报错。

辨析：`null?` 在参数为非序对时，不报错，只是返回 `#f`。

tests/goldfish/liii/list-test.scm △ 10 ▾

```
(check (null? 1) => #f)
```

如果已经确定需要判别的对象是列表，使用 `null-list?` 更加合适。`null?` 无法分辨非空的序对和非空的列表，命名上偏模糊，不推荐使用。

### 7.4.3 选择器

R7RS 索引

**car**

一个序对由 `<car>` 部分与 `<cdr>` 部分组成，形如：(`<car>` . `<cdr>`)。 `car` 是 S7 Scheme 内置的 R7RS 定义的函数，用于返回序对的 `<car>` 部分。 `car` 的参数必须是序对，否则报错；特别地，空列表不是序对，报错。

tests/goldfish/liii/base-test.scm △ 83 ▾

```
(check (car '(a b c . d)) => 'a)
(check (car '(a b c)) => 'a)
```

```
(check-catch 'wrong-type-arg (car '()))
```

R7RS 索引

**cdr**

`cdr` 是 S7 Scheme 内置的 R7RS 定义的函数，用于返回序对的 `<cdr>` 部分。 `cdr` 的参数必须是序对，否则报错；特别地，空列表不是序对，报错。

tests/goldfish/liii/base-test.scm △ 84 ▾

```
(check (cdr '(a b c . d)) => '(b c . d))
(check (cdr '(a b c)) => '(b c))
```

```
(check-catch 'wrong-type-arg (cdr '()))
```

**caar** 索引 (caar pair) => obj

caar是S7 Scheme内置的R7RS定义的函数，用于返回序对<car>部分的<car>部分。参数必须是序对，且该序对的<car>部分的内容也要是序对，否则报错。特别地，空列表不是序对，报错。

tests/goldfish/liii/base-test.scm

△ 85 ▽

```
(check (caar '(a . b) . c)) => 'a)
```

```
(check-catch 'wrong-type-arg (caar '(a b . c)))
```

```
(check-catch 'wrong-type-arg (caar '()))
```

**list-ref** 索引

list-ref是S7 Scheme内置的R7RS定义的函数，接收一个列表和一个称为索引值的非负整数k为参数，通过索引值k返回列表的第k个元素（从0开始计数）。当参数为空列表时，报错。当k为负数，报错。当k大于等于列表中元素数时，报错。

注意(cons '(1 2) '(3 4))其实是一个列表，但这个列表的元素不都是整数。这个例子容易理解成结果是一个序对且不是列表。

```
> (cons '(1 2) '(3 4))
```

```
>
```

tests/goldfish/liii/base-test.scm

△ 86 ▽

```
(check (list-ref (cons '(1 2) '(3 4)) 1) => 3)
```

tests/goldfish/liii/base-test.scm

△ 87 ▽

```
(check (list-ref '(a b c) 2) => 'c)
```

```
(check-catch 'wrong-type-arg (list-ref '() 0))
```

```
(check-catch 'out-of-range (list-ref '(a b c) -1))
```

```
(check-catch 'out-of-range (list-ref '(a b c) 3))
```

**first** 索引

first是一个函数，car的同义词，用于返回列表的第1个元素。当列表元素不足1个，报错。

tests/goldfish/liii/list-test.scm

△ 11 ▽

```
(check (first '(1 2 3 4 5 6 7 8 9 10)) => 1)
```

```
(check (first '(left . right)) => 'left)
```

```
(check-catch 'wrong-type-arg (first '()))
```

goldfish/srfi/srfi-1.scm

△ 7 ▽

```
(define first car)
```

**second** 索引

second是一个函数，cadr的同义词，用于返回列表的第2个元素。当列表元素不足2个，报错。

tests/goldfish/liii/list-test.scm

△ 12 ▽

```
(check (second '(1 2 3 4 5 6 7 8 9 10)) => 2)
```

```
(check-catch 'wrong-type-arg (second '(left . right)))
```

```
(check-catch 'wrong-type-arg (second '(1)))
```

goldfish/srfi/srfi-1.scm

△ 8 ▽

```
(define second cadr)
```



**third** 

`third`是一个函数，`caddr`的同义词，用于返回列表的第3个元素。当列表元素不足3个，报错。

[tests/goldfish/liii/list-test.scm](#) △ 13 ▾

```
(check (third '(1 2 3 4 5 6 7 8 9 10)) => 3)
```

```
(check-catch 'wrong-type-arg (third '(1 2)))
```

[goldfish/srfi/srfi-1.scm](#) △ 9 ▾

```
(define third caddr)
```

**fourth** 

[goldfish/srfi/srfi-1.scm](#) △ 10 ▾

```
(define (fourth x) (list-ref x 3))
```

[tests/goldfish/liii/list-test.scm](#) △ 14 ▾

```
(check (fourth '(1 2 3 4 5 6)) => 4)
```

**fifth** 

[goldfish/srfi/srfi-1.scm](#) △ 11 ▾

```
(define (fifth x) (list-ref x 4))
```

[tests/goldfish/liii/list-test.scm](#) △ 15 ▾

```
(check (fifth '(1 2 3 4 5 6 7 8 9 10)) => 5)
```

**sixth** 

[goldfish/srfi/srfi-1.scm](#) △ 12 ▾

```
(define (sixth x) (list-ref x 5))
```

[tests/goldfish/liii/list-test.scm](#) △ 16 ▾

```
(check (sixth '(1 2 3 4 5 6 7 8 9 10)) => 6)
```

**seventh** 

[goldfish/srfi/srfi-1.scm](#) △ 13 ▾

```
(define (seventh x) (list-ref x 6))
```

[tests/goldfish/liii/list-test.scm](#) △ 17 ▾

```
(check (seventh '(1 2 3 4 5 6 7 8 9 10)) => 7)
```

**eighth** 

[goldfish/srfi/srfi-1.scm](#) △ 14 ▾

```
(define (eighth x) (list-ref x 7))
```

[tests/goldfish/liii/list-test.scm](#) △ 18 ▾

```
(check (eighth '(1 2 3 4 5 6 7 8 9 10)) => 8)
```

**ninth** 

---

```
goldfish/srfi/srfi-1.scm △ 15 ▾
(define (ninth x) (list-ref x 8))
```

---

```
tests/goldfish/liii/list-test.scm △ 19 ▾
(check (ninth '(1 2 3 4 5 6 7 8 9 10)) => 9)
```

---

## tenth 索引

**tenth**是一个函数，用于返回列表的第10个元素。当列表元素不足10个，报错。

```
tests/goldfish/liii/list-test.scm △ 20 ▾
(check (tenth '(1 2 3 4 5 6 7 8 9 10)) => 10)
```

---

```
goldfish/srfi/srfi-1.scm △ 16 ▾
(define (tenth x)
  (cadr (cddddr (cddddr x))))
```

---

## take 索引

**take**是一个函数，接收一个列表和一个非负整数k为参数，返回列表的前k个元素组成的新列表。当列表元素数量不足k个，报错。

### 测试

```
tests/goldfish/liii/list-test.scm △ 21 ▾
(check (take '(1 2 3 4) 3) => '(1 2 3))
(check (take '(1 2 3 4) 4) => '(1 2 3 4))
(check (take '(1 2 3 . 4) 3) => '(1 2 3))

(check-catch 'wrong-type-arg (take '(1 2 3 4) 5))
(check-catch 'wrong-type-arg (take '(1 2 3 . 4) 4))
```

---

### 实现

```
goldfish/srfi/srfi-1.scm △ 17 ▾
(define (take l k)
  (let loop ((l l) (k k))
    (if (zero? k)
        '()
        (cons (car l)
              (loop (cdr l) (- k 1))))))
```

---

## list-tail 索引

## drop 索引

**list-tail**是S7 Scheme的内置函数。**drop**是SRFI定义的函数，语义和**list-tail**一致，接收一个列表和一个非负整数k为参数，返回去掉列表前k个元素组成的新列表。当列表元素数量不足k个，报错。

```
tests/goldfish/liii/list-test.scm △ 22 ▾
(check (drop '(1 2 3 4) 2) => '(3 4))
(check (drop '(1 2 3 4) 4) => '())
(check (drop '(1 2 3 . 4) 3) => 4)

(check-catch 'out-of-range (drop '(1 2 3 4) 5))
(check-catch 'out-of-range (drop '(1 2 3 . 4) 4))
```

---

goldfish/srfi/srfi-1.scm

△ 18 ▽

```
(define drop list-tail)
```

**take-right**

索引

**take-right**是一个函数，接收一个列表和一个非负整数k为参数，取出列表的后k个元素组成新列表，返回这个新列表。当列表元素数量不足k个，报错。

tests/goldfish/liii/list-test.scm

△ 23 ▽

```
(check (take-right '(1 2 3 4) 3) => '(2 3 4))
(check (take-right '(1 2 3 4) 4) => '(1 2 3 4))
(check (take-right '(1 2 3 . 4) 3) => '(1 2 3 . 4))

(check-catch 'out-of-range (take-right '(1 2 3 4) 5))
(check-catch 'out-of-range (take-right '(1 2 3 . 4) 4))
```

goldfish/srfi/srfi-1.scm

△ 19 ▽

```
(define (take-right l k)
  (let loop ((lag l)
            (lead (drop l k)))
    (if (pair? lead)
        (loop (cdr lag) (cdr lead))
        lag)))
```

**drop-right**

索引

**drop-right**是一个函数，接收一个列表和一个非负整数k为参数，去掉列表的后k个元素组成新列表，返回这个新列表。当列表元素数量不足k个，报错。当k为负数，报错。

goldfish/srfi/srfi-1.scm

△ 20 ▽

```
(define (drop-right l k)
  (let loop ((lag l) (lead (drop l k)))
    (if (pair? lead)
        (cons (car lag) (loop (cdr lag) (cdr lead)))
        '())))
```

tests/goldfish/liii/list-test.scm

△ 24 ▽

```
(check (drop-right '(1 2 3 4) 2) => '(1 2))
(check (drop-right '(1 2 3 4) 4) => '())
(check (drop-right '(1 2 3 . 4) 3) => '())

(check-catch 'out-of-range (drop-right '(1 2 3 4) 5))
(check-catch 'out-of-range (drop-right '(1 2 3 4) -1))
(check-catch 'out-of-range (drop-right '(1 2 3 . 4) 4))
```

**split-at** (lst i) => (values taked dropped)

索引

lst. 列表，需要被分割的列表。

i. 整数，指定分割的位置。

(values taked dropped). 两个值，第一个是包含原列表前 i 个元素的列表，第二个是原列表剩余元素的列表。

将一个列表分割成两部分，返回两个值。第一个值是包含原列表前 i 个元素的列表，第二个值是原列表剩余元素的列表。

实现细节：该函数通过迭代方式工作。它遍历列表 `lst`，每次迭代将当前元素添加到结果列表中，直到达到指定的分割位置 `i`。如果 `i` 为 0，函数返回空列表和原始列表。如果 `i` 大于列表长度或小于 0，函数报错。函数使用 `values` 返回两个值，这样可以同时返回两个列表。

goldfish/srfi/srfi-1.scm

△ 21 ▽

```
(define (split-at lst i)
  (when (< i 0)
    (value-error "require a index greater than 0, but got ~A--" i))
  (let ((result (cons #f '())))
    (do ((j i (- j 1))
        (rest lst (cdr rest))
        (node result (cdr node)))
      ((zero? j)
       (values (cdr result) rest))
      (when (not (pair? rest))
        (value-error "list length cannot be greater than i, where lst is ~A, but i is ~A--" lst i))
      (set-cdr! node (cons (car rest) '())))))
```

tests/goldfish/liii/list-test.scm

△ 25 ▽

```
(check (list (split-at '(1 2 3 4 5) 3)) => '((1 2 3) (4 5)))
(check (list (split-at '(1 2 3 4 5) 0)) => '(() (1 2 3 4 5)))

(check-catch 'value-error (split-at '(1 2 3 4 5) 10))
(check-catch 'value-error (split-at '(1 2 3 4 5) -1))

(check (list (split-at '(1 2 3 4 . 5) 0)) => '(() (1 2 3 4 . 5)))
(check (list (split-at '(1 2 3 4 . 5) 3)) => '((1 2 3) (4 . 5)))
(check (list (split-at '(1 2 3 4 . 5) 4)) => '((1 2 3 4) 5))

(check-catch 'value-error (split-at '(1 2 3 4 . 5) 10))
(check-catch 'value-error (split-at '(1 2 3 4 . 5) -1))

(check (list (split-at '() 0)) => '(() ()))
(check-catch 'value-error (split-at '() 10))
(check-catch 'value-error (split-at '() -1))
```

## last-pair

索引

`last-pair` 是一个函数，以序对形式返回列表的最后一个元素，参数必须是序对，空列表报错。

goldfish/srfi/srfi-1.scm

△ 22 ▽

```
(define (last-pair l)
  (if (pair? (cdr l))
      (last-pair (cdr l))
      l))
```

tests/goldfish/liii/list-test.scm

△ 26 ▽

```
(check (last-pair '(a b c)) => '(c))
(check (last-pair '(c)) => '(c))

(check (last-pair '(a b . c)) => '(b . c))
(check (last-pair '(b . c)) => '(b . c))

(check-catch 'wrong-type-arg (last-pair '()))
```

**last**

索引

`last`是一个函数，以符号形式返回列表的最后一个元素，参数必须是序对，空列表报错。

goldfish/srfi/srfi-1.scm

△ 23 ▾

```
(define (last l)
  (car (last-pair l)))
```

tests/goldfish/liii/list-test.scm

△ 27 ▾

```
(check (last '(a b c)) => 'c)
(check (last '(c)) => 'c)

(check (last '(a b . c)) => 'b)
(check (last '(b . c)) => 'b)

(check-catch 'wrong-type-arg (last '()))
```

## 7.4.4 常用函数

R7RS

**length** (lst) -> integer

索引

`length`是一个S7内置函数，它接收一个列表为参数，返回该列表中元素的数量。如果参数不是列表，返回0。

tests/goldfish/liii/base-test.scm

△ 88 ▾

```
(check (length ()) => 0)
(check (length '(a b c)) => 3)
(check (length '(a (b) (c d e))) => 3)

(check (length 2) => #f)
(check (length '(a . b)) => -1)
```

R7RS

**append**

索引

`append`是一个S7内置的R7RS定义的函数，它接收多个列表为参数，按顺序拼接在一起，返回一个新的列表。`append`没有参数时，返回空列表。

tests/goldfish/liii/base-test.scm

△ 89 ▾

```
(check (append '(a) '(b c d)) => '(a b c d))
(check (append '(a b) 'c) => '(a b . c))

(check (append () 'c) => 'c)
(check (append) => '())
```

R7RS

**reverse**

索引

tests/goldfish/liii/base-test.scm

△ 90 ▾

```
(check (reverse '()) => '())
(check (reverse '(a)) => '(a))
(check (reverse '(a b)) => '(b a))
```

**count**

索引

`count`是一个高阶函数，它接收两个参数：一个谓词和一个列表；返回满足谓词条件的元素在列表中出现的次数。

---

```
goldfish/srifi/srifi-1.scm △ 24 ▾
(define (count pred list1 . lists)
  (let lp ((lis list1) (i 0))
    (if (null-list? lis) i
        (lp (cdr lis) (if (pred (car lis)) (+ i 1) i)))))
```

---

```
tests/goldfish/liii/list-test.scm △ 28 ▾
(check (count even? '(3 1 4 1 5 9 2 5 6)) => 3)
```

---

### 7.4.5 折叠和映射

R7RS 索引  
**map**

`map`是S7的内置高阶函数，它接收一个函数和一个列表为从参数，把该函数应用于该列表的每个元素上，并返回一个新列表。

---

```
tests/goldfish/liii/base-test.scm △ 91 ▾
(check (map square (list 1 2 3 4 5)) => '(1 4 9 16 25))
```

---

R7RS 索引  
**for-each** (`for-each proc list1 [list2 ...]`)

`for-each`是S7内置的高阶函数。`for-each`的参数与`map`的参数类似，但`for-each`调用`proc`是为了它的副作用，而不是为了它的返回值。与`map`不同，`for-each`保证会按照从第一个元素到最后一个元素的顺序调用`proc`，并且`for-each`返回的值是未指定的。如果给出了多个列表，并且不是所有列表的长度都相同，`for-each`会在最短的列表用尽时终止。当`proc`不接受与`lists`数量相同的参数，报错。

---

```
tests/goldfish/liii/base-test.scm △ 92 ▾
(check
  (let ((v (make-vector 5)))
    (for-each (lambda (i) (vector-set! v i (* i i)))
              (iota 5))
    v)
  => #(0 1 4 9 16))

(check
  (let ((v (make-vector 5 #f)))
    (for-each (lambda (i) (vector-set! v i (* i i)))
              (iota 4))
    v)
  => #(0 1 4 9 #f))

(check
  (let ((v (make-vector 5 #f)))
    (for-each (lambda (i) (vector-set! v i (* i i)))
              (iota 0))
    v)
  => #( #f #f #f #f #f ))
```

---

## fold

索引

`fold`是一个高阶函数，它接受三个参数：一个函数、一个初始值和一个列表，将函数累积地应用到一个列表的所有元素上，从左到右，从而将列表折叠成一个单一的值。

goldfish/srfi/srfi-1.scm

△ 25 ▽

```
(define (fold f initial l)
  (when (not (procedure? f))
    (error 'type-error "The first param must be a procedure"))
  (if (null? l)
      initial
      (fold f
            (f (car l) initial)
            (cdr l))))
```

这是SRFI-1官方提供的实现，我们暂时不用。

```
(define (fold kons knil lis1 . lists)
  (if (pair? lists)
      (let lp ((lists (cons lis1 lists)) (ans knil))
        (receive (cars+ans cdrs) (%cars+cdrs+ lists ans)
          (if (null? cars+ans) ans
              (lp cdrs (apply kons cars+ans)))))
      (let lp ((lis lis1) (ans knil))
        (if (null-list? lis) ans
            (lp (cdr lis) (kons (car lis) ans))))))
```

常见的用法：

从初始值开始，依次累加列表中的元素，返回一个数；当列表为空列表时，返回初始值。

tests/goldfish/liii/list-test.scm

△ 29 ▽

```
(check (fold + 0 '(1 2 3 4)) => 10)
(check (fold + 0 '()) => 0)

(check-catch 'type-error (fold 0 + '(1 2 3 4)))
```

反转列表中的元素，返回一个新列表。

tests/goldfish/liii/list-test.scm

△ 30 ▽

```
(check (fold cons () '(1 2 3 4)) => '(4 3 2 1))
```

统计列表中满足谓词的元素数量，返回这个数量。

tests/goldfish/liii/list-test.scm

△ 31 ▽

```
(check
  (fold (lambda (x count) (if (symbol? x) (+ count 1) count))
        0
        '(a b 1 2 3 4))
=> 2)
```

## fold-right

索引

`fold-right`与`fold`类似，不同的是，`fold-right`是从右到左折叠。

goldfish/srfi/srfi-1.scm

△ 26 ▾

---

```
(define (fold-right f initial l)
  (if (null? l)
      initial
      (f (car l)
         (fold-right f
                     initial
                     (cdr l))))))
```

---

这是SRFI-1官方提供的实现，我们暂时不用：

```
(define (fold-right kons knil lis1 . lists)
  (if (pair? lists)
      (let recur ((lists (cons lis1 lists)))
        (let ((cdrs (%cdrs lists)))
          (if (null? cdrs) knil
              (apply kons (%cars+ lists (recur cdrs))))))
      (let recur ((lis lis1))
        (if (null-list? lis) knil
            (let ((head (car lis)))
              (kons head (recur (cdr lis))))))))))
```

在用作累加、统计时，`fold-right`与`fold`的结果是相同的。

tests/goldfish/liii/list-test.scm

△ 32 ▾

---

```
(check (fold-right + 0 '(1 2 3 4)) => 10)
```

```
(check (fold-right + 0 '()) => 0)
```

```
(check
  (fold-right (lambda (x count) (if (symbol? x) (+ count 1) count))
              0
              '(a b 1 2 3 4))
=>
2)
```

---

但`fold-right`与`fold`的折叠方向是相反的，这就使得列表原本的顺序得以保持，不会反转。

tests/goldfish/liii/list-test.scm

△ 33 ▾

---

```
(check (fold-right cons () '(1 2 3 4)) => '(1 2 3 4))
```

---

## reduce 索引

`reduce`与`fold`类似，但有微妙且关键的不同。只有在列表为空列表时，才会使用这个初始值。在列表不是空列表时，则把列表的`<car>`部分取出作为`fold`的初始值，又把列表的`<cdr>`部分取出作为`fold`的列表。

goldfish/srfi/srfi-1.scm

△ 27 ▾

---

```
(define (reduce f initial l)
  (if (null-list? l) initial
      (fold f (car l) (cdr l))))
```

---

在用作累加时，`reduce`与`fold`的结果是相同的。

tests/goldfish/liii/list-test.scm

△ 34 ▾

---

```
(check (reduce + 0 '(1 2 3 4)) => 10)
```

```
(check (reduce + 0 '()) => 0)
```

---



不适用于反转列表中的元素，但当列表非空，返回的不再是列表，而是序对。因为`reduce`会把非空列表的第一个元素取出来作为`fold`的初始值。

```
tests/goldfish/liii/list-test.scm
```

△ 35 ▾

```
(check (reduce cons () '(1 2 3 4)) => '(4 3 2 . 1))
```

不适用于统计列表中满足谓词的元素数量，因为`reduce`会把非空列表的第一个元素取出来作为`fold`的初始值，引发错误。

```
tests/goldfish/liii/list-test.scm
```

△ 36 ▾

```
(check-catch 'wrong-type-arg
  (reduce (lambda (x count) (if (symbol? x) (+ count 1) count))
    0
    '(a b 1 2 3 4)))
```

## reduce-right

索引

`reduce-right`与`reduce`类似，不同的是，`reduce-right`是从右到左规约。

```
goldfish/srfi/srfi-1.scm
```

△ 28 ▾

```
(define (reduce-right f initial l)
  (if (null-list? l) initial
      (let recur ((head (car l)) (l (cdr l)))
        (if (pair? l)
            (f head (recur (car l) (cdr l)))
            head))))
```

在用作累加时，`reduce-right`与`fold`的结果是相同的。

```
tests/goldfish/liii/list-test.scm
```

△ 37 ▾

```
(check (reduce-right + 0 '(1 2 3 4)) => 10)
```

```
(check (reduce-right + 0 '()) => 0)
```

也不适用于重列列表中的元素，以及统计列表中满足谓词的元素数量。

```
tests/goldfish/liii/list-test.scm
```

△ 38 ▾

```
(check (reduce-right cons () '(1 2 3 4))
  => '(1 2 3 . 4) )

(check
  (reduce-right (lambda (x count) (if (symbol? x) (+ count 1) count))
    0
    '(a b 1 2 3 4))
  => 6)
```

## append-map

索引

```
goldfish/srfi/srfi-1.scm
```

△ 29 ▾

```
(define append-map
  (typed-lambda ((proc procedure?) (lst list?))
    (let loop ((rest lst)
              (result '()))
      (if (null? rest)
          result
          (loop (cdr rest)
                (append result (proc (car rest)))))))
```

## 7.4.6 过滤和分组

### filter 索引

`filter`是一个高阶函数，接收一个谓词和一个列表为参数，从这个列表中筛出满足谓词的元素，组成一个新列表，返回这个新列表。

goldfish/srfi/srfi-1.scm △ 30 ▾

```
(define (filter pred l)
  (let recur ((l l))
    (if (null-list? l) l
        (let ((head (car l))
              (tail (cdr l)))
          (if (pred head)
              (let ((new-tail (recur tail)))
                (if (eq? tail new-tail) l
                    (cons head new-tail)))
              (recur tail))))))
```

tests/goldfish/liii/list-test.scm △ 39 ▾

```
(check (filter even? '(-2 -1 0 1 2)) => '(-2 0 2))
```

### partition 索引

`partition`是一个高阶函数，接收一个谓词和一个列表为参数，从这个列表中分别筛出满足和不满足谓词的元素，各组成一个新列表，返回以这两个新列表组成的序对。

goldfish/srfi/srfi-1.scm △ 31 ▾

```
(define (partition pred l)
  (let loop ((lst l) (satisfies '()) (dissatisfies '()))
    (cond ((null? lst)
           (cons satisfies dissatisfies))
          ((pred (car lst))
           (loop (cdr lst) (cons (car lst) satisfies) dissatisfies))
          (else
           (loop (cdr lst) satisfies (cons (car lst) dissatisfies))))))
```

tests/goldfish/liii/list-test.scm △ 40 ▾

```
(check
 (partition symbol? '(one 2 3 four five 6))
 => (cons '(five four one) '(6 3 2)))
```

### remove 索引

`remove`是一个高阶函数，接收一个谓词和一个列表为参数，从这个列表中去掉满足谓词的元素，组成一个新列表，返回这个新列表。

goldfish/srfi/srfi-1.scm △ 32 ▾

```
(define (remove pred l)
  (filter (lambda (x) (not (pred x))) l))
```

tests/goldfish/liii/list-test.scm △ 41 ▾

```
(check (remove even? '(-2 -1 0 1 2)) => '(-1 1))
```

## 7.4.7 搜索

### memq

索引

`memq`是一个S7内置函数，和`member`类似，但判断元素等价的谓词是`eq?`。也就是说，检查的是两个元素在是否是同一个实例，即它们是否具有相同的内存地址。

对比使用`equal?`的`member`或使用`eqv?`的`memv`，这种检查最为“严格”，适用于判断的元素类型最少，速度最快，适用于判断布尔值（`#t`、`#f`）、符号、整数（浮点数和复数不行）、函数（的值）这些类型的元素是否在列表中。

```
tests/goldfish/liii/base-test.scm △ 93 ▾


---


(check (memq #f '(1 #f 2 3)) => '(#f 2 3))
(check (memq 'a '(1 a 2 3)) => '(a 2 3))
(check (memq 2 '(1 2 3)) => '(2 3))

(check (memq 2.0 '(1 2.0 3)) => #f)
(check (memq 2+0i '(1 2+0i 3)) => #f)

(define num1 3)
(define num2 2)
(check (memq num1 '(3 num2)) => '(3 num2))
(check (memq 3 '(num1 num2)) => #f)
(check (memq 'num1 '(num1 num2)) => '(num1 num2))

(check (memq (+ 1 1) '(1 2 3)) => '(2 3))
```

### memv

索引

`memv`是一个S7内置函数，和`member`类似，但判断元素元素的谓词是`eqv?`。也就是说，检查的是两个元素是否相同或在数值上等价。

比使用`equal?`的`member`“严格”，但比使用`eq?`的`memq`“宽松”，适用于判断的元素类型更全（`memv`能实现的功能`member`都能实现），速度中等，适用于判断数值类元素（整数、浮点数、复数）是否在列表中。但是注意，即使数值相同也不视为同一元素。

```
tests/goldfish/liii/base-test.scm △ 94 ▾


---


(check (memv 2 '(1 2 3)) => '(2 3))
(check (memv 2.0 '(1 2.0 3)) => '(2.0 3))
(check (memv 2+0i '(1 2+0i 3)) => '(2+0i 3))

(check (memv 2 '(1 2.0 3)) => #f)
(check (memv 2 '(1 2+0i 3)) => #f)
```

### member

索引

`member`是一个S7内置函数，接收一个元素和一个列表为参数，返回包含该元素的第一个子列表。当元素不在列表中，返回`#f`。当列表为空列表，返回`#f`。

```
tests/goldfish/liii/base-test.scm △ 95 ▾


---


(check (member 2 '(1 2 3)) => '(2 3))
(check (member 0 '(1 2 3)) => #f)
(check (member 0 '()) => #f)
```

注意，判断两个元素等价的谓词是`equal?`。也就是说，检查的是两个元素在结构和内容上是否等价。

对比使用`eq?`的`memq`或使用`eqv?`的`memv`，这种检查最为“宽松”，适用判断的元素类型最全，但速度最慢（因为它会递归地比较复合数据结构的每个部分），建议用于判断字符串、序对、列表这些类型的元素是否在列表中，如果无需判断这些类型，建议选用`memv`或`memq`。

tests/goldfish/liii/base-test.scm △ 96 ▾

```
(check (member "1" '(0 "1" 2 3)) => '("1" 2 3))
(check (member '(1 . 2) '(0 (1 . 2) 3)) => '((1 . 2) 3))
(check (member '(1 2) '(0 (1 2) 3)) => '((1 2) 3))
```

## find 索引

`find`是一个高阶函数，接收一个谓词和一个列表为参数，返回该列表中第一个满足谓词的元素。当列表为空列表，或列表中没有满足谓词的元素，返回`#f`。

goldfish/srfi/srfi-1.scm △ 33 ▾

```
(define (find pred l)
  (cond ((null? l) #f)
        ((pred (car l)) (car l))
        (else (find pred (cdr l)))))
```

tests/goldfish/liii/list-test.scm △ 42 ▾

```
(check (find even? '(3 1 4 1 5 9)) => 4)

(check (find even? '()) => #f)

(check (find even? '(1 3 5 7 9)) => #f)
```

## take-while 索引

`take-while`是一个高阶函数，接收一个谓词和一个列表为参数，按列表顺序筛出满足谓词的元素，直到不满足谓词的那个一个就停止筛选，返回筛出的元素组成的列表。

goldfish/srfi/srfi-1.scm △ 34 ▾

```
(define (take-while pred lst)
  (if (null? lst)
      '()
      (if (pred (car lst))
          (cons (car lst) (take-while pred (cdr lst)))
          '())))
```

当参数的列表为空列表，无论谓词是什么都返回空列表。

tests/goldfish/liii/list-test.scm △ 43 ▾

```
(check (take-while even? '()) => '())
```

当列表中所有元素都满足谓词，返回原列表。

tests/goldfish/liii/list-test.scm △ 44 ▾

```
(check (take-while (lambda (x) #t) '(1 2 3))
      => '(1 2 3))
```

当列表中没有元素满足谓词，返回空列表。

tests/goldfish/liii/list-test.scm △ 45 ▾

```
(check
  (take-while (lambda (x) #f) '(1 2 3))
  => '())
```

当列表的第一个元素就不满足谓词，返回空列表。

```
tests/goldfish/liii/list-test.scm △ 46 ▾
(check
  (take-while (lambda (x) (not (= x 1))) '(1 2 3))
  => '())
```

筛出元素的过程按照列表的顺序进行，当一个元素已经不再满足谓词，那么这个元素之后的元素不会被筛出。

```
tests/goldfish/liii/list-test.scm △ 47 ▾
(check
  (take-while (lambda (x) (< x 3)) '(1 2 3 0))
  => '(1 2))
```

### drop-while 索引

`drop-while`是一个高阶函数，接收一个谓词和一个列表为参数，按列表顺序丢掉满足谓词的元素，直到不满足谓词的那个一个就停止丢掉，返回剩下的元素组成的列表。

```
goldfish/srfi/srfi-1.scm △ 35 ▾
(define (drop-while pred l)
  (if (null? l)
      '()
      (if (pred (car l))
          (drop-while pred (cdr l))
          l)))
```

当列表为空列表，返回空列表。

```
tests/goldfish/liii/list-test.scm △ 48 ▾
(check (drop-while even? '()) => '())
```

当列表中所有元素都满足谓词，返回空列表。

```
tests/goldfish/liii/list-test.scm △ 49 ▾
(check (drop-while (lambda (x) #t) '(1 2 3)) => '())
```

当列表中没有元素满足谓词，返回原列表。

```
tests/goldfish/liii/list-test.scm △ 50 ▾
(check (drop-while (lambda (x) #f) '(1 2 3)) => '(1 2 3))
```

当列表的第一个元素就不满足谓词，返回原列表。

```
tests/goldfish/liii/list-test.scm △ 51 ▾
(check
  (drop-while (lambda (x) (not (= x 1))) '(1 2 3))
  => '(1 2 3))
```

### list-index 索引

`list-index`是一个高阶函数，接收一个谓词和一个列表为参数，返回第一个符合谓词要求元素的位置索引。当列表为空列表，或列表中没有满足谓词的元素，返回`#f`。

goldfish/srfi/srfi-1.scm

△ 36 ▾

---

```
(define (list-index pred l)
  (let loop ((index 0) (l l))
    (if (null? l)
        #f
        (if (pred (car l))
            index
            (loop (+ index 1) (cdr l))))))
```

---

tests/goldfish/liii/list-test.scm

△ 52 ▾

---

```
(check (list-index even? '(3 1 4 1 5 9)) => 2)
(check (list-index even? '()) => #f)
(check (list-index even? '(1 3 5 7 9)) => #f)
```

---

any

索引

goldfish/srfi/srfi-1.scm

△ 37 ▾

---

```
(define (any pred? l)
  (cond ((null? l) #f)
        ((pred? (car l)) #t)
        (else (any pred? (cdr l)))))
```

---

tests/goldfish/liii/list-test.scm

△ 53 ▾

---

```
(check (any integer? '()) => #f)
(check (any integer? '(a 3.14 "3")) => #f)
(check (any integer? '(a 3.14 3)) => #t)
```

---

every

索引

goldfish/srfi/srfi-1.scm

△ 38 ▾

---

```
(define (every pred? l)
  (cond ((null? l) #t)
        ((not (pred? (car l))) #f)
        (else (every pred? (cdr l)))))
```

---

tests/goldfish/liii/list-test.scm

△ 54 ▾

---

```
(check (every integer? '()) => #t)
(check (every integer? '(a 3.14 3)) => #f)
(check (every integer? '(1 2 3)) => #t)
```

---

## 7.4.8 删除

公共子函数，用于处理可选的maybe-equal参数。

goldfish/srfi/srfi-1.scm

△ 39 ▾

---

```
(define (%extract-maybe-equal maybe-equal)
  (let ((my-equal (if (null-list? maybe-equal)
                     =
                     (car maybe-equal))))
    (if (procedure? my-equal)
        my-equal
        (error 'wrong-type-arg "maybe-equal_must_be_procedure"))))
```

---

delete

索引

goldfish/srfi/srfi-1.scm

△ 40 ▾

---

```
(define (delete x l . maybe-equal)
  (let ((my-equal (%extract-maybe-equal maybe-equal)))
    (filter (lambda (y) (not (my-equal x y))) l)))
```

---

## 测试用例

tests/goldfish/liii/list-test.scm

△ 55 ▾

```
(check (delete 1 (list 1 2 3 4)) => (list 2 3 4))

(check (delete 0 (list 1 2 3 4)) => (list 1 2 3 4))

(check (delete #\a (list #\a #\b #\c) char=?)
=> (list #\b #\c))

(check (delete #\a (list #\a #\b #\c) (lambda (x y) #f))
=> (list #\a #\b #\c))

(check (delete 1 (list )) => (list ))

(check
  (catch 'wrong-type-arg
    (lambda ()
      (check (delete 1 (list 1 2 3 4) 'not-pred) => 1))
    (lambda args #t))
=> #t)
```

---

## delete-duplicates

索引

goldfish/srfi/srfi-1.scm

△ 41 ▾

```
;;; right-duplicate deletion
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; delete-duplicates delete-duplicates!
;;;
;;; Beware -- these are N^2 algorithms. To efficiently remove duplicates
;;; in long lists, sort the list to bring duplicates together, then use a
;;; linear-time algorithm to kill the dups. Or use an algorithm based on
;;; element-marking. The former gives you O(n lg n), the latter is linear.

(define (delete-duplicates lis . maybe-equal)
  (let ((my-equal (%extract-maybe-equal maybe-equal)))
    (let recur ((lis lis))
      (if (null-list? lis)
          lis
          (let* ((x (car lis))
                 (tail (cdr lis))
                 (new-tail (recur (delete x tail my-equal))))
            (if (eq? tail new-tail)
                lis
                (cons x new-tail)))))))
```

---

## 测试用例

tests/goldfish/liii/list-test.scm △ 56 ▾

```
(check (delete-duplicates (list 1 1 2 3)) => (list 1 2 3))
(check (delete-duplicates (list 1 2 3)) => (list 1 2 3))
(check (delete-duplicates (list 1 1 1)) => (list 1))
```

```
(check (delete-duplicates (list )) => (list ))
```

```
(check (delete-duplicates (list 1 1 2 3) (lambda (x y) #f))
=> (list 1 1 2 3))
```

如果判断相等的函数类型不正确，会报错：

tests/goldfish/liii/list-test.scm △ 57 ▾

```
(check
  (catch 'wrong-type-arg
    (lambda
      ()
      (check (delete-duplicates (list 1 1 2 3) 'not-pred) => 1))
    (lambda args #t))
  => #t)
```

## 7.4.9 关联列表

关联列表 (association list) 的每一个元素都是序对。

R7RS 索引  
**assoc**  
R7RS 索引  
**assq**  
R7RS 索引  
**assv**

tests/goldfish/liii/list-test.scm △ 58 ▾

```
(let1 l '((a 1) (b 2) (c . 3))
  (check (assq 'a 1) => '(a 1))
  (check-true (eq? (assq 'a 1) (1 0)))
  (check (assq 'b 1) => '(b 2))
  (check (assq 'c 1) => '(c . 3))
  (check (assq 'd 1) => #f))
```

tests/goldfish/liii/list-test.scm △ 59 ▾

```
(let1 l '((2 3) (5 7) (11 . 13))
  (check (assv 5 1) => '(5 7))
  (check (assv 11 1) => '(11 . 13)))
```

tests/goldfish/liii/list-test.scm △ 60 ▾

```
(let1 l '(((a)) ((b)) ((c)))
  (check (assoc '(a) 1) => '(((a))))
  (check (assq '(a) 1) => #f)
  (check (assv '(a) 1) => #f))
```

**alist-cons** 索引

goldfish/srfi/srfi-1.scm △ 42 ▾

```
(define (alist-cons key value alist)
  (cons (cons key value) alist))
```



tests/goldfish/liii/list-test.scm

△ 61 ▾

```
(check (alist-cons 'a 1 '()) => '((a . 1)))
(check (alist-cons 'a 1 '((b . 2))) => '((a . 1) (b . 2)))
```

## 7.4.10 循环列表

### circular-list

索引

使用该构造器构造的列表是循环列表，每一个循环的单元包含构造器中的所有参数。

goldfish/srfi/srfi-1.scm

△ 43 ▾

```
(define (circular-list val1 . vals)
  (let ((ans (cons val1 vals)))
    (set-cdr! (last-pair ans) ans)
    ans))
```

tests/goldfish/liii/list-test.scm

△ 62 ▾

```
(let1 cl (circular-list 1 2 3)
  (check (cl 3) => 1)
  (check (cl 4) => 2)
  (check (cl 5) => 3)
  (check (cl 6) => 1))
```

### circular-list?

索引

判断一个列表是不是循环列表，这里采用的是判断链表这个数据结构的是否循环的经典算法，使用两个指针，第一个指针每轮迭代移动两次，第二个指针每轮迭代移动一次，如果两个指针在某一次迭代中指向的是同一个位置，那么该列表就是循环列表。

goldfish/srfi/srfi-1.scm

△ 44 ▾

```
(define (circular-list? x)
  (let loop ((x x) (lag x))
    (and (pair? x)
         (let ((x (cdr x)))
           (and (pair? x)
                (let ((x (cdr x))
                      (lag (cdr lag)))
                  (or (eq? x lag) (loop x lag))))))))
```

### 测试

tests/goldfish/liii/list-test.scm

△ 63 ▾

```
(check-true (circular-list? (circular-list 1 2)))
(check-true (circular-list? (circular-list 1)))

(let* ((l (list 1 2 3))
       (end (last-pair l)))
  (set-cdr! end (cdr l))
  (check-true (circular-list? l)))

(check-false (circular-list? (list 1 2)))
```

## 7.5 三鲤扩展函数

length=? <sup>索引</sup> (x l) => boolean

x. 期望的列表长度，如果长度为负数，该函数会抛出 **value-error**

## 1. 列表

快速判断一个列表`l`的长度是否为`x`。由于`(= x (length l))`这种判断方式的复杂度是 $O(n)$ ，故而需要`length=?`这种快速的判断方式。

tests/goldfish/liii/list-test.scm △ 64 ▽

```
(check-true (length=? 3 (list 1 2 3)))
(check-false (length=? 2 (list 1 2 3)))
(check-false (length=? 4 (list 1 2 3)))

(check-true (length=? 0 (list )))
(check-catch 'value-error (length=? -1 (list )))
```

goldfish/liii/list.scm △ 3 ▽

```
(define (length=? x scheme-list)
  (when (< x 0)
    (value-error "length=?:_expected_non-negative_integer_x_but_received_~d" x))
  (cond ((and (= x 0) (null? scheme-list)) #t)
        ((or (= x 0) (null? scheme-list)) #f)
        (else (length=? (- x 1) (cdr scheme-list)))))
```

`length>?` (lst len) => bool 索引

使用贪心策略，先访问列表的前`len`个元素，如果列表长度不大于`len`，那么返回布尔值假，否则返回布尔值真。`lst`并不一定是严格意义上的列表（最后一个元素是空列表），也可能是序对。

goldfish/liii/list.scm △ 4 ▽

```
(define (length>? lst len)
  (let loop ((lst lst)
            (cnt 0))
    (cond ((null? lst) (< len cnt))
          ((pair? lst) (loop (cdr lst) (+ cnt 1)))
          (else (< len cnt)))))
```

tests/goldfish/liii/list-test.scm △ 65 ▽

```
(check-true (length>? '(1 2 3 4 5) 3))
(check-false (length>? '(1 2) 3))
(check-false (length>? '() 0))

(check-true (length>? '(1) 0))
(check-false (length>? '() 1))

(check-false (length>? '(1 2 . 3) 2))
(check-true (length>? '(1 2 . 3) 1))
```

`length>=?` (lst len) => bool 索引

使用贪心策略，先访问列表的前`len`个元素，如果列表长度小于`len`，那么返回布尔值假，否则返回布尔值真。`lst`并不一定是严格意义上的列表（最后一个元素是空列表），也可能是序对。

goldfish/liii/list.scm △ 5 ▽

```
(define (length>=? lst len)
  (let loop ((lst lst)
            (cnt 0))
    (cond ((null? lst) (<= len cnt))
          ((pair? lst) (loop (cdr lst) (+ cnt 1)))
          (else (<= len cnt)))))
```

tests/goldfish/liii/list-test.scm

△ 66 ▽

```
(check-true (length>=? '(1 2 3 4 5) 3))
(check-false (length>=? '(1 2) 3))
(check-true (length>=? '() 0))

(check-true (length>=? '(1) 0))
(check-false (length>=? '() 1))

(check-false (length>=? '(1 2 . 3) 3))
(check-true (length>=? '(1 2 . 3) 2))
```

**list-view**

索引

由于Scheme的List和数据的流向是相反的：

```
(map (lambda (x) (* x x))
     (map (lambda (x) (+ x 1))
          (list 1 2 3)))
```

所以我们实现了list-view，采用和Scala的List类似的语法来处理数据：

tests/goldfish/liii/list-test.scm

△ 67 ▽

```
(check ((list-view (list 1 2 3))) => (list 1 2 3))
```

```
(check (((list-view (list 1 2 3))
         map (lambda (x) (+ x 1)))) => (list 2 3 4))
```

```
(check (((list-view (list 1 2 3))
         map (lambda (x) (+ x 1))
         map (lambda (x) (* x x))))
=> (list 4 9 16))
```

(list-view 1 2 3)得到的是函数，需要在外面再加一层括号才能得到(list 1 2 3)。

```
(map (lambda (x) (* x x))          (((list-view 1 2 3)
                                     map (lambda (x) (+ x 1))
                                     map (lambda (x) (* x x))))
```

图 7.1. 使用list处理数据和使用list-view处理数据的对比

实现list-view时需要考虑三种情况和一种例外情况。

**无参数.**也就是直接在list-view得到的结果外面添加括号，此时得到的是list-view对应的list

**有两个参数.**这里举例说明，((list-view 1 2 3) map (lambda (x) (+ x 1)))实际的计算过程是：

1. 计算并得到结果(map (lambda (x) (+ x 1)) (list 1 2 3)) => (list 2 3 4)
2. 将计算结果包装到 list-view 里面，这里使用了apply这个内置函数

其实也是树的转换：

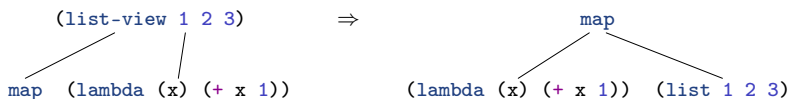


图 7.2. 原理的可视化

**偶数个参数.** 在上述两个递归退出条件写好的情况下, 在思考这种一般的情况。

需要计算((list-view 1 2 3) hf1 f1 hf2 f2 ... hfn fn), 其中hf指的是high-order function, 也就是高阶函数。也就是需要计算:

```
(((list-view 1 2 3) hf1 f1) hf2 f2) ... hfn fn)
```

goldfish/liii/list.scm

△ 6 ▽

```
(define (list-view scheme-list)
  (define (f-inner-reducer scheme-list filter filter-func rest-funcs)
    (cond ((null? rest-funcs) (list-view (filter filter-func scheme-list)))
          (else
           (f-inner-reducer (filter filter-func scheme-list)
                             (car rest-funcs)
                             (cadr rest-funcs)
                             (cddr rest-funcs)))))
  (define (f-inner . funcs)
    (cond ((null? funcs) scheme-list)
          ((length=? 2 funcs)
           (list-view ((car funcs) (cadr funcs) scheme-list)))
          ((even? (length funcs))
           (f-inner-reducer scheme-list
                             (car funcs)
                             (cadr funcs)
                             (cddr funcs)))
          (else (error 'wrong-number-of-args
                       "list-view只 accepts even number of args"))))
  f-inner)
```

## flatmap

goldfish/liii/list.scm

△ 7 ▽

```
(define flatmap append-map)
```

tests/goldfish/liii/list-test.scm

△ 68 ▽

```
(check (flatmap (lambda (x) (list x x))
               (list 1 2 3))
      => (list 1 1 2 2 3 3))

(check-catch 'type-error (flatmap 1 (list 1 2 3)))
```

## not-null-list?

null-list?的反面, 会抛出异常。

goldfish/liii/list.scm

△ 8 ▽

```
(define (not-null-list? l)
  (cond ((pair? l)
        (or (null? (cdr l)) (pair? (cdr l))))
        ((null? l) #f)
        (else
         (error 'type-error "type_mismatch"))))
```

## list-null?

null-list?的没有异常的版本, 只要不是list, 都是#f。

goldfish/liii/list.scm

△ 9 ▾

```
(define (list-null? l)
  (and (not (pair? l)) (null? l)))
```

**list-not-null?** 索引

not-null-list? 的没有异常的版本。

goldfish/liii/list.scm

△ 10 ▾

```
(define (list-not-null? l)
  (and (pair? l)
       (or (null? (cdr l)) (pair? (cdr l)))))
```

tests/goldfish/liii/list-test.scm

△ 69 ▾

```
(check (not-null-list? (list 1)) => #t)
(check (list-not-null? (list 1)) => #t)
(check (list-null? (list 1)) => #f)

(check (not-null-list? (list 1 2 3)) => #t)
(check (list-not-null? (list 1 2 3)) => #t)
(check (list-null? (list 1 2 3)) => #f)

(check (not-null-list? '(a)) => #t)
(check (list-not-null? '(a)) => #t)
(check (list-null? '(a)) => #f)

(check (not-null-list? '(a b c)) => #t)
(check (list-not-null? '(a b c)) => #t)
(check (list-null? '(a b c)) => #f)

(check (not-null-list? ()) => #f)
(check (list-not-null? ()) => #f)
(check (list-null? ()) => #t)

; '(a) is a pair and a list
; '(a . b) is a pair but not a list
(check (not-null-list? '(a . b)) => #f)
(check (list-not-null? '(a . b)) => #f)
(check (list-null? '(a . b)) => #f)

(check-catch 'type-error (not-null-list? 1))
(check (list-not-null? 1) => #f)
(check (list-null? 1) => #f)
```

**flatten** 索引 (lst [depth]) => lst

(lst [depth]) => lst

压平 lst，压为更扁的形状。直观来说，去掉内层列表的括号。例如，(flatten '(a b (c (d)) (e) ((f))) 'deepest) => (a b c d e f)。参数 depth 可选，默认为 1，指定最深压平层数，depth 为 n 时，最深压平 n 层列表，depth 为 'deepest 时，压平整个列表。

注意，定义 flatten 应当使用 **define\*** 而非 **define** 以支持可选参数。

goldfish/liii/list.scm

△ 11 ▾

```
(define* (flatten lst (depth 1))
```

包装一般化的 flatten，使用 **set-cdr!** 加速构建列表，

goldfish/liii/list.scm

△ 12 ▽

---

```
(define (flatten-depth-iter rest depth res-node)
  (if (null? rest)
      res-node
      (let ((first (car rest))
            (tail (cdr rest)))
        (cond ((and (null? first) (not (= 0 depth)))
              (flatten-depth-iter tail depth res-node))
              ((or (= depth 0) (not (pair? first)))
               (set-cdr! res-node (cons first '())))
              (flatten-depth-iter tail depth (cdr res-node)))
          (else
           (flatten-depth-iter
            tail
            depth
            (flatten-depth-iter
             first
             (- depth 1)
             res-node))))))))))
```

---

因为要 `set-cdr!` 参数 `res-node`，所以必须保证后者是 `pair`，不能是 `null`。为达成这个目的，可以先 `cons` 一个 `pair`，之后返回它的 `<cdr>`。

goldfish/liii/list.scm

△ 13 ▽

---

```
(define (flatten-depth lst depth)
  (let ((res (cons #f '())))
    (flatten-depth-iter lst depth res)
    (cdr res)))
```

---

注意到 `flatten-depth-iter` 迭代中 (`cond` 第二个子句)，不断传递 `depth` 参数，只是重新绑定而不加改变。然而 `goldfish` 访问自由变量不比直接传递参数效率更高，只好辗转传递 `depth`。`(flatten-depth lst -1)` 固然可以用来实现 (`flatten lst -1`)，但总是额外传递一个无意义的参数 `depth`，效率较低，可以针对这一点进行优化。

goldfish/liii/list.scm

△ 14 ▽

---

```
(define (flatten-deepest-iter rest res-node)
  (if (null? rest)
      res-node
      (let ((first (car rest))
            (tail (cdr rest)))
        (cond ((pair? first)
              (flatten-deepest-iter
               tail
               (flatten-deepest-iter
                first
                res-node)))
              ((null? first)
               (flatten-deepest-iter tail res-node))
              (else
               (set-cdr! res-node (cons first '())))
              (flatten-deepest-iter tail (cdr res-node))))))
  (define (flatten-deepest lst)
    (let ((res (cons #f '())))
      (flatten-deepest-iter lst res)
      (cdr res)))
```

---

最后，分派 `flatten-depth` 和 `flatten-deepest`。注意 `flatten` 参数 `depth` 不应该是除 `n` 和 `'deepest` 以外的任何值，因而，当出现其它值时抛出错误。

goldfish/liii/list.scm

△ 15 ▽

```
(cond ((eq? depth 'deepest)
      (flatten-deepest lst))
      ((integer? depth)
      (flatten-depth lst depth))
      (else
      (type-error
       (string-append
        "flatten: the second argument depth should be symbol"
        "'deepest' or an integer, which will be used as depth,"
        " but got a ~A" depth)))
      ) ; end of (define* (flatten))
```

测试代码如下。

tests/goldfish/liii/list-test.scm

△ 70 ▽

```
; deepest flatten
(check (flatten '(a) () (b ()) () (c)) 'deepest) => '(a b c)
(check (flatten '(a b) c ((d) e)) 'deepest) => '(a b c d e)
(check (flatten 'a b ((c))) 'deepest) => '(a b c))
; depth flatten
(check (flatten '(a) () (b ()) () (c)) 0) => '((a) () (b ()) () (c))
(check (flatten '(a) () (b ()) () (c)) 1) => '(a b () c)
(check (flatten '(a) () (b ()) () (c))) => '(a b () c)
(check (flatten '(a) () (b ()) () (c)) 2) => '(a b c)
(check (flatten '(a) () (b ()) () (c)) -1) => '(a b c)
(check (flatten '(a b) c ((d) e)) 0) => '((a b) c ((d) e))
(check (flatten '(a b) c ((d) e)) 1) => '(a b c ((d) e))
(check (flatten '(a b) c ((d) e))) => '(a b c ((d) e))
(check (flatten '(a b) c ((d) e)) 2) => '(a b c (d) e)
(check (flatten '(a b) c ((d) e)) 3) => '(a b c d e)
(check (flatten '(a b) c ((d) e)) -1) => '(a b c d e)
(check (flatten 'a b ((c))) 0) => '(a b ((c)))
(check (flatten 'a b ((c))) 1) => '(a b () (c))
(check (flatten 'a b ((c)))) => '(a b () (c))
(check (flatten 'a b ((c)) 2) => '(a b c)
(check (flatten 'a b ((c)) -1) => '(a b c))
; error depth flatten
(check-catch 'type-error (flatten '(a) () (b ()) () (c)) 'a)
(check-catch 'type-error (flatten '(a) () (b ()) () (c)) (make-vector 1 1)))
```

## 7.6 结尾

goldfish/liii/list.scm

△ 16

```
) ; end of begin
) ; end of library
```

goldfish/srfi/srfi-1.scm

△ 45

```
) ; end of begin
) ; end of define-library
```

`tests/goldfish/liii/list-test.scm`

△ 71

---

(check-report)

---



# 第 8 章

## (liii bitwise) chapter:liii\_bitwise

### 8.1 概述

位运算的基本规则如下表所示：

表格 8.1. 二进制1位位运算表

	1 op 1	1 op 0	0 op 1	0 op 0
and	1	0	0	0
or	1	1	1	0
xor	0	1	1	0

在金鱼Scheme中，整数类型是64位的有符号整数，上述运算规则执行64次，就是金鱼Scheme中整数的位运算具体规则。

### 8.2 许可证

[goldfish/liiii/bitwise.scm](http://goldfish/liiii/bitwise.scm)

1 ▽

```
;  
; Copyright (C) 2024 The Goldfish Scheme Authors  
;  
; Licensed under the Apache License, Version 2.0 (the "License");  
; you may not use this file except in compliance with the License.  
; You may obtain a copy of the License at  
;  
; http://www.apache.org/licenses/LICENSE-2.0  
;  
; Unless required by applicable law or agreed to in writing, software  
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT  
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the  
; License for the specific language governing permissions and limitations  
; under the License.  
;
```

[goldfish/srfi/srfi-151.scm](#)

1 ▾

---

```

;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;

```

---

[tests/goldfish/liii/bitwise-test.scm](#)

1 ▾

---

```

;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;

```

---

## 8.3 接口

[goldfish/liii/bitwise.scm](#)

△ 2 ▾

---

```

(define-library (liii bitwise)
  (import (srfi srfi-151)
          (liii error))
  (export
   ; from (srfi srfi-151)
   bitwise-not bitwise-and bitwise-ior bitwise-xor bitwise-or bitwise-nor bitwise-nand
   bit-count bitwise-orc1 bitwise-orc2 bitwise-andc1 bitwise-andc2
   arithmetic-shift
   ; S7 built-in
   lognot logand logior logxor
   ash
  )
  (begin

```

---

goldfish/srfi/srfi-151.scm

△ 2 ▾

```
(define-library (srfi srfi-151)
  (export
    bitwise-not bitwise-and bitwise-ior bitwise-xor bitwise-nor bitwise-nand bit-count
    bitwise-orc1 bitwise-orc2 bitwise-andc1 bitwise-andc2
    arithmetic-shift
  )
  (begin
```

## 8.4 测试

tests/goldfish/liii/bitwise-test.scm

△ 2 ▾

```
(import (liii check)
  (liii bitwise))

(check-set-mode! 'report-failed)
```

## 8.5 实现

### 8.5.1 基础运算

lognot

索引

**bitwise-not** `((x integer?)) => boolean?`

使用S7内置的lognot实现，不推荐使用lognot。

goldfish/srfi/srfi-151.scm

△ 3 ▾

```
(define bitwise-not lognot)
```

#### 测试

tests/goldfish/liii/bitwise-test.scm

△ 3 ▾

```
(check (bitwise-not 0) => -1)
(check (bitwise-not 1) => -2)
(check (bitwise-not #b1000) => -9)
(check (bitwise-not -1) => 0)
```

logand

索引

**bitwise-and** `((a integer?) (b integer?)) => boolean?`

使用S7内置的logand实现，不推荐使用logand。此函数应用and运算在整数的二进制表示上。

goldfish/srfi/srfi-151.scm

△ 4 ▾

```
(define bitwise-and logand)
```

#### 测试

tests/goldfish/liii/bitwise-test.scm

△ 4 ▾

```
(check (bitwise-and 5 3) => 1) ; 5 (101) AND 3 (011) = 1 (001)
(check (bitwise-and 8 4) => 0) ; 8 (1000) AND 4 (0100) = 0 (0000)
(check (bitwise-and #b101 #b011) => 1) ; 5 (101) AND 3 (011) = 1 (001)
(check (bitwise-and #b1000 #b0100) => 0) ; 8 (1000) AND 4 (0100) = 0 (0000)
(check (bitwise-and #b1100 #b1010) => 8)
```

**logior**

索引

**bitwise-or** `((a integer?) (b integer?)) => boolean?`

使用S7内置的**logior**实现，不推荐使用**logior**。此函数应用**or**运算在整数的二进制表示上。

goldfish/srfi/srfi-151.scm

△ 5 ▽

```
(define bitwise-ior logior)
```

goldfish/liii/bitwise.scm

△ 3 ▽

```
(define bitwise-or bitwise-ior)
```

**测试**

tests/goldfish/liii/bitwise-test.scm

△ 5 ▽

```
(check (bitwise-ior 5 3) => 7) ; 5 (101) OR 3 (011) = 7 (111)
(check (bitwise-or 5 3) => 7)
(check (bitwise-ior 8 4) => 12) ; 8 (1000) OR 4 (0100) = 12 (1100)
(check (bitwise-ior #b101 #b011) => 7) ; 5 (101) AND 3 (011) = 1 (001)
(check (bitwise-ior #b1000 #b0100) => 12) ; 8 (1000) AND 4 (0100) = 0 (0000)
(check (bitwise-ior #b1100 #b0001) => 13)
```

**logxor**

索引

**bitwise-xor** `((a integer?) (b integer?)) => boolean?`

使用S7内置的**logxor**实现，不推荐使用**logxor**，推荐使用**bitwise-xor**。

goldfish/srfi/srfi-151.scm

△ 6 ▽

```
(define bitwise-xor logxor)
```

**测试**

tests/goldfish/liii/bitwise-test.scm

△ 6 ▽

```
(check (bitwise-xor 1 1) => 0)
(check (bitwise-xor #b10 #b11) => #b01) ; 2 xor 3 = 1
(check (bitwise-xor #b101010 #b110100) => #b011110) ; 42 xor 20 = 34
(check (bitwise-xor #b0 #b0) => #b0) ; 0 xor 0 = 0
(check (bitwise-xor #b1 #b1) => #b0) ; 1 xor 1 = 0
(check (bitwise-xor #b101 #b111) => #b010) ; 5 xor 7 = 2
(check (bitwise-xor #b1000 #b1001) => #b0001) ; 8 xor 9 = 1
(check (bitwise-xor #b10010101 #b01111001) => #b11101100)
```

**lognot**

索引

**bitwise-nor** `((a integer?) (b integer?)) => boolean?`

使用S7内置的**lognot**和**bitwise-ior**实现。

goldfish/srfi/srfi-151.scm

△ 7 ▽

```
(define (bitwise-nor a b)
  (lognot (bitwise-ior a b)))
```

**测试**

tests/goldfish/liii/bitwise-test.scm △ 7 ▾

```
(check (bitwise-nor 2 4) => -7)
(check (bitwise-nor 3 1) => -4)
(check (bitwise-nor #b111 #b011) => -8)
(check (bitwise-nor #b1101 #b1011) => -16)
(check (bitwise-nor #b1100 #b0000) => -13)
```

lognot 索引

SRFI bitwise-nand 索引 `((a integer?) (b integer?)) => boolean?`

使用S7内置的lognot和bitwise-and实现。

goldfish/srfi/srfi-151.scm △ 8 ▾

```
(define (bitwise-nand a b)
  (lognot (bitwise-and a b)))
```

测试

tests/goldfish/liii/bitwise-test.scm △ 8 ▾

```
(check (bitwise-nand 1 1) => -2)
(check (bitwise-nand 3 1) => -2)
(check (bitwise-nand #b110 #b001) => -1)
(check (bitwise-nand #b1001 #b0111) => -2)
(check (bitwise-nand #b1011 #b0101) => -2)
```

SRFI bit-count 索引 `((i integer?) => integer?`

如果i是正数，统计i的二进制表示中的1的数量。如果i是负数，统计i的二进制表示中的0的数量。

goldfish/srfi/srfi-151.scm △ 9 ▾

```
(define bit-count
  (typed-lambda ((i integer?))
    (define (bit-count-positive i)
      (let loop ((n i) (cnt 0))
        (if (= n 0)
            cnt
            (loop (logand n (- n 1)) (+ cnt 1)))))
    (cond ((zero? i) 0)
          ((positive? i) (bit-count-positive i))
          (else (bit-count-positive (lognot i)))))
```

测试

tests/goldfish/liii/bitwise-test.scm △ 9 ▾

```
(check (bit-count 0) => 0)
(check (bit-count -1) => 0)
(check (bit-count 7) => 3)
(check (bit-count 13) => 3)
(check (bit-count -13) => 2)
(check (bit-count 30) => 4)
(check (bit-count -30) => 4)
(check (bit-count (arithmetic-shift #b10 61)) => 1)
```

**SRFI** **bitwise-orc1** [索引](#) `((a integer?) (b integer?)) => boolean?`

使用S7内置的bitwise-ior和bitwise-not实现。

goldfish/srfi/srfi-151.scm

△ 10 ▾

```
(define (bitwise-orc1 i j)
  (bitwise-ior (bitwise-not i) j))
```

## 测试

tests/goldfish/liii/bitwise-test.scm

△ 10 ▾

```
(check (bitwise-orc1 1 1) => -1)
(check (bitwise-orc1 3 1) => -3)
(check (bitwise-orc1 11 26) => -2)
(check (bitwise-orc1 #b110 #b001) => -7)
(check (bitwise-orc1 #b1001 #b0111) => -9)
(check (bitwise-orc1 #b1011 #b0101) => -11)
```

**SRFI** **bitwise-orc2** [索引](#) `((a integer?) (b integer?)) => boolean?`

使用S7内置的bitwise-ior和bitwise-not实现。

goldfish/srfi/srfi-151.scm

△ 11 ▾

```
(define (bitwise-orc2 i j)
  (bitwise-ior i (bitwise-not j)))
```

## 测试

tests/goldfish/liii/bitwise-test.scm

△ 11 ▾

```
(check (bitwise-orc2 11 26) => -17)
(check (bitwise-orc2 3 1) => -1)
(check (bitwise-orc2 #b110 #b001) => -2)
(check (bitwise-orc2 #b1001 #b0111) => -7)
(check (bitwise-orc2 #b1011 #b0101) => -5)
```

**SRFI** **bitwise-andc1** [索引](#) `((a integer?) (b integer?)) => boolean?`

使用S7内置的bitwise-and和bitwise-not实现。

goldfish/srfi/srfi-151.scm

△ 12 ▾

```
(define (bitwise-andc1 i j)
  (bitwise-and (bitwise-not i) j))
```

## 测试

tests/goldfish/liii/bitwise-test.scm

△ 12 ▾

```
(check (bitwise-andc1 11 26) => 16)
(check (bitwise-andc1 5 3) => 2)
(check (bitwise-andc1 #b1100 #b1010) => 2)
(check (bitwise-andc1 0 15) => 15)
(check (bitwise-andc1 15 0) => 0)
(check (bitwise-andc1 7 1) => 0)
```

**SRFI** **bitwise-andc2** [索引](#) `((a integer?) (b integer?)) => boolean?`

使用S7内置的bitwise-and和bitwise-not实现。

goldfish/srfi/srfi-151.scm

△ 13 ▾

```
(define (bitwise-andc2 i j)
  (bitwise-and i (bitwise-not j)))
```

### 测试

tests/goldfish/liii/bitwise-test.scm

△ 13 ▾

```
(check (bitwise-andc2 11 26) => 1)
(check (bitwise-andc2 5 3) => 4)
(check (bitwise-andc2 #b1100 #b1010) => 4)
(check (bitwise-andc2 0 15) => 0)
(check (bitwise-andc2 15 0) => 15)
(check (bitwise-andc2 7 1) => 6)
```

## 8.5.2 整数运算

ash

索引

**SRFI** **索引**  
**arithmetic-shift** `((i integer?) (shift integer?)) => integer?`

log开头的S7内置函数不推荐使用，原因是log不符合位运算的语义，ash是arithmetic-shift的简写，能够有效避免arithmetic这个单词拼写错误的可能性，仍旧是推荐使用的。从代码可读性上考虑，还是使用arithmetic-shift更加合适。

goldfish/srfi/srfi-151.scm

△ 14 ▾

```
(define arithmetic-shift ash)
```

### 测试

tests/goldfish/liii/bitwise-test.scm

△ 14 ▾

```
(check (arithmetic-shift #b10 -1) => #b1) ; 2 >> 1 = 1
(check (arithmetic-shift #b10 1) => #b100) ; 2 << 1 = 4
(check (arithmetic-shift #b1000 -2) => #b10) ; 8 >> 2 = 2
(check (arithmetic-shift #b1000 2) => #b100000)
(check (arithmetic-shift #b1000000000000000 -3) => #b1000000000000000)
(check (arithmetic-shift #b1000000000000000 3) => #b1000000000000000000)
```

## 8.5.3 单位运算

## 8.5.4 按位运算

## 8.5.5 按位转换

## 8.5.6 高阶函数

## 8.6 结束

SRFI-151 End

---

[goldfish/srfi/srfi-151.scm](#)

△ 15

---

) ; end of begin

) ; end of define-library

---

[goldfish/liii/bitwise.scm](#)

△ 4

---

) ; end of begin

) ; end of library

---

## Report Test Result

[tests/goldfish/liii/bitwise-test.scm](#)

△ 15

---

(check-report)

---



# 第 9 章

## (liii string)<sup>chapter:liii\_string</sup>

### 9.1 许可证

[goldfish/liiii/string.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS_IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

[goldfish/srfi/srfi-13.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS_IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

tests/goldfish/liii/string-test.scm

1 ▾

---

```

;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;

```

---

## 9.2 接口

goldfish/liii/string.scm

△ 2 ▾

---

```

(define-library (liii string)
  (export
    ; S7 built-in
    string? string-ref string-length
    ; from (scheme base)
    string-copy string-for-each string-map
    ; from (srfi srfi-13)
    string-null? string-join
    string-every string-any
    string-take string-take-right string-drop string-drop-right
    string-pad string-pad-right
    string-trim string-trim-right string-trim-both
    string-index string-index-right
    string-contains string-count
    string-upcase string-downcase
    string-reverse
    string-tokenize
    ; Liii extras
    string-starts? string-ends?
    string-remove-prefix string-remove-suffix
  )
  (import (srfi srfi-13)
          (liiii base)
          (liiii error))
  (begin

```

---

goldfish/srfi/srfi-13.scm

△ 2 ▾

---

```
(define-library (srfi srfi-13)
  (import
    (liii base)
    (srfi srfi-1))
  (export
    string-null? string-copy string-join
    string-every string-any
    string-take string-take-right string-drop string-drop-right
    string-pad string-pad-right
    string-trim string-trim-right string-trim-both
    string-prefix? string-suffix?
    string-index string-index-right
    string-contains string-count
    string-upcase string-downcase
    string-reverse
    string-tokenize)
  (begin
```

---

## 9.3 测试

tests/goldfish/liiii/string-test.scm

△ 2 ▾

---

```
(import (liiii check)
  (liiii string))

(check-set-mode! 'report-failed)
```

---

## 9.4 SRFI 13

### 9.4.1 内部公共子函数

goldfish/srfi/srfi-13.scm

△ 3 ▾

---

```
(define (%string-from-range str start_end)
  (cond ((null-list? start_end) str)
        ((= (length start_end) 1)
         (substring str (car start_end)))
        ((= (length start_end) 2)
         (substring str (first start_end) (second start_end)))
        (else (error 'wrong-number-of-args "%string-from-range"))))
```

---

[测试一下边界条件，这样后面漏测也没关系了。]

goldfish/srfi/srfi-13.scm

△ 4 ▽

---

```
(define (%make-criterion char/pred?)
  (cond ((char? char/pred?) (lambda (x) (char=? x char/pred?)))
        ((procedure? char/pred?) char/pred?)
        (else (error 'wrong-type-arg "%make-criterion"))))
```

---

## 9.4.2 谓词

R7RS 索引  
**string?**

`string?` 是一个 S7 内置的谓词函数，当且仅当参数对象类型是字符串（例如 "MathAgape"）时返回 #t，否则都返回 #f。当参数为符号（例如 'MathAgape）、字符（例如 #/MathAgape）、数字（例如 123）、列表（例如 '(1 2 3)）这些非字符串类型时，都返回 #f。`string?` 用于确定对象是否可以被当作字符串处理，在需要执行对字符串特定操作时特别有用，避免类型错误。

tests/goldfish/liii/base-test.scm

△ 97 ▽

---

```
(check (string? "MathAgape") => #t)
(check (string? "") => #t)

(check (string? 'MathAgape) => #f)
(check (string? #/MathAgape) => #f)
(check (string? 123) => #f)
(check (string? '(1 2 3)) => #f)
```

---

## 9.4.3 列表-字符串转换

R7RS 索引  
**string->list**

`string->list` 是一个 S7 内置的函数，用于将字符串转换为字符列表。

tests/goldfish/liii/base-test.scm

△ 98 ▽

---

```
(check (string->list "MathAgape")
=> '(#\M #\a #\t #\h #\A #\g #\a #\p #\e))

(check (string->list "") => '())
```

---

R7RS 索引  
**list->string**

`list->string` 是一个 S7 内置的函数，用于将字符列表转换为字符串。

tests/goldfish/liii/base-test.scm

△ 99 ▽

---

```
(check
  (list->string '(#\M #\a #\t #\h #\A #\g #\a #\p #\e))
  => "MathAgape")

(check (list->string '()) => "")
```

---

**string-join** (l [delim [grammar]]) => string 索引

实现

goldfish/srfi/srfi-13.scm

△ 5 ▽

---

```

(define (string-join l . delim+grammar)
  (define (extract-params params-l)
    (cond ((null-list? params-l)
           (list "" 'infix))
          ((and (= (length params-l) 1)
                 (string? (car params-l)))
           (list (car params-l) 'infix))
          ((and (= (length params-l) 2)
                 (string? (first params-l))
                 (symbol? (second params-l)))
           params-l)
          (> (length params-l) 2)
           (error 'wrong-number-of-args "optional_params_in_string-join"))
          (else (error 'type-error "optional_params_in_string-join"))))

  (define (string-join-sub l delim)
    (cond ((null-list? l) "")
          ((= (length l) 1) (car l))
          (else
           (string-append
            (car l)
            delim
            (string-join-sub (cdr l) delim)))))

  (let* ((params (extract-params delim+grammar))
         (delim (first params))
         (grammar (second params))
         (ret (string-join-sub l delim)))
    (case grammar
      ('infix ret)
      ('strict-infix
       (if (null-list? l)
           (error 'value-error "empty_list_not_allowed")
           ret))
      ('suffix
       (if (null-list? l) "" (string-append ret delim)))
      ('prefix
       (if (null-list? l) "" (string-append delim ret)))
      (else (error 'value-error "invalid_grammar")))))

```

---

测试

tests/goldfish/liii/string-test.scm

△ 3 ▽

---

```

(check (string-join '("a" "b" "c")) => "abc")

(check (string-join '("a" "b" "c") ":") => "a:b:c")
(check (string-join '("a" "b" "c") ":" 'infix) => "a:b:c")
(check (string-join '("a" "b" "c") ":" 'suffix) => "a:b:c:")
(check (string-join '("a" "b" "c") ":" 'prefix) => ":a:b:c")

(check (string-join '() ":") => "")
(check (string-join '() ":" 'infix) => "")
(check (string-join '() ":" 'prefix) => "")
(check (string-join '() ":" 'suffix) => "")

(check-catch 'value-error (string-join '() ":" 'strict-infix))
(check-catch 'type-error (string-join '() ":" 2))
(check-catch 'value-error (string-join '() ":" 'no-such-grammar))
(check-catch 'wrong-number-of-args (string-join '() ":" 1 2 3))

```

---

## 9.4.4 谓词

**string-null?** <sup>索引</sup> (str) => boolean

**string-null?**是一个谓词函数，当且仅当参数字符串为空（即长度为0）时返回#t，否则都返回#f，也就是说，当参数是非字符串或长度不为0的字符串时，返回#f。

goldfish/srfi/srfi-13.scm

△ 6 ▽

---

```

(define (string-null? str)
  (and (string? str)
        ((lambda (x) (= x 0)) (string-length str))))

```

---

tests/goldfish/liii/string-test.scm

△ 4 ▾

```
(check-true (string-null? ""))

(check-false (string-null? "MathAgape"))

(check-false (string-null? 'not-a-string))
```

## string-every

索引

`string-every` 是一个谓词函数，它接收一个字符串 `str` 和一个评估规则 `criterion` 为参数，以及最多 2 个非负整数为可选参数。 `start_end` 用于指定搜索的起始位置和结束位置，检查指定范围内字符串中的每个字符是否都满足评估规则 `criterion`。

goldfish/srfi/srfi-13.scm

△ 7 ▾

```
(define (string-every char/pred? str . start+end)
  (define (string-every-sub pred? str)
    (let
      (loop ((i 0) (len (string-length str)))
        (or (= i len)
            (and (pred? (string-ref str i))
                 (loop (+ i 1) len))))))
  (let ((str-sub (%string-from-range str start+end))
        (criterion (%make-criterion char/pred?)))
    (string-every-sub criterion str-sub)))
```

tests/goldfish/liii/string-test.scm

△ 5 ▾

```
(check-true (string-every #\x "xxxxxxx"))
(check-false (string-every #\x "xxx0xx"))

(check-true (string-every char-numeric? "012345"))
(check-false (string-every char-numeric? "012d45"))
```

注意，评估规则 `criterion` 必须是谓词或字符，否则报错。

tests/goldfish/liii/string-test.scm

△ 6 ▾

```
(check-catch 'wrong-type-arg (string-every 1 "012345"))
(check-catch 'wrong-type-arg (string-every #\012345 "012345"))
(check-catch 'wrong-type-arg (string-every "012345" "012345"))
```

注意，谓词要使用字符属性测试函数（例如 `char-numeric?`），不要用类型检查函数（例如 `number?`），否则失去了检查的意义。

tests/goldfish/liii/string-test.scm

△ 7 ▾

```
(check-true (string-every char-numeric? "012345"))
(check-false (string-every number? "012345"))
```

对于可选参数 `start_end`：没有可选参数时，默认搜索整个字符串；只有 1 个可选参数时，该数指定的是起始位置，结束位置默认为字符串的最后一个字符。注意，有 2 个可选参数时，第二个数不能小于第一个数，否则报错；指定的起始位置和结束位置不能超过字符串 `str` 的起始范围，否则报错；可选参数超过 2 个时，报错。（注意，可选参数的规则适用于后面所有调用到 `%string-from-range` 的函

数)

```
tests/goldfish/liii/string-test.scm △ 8 ▽
(check-true (string-every char-numeric? "ab2345" 2))
(check-false (string-every char-numeric? "ab2345" 1))
(check-false (string-every char-numeric? "ab234f" 2))
(check-true (string-every char-numeric? "ab234f" 2 4))
(check-true (string-every char-numeric? "ab234f" 2 2))
(check-false (string-every char-numeric? "ab234f" 1 4))
(check-true (string-every char-numeric? "ab234f" 2 5))
(check-false (string-every char-numeric? "ab234f" 2 6))

(check-catch 'out-of-range (string-every char-numeric? "ab234f" 2 7))
(check-catch 'out-of-range (string-every char-numeric? "ab234f" 2 1))
```

## string-any 索引

`string-any`是一个谓词函数，它接收一个谓词和一个字符串为参数，用于检查字符串中是否存在字符都满足谓词。注意，这里的谓词必须是字符属性测试函数，否则报错。

```
goldfish/srfi/srfi-13.scm △ 8 ▽
(define (string-any char/pred? str . start+end)
  (define (string-any-sub pred? str)
    (let loop ((i 0) (len (string-length str)))
      (if (= i len)
          #f
          (or (pred? (string-ref str i))
              (loop (+ i 1) len))))))
  (let ((str_sub (%string-from-range str start+end))
        (criterion (%make-criterion char/pred?)))
    (string-any-sub criterion str_sub)))
```

```
tests/goldfish/liii/string-test.scm △ 9 ▽
(check-true (string-any #\0 "xxx0xx"))
(check-false (string-any #\0 "xxxxxx"))
(check-true (string-any char-numeric? "xxx0xx"))
(check-false (string-any char-numeric? "xxxxxx"))
```

注意，评估规则 `criterion` 必须是谓词或字符，否则报错。

```
tests/goldfish/liii/string-test.scm △ 10 ▽
(check-catch 'wrong-type-arg (string-every 0 "xxx0xx"))
(check-catch 'wrong-type-arg (string-any (lambda (n) (= n 0)) "xxx0xx"))
(check-catch 'wrong-type-arg (string-every '0 "xxx0xx"))
```

对于可选参数：



tests/goldfish/liii/string-test.scm

△ 11 ▾

```
(check-true (string-any char-alphabetic? "01c345" 2))
(check-false (string-any char-alphabetic? "01c345" 3))
(check-true (string-any char-alphabetic? "01c345" 2 4))
(check-false (string-any char-alphabetic? "01c345" 2 2))
(check-false (string-any char-alphabetic? "01c345" 3 4))
(check-true (string-any char-alphabetic? "01c345" 2 6))
```

```
(check
  (catch 'out-of-range
    (lambda ()
      (string-any
        char-alphabetic?
        "01c345"
        2
        7))
    (lambda args #t))
  =>
  #t)
```

```
(check
  (catch 'out-of-range
    (lambda ()
      (string-any
        char-alphabetic?
        "01c345"
        2
        1))
    (lambda args #t))
  =>
  #t)
```

### 9.4.5 选择器

R7RS 索引  
**string-length**

`string-length`是一个S7内置的函数，它接收一个字符串作为参数，返回一个表示该字符串长度的整数，即字符串中包含的字符数。当参数不是字符串，报错。

tests/goldfish/liii/base-test.scm

△ 100 ▾

```
(check (string-length "MathAgape") => 9)
(check (string-length "") => 0)
```

```
(check
  (catch 'wrong-type-arg
    (lambda () (string-length 'not-a-string))
    (lambda args #t))
  =>
  #t)
```

R7RS 索引  
**string-ref**

`string-ref`是一个S7内置的函数，接收一个字符串和一个称为索引值的非负整数 $k$ 为参数，通过索引 $k$ 返回字符串的第 $k$ 个元素（从0开始计数）。当参数为空字符串时，报错。当 $k$ 为负数，报错。当

k大于等于字符串中字符数时，报错。

```
tests/goldfish/liii/base-test.scm △ 101 ▽
(check (string-ref "MathAgape" 0) => #\M)
(check (string-ref "MathAgape" 2) => #\t)

(check-catch 'out-of-range (string-ref "MathAgape" -1))
(check-catch 'out-of-range (string-ref "MathAgape" 9))
(check-catch 'out-of-range (string-ref "" 0))
```

## R7RS 索引 string-copy

`string-copy`是一个S7内置的函数，用于创建一个现有字符串的副本。这个函数返回一个与原始字符串内容完全相同的新字符串，但它们在内存中是两个独立的实体。也就是说，原始字符串和副本字符串内容相同，但它们是不同的对象，这可以用`equal?`和`eq?`对比出来。

`string-copy`和`%string-from-range`的实现比较接近，区别在于：在没有指定字符串的范围的时候，我们需要使用`substring`来模拟整个字符串的拷贝。S7内置的函数只提供了整个字符串的拷贝，R7RS和SRFI-13是需要额外的字符串的范围的，所以需要重新实现。

```
goldfish/scheme/base.scm △ 40 ▽
(define (string-copy str . start_end)
  (cond ((null? start_end)
         (substring str 0))
        ((= (length start_end) 1)
         (substring str (car start_end)))
        ((= (length start_end) 2)
         (substring str (car start_end) (cadr start_end)))
        (else (error 'wrong-number-of-args))))
```

## 测试

```
tests/goldfish/liii/string-test.scm △ 12 ▽
(define original-string "MathAgape")
(define copied-string (string-copy original-string))

(check-true (equal? original-string copied-string))
(check-false (eq? original-string copied-string))

(check-true
  (equal? (string-copy "MathAgape" 4)
          (string-copy "MathAgape" 4)))

(check-false
  (eq? (string-copy "MathAgape" 4)
       (string-copy "MathAgape" 4)))

(check-true
  (equal? (string-copy "MathAgape" 4 9)
          (string-copy "MathAgape" 4 9)))

(check-false
  (eq? (string-copy "MathAgape" 4 9)
       (string-copy "MathAgape" 4 9)))
```

**string-take**

`string-take` 是一个函数，接收一个字符串和一个非负整数  $k$  为参数，返回字符串的前  $k$  个字符组成的新字符串。当字符串字符数量不足  $k$  个，报错。

goldfish/srfi/srfi-13.scm

△ 9 ▾

```
(define (string-take str k)
  (substring str 0 k))
```

tests/goldfish/liii/string-test.scm

△ 13 ▾

```
(check (string-take "MathAgape" 4) => "Math")

(check-catch 'out-of-range (string-take "MathAgape" 20))
```

**string-take-right**

`string-take-right` 是一个函数，接收一个字符串和一个非负整数  $k$  为参数，取出字符串的后  $k$  个字符组成新字符串，返回这个新字符串。当字符串字符数量不足  $k$  个，报错。

goldfish/srfi/srfi-13.scm

△ 10 ▾

```
(define (string-take-right str k)
  (let ((N (string-length str)))
    (if (> k N)
        (error 'out-of-range "k must be ≤ N" k N)
        (substring str (- N k) N))))
```

tests/goldfish/liii/string-test.scm

△ 14 ▾

```
(check (string-take-right "MathAgape" 0) => "")
(check (string-take-right "MathAgape" 1) => "e")
(check (string-take-right "MathAgape" 9) => "MathAgape")

(check-catch 'out-of-range (string-take-right "MathAgape" 20))
```

**string-drop**

`string-drop` 是一个函数，接收一个字符串和一个非负整数  $k$  为参数，返回去掉字符串前  $k$  个字符组成的新字符串。当字符串字符数量不足  $k$  个，报错。

goldfish/srfi/srfi-13.scm

△ 11 ▾

```
(define string-drop
  (typed-lambda ((str string?) (k integer?))
    (when (< k 0)
      (error 'out-of-range "k must be non-negative" k))
    (let ((N (string-length str)))
      (if (> k N)
          (error 'out-of-range "k must be ≤ N" k N)
          (substring str k N))))))
```

tests/goldfish/liii/string-test.scm

△ 15 ▾

```
(check (string-drop "MathAgape" 8) => "e")
(check (string-drop "MathAgape" 9) => "")
(check (string-drop "MathAgape" 0) => "MathAgape")

(check-catch 'out-of-range (string-drop "MahtAgape" -1))
(check-catch 'out-of-range (string-drop "MathAgape" 20))
```

**string-drop-right** 索引 (str k) => string

`string-drop-right` 是一个函数，接收一个字符串和一个非负整数  $k$  为参数，去掉字符串的后  $k$  个字符组成新字符串，返回这个新字符串。当字符串字符数量不足  $k$  个，报错。当  $k$  为负数，报错。

goldfish/srfi/srfi-13.scm

△ 12 ▾

```
(define string-drop-right
  (typed-lambda ((str string?) (k integer?))
    (when (< k 0)
      (error 'out-of-range "k must be non-negative" k))
    (let ((N (string-length str)))
      (if (> k N)
          (error 'out-of-range "k must be ≤ N" k N)
          (substring str 0 (- N k))))))
```

tests/goldfish/liii/string-test.scm

△ 16 ▾

```
(check (string-drop-right "MathAgape" 5) => "Math")
(check (string-drop-right "MathAgape" 9) => "")
(check (string-drop-right "MathAgape" 0) => "MathAgape")

(check-catch 'out-of-range (string-drop-right "MathAgape" -1))
(check-catch 'out-of-range (string-drop-right "MathAgape" 20))
```

**string-pad** 索引

`string-pad` 是一个函数，接收一个字符串、一个指定长度（非负整数）为参数，从左填充空格字符直到指定长度，返回这个新字符。当指定长度等于字符串长度，返回的字符串和原字符串内容相同。当指定长度小于字符串长度，则从左去掉字符直到指定长度，返回这个新字符。当参数的指定长度为负数时，报错。

goldfish/srfi/srfi-13.scm

△ 13 ▾

```
(define (string-pad str len . char+start+end)
  (define (string-pad-sub str len ch)
    (let ((orig-len (string-length str)))
      (if (< len orig-len)
          (string-take-right str len)
          (string-append (make-string (- len orig-len) ch) str))))

  (cond ((null-list? char+start+end)
         (string-pad-sub str len #\ ))
        ((list? char+start+end)
         (string-pad-sub
          (%string-from-range str (cdr char+start+end))
          len
          (car char+start+end)))
        (else (error 'wrong-type-arg "string-pad"))))
```

tests/goldfish/liii/string-test.scm

△ 17 ▾

```
(check (string-pad "MathAgape" 15) => "MathAgape")
(check (string-pad "MathAgape" 12 #\1) => "111MathAgape")
(check (string-pad "MathAgape" 6 #\1 0 4) => "11Math")
(check (string-pad "MathAgape" 9) => "MathAgape")
(check (string-pad "MathAgape" 5) => "Agape")
(check (string-pad "MathAgape" 2 #\1 0 4) => "th")

(check-catch 'out-of-range (string-pad "MathAgape" -1))
```

## string-pad-right

索引

`string-pad-right` 是一个函数，接收一个字符串、一个指定长度（非负整数）为参数，从右填充空格字符直到指定长度，返回这个新字符串。当指定长度等于字符串长度，返回的字符串和原字符串内容相同。当指定长度小于字符串长度，则从右去掉字符直到指定长度，返回这个新字符串。当参数的指定长度为负数时，报错。

goldfish/srfi/srfi-13.scm

△ 14 ▾

```
(define (string-pad-right str len . char+start+end)
  (define (string-pad-right-sub str len ch)
    (let ((orig-len (string-length str)))
      (if (< len orig-len)
          (string-take str len)
          (string-append str (make-string (- len orig-len) ch))))))

  (cond ((null-list? char+start+end)
         (string-pad-right-sub str len #\ ))
        ((list? char+start+end)
         (string-pad-right-sub
          (%string-from-range str (cdr char+start+end))
          len
          (car char+start+end)))
        (else (error 'wrong-type-arg "string-pad"))))
```

## 测试

tests/goldfish/liii/string-test.scm

△ 18 ▾

```
(check (string-pad-right "MathAgape" 15) => "MathAgape")
(check (string-pad-right "MathAgape" 12 #\1) => "MathAgape111")
(check (string-pad-right "MathAgape" 6 #\1 0 4) => "Math11")
(check (string-pad-right "MathAgape" 9) => "MathAgape")
(check (string-pad-right "MathAgape" 9 #\1) => "MathAgape")
(check (string-pad-right "MathAgape" 4) => "Math")
(check (string-pad "MathAgape" 2 #\1 0 4) => "th")

(check-catch 'out-of-range (string-pad-right "MathAgape" -1))
```

## string-trim

索引

`string-trim` 是一个函数，用于去除字符串左端的空白字符。空白字符通常包括空格、制表符、换行符等。当原字符串只包含空白字符，返回空字符串。

goldfish/srfi/srfi-13.scm

△ 15 ▽

```
(define (%trim-do str string-trim-sub criterion+start+end)
  (cond ((null-list? criterion+start+end)
        (string-trim-sub str #\ ))
        ((list? criterion+start+end)
         (if (char? (car criterion+start+end))
             (string-trim-sub
              (%string-from-range str (cdr criterion+start+end))
              (car criterion+start+end))
             (string-trim-sub
              (%string-from-range str criterion+start+end)
              #\ )))
        (else (error 'wrong-type-arg "string-trim"))))

(define (string-trim str . criterion+start+end)
  (define (string-trim-sub str space-or-char)
    (let loop ((i 0)
               (len (string-length str)))
      (if (or (= i len) (not (char=? space-or-char (string-ref str i))))
          (substring str i len)
          (loop (+ i 1) len))))
  (%trim-do str string-trim-sub criterion+start+end))
```

tests/goldfish/liii/string-test.scm

△ 19 ▽

```
(check (string-trim "  2 4  ") => "2 4 ")
(check (string-trim "  2 4  " 2) => "2 4 ")
(check (string-trim "  2 4  " 3) => "4 ")
(check (string-trim "  2 4  " 4) => "4 ")
(check (string-trim "  2 4  " 5) => "")

(check-catch 'out-of-range (string-trim "  2 4  " 8))

(check (string-trim "  2 4  " 0 4) => "2 ")
(check (string-trim "  2 4  " 0 7) => "2 4 ")

(check-catch 'out-of-range (string-trim "  2 4  " 0 8))

(check (string-trim "  2 4  " #\ ) => "2 4 ")
(check (string-trim "-- 2 4 --" #\ -) => "_ 2 4 _ -")
(check (string-trim "_- 345" #\ - 1) => "_ 345")
(check (string-trim "_- 345" #\ - 1 4) => "_ 3")
```

## string-trim-right 索引

`string-trim-right` 是一个函数，用于去除字符串右端的空白字符。当原字符串只包含空白字符，返回空字符串。

goldfish/srfi/srfi-13.scm

△ 16 ▽

```
(define (string-trim-right str . criterion+start+end)
  (define (string-trim-right-sub str space-or-char)
    (let loop ((i (- (string-length str) 1))
               (cond ((negative? i) "")
                      ((char=? space-or-char (string-ref str i)) (loop (- i 1)))
                      (else (substring str 0 (+ i 1))))))
  (%trim-do str string-trim-right-sub criterion+start+end))
```

tests/goldfish/liii/string-test.scm

△ 20 ▾

```
(check (string-trim-right "  2 4  ") => "2 4")
(check (string-trim-right "  2 4  " 1) => " 2 4")
(check (string-trim-right "  2 4  " 2) => "2 4")
(check (string-trim-right "  2 4  " 3) => " 4")
(check (string-trim-right "  2 4  " 4) => "4")
(check (string-trim-right "  2 4  " 5) => "")
(check (string-trim-right "  2 4  " 6) => "")
(check (string-trim-right "  2 4  " 7) => "")

(check-catch 'out-of-range (string-trim-right "  2 4  " 8))

(check (string-trim-right "  2 4  " 0 4) => " 2 2")
(check (string-trim-right "  2 4  " 0 7) => " 2 2 4")

(check-catch 'out-of-range (string-trim-right "  2 4  " 0 8))

(check (string-trim-right "  2 4  " #\ ) => " 2 2 4")
(check (string-trim-right "-- 2 4 --" #\ -) => "-- 2 4")
(check (string-trim-right "012-" #\ - 1) => "12")
(check (string-trim-right "012-4" #\ - 0 4) => "012")
```

## string-trim-both

索引

`string-trim-both` 是一个函数，用于去除字符串两端的空白字符。当原字符串只包含空白字符，报错。

goldfish/srfi/srfi-13.scm

△ 17 ▾

```
(define (string-trim-both str . criterion+start+end)
  (define (string-trim-both-sub str space-or-char)
    (let loop ((i 0)
              (len (string-length str)))
      (if (or (= i len) (not (char=? space-or-char (string-ref str i))))
          (let loop-end ((j (- len 1)))
            (if (or (< j 0) (not (char=? space-or-char (string-ref str j))))
                (substring str i (+ j 1))
                (loop-end (- j 1))))
          (loop (+ i 1) len))))
  (%trim-do str string-trim-both-sub criterion+start+end))
```

tests/goldfish/liii/string-test.scm

△ 21 ▾

```
(check (string-trim-both "  2 4  ") => "2 4")
(check (string-trim-both "-- 2 4 --" #\ -) => "2 4")
```

## 9.4.6 前缀和后缀

### string-prefix?

索引

`string-prefix?` 是一个谓词函数，用于检查一个字符串是否是另一个字符串的前缀。若第一个字符串是第二个字符串的前缀，则函数返回 `#t`；否则返回 `#f`。特别地，空字符串是任意字符串的前缀；两

个内容相同的字符串互为前缀。

[goldfish/srfi/srfi-13.scm](#)

△ 18 ▾

```
(define (string-prefix? prefix str)
  (let* ((prefix-len (string-length prefix))
        (str-len (string-length str)))
    (and (<= prefix-len str-len)
         (let loop ((i 0))
           (or (= i prefix-len)
               (and (char=? (string-ref prefix i)
                          (string-ref str i))
                    (loop (+ i 1))))))))
```

string-suffix?

索引

string-suffix?是一个谓词函数，用于检查一个字符串是否是另一个字符串的后缀。若第一个字符串是第二个字符串的后缀，则函数返回#t；否则返回#f。特别地，空字符串是任意字符串的后缀；两个内容相同的字符串互为后缀。

[goldfish/srfi/srfi-13.scm](#)

△ 19 ▾

```
(define (string-suffix? suffix str)
  (let* ((suffix-len (string-length suffix))
        (str-len (string-length str)))
    (and (<= suffix-len str-len)
         (let loop ((i 0))
           (j (- str-len suffix-len))
           (or (= i suffix-len)
               (and (char=? (string-ref suffix i)
                          (string-ref str j))
                    (loop (+ i 1) (+ j 1))))))))
```

## 9.4.7 搜索

string-index

索引

[goldfish/srfi/srfi-13.scm](#)

△ 20 ▾

```
(define (string-index str char/pred? . start+end)
  (define (string-index-sub str pred?)
    (let loop ((i 0))
      (cond (>= i (string-length str)) #f
            ((pred? (string-ref str i)) i)
            (else (loop (+ i 1)))))

  (let* ((start (if (null-list? start+end) 0 (car start+end)))
        (str-sub (%string-from-range str start+end))
        (pred? (%make-criterion char/pred?))
        (ret (string-index-sub str-sub pred?)))
    (if ret (+ start ret) ret)))
```



tests/goldfish/liii/string-test.scm

△ 22 ▾

```
(check (string-index "0123456789" #\2) => 2)
(check (string-index "0123456789" #\2 2) => 2)
(check (string-index "0123456789" #\2 3) => #f)
(check (string-index "01x3456789" char-alphabetic?) => 2)
```

**string-index-right**

索引

goldfish/srfi/srfi-13.scm

△ 21 ▾

```
(define (string-index-right str char/pred? . start+end)
  (define (string-index-right-sub str pred?)
    (let loop ((i (- (string-length str) 1)))
      (cond ((< i 0) #f)
            ((pred? (string-ref str i)) i)
            (else (loop (- i 1))))))
  (let* ((start (if (null-list? start+end) 0 (car start+end)))
         (str-sub (%string-from-range str start+end))
         (pred? (%make-criterion char/pred?))
         (ret (string-index-right-sub str-sub pred?)))
    (if ret (+ start ret) ret)))
```

tests/goldfish/liii/string-test.scm

△ 23 ▾

```
(check (string-index-right "0123456789" #\8) => 8)
(check (string-index-right "0123456789" #\8 2) => 8)
(check (string-index-right "0123456789" #\8 9) => #f)
(check (string-index-right "01234567x9" char-alphabetic?) => 8)
```

**string-contains**

索引

`string-contains` 是一个函数，用于检查一个字符串是否包含另一个子字符串，包含则返回 `#t`，返回 `#f`。

goldfish/srfi/srfi-13.scm

△ 22 ▾

```
(define (string-contains str sub-str)
  (let loop ((i 0))
    (let ((len (string-length str))
          (sub-str-len (string-length sub-str)))
      (if (> i (- len sub-str-len))
          #f
          (if (string=?
                (substring
                 str
                 i
                 (+ i sub-str-len))
                sub-str)
              #t
              (loop (+ i 1)))))))
```

tests/goldfish/liii/string-test.scm

△ 24 ▾

```
(check-true (string-contains "0123456789" "3"))
(check-true (string-contains "0123456789" "34"))
(check-false (string-contains "0123456789" "24"))
```

**string-count**

索引

---

```
goldfish/srfi/srfi-13.scm △ 23 ▾
(define (string-count str char/pred? . start+end)
  (let ((str-sub (%string-from-range str start+end))
        (criterion (%make-criterion char/pred?)))
    (count criterion (string->list str-sub))))
```

---

```
tests/goldfish/liii/string-test.scm △ 25 ▾
(check (string-count "xyz" #\x) => 1)
(check (string-count "xyz" #\x 0 1) => 1)
(check (string-count "xyz" #\y 0 1) => 0)
(check (string-count "xyz" #\x 0 3) => 1)
(check (string-count "xyz" (lambda (x) (char=? x #\x))) => 1)
```

---

## 9.4.8 大写小写转换

### string-titlecase

SRFI 索引  
**string-upcase**

S7内置的`string-upcase`是符合R7RS标准的，这里实现的是SRFI 13定义的`string-upcase`，需要额外实现`start`和`end`这两个可选参数。

---

```
goldfish/srfi/srfi-13.scm △ 24 ▾
(define s7-string-upcase string-upcase)

(define* (string-upcase str (start 0) (end (string-length str)))
  (let* ((left (substring str 0 start))
         (middle (substring str start end))
         (right (substring str end)))
    (string-append left (s7-string-upcase middle) right)))
```

---

```
tests/goldfish/liii/string-test.scm △ 26 ▾
(check (string-upcase "abc") => "ABC")
(check (string-upcase "abc" 0 1) => "Abc")

(check-catch 'out-of-range (string-upcase "abc" 0 4))
```

---

SRFI 索引  
**string-downcase**

S7内置的`string-downcase`是符合R7RS标准的，这里实现的是SRFI 13定义的`string-downcase`，需要额外实现`start`和`end`这两个可选参数。

---

```
goldfish/srfi/srfi-13.scm △ 25 ▾
(define s7-string-downcase string-downcase)

(define* (string-downcase str (start 0) (end (string-length str)))
  (let* ((left (substring str 0 start))
         (middle (substring str start end))
         (right (substring str end)))
    (string-append left (s7-string-downcase middle) right)))
```

---

```
tests/goldfish/liii/string-test.scm △ 27 ▾
(check (string-downcase "ABC") => "abc")
(check (string-downcase "ABC" 0 1) => "aBc")

(check-catch 'out-of-range (string-downcase "ABC" 0 4))
```

---

## 9.4.9 翻转和追加

### string-reverse 索引

goldfish/srfi/srfi-13.scm

△ 26 ▾

```
(define (string-reverse str . start+end)
  (cond ((null-list? start+end)
        (reverse str))
        ((= (length start+end) 1)
         (let ((start (first start+end)))
           (string-append (substring str 0 start)
                          (reverse (substring str start))))))
        ((= (length start+end) 2)
         (let ((start (first start+end))
               (end (second start+end)))
           (string-append (substring str 0 start)
                          (reverse (substring str start end))
                          (substring str end))))
        (else (error 'wrong-number-of-args "string-reverse"))))
```

tests/goldfish/liii/string-test.scm

△ 28 ▾

```
(check (string-reverse "01234") => "43210")

(check-catch 'out-of-range (string-reverse "01234" -1))

(check (string-reverse "01234" 0) => "43210")
(check (string-reverse "01234" 1) => "04321")
(check (string-reverse "01234" 5) => "01234")

(check-catch 'out-of-range (string-reverse "01234" 6))

(check (string-reverse "01234" 0 2) => "10234")
(check (string-reverse "01234" 1 3) => "02134")
(check (string-reverse "01234" 1 5) => "04321")
(check (string-reverse "01234" 0 5) => "43210")

(check-catch 'out-of-range (string-reverse "01234" 1 6))

(check-catch 'out-of-range (string-reverse "01234" -1 3))
```

RTRS

### string-append 索引

`string-append` 是一个 S7 内置的函数，用于连接两个或多个字符串。它会将所有提供的字符串参数逐个拼接在一起，并返回一个新的字符串，原始字符串不会被修改。当没有参数时，返回空字符串。

tests/goldfish/liii/base-test.scm

△ 102 ▾

```
(check (string-append "Math" "Agape") => "MathAgape")

(check (string-append) => "")
```

## 9.4.10 高阶函数

RTRS

### string-map 索引

实现

[goldfish/scheme/base.scm](#)

△ 41 ▾

---

```
(define (string-map p . args) (apply string (apply map p args)))
```

---

**测试**[tests/goldfish/liii/string-test.scm](#)

△ 29 ▾

```
(check
  (string-map
    (lambda (ch) (integer->char (+ 1 (char->integer ch))))
    "HAL")
  => "IBM")
```

---

[RTRS](#)**string-for-each** (proc str1 [str2 ...])[索引](#)[goldfish/scheme/base.scm](#)

△ 42 ▾

---

```
(define string-for-each for-each)
```

---

[tests/goldfish/liii/string-test.scm](#)

△ 30 ▾

```
(check
  (let ((lst '()))
    (string-for-each
      (lambda (x) (set! lst (cons (char->integer x) lst)))
      "12345")
    lst)
  => '(53 52 51 50 49))

(check
  (let ((lst '()))
    (string-for-each
      (lambda (x) (set! lst (cons (- (char->integer x) (char->integer #\0)) lst)))
      "12345")
    lst)
  => '(5 4 3 2 1))

(check
  (let ((lst '()))
    (string-for-each
      (lambda (x) (set! lst (cons (- (char->integer x) (char->integer #\0)) lst)))
      "123")
    lst)
  => '(3 2 1))

(check
  (let ((lst '()))
    (string-for-each
      (lambda (x) (set! lst (cons (- (char->integer x) (char->integer #\0)) lst)))
      "")
    lst)
  => '())
```

---

**9.4.11 插入和解析****string-tokenize**[索引](#)

实现

goldfish/srfi/srfi-13.scm

△ 27 ▾

```

(define (string-tokenize str . char+start+end)

  (define (string-tokenize-sub str char)

    (define (tokenize-helper tokens cursor)
      (let ((sep-pos/false (string-index str char cursor)))
        (if (not sep-pos/false)
            (reverse (cons (substring str cursor) tokens))
            (let ((new-tokens
                    (if (= cursor sep-pos/false)
                        tokens
                        (cons (substring str cursor sep-pos/false) tokens))))
              (next-cursor (+ sep-pos/false 1)))
              (tokenize-helper new-tokens next-cursor))))))

    (tokenize-helper '() 0))

  (cond ((null-list? char+start+end)
         (string-tokenize-sub str #\ ))
        ((list? char+start+end)
         (string-tokenize-sub
          (%string-from-range str (cdr char+start+end)
                              (car char+start+end)))
         (else (error 'wrong-type-arg "string-tokenize"))))

```

## 测试

tests/goldfish/liii/string-test.scm

△ 31 ▾

```

(check (string-tokenize "1_22_333") => '("1" "22" "333"))
(check (string-tokenize "1_22_333" #\2) => '("1_" "333"))
(check (string-tokenize "1_22_333" #\ 2) => '("22" "333"))

```

## 9.5 三鲤扩展函数

### string-starts?



SRFI 13定义的string-prefix?的顺序容易弄错，故而定义string-starts?将字符串放在前面，前缀放在后面。

goldfish/liii/string.scm

△ 3 ▾

```

(define (string-starts? str prefix)
  (string-prefix? prefix str))

```

tests/goldfish/liii/string-test.scm

△ 32 ▾

```

(check-true (string-starts? "MathAgape" "Ma"))
(check-true (string-starts? "MathAgape" ""))
(check-true (string-starts? "MathAgape" "MathAgape"))

(check-false (string-starts? "MathAgape" "a"))

```

### string-ends?



SRFI 13定义的string-suffix?的顺序容易弄错，故而定义string-end?将字符串放在前面，前缀放在后面。

[goldfish/liii/string.scm](#) △ 4 ▾

```
(define (string-ends? str suffix)
  (string-suffix? suffix str))
```

[tests/goldfish/liii/string-test.scm](#) △ 33 ▾

```
(check-true (string-ends? "MathAgape" "e"))
(check-true (string-ends? "MathAgape" ""))
(check-true (string-ends? "MathAgape" "MathAgape"))

(check-false (string-ends? "MathAgape" "p"))
```

## string-remove-prefix 索引

在编程实践中，移除字符串的前缀是非常常用的功能。

[goldfish/liii/string.scm](#) △ 5 ▾

```
(define string-remove-prefix
  (typed-lambda ((str string?) (prefix string?))
    (if (string-prefix? prefix str)
        (substring str (string-length prefix)
                  str)))
```

[tests/goldfish/liii/string-test.scm](#) △ 34 ▾

```
(check (string-remove-prefix "浙江省杭州市西湖区" "浙江省") => "杭州市西湖区")
(check (string-remove-prefix "aaa" "a") => "aa")
(check (string-remove-prefix "abc" "bc") => "abc")
(check (string-remove-prefix "abc" "") => "abc")
```

## string-remove-suffix 索引

[goldfish/liii/string.scm](#) △ 6 ▾

```
(define string-remove-suffix
  (typed-lambda ((str string?) (suffix string?))
    (if (string-suffix? suffix str)
        (substring str 0 (- (string-length str) (string-length suffix))
                  (string-copy str))))
```

[tests/goldfish/liii/string-test.scm](#) △ 35 ▾

```
(check (string-remove-suffix "aaa" "a") => "aa")
(check (string-remove-suffix "aaa" "") => "aaa")
(check (string-remove-suffix "Goldfish.tmu" ".tmu") => "Goldfish")
```

## 9.6 结尾

[goldfish/liii/string.scm](#) △ 7

```
) ; end of begin
) ; end of define-library
```

[goldfish/srfi/srfi-13.scm](#) △ 28

```
) ; end of begin
) ; end of define-library
```

[tests/goldfish/liii/string-test.scm](#) △ 36

```
(check-report)
```

# 第 10 章

## (liii vector) chapter:liiii\_vector

### 10.1 许可证

[goldfish/srfi/srfi-133.scm](#)

1 ▾

---

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

---

[goldfish/liiii/vector.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

---

tests/goldfish/liii/vector-test.scm

1 ▾

---

```

;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;

```

---

## 10.2 接口

goldfish/srfi/srfi-133.scm

△ 2 ▾

---

```

(define-library (srfi srfi-133)
  (import (liii base))
  (export
    vector-empty?
    vector-count
    vector-any vector-every vector-copy vector-copy!
    vector-index vector-index-right vector-partition
    vector-swap! vector-cumulate reverse-list->vector
    vector=)
  (begin

```

---

goldfish/liii/vector.scm

△ 2 ▾

---

```

(define-library (liii vector)
  (import (srfi srfi-133)
    (liii base))
  (export
    ; S7 Scheme built-in
    make-vector vector vector-length vector-ref vector-set! vector->list list->vector
    ; from (scheme base)
    vector-copy vector-fill! vector-copy! vector->string string->vector
    vector-map vector-for-each
    ; from (srfi srfi-133)
    vector-empty?
    vector-count
    vector-any vector-every vector-copy vector-copy!
    vector-index vector-index-right vector-partition
    vector-swap! vector-cumulate reverse-list->vector
    vector=)
  (begin

```

---



## 10.3 测试

tests/goldfish/liii/vector-test.scm

△ 2 ▽

```
(import (liii list)
        (liii check)
        (liii vector)
        (only (scheme base) let-values))

(check-set-mode! 'report-failed)

(for-each (lambda (p) (check (procedure? p) => #t))
  (list
   vector-empty?
   vector-count
   vector-any vector-every vector-copy vector-copy!
   vector-index vector-index-right vector-partition
   vector-swap! vector-cumulate reverse-list->vector
   vector=))
```

## 10.4 实现

### 10.4.1 构造器

R7RS **make-vector** (k [fill]) => **vector** 索引

返回长度为k的向量，如果提供了fill，则采用fill作为每一个元素的初始值，否则每一个元素的初始值是未指定。

tests/goldfish/liii/base-test.scm

△ 103 ▽

```
(check (make-vector 1 1) => (vector 1))
(check (make-vector 3 'a) => (vector 'a 'a 'a))

(check (make-vector 0) => (vector ))
(check (vector-ref (make-vector 1) 0) => #<unspecified>)
```

R7RS **vector** (obj1 obj2 ...) => **vector** 索引

返回一个新分配的向量，其元素包含给定的参数。类似于list。

tests/goldfish/liii/base-test.scm

△ 104 ▽

```
(check (vector 'a 'b 'c) => #(a b c))
(check (vector) => #())
```

索引 **int-vector** (integer integer ...) => **vector**

返回一个所有元素都是integer类型的int-vector。int-vector是vector的子类型。

tests/goldfish/liii/vector-test.scm

△ 3 ▽

```
(check-true (vector? (int-vector 1 2 3)))
(check-catch 'wrong-type-arg (int-vector 1 2 'a))

(let1 v (int-vector 1 2 3)
  (check (vector-ref v 0) => 1)
  (check (vector-ref v 1) => 2)
  (check (vector-ref v 2) => 3))
```

**vector-unfold****vector-unfold-right**

**vector-copy** (v [start [end]]) => vector 索引

返回一个新的分配副本，包含给定向量从开始到结束的元素。新向量的元素与旧向量的元素相同（在eqv?的意义上）。

```
(define (vector-copy v)
  (let ((new-v (make-vector (vector-length v))))
    (let loop ((i 0))
      (if (= i (vector-length v))
          new-v
          (begin
             (vector-set! new-v i (vector-ref v i))
             (loop (+ i 1)))))))
```

goldfish/scheme/base.scm

△ 43 ▽

```
(define* (vector-copy v (start 0) (end (vector-length v)))
  (if (or (> start end) (> end (vector-length v)))
      (error 'out-of-range "vector-copy")
      (let ((new-v (make-vector (- end start))))
        (let loop ((i start) (j 0))
          (if (>= i end)
              new-v
              (begin
                 (vector-set! new-v j (vector-ref v i))
                 (loop (+ i 1) (+ j 1))))))))
```

tests/goldfish/liii/vector-test.scm

△ 4 ▽

```
(check (vector-copy #(0 1 2 3)) => #(0 1 2 3))
(check (vector-copy #(0 1 2 3) 1) => #(1 2 3))
(check (vector-copy #(0 1 2 3) 3) => #(3))
(check (vector-copy #(0 1 2 3) 4) => #())

(check-catch 'out-of-range (vector-copy #(0 1 2 3) 5))
(check-catch 'out-of-range (vector-copy #(0 1 2 3) 1 5))

(define my-vector #(0 1 2 3))
(check (eqv? my-vector (vector-copy #(0 1 2 3))) => #f)
(check-true
  (eqv? (vector-ref my-vector 2)
        (vector-ref (vector-copy #(0 1 2 3)) 2)))

(check (vector-copy #(0 1 2 3) 1 1) => #())
(check (vector-copy #(0 1 2 3) 1 2) => #(1))
(check (vector-copy #(0 1 2 3) 1 4) => #(1 2 3))
```

**vector-reverse-copy**

**vector-append** (v1 v2 v3 ...) => vector 索引

返回一个新分配的向量，其元素是给定向量的元素的拼接。这是一个S7内置的函数。

tests/goldfish/liii/base-test.scm

△ 105 ▽

```
(check (vector-append #(0 1 2) #(3 4 5)) => #(0 1 2 3 4 5))
```

## 10.4.2 谓词

**vector?** 索引 (vector? obj) => bool

如果obj是一个向量返回#t, 否则返回#f。

tests/goldfish/liii/base-test.scm △ 106 ▽

```
(check (vector? #(1 2 3)) => #t)
(check (vector? #()) => #f)
(check (vector? '(1 2 3)) => #f)
```

**int-vector?** (int-vector? obj) => bool

只有使用int-vector构造的vector, 才能在判定为真。

tests/goldfish/liii/vector-test.scm △ 5 ▽

```
(check-true (int-vector? (int-vector 1 2 3)))
(check-false (int-vector? (vector 1 2 3)))
```

**vector-empty?** 索引

goldfish/srfi/srfi-133.scm △ 3 ▽

```
(define (vector-empty? v)
  (when (not (vector? v))
    (error 'type-error "v is not a vector")))
(zero? (vector-length v))
```

tests/goldfish/liii/vector-test.scm △ 6 ▽

```
(check-true (vector-empty? (vector)))
(check-false (vector-empty? (vector 1)))
(check-catch 'type-error (vector-empty? 1))
```

**vector=**

使用一个元素比较器 (elt=?) 来比较传入的一组向量是否相等。当传入的向量不足两个的时候默认返回相等。元素比较器 elt=? 以及内置的比较器运算的结果都是 boolean, 否则都会抛出类型错误。

因为比较一组向量是否全部相等, 可以通过不断比较相邻两个来实现。所以这里的具体实现中, 定义了一个名为 compare2vecs 的内部 procedure, 它可以使用 elt=? 进行向量的两两比较。

goldfish/srfi/srfi-133.scm △ 4 ▽

```
(define (vector= elt=? . rest)
  (define compare2vecs
    (typed-lambda ((cmp procedure?) (vec1 vector?) (vec2 vector?))
      (let* ((len1 (vector-length vec1))
             (len2 (vector-length vec2)))
        (if (not (= len1 len2)) #f
            (let loop ((ilhs 0) (irhs 0) (len len1))
              (if (= ilhs len) #t
                  (if (not (cmp (vec1 ilhs) (vec2 irhs))) #f
                      (loop (+ 1 ilhs) (+ 1 irhs) len)))))))
    (when (not (procedure? elt=?)) (error 'type-error "elt=? should be a procedure"))
    (if (or(null? rest) (= 1 (length rest))) #t
        (let loop ((vec1 (car rest)) (vec2 (car (cdr rest))) (vrest (cdr (cdr rest))))
          (let1 rst (compare2vecs elt=? vec1 vec2)
            (when (not (boolean? rst)) (error 'type-error "elt=? should return bool"))
            (if (compare2vecs elt=? vec1 vec2)
                (if (null? vrest) #t
                    (loop vec2 (car vrest) (cdr vrest)))
                #f))))))
```

tests/goldfish/liii/vector-test.scm

△ 7 ▽

```

; trivial cases
(check-true (vector= eq?))
(check-true (vector= eq? '#(a)))
; basic cases
(check-true (vector= eq? '#(a b c d) '#(a b c d)))
(check-false (vector= eq? '#(a b c d) '#(a b d c)))
(check-false (vector= = '#(1 2 3 4 5) '#(1 2 3 4)))
(check-true (vector= = '#(1 2 3 4) '#(1 2 3 4)))
(check-true (vector= equal? '#(1 2 3) '#(1 2 3) '#(1 2 3)))
(check-false (vector= equal? '#(1 2 3) '#(1 2 3) '#(1 2 3 4)))
; error cases
(check-catch 'type-error (vector= 1 (vector (vector 'a)) (vector (vector 'a))))
; complex cases in srfi-133
(check-true (vector= equal? (vector (vector 'a)) (vector (vector 'a))))
(check-false (vector= eq? (vector (vector 'a)) (vector (vector 'a))))

```

### 10.4.3 选择器

R7RS

索引

**vector-length** (*v*) => integer

以整数返回向量中元素的数量。

tests/goldfish/liii/base-test.scm

△ 107 ▽

```

(check (vector-length #(1 2 3)) => 3)
(check (vector-length #()) => 0)

```

R7RS

索引

**vector-ref** ((*v* vector?) (*k* integer?)) => object

返回向量中索引为 *k* 的元素的内容。当 *k* 不是向量的有效索引，报错。

**注意 10.1.** 在金鱼Scheme中，我们推荐使用(`v 0`)而不是(`vector-ref v 0`)，因为前者更加简洁。在使用前者访问向量中的元素时，注意变量的命名需要能够标明变量的类型，比如在(`x 0`)中，我们无法有效分辨`x`是什么类型的变量，而(`vector-ref x 0`)能够有效地标明`x`是向量。故而，在使用前者这种简洁的语法时，建议使用`v-`前缀或者`-v`后缀来标明`x`的类型，比如(`v-x 0`)或者(`x-v 0`)。

tests/goldfish/liii/base-test.scm

△ 108 ▽

```

(let1 v #(1 2 3)
  (check (vector-ref v 0) => 1)
  (check (v 0) => 1)

  (check (vector-ref v 2) => 3)
  (check (v 2) => 3))

(check-catch 'out-of-range (vector-ref #(1 2 3) 3))
(check-catch 'out-of-range (vector-ref #() 0))

(check-catch 'wrong-type-arg (vector-ref #(1 2 3) 2.0))
(check-catch 'wrong-type-arg (vector-ref #(1 2 3) "2"))

```

### 10.4.4 迭代

R7RS

索引

**vector-map**

[goldfish/scheme/base.scm](#)

△ 44 ▾

---

```
(define (vector-map p . args) (apply vector (apply map p args)))
```

---

**RTFS** `vector-for-each` (proc vector1 [vector2 ...]) **索引**

实现

[goldfish/scheme/base.scm](#)

△ 45 ▾

---

```
(define vector-for-each for-each)
```

---

测试

[tests/goldfish/liii/vector-test.scm](#)

△ 8 ▾

---

```
(check
  (let ((lst (make-list 5)))
    (vector-for-each
      (lambda (i) (list-set! lst i (* i i)))
      #(0 1 2 3 4))
    lst)
  => '(0 1 4 9 16))
```

```
(check
  (let ((lst (make-list 5)))
    (vector-for-each
      (lambda (i) (list-set! lst i (* i i)))
      #(0 1 2))
    lst)
  => '(0 1 4 #f #f))
```

```
(check
  (let ((lst (make-list 5)))
    (vector-for-each
      (lambda (i) (list-set! lst i (* i i)))
      #())
    lst)
  => '(#f #f #f #f #f))
```

---

**vector-count** **索引**

实现

[goldfish/srfi/srfi-133.scm](#)

△ 5 ▾

---

```
; TODO optional parameters
(define (vector-count pred v)
  (let loop ((i 0) (count 0))
    (cond ((= i (vector-length v)) count)
          ((pred (vector-ref v i))
           (loop (+ i 1) (+ count 1)))
          (else (loop (+ i 1) count))))))
```

---

测试

[tests/goldfish/liii/vector-test.scm](#)

△ 9 ▾

---

```
(check (vector-count even? #()) => 0)
(check (vector-count even? #(1 3 5 7 9)) => 0)
(check (vector-count even? #(1 3 4 7 8)) => 2)
```

---

**vector-cumulate**

索引

**实现**

使用函数 `fn` 和初始值 `knil` 在向量 `vector` 上迭代，迭代结果将在一个等长的新向量中返回。

每次迭代中，`fn` 被传入上次迭代的结果 `lhs` 和向量中的元素 `vec i`，产生的结果 `cumu-i` 将存储在结果向量 `v-rst` 中。

一般而言，`knil`, `cumu-i` 的类型未必与 `vector` 中的元素相同，只需要保证 `fn cumu-i (vec i)` 能够得到 `cumu-i` 同类型的结果即可。

迭代过程可以形式地表示为：

```
vector-cumulate fn knil vec => #(fn knil (vec 0), fn cumu-0 (vec 1), fn cumu-1 (vec 2), ...)
```

[goldfish/srfi/srfi-133.scm](#)

△ 6 ▽

---

```
; Return a new vector v-rst with same length of input vector vec.
; Every element of the result is the result the i-th iteration of fn cumu_i vec_i.
; Where fn should be a procedure with 2 args.
; The type of knil and vector could be different.
; In the i-th iteration, cumu_i = fn cumu_(i-1) vec_i, with cumu_0 = fn knil vec_0.
```

```
(define vector-cumulate
  (typed-lambda ((fn procedure?) knil (vec vector?))
    (let* ((len (vector-length vec))
           (v-rst (make-vector len)))
      (let loop ((i 0) (lhs knil))
        (if (= i len)
            v-rst
            (let1 cumu-i (fn lhs (vec i))
                (begin
                  (vector-set! v-rst i cumu-i)
                  (loop (+ 1 i) cumu-i))))))))
```

---

**测试**

[tests/goldfish/liii/vector-test.scm](#)

△ 10 ▽

---

```
; Trivial cases.
(check (vector-cumulate + 0 '(1 2 3 4)) => #(1 3 6 10))
(check (vector-cumulate - 0 '(1 2 3 4)) => #(-1 -3 -6 -10))
(check (vector-cumulate * 1 '(-1 -2 -3 -4)) => #(-1 2 -6 24))

;;; Test cases of vec.
; Not a vec input.
(check-catch 'type-error (vector-cumulate + 0 'a))
; Empty vec test.
(check (vector-cumulate + 0 '()) => #())

;;; Test cases of fn.
; A case with constant fn.
(check (vector-cumulate (lambda (x y) 'a) 0 '(1 2 3)) => #(a a a))
; A wrong-number-of-args case with 1-arg fn.
(check-catch 'wrong-number-of-args (vector-cumulate (lambda (x) 'a) 0 '(1 2 3)))
; A wrong-type-arg case with args can't be mapped by fn.
(check-catch 'wrong-type-arg (vector-cumulate + '(1) '(1 2 3)))

;;; Test cases of knil.
; A case of different type of knil/cumu and vec-i.
(check (vector-cumulate (lambda (x y) (+ x 2)) 0 '(a b c)) => #(2 4 6))
```

---

## 10.4.5 搜索

### vector-any

#### 实现

```
goldfish/srfi/srfi-133.scm △ 7 ▾
; TODO optional parameters
(define (vector-any pred v)
  (let loop ((i 0))
    (cond ((= i (vector-length v)) #f)
          ((pred (vector-ref v i)) #t)
          (else (loop (+ i 1))))))
```

#### 测试

```
tests/goldfish/liii/vector-test.scm △ 11 ▾
(check (vector-any even? #()) => #f)
(check (vector-any even? #(1 3 5 7 9)) => #f)
(check (vector-any even? #(1 3 4 7 8)) => #t)
```

### vector-every

```
goldfish/srfi/srfi-133.scm △ 8 ▾
; TODO optional parameters
(define (vector-every pred v)
  (let loop ((i 0))
    (cond ((= i (vector-length v)) #t)
          ((not (pred (vector-ref v i))) #f)
          (else (loop (+ i 1))))))
```

```
tests/goldfish/liii/vector-test.scm △ 12 ▾
(check (vector-every odd? #()) => #t)
(check (vector-every odd? #(1 3 5 7 9)) => #t)
(check (vector-every odd? #(1 3 4 7 8)) => #f)
```

### vector-index

```
goldfish/srfi/srfi-133.scm △ 9 ▾
; TODO optional parameters
(define (vector-index pred v)
  (let loop ((i 0))
    (cond ((= i (vector-length v)) #f)
          ((pred (vector-ref v i)) i)
          (else (loop (+ i 1))))))
```

```
tests/goldfish/liii/vector-test.scm △ 13 ▾
(check (vector-index even? #()) => #f)
(check (vector-index even? #(1 3 5 7 9)) => #f)
(check (vector-index even? #(1 3 4 7 8)) => 2)
```

### vector-index-right

goldfish/srfi/srfi-133.scm

△ 10 ▾

```

; TODO optional parameters
(define (vector-index-right pred v)
  (let ((len (vector-length v))
        (loop ((i (- len 1))
               (cond ((< i 0) #f)
                     ((pred (vector-ref v i)) i)
                     (else (loop (- i 1)))))))
    ))

```

tests/goldfish/liii/vector-test.scm

△ 14 ▾

```

(check (vector-index-right even? #()) => #f)
(check (vector-index-right even? #(1 3 5 7 9)) => #f)
(check (vector-index-right even? #(1 3 4 7 8)) => 4)

```

vector-skip

索引

vector-skip-right

索引

vector-partition

索引

goldfish/srfi/srfi-133.scm

△ 11 ▾

```

(define (vector-partition pred v)
  (let* ((len (vector-length v))
        (cnt (vector-count pred v))
        (ret (make-vector len)))
    (let loop ((i 0) (yes 0) (no cnt))
      (if (= i len)
          (values ret cnt)
          (let ((elem (vector-ref v i)))
            (if (pred elem)
                (begin
                  (vector-set! ret yes elem)
                  (loop (+ i 1) (+ yes 1) no))
                (begin
                  (vector-set! ret no elem)
                  (loop (+ i 1) yes (+ no 1))))))))))

```

tests/goldfish/liii/vector-test.scm

△ 15 ▾

```

(define (vector-partition->list pred v)
  (let-values (((ret cnt) (vector-partition pred v))) (list ret cnt)))

(check (vector-partition->list even? #()) => '(#() 0))
(check (vector-partition->list even? #(1 3 5 7 9)) => '(#(1 3 5 7 9) 0))
(check (vector-partition->list even? #(1 3 4 7 8)) => '(#(4 8 1 3 7) 2))

```

## 10.4.6 修改器

**vector-set!** (vector-set! v k obj)

索引

该函数将对象 obj 存储到向量中索引为 k 的元素里。注意，返回的不是向量，而是那个 obj。当 k 不是向量的有效索引，报错。



tests/goldfish/liii/base-test.scm

△ 109 ▾

```
(define my-vector #(0 1 2 3))
(check my-vector => #(0 1 2 3))

(check (vector-set! my-vector 2 10) => 10)
(check my-vector => #(0 1 10 3))

(check-catch 'out-of-range (vector-set! my-vector 4 10))
```

### vector-swap!

索引

goldfish/srfi/srfi-133.scm

△ 12 ▾

```
(define (vector-swap! vec i j)
  (let ((elem-i (vector-ref vec i))
        (elem-j (vector-ref vec j)))
    (vector-set! vec i elem-j)
    (vector-set! vec j elem-i)
    ))
```

tests/goldfish/liii/vector-test.scm

△ 16 ▾

```
(define my-vector (vector 0 1 2 3))
(vector-swap! my-vector 1 2)
(check my-vector => #(0 2 1 3))

(define my-vector (vector 0 1 2 3))
(vector-swap! my-vector 1 1)
(check my-vector => #(0 1 2 3))

(define my-vector (vector 0 1 2 3))
(vector-swap! my-vector 0 (- (vector-length my-vector) 1))
(check my-vector => #(3 1 2 0))

(check-catch 'out-of-range
  (vector-swap! my-vector 1 (vector-length my-vector)))
```

R7RS

### vector-fill!

索引

goldfish/scheme/base.scm

△ 46 ▾

```
(define vector-fill! fill!)
```

tests/goldfish/liii/vector-test.scm

△ 17 ▾

```
(define my-vector (vector 0 1 2 3 4))
(fill! my-vector #f)
(check my-vector => #( #f #f #f #f #f))

(define my-vector (vector 0 1 2 3 4))
(fill! my-vector #f 1 2)
(check my-vector => #(0 #f 2 3 4))
```

R7RS

### vector-copy! (vector-copy! to at from [start [end]])

索引

to和from都是向量，vector-copy!就是从from向量复制元素依次粘贴到to向量，粘贴从to向量的第at个索引位置开始；start和end是可选参数，用于指定从from向量选取元素的范围。

```
a (vector "a0" "a1" "a2" "a3" "a4")
```

```
b (vector "b0" "b1" "b2" "b3" "b4")
```

(vector-copy! b 1 a 0 2) 就是：

第一步：从a中选取索引为[0,2)的元素，即"a0" "a1"

第二步：定位到b的索引为 1 的位置，即"b1"所在的那个位置

第三步：以刚刚定位到的那个"b1"位置为起点，把刚刚选出的元素"a0" "a1"依次替换，直到

代码实现时要注意

at、to、from的边界条件，按顺序写它们会导致报错的条件

要注意每一个参数在后面会遇到怎样的使用

第一步：只涉及单个参数的报错条件

```
(< at 0) 可省去
```

```
(> at (vector-length to)) 可省去
```

```
(< start 0)
```

```
(< start 0)
```

```
(> start (vector-length from))
```

```
(< end 0)
```

```
(> end (vector-length from))
```

第二步：两个依赖关系的参数的报错条件

```
(> start end)
```

第三步：三个依赖关系参数的报错条件

```
(> (+ at (- end start)) (vector-length to))
```

综合上方的报错条件，去掉一些多余的条件：

```
(< start 0)可省去
```

因为有(> (+ at (- end start)) (vector-length to))，所以(> at (vector-length to))可省去

```
> (define a (vector "a0" "a1" "a2" "a3" "a4"))
```

```
> (define b (vector "b0" "b1" "b2" "b3" "b4"))
```

```
> (vector-copy! b 0 a 1)
```

```
> b
```

```
> a
```

```
>
```

[goldfish/scheme/base.scm](http://goldfish/scheme/base.scm)

△ 47 ▽

```
(define* (vector-copy! to at from (start 0) (end (vector-length from)))
```

```
  (if (or (< at 0)
```

```
        (> start (vector-length from))
```

```
        (< end 0)
```

```
        (> end (vector-length from))
```

```
        (> start end)
```

```
        (> (+ at (- end start)) (vector-length to))))
```

```
(error 'out-of-range "vector-copy!")
```

```
(let loop ((to-i at) (from-i start))
```

```
  (if (>= from-i end)
```

```
      to
```

```
      (begin
```

```
        (vector-set! to to-i (vector-ref from from-i))
```

```
        (loop (+ to-i 1) (+ from-i 1))))))
```

tests/goldfish/liii/vector-test.scm

△ 18 ▾

```

(define a (vector "a0" "a1" "a2" "a3" "a4"))
(define b (vector "b0" "b1" "b2" "b3" "b4"))

;< at 0)
(check-catch 'out-of-range (vector-copy! b -1 a))

;< start 0)
(check-catch 'out-of-range (vector-copy! b 0 a -1))

;> start (vector-length from))
(check-catch 'out-of-range (vector-copy! b 0 a 6))

;> end (vector-length from))
(check-catch 'out-of-range (vector-copy! b 0 a 0 6))

;> start end)
(check-catch 'out-of-range (vector-copy! b 0 a 2 1))

;> (+ at (- end start)) (vector-length to))
(check-catch 'out-of-range (vector-copy! b 6 a))

(check-catch 'out-of-range (vector-copy! b 1 a))

(define a (vector "a0" "a1" "a2" "a3" "a4"))
(define b (vector "b0" "b1" "b2" "b3" "b4"))
(vector-copy! b 0 a 1)
(check b => #("a1" "a2" "a3" "a4" "b4"))

(define a (vector "a0" "a1" "a2" "a3" "a4"))
(define b (vector "b0" "b1" "b2" "b3" "b4"))
(vector-copy! b 0 a 0 5)
(check b => #("a0" "a1" "a2" "a3" "a4"))

```

## 10.4.7 转换

R7RS **vector->list** (vector->list v [start [end]]) 索引

返回一个新分配的列表，包含向量中从 start 到 end 之间的元素中的对象。当 start、end 不是向量的有效索引，报错。

tests/goldfish/liii/base-test.scm

△ 110 ▾

```

(check (vector->list #()) => '())
(check (vector->list #() 0) => '())

(check-catch 'out-of-range (vector->list #() 1))

(check (vector->list #(0 1 2 3)) => '(0 1 2 3))
(check (vector->list #(0 1 2 3) 1) => '(1 2 3))
(check (vector->list #(0 1 2 3) 1 1) => '())
(check (vector->list #(0 1 2 3) 1 2) => '(1))

```

R7RS **list->vector** (list->vector l) 索引

返回一个新创建的向量，其元素初始化为列表 `list` 中的元素。注意，`vector` 不像 `list` 那样可用接收索引参数。

```
tests/goldfish/liii/base-test.scm △ 111
(check (list->vector '(0 1 2 3)) => #(0 1 2 3))
(check (list->vector '()) => #())
```

**R7RS** `reverse-list->vector` (reverse-list->vector `l`) 索引

返回一个新创建的向量，其元素初始化为反转后的列表 `list` 中的元素。  
传入的列表应当是一个 `proper-list`。

实现

```
goldfish/srfi/srfi-133.scm △ 13 ▾
; Input a proper-list, return a vector with inversed order elements.
(define reverse-list->vector
  (typed-lambda ((lst proper-list?))
    (let* ((len (length lst)) (v-rst (make-vector len)))
      (let loop ((l lst) (i (- len 1)))
        (if (null? l) v-rst
            (begin
              (vector-set! v-rst i (car l))
              (loop (cdr l) (- i 1))))))))
```

测试

常见情况：

```
tests/goldfish/liii/vector-test.scm △ 19 ▾
(check (reverse-list->vector '()) => '#())
(check (reverse-list->vector '(1 2 3)) => '#(3 2 1))
```

点状列表不是正规列表，故而会直接抛出 `type-error`：

```
tests/goldfish/liii/vector-test.scm △ 20 ▾
(check-catch 'type-error (reverse-list->vector '(1 2 . 3)))
```

循环列表不是正规列表，故而会直接抛出 `type-error`：

```
tests/goldfish/liii/vector-test.scm △ 21 ▾
(check-catch 'type-error (reverse-list->vector (circular-list 1 2 3)))
```

**R7RS** `vector->string` (`v` [`start` [`end`]]) => `string` 索引

将向量 `v` 转化为字符串，如果指定了起始索引和终止索引，则只将指定范围内的子向量转化为字符串。

```
goldfish/scheme/base.scm △ 48 ▾
; 0-clause BSD
; Bill Schottstaedt
; from S7 source repo: r7rs.scm
(define* (vector->string v (start 0) end)
  (let ((stop (or end (length v))))
    (copy v (make-string (- stop start)) start stop)))
```

tests/goldfish/liii/vector-test.scm

△ 22 ▾

```
(check (vector->string (vector #\0 #\1 #\2 #\3)) => "0123")
(check (vector->string (vector #\a #\b #\c)) => "abc")

(check (vector->string (vector #\0 #\1 #\2 #\3) 0 4) => "0123")
(check (vector->string (vector #\0 #\1 #\2 #\3) 1) => "123")
(check (vector->string (vector #\0 #\1 #\2 #\3) 1 4) => "123")
(check (vector->string (vector #\0 #\1 #\2 #\3) 1 3) => "12")
(check (vector->string (vector #\0 #\1 #\2 #\3) 1 2) => "1")

(check-catch 'out-of-range (vector->string (vector #\0 #\1 #\2 #\3) 2 10))

(check (vector->string (vector 0 1 #\2 3 4) 2 3) => "2")

(check-catch 'wrong-type-arg (vector->string (vector 0 1 #\2 3 4) 1 3))
```

R7RS

索引

**string->vector** (s [start [end]]) => vector

将字符串s转化为向量，如果指定了起始索引和终止索引，则只将指定范围内的子字符串转化为向量。

goldfish/scheme/base.scm

△ 49 ▾

```
; 0-clause BSD
; Bill Schottstaedt
; from S7 source repo: r7rs.scm
(define* (string->vector s (start 0) end)
  (let ((stop (or end (length s))))
    (copy s (make-vector (- stop start)) start stop)))
```

tests/goldfish/liii/vector-test.scm

△ 23 ▾

```
(check (string->vector "0123") => (vector #\0 #\1 #\2 #\3))
(check (string->vector "abc") => (vector #\a #\b #\c))

(check (string->vector "0123" 0 4) => (vector #\0 #\1 #\2 #\3))
(check (string->vector "0123" 1) => (vector #\1 #\2 #\3))
(check (string->vector "0123" 1 4) => (vector #\1 #\2 #\3))
(check (string->vector "0123" 1 3) => (vector #\1 #\2))
(check (string->vector "0123" 1 2) => (vector #\1))

(check-catch 'out-of-range (string->vector "0123" 2 10))
```

## 10.5 结尾

goldfish/srfi/srfi-133.scm

△ 14

```
) ; end of begin
) ; end of define-library
```

goldfish/liii/vector.scm

△ 3

```
) ; end of begin
) ; end of define-library
```

tests/goldfish/liii/vector-test.scm

△ 24

```
(check-report)
```



# 第 11 章

## (liii stack)

栈是一个先进后出（FILO）的数据结构。

这个函数库是三鲤自定义的库，参考了C++和Java的栈相关的函数库的接口。

### 11.1 许可证

[goldfish/liii/stack.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

---

### 11.2 接口

[goldfish/liii/stack.scm](#)

△ 2 ▾

```
(define-library (liii stack)
  (import (srfi srfi-9)
          (liii base)
          (liii error))
  (export
   stack
   stack? stack-empty?
   stack-size stack-top
   stack-push! stack-pop!
   stack->list
  )
  (begin
```

---

## 11.3 测试

```
tests/goldfish/liii/stack-test.scm
```

---

```
(import (liii stack)
        (liii check))

(check-set-mode! 'report-failed)

(define stack1 (stack))
(check (stack-empty? stack1) => #t)
(check (stack->list stack1) => '())
(check-catch 'value-error (stack-pop! stack1))

(stack-push! stack1 1)
(check (stack->list stack1) => '(1))
(check (stack-top stack1) => 1)
(check (stack-size stack1) => 1)
(check (stack-pop! stack1) => 1)
(check (stack-empty? stack1) => #t)
(check (stack-size stack1) => 0)

(define stack2 (stack 1 2 3))
(check (stack->list stack2) => '(1 2 3))
(check (stack-size stack2) => 3)
(check (stack-pop! stack2) => 1)
(check (stack-pop! stack2) => 2)
(check (stack-pop! stack2) => 3)

(define stack3 (stack ))
(stack-push! stack3 1)
(stack-push! stack3 2)
(stack-push! stack3 3)
(check (stack-pop! stack3) => 3)
(check (stack-pop! stack3) => 2)
(check (stack-pop! stack3) => 1)

(check-catch 'type-error (stack-empty? 1))
```

---

## 11.4 实现

```
goldfish/liii/stack.scm
```

---

△ 3 ▽

```
(define-record-type :stack
  (make-stack data)
  stack?
  (data get-data set-data!))

(define (%stack-check-parameter st)
  (when (not (stack? st))
    (error 'type-error "Parameter_␣st_␣is_␣not_␣a_␣stack")))

```

---

**stack** 索引 (x1 x2 ...) => stack

传入参数，构造一个栈，第一个参数是栈顶。如果没有参数，则构造的是空栈。



[goldfish/liii/stack.scm](#)[△ 4 ▾](#)


---

```
(define (stack . l)
  (if (null? l)
      (make-stack '())
      (make-stack l)))
```

---

**stack-empty?** <sup>索引</sup> (st) => bool[goldfish/liii/stack.scm](#)[△ 5 ▾](#)


---

```
(define (stack-empty? st)
  (%stack-check-parameter st)
  (null? (get-data st)))
```

---

**stack-size** <sup>索引</sup> (st) => int[goldfish/liii/stack.scm](#)[△ 6 ▾](#)


---

```
(define (stack-size st)
  (%stack-check-parameter st)
  (length (get-data st)))
```

---

**stack-top** <sup>索引</sup> (st) => x[goldfish/liii/stack.scm](#)[△ 7 ▾](#)


---

```
(define (stack-top st)
  (%stack-check-parameter st)
  (car (get-data st)))
```

---

**stack-push!** <sup>索引</sup> (st x) => #<unspecified>[goldfish/liii/stack.scm](#)[△ 8 ▾](#)


---

```
(define (stack-push! st elem)
  (%stack-check-parameter st)
  (set-data! st (cons elem (get-data st))))
```

---

**stack-pop!** <sup>索引</sup> (st) => x[goldfish/liii/stack.scm](#)[△ 9 ▾](#)


---

```
(define (stack-pop! st)
  (%stack-check-parameter st)
  (when (stack-empty? st)
    (error 'value-error "Failed to pop! on empty stack"))
  (let1 data (get-data st)
    (set-data! st (cdr data))
    (car data)))
```

---

**stack->list** <sup>索引</sup> (st) => list[goldfish/liii/stack.scm](#)[△ 10 ▾](#)


---

```
(define (stack->list st)
  (%stack-check-parameter st)
  (get-data st))
```

---

## 11.5 结尾

[goldfish/liii/stack.scm](#)[△ 11](#)


---

```
) ; end of begin
) ; end of library
```

---



# 第 12 章

## (liii queue)

这个函数库是三鲤自定义的库，参考了C++和Java的队列相关的函数库的接口。目前基于Scheme的列表实现，从队列取出数据的复杂度是 $O(1)$ ，从队列存入数据的复杂度是 $O(n)$ 。

### 12.1 许可证

[goldfish/liiii/queue.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

[tests/goldfish/liiii/queue-test.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

## 12.2 接口

goldfish/liii/queue.scm

△ 2 ▽

---

```
(define-library (liii queue)
  (import (liii list)
          (liii base)
          (srfi srfi-9)
          (liii error))
  (export
   queue
   queue? queue-empty?
   queue-size queue-front queue-back
   queue-pop! queue-push!
   queue->list)
  (begin
```

---

## 12.3 测试

tests/goldfish/liii/queue-test.scm

△ 2 ▽

---

```
(import (liii queue)
        (liii base)
        (liii check))

(check-set-mode! 'report-failed)

(let1 q1 (queue)
  (check-true (queue-empty? q1))
  (check (queue-size q1) => 0)
  (check-catch 'value-error (queue-pop! q1))

  (queue-push! q1 1)
  (check (queue-size q1) => 1)
  (check (queue-front q1) => 1)
  (check (queue-back q1) => 1)
  (check (queue-pop! q1) => 1)
  (check-true (queue-empty? q1))
)

(let1 q2 (queue 1 2 3)
  (check (queue-size q2) => 3)
  (check (queue-front q2) => 1)
  (check (queue-back q2) => 3)
  (check (queue-pop! q2) => 1)
  (check (queue-pop! q2) => 2)
  (check (queue-pop! q2) => 3)
)
)
```

---

## 12.4 实现

goldfish/liii/queue.scm

△ 3 ▾

```
(define-record-type :queue
  (make-queue data)
  queue?
  (data get-data set-data!))

(define (%queue-assert-type q)
  (when (not (queue? q))
    (type-error "Parameter q is not a queue")))

(define (%queue-assert-value q)
  (when (queue-empty? q)
    (value-error "q must be non-empty")))
```

**queue**

索引

第一个参数是队列的头部，最后一个参数是队列的尾部。

goldfish/liii/queue.scm

△ 4 ▾

```
(define (queue . l)
  (if (null? l)
      (make-queue '())
      (make-queue l)))
```

**queue-empty?** (queue) => bool

索引

goldfish/liii/queue.scm

△ 5 ▾

```
(define (queue-empty? q)
  (%queue-assert-type q)
  (null? (get-data q)))
```

**queue-size** (queue) => int

索引

goldfish/liii/queue.scm

△ 6 ▾

```
(define (queue-size q)
  (%queue-assert-type q)
  (length (get-data q)))
```

**queue-front** (queue) => x

索引

goldfish/liii/queue.scm

△ 7 ▾

```
(define (queue-front q)
  (%queue-assert-type q)
  (%queue-assert-value q)
  (first (get-data q)))
```

**queue-back** (queue) => x

索引

goldfish/liii/queue.scm

△ 8 ▾

```
(define (queue-back q)
  (%queue-assert-type q)
  (%queue-assert-value q)
  (last (get-data q)))
```

**queue-push!**  (queue x) => queue

[goldfish/liii/queue.scm](#)

△ 9 ▾

---

```
(define (queue-push! q x)
  (%queue-assert-type q)
  (let1 data (get-data q)
    (set-data! q (append data (list x))))))
```

---

**queue-pop!**  (queue) => x


[goldfish/liii/queue.scm](#)

△ 10 ▾

---

```
(define (queue-pop! q)
  (%queue-assert-type q)
  (%queue-assert-value q)
  (let1 data (get-data q)
    (set-data! q (cdr data))
    (car data)))
```

---

**queue->list**  (queue) => list

[goldfish/liii/queue.scm](#)

△ 11 ▾

---

```
(define (queue->list q)
  (get-data q))
```

---

## 12.5 结尾

[goldfish/liii/queue.scm](#)

△ 12

---

```
) ; end of begin
) ; end of library
```

---

[tests/goldfish/liii/queue-test.scm](#)

△ 3

---

```
(check-report)
```

---

# 第 13 章

## (liii comparator)

### 13.1 许可证

SRFI 128的参考实现依赖于SRFI 39。我们可以移除SRFI 128的参考实现对于SRFI 39的依赖。

`goldfish/liii/comparator.scm`

1 ▽

---

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

---

[goldfish/srfi/srfi-128.scm](#)

1 ▾

---

```

;;; SPDX-License-Identifier: MIT
;;;
;;; Copyright (C) John Cowan (2015). All Rights Reserved.
;;;
;;; Permission is hereby granted, free of charge, to any person
;;; obtaining a copy of this software and associated documentation
;;; files (the "Software"), to deal in the Software without
;;; restriction, including without limitation the rights to use,
;;; copy, modify, merge, publish, distribute, sublicense, and/or
;;; sell copies of the Software, and to permit persons to whom the
;;; Software is furnished to do so, subject to the following
;;; conditions:
;;;
;;; The above copyright notice and this permission notice shall be
;;; included in all copies or substantial portions of the Software.
;;;
;;; THE SOFTWARE IS PROVIDED "AS-IS", WITHOUT WARRANTY OF ANY KIND,
;;; EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
;;; OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
;;; NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
;;; HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
;;; WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
;;; FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
;;; OTHER DEALINGS IN THE SOFTWARE.

;;; Main part of the SRFI 114 reference implementation

;;; "There are two ways of constructing a software design: One way is to
;;; make it so simple that there are obviously no deficiencies, and the
;;; other way is to make it so complicated that there are no *obvious*
;;; deficiencies." --Tony Hoare

```

---

## 13.2 接口

[goldfish/liii/comparator.scm](#)

△ 2 ▾

---

```

(define-library (liii comparator)
  (import (srfi srfi-128))
  (export
    comparator? comparator-ordered? comparator-hashable?
    make-comparator make-pair-comparator make-list-comparator
    make-vector-comparator make-eq-comparator make-eqv-comparator make-equal-comparator
    boolean-hash char-hash char-ci-hash string-hash string-ci-hash
    symbol-hash number-hash
    make-default-comparator default-hash
    comparator-type-test-predicate comparator-equality-predicate
    comparator-ordering-predicate comparator-hash-function
    comparator-test-type comparator-check-type comparator-hash
    =? <? >? <=? >=?
  )
  (begin

```

---






[goldfish/srfi/srfi-128.scm](#)

△ 2 ▾

```

(define-library (srfi srfi-128)
  (import (scheme base)
          (lisp error))
  (export
   comparator? comparator-ordered? comparator-hashable?
   make-comparator make-pair-comparator make-list-comparator
   make-vector-comparator make-eq-comparator make-eqv-comparator make-equal-comparator
   boolean-hash char-hash char-ci-hash string-hash string-ci-hash
   symbol-hash number-hash
   make-default-comparator default-hash
   comparator-type-test-predicate comparator-equality-predicate
   comparator-ordering-predicate comparator-hash-function
   comparator-test-type comparator-check-type comparator-hash
   =? <? >? <=? >=?
  )
  (begin

```

comparator? comparator-ordered? comparator-hashable? comparator-type-test-predicate comparator-equality-predicate comparator-ordering-predicate comparator-hash-function [goldfish/srfi/srfi-128.scm](#)

△ 3 ▾

```

(define-record-type comparator
  (make-raw-comparator type-test equality ordering hash ordering? hash?)
  comparator?
  (type-test comparator-type-test-predicate)
  (equality comparator-equality-predicate)
  (ordering comparator-ordering-predicate)
  (hash comparator-hash-function)
  (ordering? comparator-ordered?)
  (hash? comparator-hashable?))

```


comparator-test-type [goldfish/srfi/srfi-128.scm](#)

△ 4 ▾

```

;; Invoke the test type
(define (comparator-test-type comparator obj)
  ((comparator-type-test-predicate comparator) obj))

```

comparator-check-type 

[goldfish/srfi/srfi-128.scm](#)[△ 5 ▾](#)


---

```
(define (comparator-check-type comparator obj)
  (if (comparator-test-type comparator obj)
      #t
      (type-error "comparator_type_check_failed" comparator obj)))
```

---

**comparator-hash**[索引](#)[goldfish/srfi/srfi-128.scm](#)[△ 6 ▾](#)


---

```
(define (comparator-hash comparator obj)
  ((comparator-hash-function comparator) obj))
```

---

## 13.3 测试

[tests/goldfish/liii/comparator-test.scm](#)[1 ▾](#)


---

```
(import (liii comparator)
        (liii check)
        (liii base))
```

```
(check-set-mode! 'report-failed)
```

---

## 13.4 内部函数

**binary=?**[goldfish/srfi/srfi-128.scm](#)[△ 7 ▾](#)


---

```
(define (binary=? comparator a b)
  ((comparator-equality-predicate comparator) a b))
```

---

**binary<?**[goldfish/srfi/srfi-128.scm](#)[△ 8 ▾](#)


---

```
(define (binary<? comparator a b)
  ((comparator-ordering-predicate comparator) a b))
```

---

**binary>?**[goldfish/srfi/srfi-128.scm](#)[△ 9 ▾](#)


---

```
(define (binary>? comparator a b)
  (binary<? comparator b a))
```

---

**binary<=?**[goldfish/srfi/srfi-128.scm](#)[△ 10 ▾](#)


---

```
(define (binary<=? comparator a b)
  (not (binary>? comparator a b)))
```

---

**binary>=?**[goldfish/srfi/srfi-128.scm](#)[△ 11 ▾](#)


---

```
(define (binary>=? comparator a b)
  (not (binary<? comparator a b)))
```

---

**%salt%**[goldfish/srfi/srfi-128.scm](#)[△ 12 ▾](#)


---

```
(define (%salt%)
  16064047)
```

---

**hash-bound** $2^{25} - 1$ [goldfish/srfi/srfi-128.scm](#)[△ 13 ▾](#)


---

```
(define (hash-bound)
  33554432)
```

---

**make-hasher**[goldfish/srfi/srfi-128.scm](#)[△ 14 ▾](#)


---

```
(define (make-hasher)
  (let ((result (%salt%)))
    (case-lambda
      (() result)
      ((n) (set! result (+ (modulo (* result 33) (hash-bound)) n)
                          result))))))
```

---

## 13.5 构造器

**make-comparator**[索引](#)[goldfish/srfi/srfi-128.scm](#)[△ 15 ▾](#)


---

```
(define (make-comparator type-test equality ordering hash)
  (make-raw-comparator
    (if (eq? type-test #t) (lambda (x) #t) type-test)
    (if (eq? equality #t) (lambda (x y) (eqv? (ordering x y) 0)) equality)
    (if ordering ordering (lambda (x y) (error "ordering not supported")))
    (if hash hash (lambda (x y) (error "hashing not supported")))
    (if ordering #t #f)
    (if hash #t #f)))
```

---


**make-eq-comparator** [goldfish/srfi/srfi-128.scm](#)

△ 16 ▾

---

```
(define (make-eq-comparator)
  (make-comparator #t eq? #f default-hash))
```

---


**make-eqv-comparator** [goldfish/srfi/srfi-128.scm](#)

△ 17 ▾

---

```
(define (make-eqv-comparator)
  (make-comparator #t eqv? #f default-hash))
```

---


**make-equal-comparator** [goldfish/srfi/srfi-128.scm](#)

△ 18 ▾

---

```
(define (make-equal-comparator)
  (make-comparator #t equal? #f default-hash))
```

---

**make-pair-comparator** [goldfish/srfi/srfi-128.scm](#)

△ 19 ▾

---

```
(define (make-pair-type-test car-comparator cdr-comparator)
  (lambda (obj)
    (and (pair? obj)
         (comparator-test-type car-comparator (car obj))
         (comparator-test-type cdr-comparator (cdr obj)))))


(define (make-pair=? car-comparator cdr-comparator)
  (lambda (a b)
    (and ((comparator-equality-predicate car-comparator) (car a) (car b))
         ((comparator-equality-predicate cdr-comparator) (cdr a) (cdr b)))))

(define (make-pair<? car-comparator cdr-comparator)
  (lambda (a b)
    (if (=? car-comparator (car a) (car b))
        (<? cdr-comparator (cdr a) (cdr b))
        (<? car-comparator (car a) (car b)))))

(define (make-pair-hash car-comparator cdr-comparator)
  (lambda (obj)
    (let ((acc (make-hash)))
      (acc (comparator-hash car-comparator (car obj)))
      (acc (comparator-hash cdr-comparator (cdr obj)))
      (acc))))

(define (make-pair-comparator car-comparator cdr-comparator)
  (make-comparator
   (make-pair-type-test car-comparator cdr-comparator)
   (make-pair=? car-comparator cdr-comparator)
   (make-pair<? car-comparator cdr-comparator)
   (make-pair-hash car-comparator cdr-comparator)))
```

---

**make-list-comparator** 

```

;; Cheap test for listness
(define (norp? obj) (or (null? obj) (pair? obj)))

(define (make-list-comparator element-comparator type-test empty? head tail)
  (make-comparator
    (make-list-type-test element-comparator type-test empty? head tail)
    (make-list=? element-comparator type-test empty? head tail)
    (make-list<? element-comparator type-test empty? head tail)
    (make-list-hash element-comparator type-test empty? head tail)))

(define (make-list-type-test element-comparator type-test empty? head tail)
  (lambda (obj)
    (and
      (type-test obj)
      (let ((elem-type-test (comparator-type-test-predicate element-comparator)))
        (let loop ((obj obj))
          (cond
            ((empty? obj) #t)
            ((not (elem-type-test (head obj))) #f)
            (else (loop (tail obj))))))))))

(define (make-list=? element-comparator type-test empty? head tail)
  (lambda (a b)
    (let ((elem=? (comparator-equality-predicate element-comparator)))
      (let loop ((a a) (b b))
        (cond
          ((and (empty? a) (empty? b) #t))
          ((empty? a) #f)
          ((empty? b) #f)
          ((elem=? (head a) (head b)) (loop (tail a) (tail b)))
          (else #f))))))

(define (make-list<? element-comparator type-test empty? head tail)
  (lambda (a b)
    (let ((elem=? (comparator-equality-predicate element-comparator))
          (elem<? (comparator-ordering-predicate element-comparator)))
      (let loop ((a a) (b b))
        (cond
          ((and (empty? a) (empty? b) #f))
          ((empty? a) #t)
          ((empty? b) #f)
          ((elem=? (head a) (head b)) (loop (tail a) (tail b)))
          ((elem<? (head a) (head b)) #t)
          (else #f))))))

(define (make-list-hash element-comparator type-test empty? head tail)
  (lambda (obj)
    (let ((elem-hash (comparator-hash-function element-comparator))
          (acc (make-hashtab)))
      (let loop ((obj obj))
        (cond
          ((empty? obj) (acc))
          (else (acc (elem-hash (head obj))) (loop (tail obj))))))))

```

---

## make-vector-comparator



[goldfish/srfi/srfi-128.scm](http://goldfish/srfi/srfi-128.scm)

△ 21 ▽

```
(define (make-vector-comparator element-comparator type-test length ref)
  (make-comparator
   (make-vector-type-test element-comparator type-test length ref)
   (make-vector=? element-comparator type-test length ref)
   (make-vector<? element-comparator type-test length ref)
   (make-vector-hash element-comparator type-test length ref)))

(define (make-vector-type-test element-comparator type-test length ref)
  (lambda (obj)
    (and
     (type-test obj)
     (let ((elem-type-test (comparator-type-test-predicate element-comparator))
           (len (length obj)))
       (let loop ((n 0))
         (cond
          ((= n len) #t)
          ((not (elem-type-test (ref obj n))) #f)
          (else (loop (+ n 1))))))))))

(define (make-vector=? element-comparator type-test length ref)
  (lambda (a b)
    (and
     (= (length a) (length b))
     (let ((elem=? (comparator-equality-predicate element-comparator))
           (len (length b)))
       (let loop ((n 0))
         (cond
          ((= n len) #t)
          ((elem=? (ref a n) (ref b n)) (loop (+ n 1)))
          (else #f))))))))

(define (make-vector<? element-comparator type-test length ref)
  (lambda (a b)
    (cond
     ((< (length a) (length b)) #t)
     ((> (length a) (length b)) #f)
     (else
      (let ((elem=? (comparator-equality-predicate element-comparator))
            (elem<=? (comparator-ordering-predicate element-comparator))
            (len (length a)))
        (let loop ((n 0))
          (cond
           ((= n len) #f)
           ((elem=? (ref a n) (ref b n)) (loop (+ n 1)))
           ((elem<=? (ref a n) (ref b n)) #t)
           (else #f))))))))))

(define (make-vector-hash element-comparator type-test length ref)
  (lambda (obj)
    (let ((elem-hash (comparator-hash-function element-comparator))
          (acc (make-hashtab))
          (len (length obj)))
      (let loop ((n 0))
        (cond
         ((= n len) (acc))
         (else (acc (elem-hash (ref obj n)) (loop (+ n 1))))))))))
```

## 13.6 标准函数

### object-type

[goldfish/srfi/srfi-128.scm](#)

△ 22 ▾

```
(define (object-type obj)
  (cond
    ((null? obj) 0)
    ((pair? obj) 1)
    ((boolean? obj) 2)
    ((char? obj) 3)
    ((string? obj) 4)
    ((symbol? obj) 5)
    ((number? obj) 6)
    ((vector? obj) 7)
    ((bytevector? obj) 8)
    (else 65535)))
```

### boolean<?

[索引](#)

[goldfish/srfi/srfi-128.scm](#)

△ 23 ▾

```
(define (boolean<? a b)
  ;; #f < #t but not otherwise
  (and (not a) b))
```

### complex<?

[goldfish/srfi/srfi-128.scm](#)

△ 24 ▾

```
(define (complex<? a b)
  (if (= (real-part a) (real-part b))
      (< (imag-part a) (imag-part b))
      (< (real-part a) (real-part b))))
```

### symbol<?

[goldfish/srfi/srfi-128.scm](#)

△ 25 ▾

```
(define (symbol<? a b)
  (string<? (symbol->string a) (symbol->string b)))
```

### boolean-hash

[索引](#)

### char-hash

[索引](#)

### char-ci-hash

[索引](#)

### string-hash

[索引](#)

### string-ci-hash

[索引](#)

### symbol-hash

[索引](#)

**number-hash**

索引

**default-hash**

索引

[goldfish/srfi/srfi-128.scm](#)

△ 26 ▾

---

```
(define boolean-hash hash-code)
(define char-hash hash-code)
(define char-ci-hash hash-code)
(define string-hash hash-code)
(define string-ci-hash hash-code)
(define symbol-hash hash-code)
(define number-hash hash-code)
(define default-hash hash-code)
```

---

**default-ordering**[goldfish/srfi/srfi-128.scm](#)

△ 27 ▾

---

```
(define (dispatch-ordering type a b)
  (case type
    ((0) 0) ; All empty lists are equal
    ((1) ((make-pair? (make-default-comparator) (make-default-comparator)) a b))
    ((2) (boolean? a b))
    ((3) (char? a b))
    ((4) (string? a b))
    ((5) (symbol? a b))
    ((6) (complex? a b))
    ((7) ((make-vector? (make-default-comparator) vector? vector-length vector-ref)
          a b))
    ((8) ((make-vector? (make-comparator exact-integer? = < default-hash)
                          bytevector? bytevector-length bytevector-u8-ref)
          a b))
    ; Add more here
    (else (binary? (registered-comparator type) a b))))

(define (default-ordering a b)
  (let ((a-type (object-type a))
        (b-type (object-type b)))
    (cond
      ((< a-type b-type) #t)
      (> a-type b-type) #f)
    (else (dispatch-ordering a-type a b)))))
```

---



**default-equality**[goldfish/srfi/srfi-128.scm](#)

△ 28 ▾

```
(define (dispatch-equality type a b)
  (case type
    ((0) #t) ; All empty lists are equal
    ((1) ((make-pair=? (make-default-comparator) (make-default-comparator)) a b))
    ((2) (boolean=? a b))
    ((3) (char=? a b))
    ((4) (string=? a b))
    ((5) (symbol=? a b))
    ((6) (= a b))
    ((7) ((make-vector=? (make-default-comparator)
                          vector? vector-length vector-ref) a b))
    ((8) ((make-vector=? (make-comparator exact-integer? = < default-hash)
                          bytevector? bytevector-length bytevector-u8-ref) a b))
    ; Add more here
    (else (binary=? (registered-comparator type) a b))))

(define (default-equality a b)
  (let ((a-type (object-type a))
        (b-type (object-type b)))
    (if (= a-type b-type)
        (dispatch-equality a-type a b)
        #f)))
```

**make-default-comparator**

索引

[goldfish/srfi/srfi-128.scm](#)

△ 29 ▾

```
(define (make-default-comparator)
  (make-comparator
   (lambda (obj) #t)
   default-equality
   default-ordering
   default-hash))
```

[tests/goldfish/liii/comparator-test.scm](#)

△ 2 ▾

```
(let1 default-comp (make-default-comparator)
  (check-false (<? default-comp #t #t))
  (check-false (<? default-comp #f #f))
  (check-true (<? default-comp #f #t))
  (check-false (<? default-comp #t #f))
  (check-true (<? default-comp (cons #f #f) (cons #t #t)))
  (check-true (<? default-comp (list 1 2) (list 2 3)))
  (check-true (<? default-comp (list 1 2) (list 1 3)))
  (check-true (<? default-comp (vector "a" "b") (vector "b" "c")))

  (check-false (<? default-comp 1 1))
  (check-true (<? default-comp 0+1i 0+2i))
  (check-true (<? default-comp 1+2i 2+2i))

  (check (comparator-hash default-comp (list 1 2)) => 42)
)
```

## 13.7 比较谓词

=? 

[goldfish/srfi/srfi-128.scm](#) △ 30 ▾

```
(define (=? comparator a b . objs)
  (let loop ((a a) (b b) (objs objs))
    (and (binary=? comparator a b)
         (if (null? objs) #t (loop b (car objs) (cdr objs))))))
```

<? 


[goldfish/srfi/srfi-128.scm](#) △ 31 ▾

```
(define (<? comparator a b . objs)
  (let loop ((a a) (b b) (objs objs))
    (and (binary<? comparator a b)
         (if (null? objs) #t (loop b (car objs) (cdr objs))))))
```

>? 


[goldfish/srfi/srfi-128.scm](#) △ 32 ▾

```
(define (>? comparator a b . objs)
  (let loop ((a a) (b b) (objs objs))
    (and (binary>? comparator a b)
         (if (null? objs) #t (loop b (car objs) (cdr objs))))))
```

<=? 

[goldfish/srfi/srfi-128.scm](#) △ 33 ▾

```
(define (<=? comparator a b . objs)
  (let loop ((a a) (b b) (objs objs))
    (and (binary<=? comparator a b)
         (if (null? objs) #t (loop b (car objs) (cdr objs))))))
```

>=? 

[goldfish/srfi/srfi-128.scm](#) △ 34 ▾

```
(define (>=? comparator a b . objs)
  (let loop ((a a) (b b) (objs objs))
    (and (binary>=? comparator a b)
         (if (null? objs) #t (loop b (car objs) (cdr objs))))))
```

## 13.8 结尾

[goldfish/liii/comparator.scm](#) △ 3

```
) ; end of begin
) ; end of define-library
```

[goldfish/srfi/srfi-128.scm](#) △ 35

```
) ; end of begin
) ; end of define-library
```

[tests/goldfish/liii/comparator-test.scm](#) △ 3

```
(check-report)
```

# 第 14 章

## (liii hash-table)

### 14.1 许可证

[goldfish/srfi/srfi-125.scm](#)

1 ▾

---

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

---

[goldfish/liii/hash-table.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

---

tests/goldfish/liii/hash-table-test.scm

1 ▾

---

```

;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
;

```

---

## 14.2 测试

tests/goldfish/liii/hash-table-test.scm

△ 2 ▾

---

```

(import (liii check)
        (liii comparator)
        (liii hash-table)
        (liii base))

(check-set-mode! 'report-failed)

(define empty-ht (make-hash-table))

```

---

## 14.3 接口

goldfish/srfi/srfi-125.scm

△ 2 ▾

---

```

(define-library (srfi srfi-125)
  (import (srfi srfi-1)
          (liii base)
          (liii error))
  (export
   make-hash-table hash-table hash-table-unfold alist->hash-table
   hash-table? hash-table-contains? hash-table-empty? hash-table=?
   hash-table-mutable?
   hash-table-ref hash-table-ref/default
   hash-table-set! hash-table-delete! hash-table-intern! hash-table-update!
   hash-table-update!/default hash-table-pop! hash-table-clear!
   hash-table-size hash-table-keys hash-table-values hash-table-entries
   hash-table-find hash-table-count
   hash-table-for-each hash-table-map->list
   hash-table->alist
  )
  (begin

```

---

goldfish/liii/hash-table.scm

△ 2 ▾

---

```
(define-library (liii hash-table)
  (import (srfi srfi-125)
          (srfi srfi-128))
  (export
    make-hash-table hash-table hash-table-unfold alist->hash-table
    hash-table? hash-table-contains? hash-table-empty? hash-table=?
    hash-table-mutable?
    hash-table-ref hash-table-ref/default
    hash-table-set! hash-table-delete! hash-table-intern! hash-table-update!
    hash-table-update!/default hash-table-pop! hash-table-clear!
    hash-table-size hash-table-keys hash-table-values hash-table-entries
    hash-table-find hash-table-count
    hash-table-for-each hash-table-map->list
    hash-table->alist
  )
  (begin
```

---

### 14.3.1 访问哈希表中的元素

除了SRFI 125定义的`hash-table-ref`和`hash-table-ref/default`之外，我们可以用S7 Scheme内置的访问方式：

tests/goldfish/liii/hash-table-test.scm

△ 3 ▾

---

```
(let1 ht (make-hash-table)
  (check (ht 'a) => #f)
  (hash-table-set! ht 'a 1)
  (check (ht 'a) => 1))
```

---

## 14.4 实现

### 14.4.1 子函数

goldfish/srfi/srfi-125.scm

△ 3 ▾

---

```
(define (assert-hash-table-type ht f)
  (when (not (hash-table? ht))
    (error 'type-error f "this parameter must be typed as hash-table")))

(define s7-hash-table-set! hash-table-set!)
(define s7-make-hash-table make-hash-table)
(define s7-hash-table-entries hash-table-entries)
```

---

### 14.4.2 构造器

make-hash-table



goldfish/srfi/srfi-125.scm

△ 4 ▽

---

```
(define (make-hash-table . args)
  (cond ((null? args) (s7-make-hash-table))
        ((comparator? (car args))
         (let* ((equiv (comparator-equality-predicate (car args)))
                (hash-func (comparator-hash-function (car args))))
           (s7-make-hash-table 8 (cons equiv hash-func) (cons #t #t))))
        (else (type-error "make-hash-table"))))
```

---

tests/goldfish/liii/hash-table-test.scm

△ 4 ▽

---

```
(let1 ht (make-hash-table (make-default-comparator))
  (hash-table-set! ht 1 2)
  (check (ht 1) => 2))

(let* ((mod10 (lambda (x) (modulo x 10)))
       (digit=? (lambda (x y) (= (modulo x 10) (modulo y 10))))
       (comp (make-comparator number? digit=? #f mod10))
       (ht (make-hash-table comp)))
  (hash-table-set! ht 1 2)
  (hash-table-set! ht 11 3)
  (check (ht 1) => 3)
  (check (ht 11) => 3)
  (check (ht 21) => 3))
```

---

## hash-table

### alist->hash-table

goldfish/srfi/srfi-125.scm

△ 5 ▽

---

```
(define alist->hash-table
  (typed-lambda ((lst list?))
    (when (odd? (length lst))
      (value-error "The length of lst must be even!"))
    (let1 ht (make-hash-table)
      (let loop ((rest lst))
        (if (null? rest)
            ht
            (begin
              (hash-table-set! ht (car rest) (cadr rest))
              (loop (cddr rest)))))))
```

---

## 14.4.3 谓词

### hash-table? (obj) => boolean?

S7内置函数。判断一个对象是不是哈希表。

### hash-table-contains? ((ht hash-table?) key) => boolean?

实现

goldfish/srfi/srfi-125.scm

△ 6 ▽

---

```
(define (hash-table-contains? ht key)
  (not (not (hash-table-ref ht key))))
```

---

## 测试

tests/goldfish/liii/hash-table-test.scm △ 5 ▾

```
(let1 ht (make-hash-table)
  (hash-table-set! ht 'brand 'liii)
  (check (hash-table-contains? ht 'brand) => #t)
  (hash-table-set! ht 'brand #f)
  (check (hash-table-contains? ht 'brand) => #f))
```

**hash-table-empty?** 索引 `((ht hash-table?) => boolean?)`

## 实现

goldfish/srfi/srfi-125.scm △ 7 ▾

```
(define (hash-table-empty? ht)
  (zero? (hash-table-size ht)))
```

## 测试

tests/goldfish/liii/hash-table-test.scm △ 6 ▾

```
(check (hash-table-empty? empty-ht) => #t)

(let1 test-ht (make-hash-table)
  (hash-table-set! test-ht 'key 'value)
  (check (hash-table-empty? test-ht) => #f))
```

**hash-table=?** 索引 `((ht1 hash-table?) (ht2 hash-table?)) => boolean?`

[改进实现, 需要使用typed-lambda保证ht1和ht2都是哈希表]

## 实现

goldfish/srfi/srfi-125.scm △ 8 ▾

```
(define (hash-table=? ht1 ht2)
  (equal? ht1 ht2))
```

## 测试

tests/goldfish/liii/hash-table-test.scm △ 7 ▾

```
(let ((empty-h1 (make-hash-table))
      (empty-h2 (make-hash-table)))
  (check (hash-table=? empty-h1 empty-h2) => #t))

(let ((t1 (make-hash-table))
      (t2 (make-hash-table)))
  (hash-table-set! t1 'a 1)
  (hash-table-set! t2 'a 1)
  (check (hash-table=? t1 t2) => #t)
  (hash-table-set! t1 'b 2)
  (check (hash-table=? t1 t2) => #f))
```

## 14.4.4 选择器

**hash-table-ref** 索引 `((ht hash-table?) key) => value`

ht. 哈希表

**key.** 键

**value.** 返回hash表中key这个键对应的值

SRFI 125定义的`hash-table-ref`的函数签名是这样的：`(hash-table-ref hash-table key [failure [success]])`。两参数形式的`hash-table-ref`是S7 Scheme的内置函数。

在S7 Scheme中，可以直接将`hash-table`视作一个单参数的函数，比如`(ht 'key)`等价于`(hash-table-ref ht 'key)`。

`tests/goldfish/liii/hash-table-test.scm`

△ 8 ▽

```
(check (hash-table-ref empty-ht 'key) => #f)
```

```
(let1 ht (make-hash-table)
  (hash-table-set! ht 'key 'value)
  (check (hash-table-ref ht 'key) => 'value)
  (check (ht 'key) => 'value))
```

**hash-table-ref/default** <sup>索引</sup> `((ht hash-table?) key default) => value`

[需要设计一种合理的方式，标注`default`参数实际上是惰性求值的]

**ht.** 哈希表

**key.** 键

**default.** 默认值，如果`key`这个键在哈希表中对应的值不存在，则返回默认值。注意，该默认值只有在`key`这个键不存在的时候，才会被求值。

**value.** 键对应的值，如果不存在，则为默认值。

测试

当键对应的值存在时，`default`不会被求值，故而测试中的`(display "hello")`实际不会被执行。

`tests/goldfish/liii/hash-table-test.scm`

△ 9 ▽

```
(let1 ht (make-hash-table)
  (check (hash-table-ref/default ht 'key 'value1) => 'value1)
  (check (hash-table-ref/default ht 'key (+ 1 2)) => 3)

  (hash-table-set! ht 'key 'value)
  (check (hash-table-ref/default ht 'key
    (begin (display "hello")
            (+ 1 2)))
    => 'value)
) ; end of let1
```

实现

`goldfish/srfi/srfi-125.scm`

△ 9 ▽

```
(define-macro (hash-table-ref/default ht key default)
  `(or (hash-table-ref ,ht ,key)
      ,default))
```

## 14.4.5 修改器

**hash-table-set!** <sup>索引</sup> `[typehint]`

测试



tests/goldfish/liii/hash-table-test.scm

△ 10 ▾

```
(let1 ht (make-hash-table)
  (hash-table-set! ht 'k1 'v1 'k2 'v2)
  (check (ht 'k1) => 'v1)
  (check (ht 'k2) => 'v2)
)
```

## 实现

goldfish/srfi/srfi-125.scm

△ 10 ▾

```
(define (hash-table-set! ht . rest)
  (assert-hash-table-type ht hash-table-set!)
  (let1 len (length rest)
    (when (or (odd? len) (zero? len))
      (error 'wrong-number-of-args len "but_must_be_even_and_non-zero"))

    (s7-hash-table-set! ht (car rest) (cadr rest))
    (when (> len 2)
      (apply hash-table-set! (cons ht (cddr rest))))))
```

hash-table-delete! <sup>索引</sup> [typehints]

## 测试

tests/goldfish/liii/hash-table-test.scm

△ 11 ▾

```
(let1 ht (make-hash-table)
  (hash-table-update! ht 'key 'value)
  (check (hash-table-delete! ht 'key) => 1)
  (check-false (hash-table-contains? ht 'key))

  (hash-table-update! ht 'key1 'value1)
  (hash-table-update! ht 'key2 'value2)
  (hash-table-update! ht 'key3 'value3)
  (hash-table-update! ht 'key4 'value4)
  (check (hash-table-delete! ht 'key1 'key2 'key3) => 3)
)
```

## 实现

goldfish/srfi/srfi-125.scm

△ 11 ▾

```
(define (hash-table-delete! ht key . keys)
  (assert-hash-table-type ht hash-table-delete!)
  (let1 all-keys (cons key keys)
    (length
     (filter
      (lambda (x)
        (if (hash-table-contains? ht x)
            (begin
              (s7-hash-table-set! ht x #f)
              #t)
            #f))
      all-keys))))
```

hash-table-update! <sup>索引</sup> [typehint]

## 实现

[goldfish/srfi/srfi-125.scm](#) △ 12 ▾

```
(define (hash-table-update! ht key value)
  (hash-table-set! ht key value))
```

## 测试

[tests/goldfish/liii/hash-table-test.scm](#) △ 12 ▾

```
(let1 ht (make-hash-table)
  (hash-table-update! ht 'key 'value)
  (check (ht 'key) => 'value)
  (hash-table-update! ht 'key 'value1)
  (check (ht 'key) => 'value1)
  (hash-table-update! ht 'key #f)
  (check (ht 'key) => #f))
```

**hash-table-clear!** 索引 [typehint]

## 实现

[goldfish/srfi/srfi-125.scm](#) △ 13 ▾

```
(define (hash-table-clear! ht)
  (for-each
   (lambda (key)
     (hash-table-set! ht key #f))
   (hash-table-keys ht)))
```

## 测试

[tests/goldfish/liii/hash-table-test.scm](#) △ 13 ▾

```
(let1 ht (make-hash-table)
  (hash-table-update! ht 'key 'value)
  (hash-table-update! ht 'key1 'value1)
  (hash-table-update! ht 'key2 'value2)
  (hash-table-clear! ht)
  (check-true (hash-table-empty? ht)))
```

## 14.4.6 哈希表整体

**hash-table-size** 索引 ((ht hash-table?)) => integer?

## 实现

[goldfish/srfi/srfi-125.scm](#) △ 14 ▾

```
(define hash-table-size s7-hash-table-entries)
```

## 测试

[tests/goldfish/liii/hash-table-test.scm](#) △ 14 ▾

```
(check (hash-table-size empty-ht) => 0)

(let1 populated-ht (make-hash-table)
  (hash-table-set! populated-ht 'key1 'value1)
  (hash-table-set! populated-ht 'key2 'value2)
  (hash-table-set! populated-ht 'key3 'value3)
  (check (hash-table-size populated-ht) => 3))
```

**hash-table-keys** 索引 ((ht hash-table?)) => list?

## 实现

goldfish/srfi/srfi-125.scm

△ 15 ▾

```
(define (hash-table-keys ht)
  (map car ht))
```

## 测试

tests/goldfish/liii/hash-table-test.scm

△ 15 ▾

```
(check (hash-table-keys empty-ht) => '())
```

```
(let1 ht (make-hash-table)
  (hash-table-set! ht 'k1 'v1)
  (check (hash-table-keys ht) => '(k1)))
```

hash-table-values 索引 |typehint|

## 实现

goldfish/srfi/srfi-125.scm

△ 16 ▾

```
(define (hash-table-values ht)
  (map cdr ht))
```

## 测试

tests/goldfish/liii/hash-table-test.scm

△ 16 ▾

```
(check (hash-table-values empty-ht) => '())
```

```
(let1 ht (make-hash-table)
  (hash-table-set! ht 'k1 'v1)
  (check (hash-table-values ht) => '(v1)))
```

hash-table-entries 索引 |typehint|

goldfish/srfi/srfi-125.scm

△ 17 ▾

```
(define hash-table-entries
  (typed-lambda ((ht hash-table?))
    (let ((ks (hash-table-keys ht))
          (vs (hash-table-values ht)))
      (values ks vs))))
```

tests/goldfish/liii/hash-table-test.scm

△ 17 ▾

```
(let1 ht (make-hash-table)
  (check (call-with-values (lambda () (hash-table-entries ht))
                        (lambda (ks vs) (list ks vs)))
    => (list (list ) (list )))

  (hash-table-set! ht 'k1 'v1)
  (check (call-with-values (lambda () (hash-table-entries ht))
                        (lambda (ks vs) (list ks vs)))
    => (list (list 'k1) (list 'v1))))
```

hash-table-find 索引 ((proc (ht hash-table?) failure) => obj)

对于哈希表中的每个关联项，调用proc函数处理它的键和值。如果proc返回真 (true)，那么 hash-table-find 函数将返回proc返回的结果。如果所有对proc的调用都返回假 (#f)，则返回调用thunk failure函数的结果。

goldfish/srfi/srfi-125.scm

△ 18 ▾

**hash-table-count** 索引 `((pred? procedure?) (ht hash-table?)) => integer?`

实现

```
(define hash-table-count
  (typed-lambda ((pred? procedure?) (ht hash-table?))
    (count (lambda (x) (pred? (car x) (cdr x)))
           (map values ht))))
```

测试

```
(check (hash-table-count (lambda (k v) #f) (hash-table)) => 0)
(check (hash-table-count (lambda (k v) #t) (hash-table 'a 1 'b 2 'c 3)) => 3)
(check (hash-table-count (lambda (k v) #f) (hash-table 'a 1 'b 2 'c 3)) => 0)

(check (hash-table-count (lambda (k v) (eq? k 'b)) (hash-table 'a 1 'b 2 'c 3)) => 1)

(check (hash-table-count (lambda (k v) (> v 1)) (hash-table 'a 1 'b 2 'c 3)) => 2)

(check (hash-table-count (lambda (k v) (string? k))
                          (hash-table "apple" 1 "banana" 2)) => 2)

(check (hash-table-count (lambda (k v) (and (symbol? k) (even? v)))
                          (hash-table 'apple 2 'banana 3 'cherry 4)) => 2)

(check (hash-table-count (lambda (k v) (eq? k v))
                          (hash-table 'a 'a 'b 'b 'c 'd)) => 2)

(check (hash-table-count (lambda (k v) (number? k))
                          (hash-table 1 100 2 200 3 300)) => 3)

(check (hash-table-count (lambda (k v) (list? v))
                          (hash-table 'a '(1 2) 'b '(3 4) 'c 3)) => 2)

(check (hash-table-count (lambda (k v)
                          (= (char->integer (string-ref (symbol->string k) 0)) v))
                          (hash-table 'a 97 'b 98 'c 99)) => 3)
```

## 14.4.7 映射和折叠

**hash-table-foreach** 索引 `((proc procedure?) (ht hash-table?)) => #<unspecified>`

对哈希表中的每个关联调用 `proc`，传递两个参数：关联的键和关联的值。`proc` 返回的值被丢弃。返回一个未指定的值。

```
(define hash-table-for-each
  (typed-lambda ((proc procedure?) (ht hash-table?))
    (for-each (lambda (x) (proc (car x) (cdr x)))
              ht)))
```

tests/goldfish/liii/hash-table-test.scm

△ 20 ▾

```
(let1 cnt 0
  (hash-table-for-each
    (lambda (k v)
      (set! cnt (+ cnt v)))
    (hash-table 'a 1 'b 2 'c 3))
  (check cnt => 6))
```

---

**hash-table-map->list** 索引 `((proc procedure?) (ht hash-table?)) => list?`

对哈希表中的每个关联调用 `proc`，传递两个参数：关联的键和关联的值。每次调用 `proc` 返回的值会被累积到一个列表中，并最终返回该列表。

goldfish/srfi/srfi-125.scm

△ 21 ▾

```
(define hash-table-map->list
  (typed-lambda ((proc procedure?) (ht hash-table?))
    (map (lambda (x) (proc (car x) (cdr x)))
         ht)))
```

tests/goldfish/liii/hash-table-test.scm

△ 21 ▾

```
(let* ((ht (hash-table 'a 1 'b 2 'c 3))
      (ks (hash-table-map->list (lambda (k v) k) ht))
      (vs (hash-table-map->list (lambda (k v) v) ht)))
  (check-true (in? 'a ks))
  (check-true (in? 'b ks))
  (check-true (in? 'c ks))
  (check-true (in? 1 vs))
  (check-true (in? 2 vs))
  (check-true (in? 3 vs)))
```

### 14.4.8 复制和转换

**hash-table->alist** 索引 `[typehint]`

goldfish/srfi/srfi-125.scm

△ 22 ▾

```
(define hash-table->alist
  (typed-lambda ((ht hash-table?))
    (append-map
     (lambda (x) (list (car x) (cdr x)))
     (map values ht))))
```

tests/goldfish/liii/hash-table-test.scm

△ 22 ▾

```
(let1 ht (make-hash-table)
  (check (hash-table->alist ht) => (list))
  (hash-table-set! ht 'k1 'v1)
  (check (hash-table->alist ht) => '(k1 v1)))

(check (hash-table->alist (alist->hash-table (list 'k1 'v1)))
      => (list 'k1 'v1))
```

## 14.5 结尾

tests/goldfish/liii/hash-table-test.scm

△ 23

```
(check-report)
```

---

[goldfish/srfi/srfi-125.scm](#)

△ 23

---

```
) ; end of begin  
) ; end of define-library
```

---

[goldfish/liii/hash-table.scm](#)

△ 3

---

```
) ; end of begin  
) ; end of library
```

---

# 第 15 章

## (liii set)

### 15.1 许可证

[goldfish/liiii/set.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS_IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

[goldfish/srfi/srfi-113.scm](#)

1 ▾

```
; Copyright (C) John Cowan (2015). All Rights Reserved.
;
; Permission is hereby granted, free of charge, to any person obtaining a copy of
; this software and associated documentation files (the "Software"), to deal in
; the Software without restriction, including without limitation the rights to
; use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
; of the Software, and to permit persons to whom the Software is furnished to do
; so, subject to the following conditions:
;
; The above copyright notice and this permission notice shall be included in all
; copies or substantial portions of the Software.
;
; THE SOFTWARE IS PROVIDED "AS_IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
; IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
; FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
; AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
; LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
; OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
; SOFTWARE.
;
```

## 15.2 接口

[goldfish/liii/set.scm](#)

[△ 2 ▽](#)

---

```
(define-library (liii set)
  (import (srfi srfi-113))
  (export set?)
  (begin
```

---

[goldfish/srfi/srfi-113.scm](#)

[△ 2 ▽](#)

---

```
(define-library (srfi srfi-113)
  (import (scheme base))
  (export set?)
  (begin
```

---

## 15.3 结尾

[goldfish/liii/set.scm](#)

[△ 3](#)

---

```
) ; end of begin
) ; end of define-library
```

---

[goldfish/srfi/srfi-113.scm](#)

[△ 3](#)

---

```
) ; end of begin
) ; end of define-library
```

---



## 第 16 章

# 三鲤扩展库说明

三鲤扩展库都是形如 (liii xyz) 的 Scheme 库。

### 16.1 结尾

`goldfish/scheme/base.scm`

△ 50

---

```
) ; end of begin
```

```
) ; end of define-library
```

---



# 第 17 章

## (liii base64)

### 17.1 协议

goldfish/liii/base64.scm

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

tests/goldfish/liii/base64-test.scm

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

### 17.2 接口

goldfish/liii/base64.scm

△ 2 ▾

```
(define-library (liii base64)
  (import (liii base)
          (liii bitwise))
  (export
    string-base64-encode bytevector-base64-encode base64-encode
    string-base64-decode bytevector-base64-decode base64-decode
  )
(begin
```

## 17.3 测试

```
tests/goldfish/liii/base64-test.scm
```

△ 2 ▽

```
(import (liii check)
        (liii base64))
```

```
(check-set-mode! 'report-failed)
```

## 17.4 Base64编解码介绍

Base64编码是一种将任意字节序列编码为可打印字符的方式，其编码和解码的过程如下：

### 编码的长度计算：

Base64的编码长度与输入数据的长度相关，长度计算公式为：

$$\text{length}(\text{output}) = \left\lceil \frac{\text{length}(\text{input})}{3} \right\rceil \times 4$$

其中，输入的每3个字节被转换为4个Base64字符。如果输入的字节数不是3的倍数，会在编码后用=符号填充，保证输出长度始终是4的倍数。

### 输入数据的分组与填充：

表格 17.1. 最后一组不足的元素需要设置为#f table:16.1

| b1    | b2    | b3    |
|-------|-------|-------|
| 0-255 | 0-255 | 0-255 |
| 0-255 | 0-255 | #f    |
| 0-255 | #f    | #f    |

输入的字节流被分为每组3个字节进行编码。对于不足3个字节的最后一组，根据不同情况补充#f（表示填充字节）。表格 17.1展示了3种情况：

- 如果有3个字节，则正常处理。
- 如果有2个字节，则第三个字节补充为#f。
- 如果只有1个字节，则后两个字节补充为#f。

### 映射规则：

每组3个字节  $[b_1, b_2, b_3]$  将映射为4个Base64字符  $[c_1, c_2, c_3, c_4]$ ，其中，Base64字符的取值范围是字母、数字、+、/，以及可能的填充符号=。映射的具体规则如下：

- 将3个字节（24位）按6位一组，分成4个部分，每个部分对应一个Base64字符。
- 如果不足3个字节（即有填充的情况），则最后一个或两个Base64字符会用=替代。

### 例子：

假设输入为"Man"，其ASCII值为77, 97, 110，其二进制表示为：

$$77 = 01001101, 97 = 01100001, 110 = 01101110$$

将24位二进制分为4组，每组6位：


$$010011, 010110, 000101, 101110$$

转换为Base64字符，分别对应T, W, F, u。因此，"Man"的Base64编码为TWFu。


解码时，过程反过来：将Base64字符转换为6位二进制，再拼接回原始的字节序列。

Base64编码的特点是保持输入的可读性并使其适合传输，但相较于原始数据，会增加大约33%的大小。

## 17.5 实现

bytevector-base64-encode 

string-base64-encode 

base64-encode 

[goldfish/liii/base64.scm](#)

△ 3 ▽

```
(define-constant BYTE2BASE64_BV
  (string->utf8 "ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"))

(define-constant BASE64_PAD_BYTE
  (char->integer #\=))

(define bytevector-base64-encode
  (typed-lambda ((bv bytevector?))
    (define (encode b1 b2 b3)
      (let* ((p1 b1)
             (p2 (if b2 b2 0))
             (p3 (if b3 b3 0))
             (combined (bitwise-ior (ash p1 16) (ash p2 8) p3))
             (c1 (bitwise-and (ash combined -18) #x3F))
             (c2 (bitwise-and (ash combined -12) #x3F))
             (c3 (bitwise-and (ash combined -6) #x3F))
             (c4 (bitwise-and combined #x3F)))
        (values
         (BYTE2BASE64_BV c1)
         (BYTE2BASE64_BV c2)
         (if b2 (BYTE2BASE64_BV c3) BASE64_PAD_BYTE)
         (if b3 (BYTE2BASE64_BV c4) BASE64_PAD_BYTE))))

    (let* ((input-N (bytevector-length bv))
           (output-N (* 4 (ceiling (/ input-N 3))))
           (output (make-bytevector output-N)))
      (let loop ((i 0) (j 0))
        (when (< i input-N)
          (let* ((b1 (bv i))
                 (b2 (if (< (+ i 1) input-N) (bv (+ i 1)) #f))
                 (b3 (if (< (+ i 2) input-N) (bv (+ i 2)) #f)))
            (receive (r1 r2 r3 r4) (encode b1 b2 b3)
              (bytevector-u8-set! output j r1)
              (bytevector-u8-set! output (+ j 1) r2)
              (bytevector-u8-set! output (+ j 2) r3)
              (bytevector-u8-set! output (+ j 3) r4)
              (loop (+ i 3) (+ j 4))))))
      output)))
```

[goldfish/liii/base64.scm](#)[△](#) [4](#) [▽](#)


---

```
(define string-base64-encode
  (typed-lambda ((str string?))
    (utf8->string (bytevector-base64-encode (string->utf8 str)))))
```

---

[goldfish/liii/base64.scm](#)[△](#) [5](#) [▽](#)


---

```
(define (base64-encode x)
  (cond ((string? x)
        (string-base64-encode x))
        ((bytevector? x)
        (bytevector-base64-encode x))
        (else
         (type-error "input must be string or bytevector"))))
```

---

## 测试

[tests/goldfish/liii/base64-test.scm](#)[△](#) [3](#) [▽](#)


---

```
(check (base64-encode "") => "")
(check (base64-encode "a") => "YQ==")
(check (base64-encode "z") => "eg==")
(check (base64-encode "f") => "Zg==")
(check (base64-encode "fo") => "Zm8=")
(check (base64-encode "foo") => "Zm9v")
(check (base64-encode "foob") => "Zm9vYg==")
(check (base64-encode "fooba") => "Zm9vYmE=")
(check (base64-encode "foobar") => "Zm9vYmFy")

(check-catch 'type-error (base64-encode 1))
```

---

[bytevector-base64-decode](#)[索引](#)

string-base64-decode 索引base64-decode 索引[goldfish/liii/base64.scm](#)

△ 6 ▽

```

(define-constant BASE64_TO_BYTE_V
  (let1 byte2base64-N (bytevector-length BYTE2BASE64_BV)
    (let loop ((i 0) (v (make-vector 256 -1)))
      (if (< i byte2base64-N)
          (begin
              (vector-set! v (BYTE2BASE64_BV i) i)
              (loop (+ i 1) v))
          v))))

(define (bytevector-base64-decode bv)
  (define (decode c1 c2 c3 c4)
    (let* ((b1 (BASE64_TO_BYTE_V c1))
           (b2 (BASE64_TO_BYTE_V c2))
           (b3 (BASE64_TO_BYTE_V c3))
           (b4 (BASE64_TO_BYTE_V c4)))
      (if (or (negative? b1) (negative? b2)
              (and (negative? b3) (!= c3 BASE64_PAD_BYTE))
              (and (negative? b4) (!= c4 BASE64_PAD_BYTE)))
          (value-error "Invalid_base64_input")
          (values
             (bitwise-ior (ash b1 2) (ash b2 -4))
             (bitwise-and (bitwise-ior (ash b2 4) (ash b3 -2)) #xFF)
             (bitwise-and (bitwise-ior (ash b3 6) b4) #xFF)
             (if (negative? b3) 1 (if (negative? b4) 2 3))))))

  (let* ((input-N (bytevector-length bv))
         (output-N (* input-N 3/4))
         (output (make-bytevector output-N)))

    (unless (zero? (modulo input-N 4))
      (value-error "length_of_the_input_bytevector_must_be_4X"))

    (let loop ((i 0) (j 0))
      (if (< i input-N)
          (receive (r1 r2 r3 cnt)
                  (decode (bv i) (bv (+ i 1)) (bv (+ i 2)) (bv (+ i 3)))
                  (bytevector-u8-set! output j r1)
                  (when (>= cnt 2)
                    (bytevector-u8-set! output (+ j 1) r2))
                  (when (>= cnt 3)
                    (bytevector-u8-set! output (+ j 2) r3))
                  (loop (+ i 4) (+ j cnt)))
          (let ((final (make-bytevector j)))
            (vector-copy! final 0 output 0 j)
            final))))))

```

[goldfish/liii/base64.scm](#)

△ 7 ▽

```

(define string-base64-decode
  (typed-lambda ((str string?))
    (utf8->string (bytevector-base64-decode (string->utf8 str)))))

```

[goldfish/liii/base64.scm](#)

△ 8 ▽

---

```
(define (base64-decode x)
  (cond ((string? x)
        (string-base64-decode x))
        ((bytevector? x)
         (bytevector-base64-decode x))
        (else
         (type-error "input must be string or bytevector"))))
```

---

[tests/goldfish/liii/base64-test.scm](#)

△ 4 ▽

---

```
(check (base64-decode "") => "")

(check (base64-decode "YQ==" ) => "a")
(check (base64-decode "eg==" ) => "z")
(check (base64-decode "Zg==" ) => "f")
(check (base64-decode "Zm8=" ) => "fo")
(check (base64-decode "Zm9v" ) => "foo")
(check (base64-decode "Zm9vYg==" ) => "foob")
(check (base64-decode "Zm9vYmE=" ) => "fooba")
(check (base64-decode "Zm9vYmFy" ) => "foobar")
```

---

## 17.6 结尾

[goldfish/liii/base64.scm](#)

△ 9

---

```
) ; end of begin
) ; end of define-library
```

---

[tests/goldfish/liii/base64-test.scm](#)

△ 5

---

```
(check-report)
```

---



# 第 18 章

## (liii os)

### 18.1 协议

tests/goldfish/liiii/os-test.scm

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS_IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

### 18.2 接口

### 18.3 测试

tests/goldfish/liiii/os-test.scm

△ 2 ▾

```
(import (liiii check)
        (liiii string)
        (liiii os)
        (liiii uuid)
        (scheme time))

(check-set-mode! 'report-failed)
```

## 18.4 实现

`os-linux?` () => boolean

`os-macos?` () => boolean

`os-windows?` () => boolean

`os-type` () => boolean

tests/goldfish/liii/os-test.scm

△ 3 ▾

```
(when (os-linux?)
  (check (os-type) => "Linux"))

(when (os-macos?)
  (check (os-type) => "Darwin"))

(when (os-windows?)
  (check (os-type) => "Windows"))

(when (not (os-windows?))
  (let ((t1 (current-second)))
    (os-call "sleep_1")
    (let ((t2 (current-second)))
      (check (>= (ceiling (- t2 t1)) 1) => #t))))
```

### os-temp-dir

tests/goldfish/liii/os-test.scm

△ 4 ▾

```
(when (os-windows?)
  (check (string-starts? (os-temp-dir) "C:") => #t))

(when (os-linux?)
  (check (os-temp-dir) => "/tmp"))
```

### mkdir

tests/goldfish/liii/os-test.scm

△ 5 ▾

```
(when (not (os-windows?))
  (check-catch 'file-exists-error
    (mkdir "/tmp"))
  (check (begin
    (let ((test_dir "/tmp/test_124"))
      (when (file-exists? test_dir)
        (rmdir "/tmp/test_124"))
      (mkdir "/tmp/test_124")))
    => #t))
```

### getcwd

tests/goldfish/liii/os-test.scm

△ 6 ▾

```
(check-false (string-null? (getcwd)))
```

### listdir

tests/goldfish/liii/os-test.scm

△ 7 ▽

---

```

(when (not (os-windows?))
  (check (> (vector-length (listdir "/usr")) 0) => #t))

(let* ((test-dir (string-append (os-temp-dir) (string (os-sep)) (uuid4)))
      (test-dir2 (string-append test-dir (string (os-sep))))
      (dir-a (string-append test-dir2 "a"))
      (dir-b (string-append test-dir2 "b"))
      (dir-c (string-append test-dir2 "c")))
  (mkdir test-dir)
  (mkdir dir-a)
  (mkdir dir-b)
  (mkdir dir-c)
  (let1 r (listdir test-dir)
    (check-true (in? "a" r))
    (check-true (in? "b" r))
    (check-true (in? "c" r)))
  (let1 r2 (listdir test-dir2)
    (check-true (in? "a" r2))
    (check-true (in? "b" r2))
    (check-true (in? "c" r2)))
  (rmdir dir-a)
  (rmdir dir-b)
  (rmdir dir-c)
  (rmdir test-dir))

(when (os-windows?)
  (check (> (vector-length (listdir "C:")) 0) => #t))

```

---

**getenv**

tests/goldfish/liii/os-test.scm

△ 8 ▽

---

```

(check (string-null? (getenv "PATH")) => #f)
(unsetenv "PATH")
(check (getenv "PATH") => #f)

```

---

**18.5 结尾**

tests/goldfish/liii/os-test.scm

△ 9

---

```

(check-report)

```

---



# 第 19 章

## (scheme case-lambda)

### 19.1 协议

tests/goldfish/scheme/case-lambda-test.scm

1 ▾

---

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

---

### 19.2 接口

在 Scheme 语言中，`case-lambda` 是一种特殊的 lambda 表达式，它可以根据不同数量的参数执行不同的代码块。`case-lambda` 允许你定义一个函数，这个函数根据传入参数的数量来选择执行不同的 lambda 表达式。

下面是一个使用 `case-lambda` 的示例：

tests/goldfish/scheme/case-lambda-test.scm

△ 2

---

```
(import (liii list)
        (liii check)
        (scheme case-lambda))

(check-set-mode! 'report-failed)

(define (my-func . args)
  (case-lambda
    (( "zero_args")
     ((x) (+ x x))
     ((x y) (+ x y))
     ((x y . rest) (reduce + 0 (cons x (cons y rest))))))

(check ((my-func)) => "zero_args")
(check ((my-func) 2) => 4)
(check ((my-func) 3 4) => 7)
(check ((my-func) 1 2 3 4) => 10)

(check-report)
```

---

## 19.3 实现

goldfish/scheme/case-lambda.scm

---

```
; 0-clause BSD
; Bill Schottstaedt
; from S7 source repo: r7rs.scm

(define-library (scheme case-lambda)
  (export case-lambda)
  (begin

;; case-lambda
(define-macro (case-lambda . choices)
  `(lambda args
     (case (length args)
       ,@(map (lambda (choice)
                (if (or (symbol? (car choice))
                        (negative? (length (car choice))))
                    `(else (apply (lambda ,(car choice) ,@(cdr choice)) args))
                    `((,(length (car choice)))
                      (apply (lambda ,(car choice) ,@(cdr choice)) args))))
              choices))))

) ; end of begin
) ; end of define-library
```

---

# 第 20 章

## (scheme char)

### 20.1 许可证

[goldfish/scheme/char.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

[tests/goldfish/scheme/char-test.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

### 20.2 接口

大部分 (scheme char) 里面的函数都是 S7 Scheme 内置函数，这里仍旧导出相关符号，以方便金鱼 Scheme 的用户搜索到相关函数。

goldfish/scheme/char.scm

△ 2 ▽

```
(define-library (scheme char)
  (export
    char-upcase char-downcase char-upper-case? char-lower-case? digit-value
  )
  (begin
```

tests/goldfish/scheme/char-test.scm

△ 2 ▽

```
(import (liii check)
        (scheme char))

(check-set-mode! 'report-failed)
```

## 20.3 实现

**R7RS**  
**char-upcase**

索引

S7内置函数。将小写英文字母转换为大写英文字母，如果不是小写英文字母，则直接返回。

tests/goldfish/scheme/char-test.scm

△ 3 ▽

```
(check (char-upcase #\z) => #\Z)
(check (char-upcase #\a) => #\A)

(check (char-upcase #\A) => #\A)
(check (char-upcase #\?) => #\?)
(check (char-upcase #\$) => #\$)
(check (char-upcase #\) => #\)
(check (char-upcase #\\) => #\\)
(check (char-upcase #\5) => #\5)
(check (char-upcase #\) => #\)
(check (char-upcase #\%) => #\%)
(check (char-upcase #\0) => #\0)
(check (char-upcase #\_ ) => #\_ )
(check (char-upcase #\?) => #\?)
(check (char-upcase #\space) => #\space)
(check (char-upcase #\newline) => #\newline)
(check (char-upcase #\null) => #\null)
```

**R7RS**  
**char-downcase**

索引

S7内置函数。将大写英文字母转换为小写英文字母，如果不是大写英文字母，则直接返回。

tests/goldfish/scheme/char-test.scm

△ 4 ▽

```
(check (char-downcase #\A) => #\a)
(check (char-downcase #\Z) => #\z)

(check (char-downcase #\a) => #\a)
```

**R7RS**  
**char-upper-case?**

索引

S7内置函数。检查是否是大写英文字母。

tests/goldfish/scheme/char-test.scm

△ 5 ▽

```
(check-true (char-upper-case? #\A))
(check-true (char-upper-case? #\Z))

(check-false (char-upper-case? #\a))
(check-false (char-upper-case? #\z))
```



[R7RS](#) [索引](#)  
**char-lower-case?**

S7内置函数。检查是否是小写英文字母。

tests/goldfish/scheme/char-test.scm

△ 6 ▾

```
(check-true (char-lower-case? #\a))
(check-true (char-lower-case? #\z))

(check-false (char-lower-case? #\A))
(check-false (char-lower-case? #\Z))
```

[R7RS](#) [索引](#)  
**digit-value**

实现

goldfish/scheme/char.scm

△ 3 ▾

```
(define (digit-value ch)
  (if (char-numeric? ch)
      (- (char->integer ch) (char->integer #\0))
      #f))
```

测试

tests/goldfish/scheme/char-test.scm

△ 7 ▾

```
(check (digit-value #\1) => 1)
(check (digit-value #\2) => 2)
(check (digit-value #\3) => 3)
(check (digit-value #\4) => 4)
(check (digit-value #\5) => 5)
(check (digit-value #\6) => 6)
(check (digit-value #\7) => 7)
(check (digit-value #\8) => 8)
(check (digit-value #\9) => 9)
(check (digit-value #\0) => 0)
(check (digit-value #\a) => #f)
(check (digit-value #\c) => #f)
```

## 20.4 结尾

goldfish/scheme/char.scm

△ 4

```
) ; end of begin
) ; end of define-library
```

tests/goldfish/scheme/char-test.scm

△ 8

```
(check-report)
```



# 第 21 章

## (scheme file)

### 21.1 许可证

[goldfish/scheme/file.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

### 21.2 接口

[goldfish/scheme/file.scm](#)

△ 2 ▾

```
(define-library (scheme file)
  (export open-binary-input-file open-binary-output-file)
  (begin
```

### 21.3 实现

[R7RS](#) [索引](#)  
**open-input-file**

S7内置函数。

[R7RS](#) [索引](#)  
**open-binary-input-file**

[goldfish/scheme/file.scm](#)

△ 3 ▾

```
(define open-binary-input-file open-input-file)
```

[R7RS](#) [索引](#)  
**open-output-file**

S7内置函数。

R7RS

**open-binary-output-file**

索引

[goldfish/scheme/file.scm](#)

△ 4 ▽

---

```
(define open-binary-output-file open-output-file)
```

---

## 21.4 结尾

[goldfish/scheme/file.scm](#)

△ 5

---

```
) ; end of begin  
) ; end of define-library
```

---

# 第 22 章

## (srfi sicmp)

### 22.1 许可证

[goldfish/srfi/srfi-216.scm](#)

1 ▾

---

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

---

[goldfish/srfi/sicmp.scm](#)

1 ▾

```
;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;
```

---

tests/goldfish/srfi/sicp-test.scm

1 ▾

---

```

;
; Copyright (C) 2024 The Goldfish Scheme Authors
;
; Licensed under the Apache License, Version 2.0 (the "License");
; you may not use this file except in compliance with the License.
; You may obtain a copy of the License at
;
; http://www.apache.org/licenses/LICENSE-2.0
;
; Unless required by applicable law or agreed to in writing, software
; distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT
; WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
; License for the specific language governing permissions and limitations
; under the License.
;

```

---

## 22.2 接口

goldfish/srfi/srfi-216.scm

△ 2 ▾

---

```

(define-library (srfi srfi-216)
  (export true false nil runtime)
  (import (scheme time))
  (begin

```

---

goldfish/srfi/sicp.scm

△ 2

---

```

(define-library (srfi sicp)
  (export true false nil runtime)
  (import (srfi srfi-216)))

```

---

## 22.3 测试

tests/goldfish/srfi/sicp-test.scm

△ 2

---

```

(import (srfi sicp)
  (liii os)
  (liiii check))

(display (runtime))
(newline)

(when (os-linux?)
  (os-call "sleep_0.01"))

(display (runtime))
(newline)

(check-true true)
(check-false false)
(check-true (null? nil))

(check-report)

```

---

## 22.4 实现

[goldfish/srfi/srfi-216.scm](#)

[△ 3 ▾](#)

---

```
(define true #t)

(define false #f)

(define nil '())

(define (runtime)
  (round (* 1000 (current-second))))
```

---

## 22.5 结尾

[goldfish/srfi/srfi-216.scm](#)

[△ 4](#)

---

```
) ; end of begin
) ; end of define-library
```

---





# 第 23 章

## C++ 部分

### 23.1 许可证

[src/goldfish.cpp](#)

1 ▾

---

```
//  
// Copyright (C) 2024 The Goldfish Scheme Authors  
//  
// Licensed under the Apache License, Version 2.0 (the "License");  
// you may not use this file except in compliance with the License.  
// You may obtain a copy of the License at  
//  
// http://www.apache.org/licenses/LICENSE-2.0  
//  
// Unless required by applicable law or agreed to in writing, software  
// distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT  
// WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the  
// License for the specific language governing permissions and limitations  
// under the License.  
//  
//
```

---

[src/goldfish.hpp](#)

1 ▾

---

```
//  
// Copyright (C) 2024 The Goldfish Scheme Authors  
//  
// Licensed under the Apache License, Version 2.0 (the "License");  
// you may not use this file except in compliance with the License.  
// You may obtain a copy of the License at  
//  
// http://www.apache.org/licenses/LICENSE-2.0  
//  
// Unless required by applicable law or agreed to in writing, software  
// distributed under the License is distributed on an "AS-IS" BASIS, WITHOUT  
// WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the  
// License for the specific language governing permissions and limitations  
// under the License.  
//  
//
```

---

### 23.2 入口

金鱼Scheme的入口是非常精简的，主要的逻辑在[src/goldfish.hpp](#)里面。这样做的好处在于方便复用hpp文件中的源代码。

---

```
src/goldfish.cpp
```

---

△ 2

```
#include "goldfish.hpp"
```

```
int
main (int argc, char** argv) {
    return goldfish::repl_for_community_edition (argc, argv);
}
```

---

## 23.3 实现

### 头文件

---

```
src/goldfish.hpp
```

---

△ 2 ▾

```
#include <algorithm>
#include <cstdlib>
#include <iostream>
#include <s7.h>
#include <string>
#include <vector>
```

```
#include <tbox/platform/file.h>
#include <tbox/platform/path.h>
#include <tbox/tbox.h>
```

```
#ifdef TB_CONFIG_OS_WINDOWS
#include <io.h>
#include <windows.h>
#else
#include <pwd.h>
#include <unistd.h>
#endif
```

```
#if !defined(TB_CONFIG_OS_WINDOWS)
#include <errno.h>
#include <wordexp.h>
#endif
```

---

### GOLDFISH\_VERSION

金鱼Scheme的版本，[后续统一使用17.10.9这个在导言区预定义的值]。文学编程的Build Buffer功能暂时不支持展开导言区预定义的值。

---

```
src/goldfish.hpp
```

---

△ 3 ▾

```
#define GOLDFISH_VERSION "17.10.9"
```

---

### GOLDFISH\_PATH\_MAXN

---

```
src/goldfish.hpp
```

---

△ 4 ▾

```
#define GOLDFISH_PATH_MAXN TB_PATH_MAXN
```

---

### 全局变量

[src/goldfish.hpp](#)[△ 5 ▾](#)


---

```
static std::vector<std::string> command_args= std::vector<std::string> ();
```

---

## goldfish命名空间的开始

[src/goldfish.hpp](#)[△ 6 ▾](#)


---

```
namespace goldfish {
using std::cerr;
using std::cout;
using std::endl;
using std::string;
using std::vector;

```

---

## 23.3.1 胶水代码

### 23.3.1.1 公共辅助函数

`string_vector_to_s7_vector`

将std::vector复制为S7的vector。

[src/goldfish.hpp](#)[△ 7 ▾](#)


---

```
inline s7_pointer
string_vector_to_s7_vector (s7_scheme* sc, vector<string> v) {
    int N = v.size ();
    s7_pointer ret= s7_make_vector (sc, N);
    for (int i= 0; i < N; i++) {
        s7_vector_set (sc, ret, i, s7_make_string (sc, v[i].c_str ()));
    }
    return ret;
}

```

---

### 23.3.1.2 Goldfish基础胶水代码

`g_version` [索引](#) => string?

[src/goldfish.hpp](#)[△ 8 ▾](#)


---

```
static s7_pointer
f_version (s7_scheme* sc, s7_pointer args) {
    return s7_make_string (sc, GOLDFISH_VERSION);
}

```

---

`g_delete_file` [索引](#) (((file-name string?)) => boolean?

[src/goldfish.hpp](#)[△ 9 ▾](#)


---

```
static s7_pointer
f_delete_file (s7_scheme* sc, s7_pointer args) {
    const char* path_c= s7_string (s7_car (args));
    return s7_make_boolean (sc, tb_file_remove (path_c));
}

```

---

## glue\_goldfish

[src/goldfish.hpp](#)

△ 10 ▾

```

inline void
glue_goldfish (s7_scheme* sc) {
    s7_pointer cur_env= s7_curlet (sc);


    const char* s_version      = "version";
    const char* d_version      = "(version)_=>_string";
    const char* s_delete_file= "g_delete-file";
    const char* d_delete_file= "(g_delete-file)_=>_boolean";

    s7_define (sc, cur_env, s7_make_symbol (sc, s_version),
              s7_make_typed_function (sc, s_version, f_version, 0, 0, false,
                                      d_version, NULL));

    s7_define (sc, cur_env, s7_make_symbol (sc, s_delete_file),
              s7_make_typed_function (sc, s_delete_file, f_delete_file, 1, 0,
                                      false, d_delete_file, NULL));
}

```

### 23.3.1.3 (scheme time)的胶水代码

**g\_current-second**  => inexact?[src/goldfish.hpp](#)

△ 11 ▾

```

static s7_pointer
f_current_second (s7_scheme* sc, s7_pointer args) {
    // TODO: use std::chrono::tai_clock::now() when using C++ 20
    tb_timeval_t tp= {0};
    tb_gettimeofday (&tp, tb_null);
    s7_double res= (time_t) tp.tv_sec + (tp.tv_usec / 1000000.0);
    return s7_make_real (sc, res);
}

```

## glue\_scheme\_time

[src/goldfish.hpp](#)

△ 12 ▾


```

inline void
glue_scheme_time (s7_scheme* sc) {
    s7_pointer cur_env= s7_curlet (sc);

    const char* s_current_second= "g_current-second";
    const char* d_current_second= "(g_current-second):_()=>_double,_return_the_"
                                   "current_unix_timestamp_in_double";
    s7_define (sc, cur_env, s7_make_symbol (sc, s_current_second),
              s7_make_typed_function (sc, s_current_second, f_current_second, 0,
                                      0, false, d_current_second, NULL));
}

```

### 23.3.1.4 (scheme process-context)的胶水代码

**g\_get-environment-variable** 

[src/goldfish.hpp](#)

△ 13 ▾

---

```

static s7_pointer
f_get_environment_variable (s7_scheme* sc, s7_pointer args) {
#ifdef _MSC_VER
    std::string path_sep= ";";
#else
    std::string path_sep= ":";
#endif
    std::string      ret;
    tb_size_t        size      = 0;
    const char*      key       = s7_string (s7_car (args));
    tb_environment_ref_t environment= tb_environment_init ();
    if (environment) {
        size= tb_environment_load (environment, key);
        if (size >= 1) {
            tb_for_all_if (tb_char_t const*, value, environment, value) {
                ret.append (value).append (path_sep);
            }
        }
        tb_environment_exit (environment);
    }
    if (size == 0) { // env key not found
        return s7_make_boolean (sc, false);
    }
    else {
        return s7_make_string (sc, ret.substr (0, ret.size () - 1).c_str ());
    }
}

```

---

g\_command-line

[索引](#)[src/goldfish.hpp](#)

△ 14 ▾

---

```

static s7_pointer
f_command_line (s7_scheme* sc, s7_pointer args) {
    s7_pointer ret = s7_nil (sc);
    int        size= command_args.size ();
    for (int i= size - 1; i >= 0; i--) {
        ret= s7_cons (sc, s7_make_string (sc, command_args[i].c_str ()), ret);
    }
    return ret;
}

```

---

glue\_scheme\_process\_context

---

 src/goldfish.hpp

△ 15 ▾

```

static s7_pointer
f_unset_environment_variable (s7_scheme* sc, s7_pointer args) {
    const char* env_name= s7_string (s7_car (args));
    return s7_make_boolean (sc, tb_environment_remove (env_name));
}

inline void
glue_scheme_process_context (s7_scheme* sc) {
    s7_pointer cur_env= s7_curlet (sc);


    const char* s_get_environment_variable= "g_get-environment-variable";
    const char* d_get_environment_variable=
        "(g_get-environemt-variable)_=>_string";
    const char* s_command_line= "g_command-line";
    const char* d_command_line= "(g_command-line)_=>_string";

    s7_define (sc, cur_env, s7_make_symbol (sc, s_get_environment_variable),
              s7_make_typed_function (sc, s_get_environment_variable,
                                      f_get_environment_variable, 1, 0, false,
                                      d_get_environment_variable, NULL));
    s7_define (sc, cur_env, s7_make_symbol (sc, s_command_line),
              s7_make_typed_function (sc, s_command_line, f_command_line, 0, 0,
                                      false, d_command_line, NULL));
}

```

---

### 23.3.1.5 (liii os)的胶水代码

 g\_os-type 


---

 src/goldfish.hpp


△ 16 ▾

```

static s7_pointer
f_os_type (s7_scheme* sc, s7_pointer args) {
#ifdef TB_CONFIG_OS_LINUX
    return s7_make_string (sc, "Linux");
#endif
#ifdef TB_CONFIG_OS_MACOSX
    return s7_make_string (sc, "Darwin");
#endif
#ifdef TB_CONFIG_OS_WINDOWS
    return s7_make_string (sc, "Windows");
#endif
    return s7_make_boolean (sc, false);
}

```

---

 g\_os-arch 


---

 src/goldfish.hpp

△ 17 ▾

```

static s7_pointer
f_os_arch (s7_scheme* sc, s7_pointer args) {
    return s7_make_string (sc, TB_ARCH_STRING);
}

```

---

## g\_os-call



src/goldfish.hpp

△ 18 ▾

---

```

static s7_pointer
f_os_call (s7_scheme* sc, s7_pointer args) {
    const char* cmd_c= s7_string (s7_car (args));
    tb_process_attr_t attr = {tb_null};
    attr.flags          = TB_PROCESS_FLAG_NO_WINDOW;
    int ret;

#ifdef _MSC_VER
    ret= (int) std::system (cmd_c);
#else
    wordexp_t p;
    ret= wordexp (cmd_c, &p, 0);
    if (ret != 0) {
        // failed after calling wordexp
    }
    else if (p.we_wordc == 0) {
        wordfree (&p);
        ret= EINVAL;
    }
    else {
        ret= (int) tb_process_run (p.we_wordv[0], (tb_char_t const**) p.we_wordv,
                                  &attr);
        wordfree (&p);
    }
#endif
    return s7_make_integer (sc, ret);
}

```

---

## g\_system



src/goldfish.hpp

△ 19 ▾

---

```

static s7_pointer
f_system (s7_scheme* sc, s7_pointer args) {
    const char* cmd_c= s7_string (s7_car (args));
    int ret = (int) std::system (cmd_c);
    return s7_make_integer (sc, ret);
}

```

---

## g\_os-temp-dir



src/goldfish.hpp

△ 20 ▾

---

```

static s7_pointer
f_os_temp_dir (s7_scheme* sc, s7_pointer args) {
    tb_char_t path[GOLDFISH_PATH_MAXN];
    tb_directory_temporary (path, GOLDFISH_PATH_MAXN);
    return s7_make_string (sc, path);
}

```


---

**g\_isdir** [src/goldfish.hpp](#)[△ 21 ▾](#)


```
static s7_pointer
f_isdir (s7_scheme* sc, s7_pointer args) {
    const char* dir_c= s7_string (s7_car (args));
    tb_file_info_t info;
    bool ret= false;
    if (tb_file_info (dir_c, &info)) {
        switch (info.type) {
            case TB_FILE_TYPE_DIRECTORY:
            case TB_FILE_TYPE_DOT:
            case TB_FILE_TYPE_DOT2:
                ret= true;
        }
    }
    return s7_make_boolean (sc, ret);
}
```

**g\_isfile** [src/goldfish.hpp](#)[△ 22 ▾](#)

```
static s7_pointer
f_isfile (s7_scheme* sc, s7_pointer args) {
    const char* dir_c= s7_string (s7_car (args));
    tb_file_info_t info;
    bool ret= false;
    if (tb_file_info (dir_c, &info)) {
        switch (info.type) {
            case TB_FILE_TYPE_FILE:
                ret= true;
        }
    }
    return s7_make_boolean (sc, ret);
}
```

**g\_mkdir** [src/goldfish.hpp](#)[△ 23 ▾](#)

```
static s7_pointer
f_mkdir (s7_scheme* sc, s7_pointer args) {
    const char* dir_c= s7_string (s7_car (args));
    return s7_make_boolean (sc, tb_directory_create (dir_c));
}
```

**g\_chdir** [src/goldfish.hpp](#)[△ 24 ▾](#)

```
static s7_pointer
f_chdir (s7_scheme* sc, s7_pointer args) {
    const char* dir_c= s7_string (s7_car (args));
    return s7_make_boolean (sc, tb_directory_current_set (dir_c));
}
```



**g\_getcwd**

索引

src/goldfish.hpp

△ 25 ▾

---

```
static s7_pointer
f_getcwd (s7_scheme* sc, s7_pointer args) {
    tb_char_t path[GOLDFISH_PATH_MAXN];
    tb_directory_current (path, GOLDFISH_PATH_MAXN);
    return s7_make_string (sc, path);
}
```

---

**g\_listdir**

索引

src/goldfish.hpp

△ 26 ▾

---

```
static tb_long_t
tb_directory_walk_func (tb_char_t const* path, tb_file_info_t const* info,
                       tb_cpointer_t priv) {
    // check
    tb_assert_and_check_return_val (path && info, TB_DIRECTORY_WALK_CODE_END);

    vector<string>* p_v_result= (vector<string>*) priv;
    p_v_result->push_back (string (path));
    return TB_DIRECTORY_WALK_CODE_CONTINUE;
}

static s7_pointer
f_listdir (s7_scheme* sc, s7_pointer args) {
    const char* path_c= s7_string (s7_car (args));
    vector<string> entries;
    s7_pointer ret= s7_make_vector (sc, 0);
    tb_directory_walk (path_c, 0, tb_false, tb_directory_walk_func, &entries);

    int entries_N = entries.size ();
    string path_s = string (path_c);
    int path_N = path_s.size ();
    int path_slash_N= path_N;
    char last_ch = path_s[path_N - 1];
    #if defined(TB_CONFIG_OS_WINDOWS)
    if (last_ch != '/' && last_ch != '\\') {
        path_slash_N= path_slash_N + 1;
    }
    #else
    if (last_ch != '/') {
        path_slash_N= path_slash_N + 1;
    }
    #endif
    for (int i= 0; i < entries_N; i++) {
        entries[i]= entries[i].substr (path_slash_N);
    }
    return string_vector_to_s7_vector (sc, entries);
}
```

---

g\_access

索引

src/goldfish.hpp

△ 27 ▾

---

```
static s7_pointer
f_access (s7_scheme* sc, s7_pointer args) {
    const char* path_c= s7_string (s7_car (args));
    int mode = s7_integer ((s7_cadr (args)));
#ifdef TB_CONFIG_OS_WINDOWS
    bool ret= (_access (path_c, mode) == 0);
#else
    bool ret= (access (path_c, mode) == 0);
#endif
    return s7_make_boolean (sc, ret);
}
```

---

g\_getlogin

索引

src/goldfish.hpp

△ 28 ▾

---

```
static s7_pointer
f_getlogin (s7_scheme* sc, s7_pointer args) {
#ifdef TB_CONFIG_OS_WINDOWS
    return s7_make_boolean (sc, false);
#else
    uid_t uid= getuid ();
    struct passwd* pwd= getpwuid (uid);
    return s7_make_string (sc, pwd->pw_name);
#endif
}
```

---

g\_getpid

索引

src/goldfish.hpp

△ 29 ▾

---

```
static s7_pointer
f_getpid (s7_scheme* sc, s7_pointer args) {
#ifdef TB_CONFIG_OS_WINDOWS
    return s7_make_integer (sc, (int) GetCurrentProcessId ());
#else
    return s7_make_integer (sc, getpid ());
#endif
}
```

---

glue\_liii\_os

```
src/goldfish.hpp
```

```
△ 30 ▽
```


```
inline void
glue_liii_os (s7_scheme* sc) {
    s7_pointer cur_env= s7_curlet (sc);

    const char* s_os_type      = "g_os-type";
    const char* d_os_type      = "(g_os-type)_=>_string";
    const char* s_os_arch      = "g_os-arch";
    const char* d_os_arch      = "(g_os-arch)_=>_string";
    const char* s_os_call      = "g_os-call";
    const char* d_os_call      = "(string)_=>_int";
    const char* s_system       = "g_system";
    const char* d_system       = "(string)_=>_int";
    const char* s_os_temp_dir  = "g_os-temp-dir";
    const char* d_os_temp_dir  = "(g_os-temp-dir)_=>_string";
    const char* s_isdir        = "g_isdir";
    const char* d_isdir        = "(g_isdir_string)_=>_boolean";
    const char* s_isfile       = "g_isfile";
    const char* d_isfile       = "(g_isfile_string)_=>_boolean";
    const char* s_mkdir        = "g_mkdir";
    const char* d_mkdir        = "(g_mkdir_string)_=>_boolean";
    const char* s_chdir        = "g_chdir";
    const char* d_chdir        = "(g_chdir_string)_=>_boolean";
    const char* s_listdir      = "g_listdir";
    const char* d_listdir      = "(g_listdir)_=>_vector";
    const char* s_getcwd       = "g_getcwd";
    const char* d_getcwd       = "(g_getcwd)_=>_string";
    const char* s_access       = "g_access";
    const char* d_access       = "(g_access_string_integer)_=>_boolean";
    const char* s_getlogin     = "g_getlogin";
    const char* d_getlogin     = "(g_getlogin)_=>_string";
    const char* s_getpid       = "g_getpid";
    const char* d_getpid       = "(g_getpid)_=>_integer";
    const char* s_unsetenv    = "g_unsetenv";
    const char* d_unsetenv    = "(g_unsetenv_string):_string_=>_boolean";

    s7_define (sc, cur_env, s7_make_symbol (sc, s_os_type),
               s7_make_typed_function (sc, s_os_type, f_os_type, 0, 0, false,
                                       d_os_type, NULL));
    s7_define (sc, cur_env, s7_make_symbol (sc, s_os_arch),
               s7_make_typed_function (sc, s_os_arch, f_os_arch, 0, 0, false,
                                       d_os_arch, NULL));
    s7_define (sc, cur_env, s7_make_symbol (sc, s_os_call),
               s7_make_typed_function (sc, s_os_call, f_os_call, 1, 0, false,
                                       d_os_call, NULL));
    s7_define (sc, cur_env, s7_make_symbol (sc, s_system),
               s7_make_typed_function (sc, s_system, f_system, 1, 0, false,
                                       d_system, NULL));
    s7_define (sc, cur_env, s7_make_symbol (sc, s_os_temp_dir),
               s7_make_typed_function (sc, s_os_temp_dir, f_os_temp_dir, 0, 0,
                                       false, d_os_call, NULL));
    s7_define (sc, cur_env, s7_make_symbol (sc, s_isdir),
               s7_make_typed_function (sc, s_isdir, f_isdir, 1, 0, false, d_isdir,
                                       NULL));
    s7_define (sc, cur_env, s7_make_symbol (sc, s_isfile),
               s7_make_typed_function (sc, s_isfile, f_isfile, 1, 0, false,
                                       d_isfile, NULL));
    s7_define (sc, cur_env, s7_make_symbol (sc, s_mkdir),
               s7_make_typed_function (sc, s_mkdir, f_mkdir, 1, 0, false, d_mkdir,
```

### 23.3.1.6 (liii uuid)的胶水代码

(liii uuid)是通过C++实现的, 本节是相关的胶水代码。

`g_uuid4`  => string?

src/goldfish.hpp △ 31 ▾

```
static s7_pointer
f_uuid4 (s7_scheme* sc, s7_pointer args) {
    tb_char_t      uuid[37];
    const tb_char_t* ret= tb_uuid4_make_cstr (uuid, tb_null);
    return s7_make_string (sc, ret);
}
```

### glue\_liii\_uuid

src/goldfish.hpp △ 32 ▾

```
inline void
glue_liii_uuid (s7_scheme* sc) {
    s7_pointer cur_env= s7_curlet (sc);
    const char* s_uuid4= "g_uuid4";
    const char* d_uuid4= "(g_uuid4)_=>_string";
    s7_define (sc, cur_env, s7_make_symbol (sc, s_uuid4),
              s7_make_typed_function (sc, s_uuid4, f_uuid4, 0, 0, false, d_uuid4,
                                      NULL));
}
```

### 23.3.1.7 胶水代码汇总接口

```
src/goldfish.hpp △ 33 ▾
void
glue_for_community_edition (s7_scheme* sc) {
    glue_goldfish (sc);
    glue_scheme_time (sc);
    glue_scheme_process_context (sc);
    glue_liii_os (sc);
    glue_liii_uuid (sc);
}
```

---

## 23.3.2 命令行

### 显示帮助

```
src/goldfish.hpp △ 34 ▾
static void
display_help () {
    cout << "Goldfish_Scheme_" << GOLDFISH_VERSION << "_byLiiiLabs" << endl;
    cout << "--version\t"
         << "Display_version" << endl;
    cout << "-mdefault\t"
         << "Allowed_mode:_default,_liii,_sicp,_r7rs,_s7" << endl;
    cout << "-e\t"
         << "Load_the_scheme_code_on_the_command_line" << endl
         << "\t\tteg.-e'(begin(display'Hello)(+12))'" << endl;
    cout << "-lFILE\t"
         << "Load_the_scheme_code_on_path" << endl;
    cout << "FILE\t"
         << "Load_the_scheme_code_on_path_and_print_the_evaluated_result" << endl;
}
```

---

### 显示版本

```
src/goldfish.hpp △ 35 ▾
static void
display_version () {
    cout << "Goldfish_Scheme_" << GOLDFISH_VERSION << "_byLiiiLabs" << endl;
    cout << "based_on_S7_Scheme_" << S7_VERSION << "(" << S7_DATE << ")" << endl;
}
```

---

### 错误的命令行选项

```
src/goldfish.hpp △ 36 ▾
static void
display_for_invalid_options () {
    cerr << "Invalid_command_line_options!" << endl << endl;
    display_help ();
}
```

---

## 执行文件中的代码

```
src/goldfish.hpp △ 37 ▾
static void
goldfish_eval_file (s7_scheme* sc, string path, bool quiet) {
    s7_pointer result= s7_load (sc, path.c_str ());
    if (!result) {
        cerr << "Failed to load" << path << endl;
        exit (-1);
    }
    if (!quiet) {
        cout << path << "=>" << s7_object_to_c_string (sc, result) << endl;
    }
}
```

---

## 执行命令行中以字符串形式传入的代码

```
src/goldfish.hpp △ 38 ▾
static void
goldfish_eval_code (s7_scheme* sc, string code) {
    s7_pointer x= s7_eval_c_string (sc, code.c_str ());
    cout << s7_object_to_c_string (sc, x) << endl;
}
```

---

## 金鱼Scheme的初始化

1. 初始化S7 Scheme
2. 为S7 Scheme设置load path
3. 初始化tbox
4. 初始化胶水代码

```
src/goldfish.hpp △ 39 ▾
s7_scheme*
init_goldfish_scheme (const char* gf_lib) {
    s7_scheme* sc= s7_init ();
    s7_add_to_load_path (sc, gf_lib);

    if (!tb_init (tb_null, tb_null)) exit (-1);

    glue_for_community_edition (sc);
    return sc;
}
```

---

## 金鱼Scheme的运行模式

-m帮助您指定预加载的标准库。

**default.** -m default等价于-m liii

liii. 预加载 (liii base) 和 (liii error) 的金鱼Scheme

**sicp.** 预加载 (srfi sicp) 和 (scheme base) 的 S7 Scheme

**r7rs.** 预加载 (scheme base) 的 S7 Scheme

**s7.** 无任何无加载库的 S7 Scheme

[src/goldfish.hpp](#)

△ 40 ▽

---

```

void
customize_goldfish_by_mode (s7_scheme* sc, string mode,
                           const char* boot_file_path) {
    if (mode != "s7") {
        s7_load (sc, boot_file_path);
    }

    if (mode == "default" || mode == "liii") {
        s7_eval_c_string (sc, "(import_(liii_base)__(liii_error))");
    }
    else if (mode == "sicp") {
        s7_eval_c_string (sc, "(import_(scheme_base)__(srfi_sicp))");
    }
    else if (mode == "r7rs") {
        s7_eval_c_string (sc, "(import_(scheme_base))");
    }
    else if (mode == "s7") {
    }
    else {
        cerr << "No_such_mode:" << mode << endl;
        exit (-1);
    }
}

```

---

## REPL

src/goldfish.hpp

△ 41 ▽

```
int
repl_for_community_edition (int argc, char** argv) {
    // Check if the standard library and boot.scm exists
    tb_char_t      data_goldfish[TB_PATH_MAXN]= {0};
    tb_char_t const* goldfish=
        tb_path_absolute (argv[0], data_goldfish, sizeof (data_goldfish));

    tb_char_t      data_bin[TB_PATH_MAXN]= {0};
    tb_char_t const* ret_bin=
        tb_path_directory (goldfish, data_bin, sizeof (data_bin));

    tb_char_t      data_root[TB_PATH_MAXN]= {0};
    tb_char_t const* gf_root=
        tb_path_directory (ret_bin, data_root, sizeof (data_root));

    tb_char_t      data_lib[TB_PATH_MAXN]= {0};
    tb_char_t const* gf_lib=
        tb_path_absolute_to (gf_root, "goldfish", data_lib, sizeof (data_lib));

    tb_char_t      data_boot[TB_PATH_MAXN]= {0};
    tb_char_t const* gf_boot= tb_path_absolute_to (gf_lib, "scheme/boot.scm",
                                                    data_boot, sizeof (data_boot));

    if (!tb_file_access (gf_lib, TB_FILE_MODE_RO)) {
        cerr << "The_load_path_for_Goldfish_Scheme_Standard_Library_does_not_exist"
              << endl;
        exit (-1);
    }
    if (!tb_file_access (gf_boot, TB_FILE_MODE_RO)) {
        cerr << "The_boot.scm_for_Goldfish_Scheme_does_not_exist" << endl;
        exit (-1);
    }
    vector<string> all_args (argv, argv + argc);
    int all_args_N= all_args.size ();
    for (int i= 0; i < all_args_N; i++) {
        command_args.push_back (all_args[i]);
    }

    // zero args
    vector<string> args (argv + 1, argv + argc);
    if (args.size () == 0) {
        display_help ();
        exit (0);
    }

    // Init the underlying S7 Scheme and add the load_path
    s7_scheme* sc= init_goldfish_scheme (gf_lib);

    const char* errmsg= NULL;
    s7_pointer old_port=
        s7_set_current_error_port (sc, s7_open_output_string (sc));
    int gc_loc= -1;
    if (old_port != s7_nil (sc)) gc_loc= s7_gc_protect (sc, old_port);

    // -m: Load the standard library by mode
    string mode_flag= "-m";
    string mode      = "default";
    int  args_N      = args.size ();
}
```



**goldfish** 命名空间的结束

```
src/goldfish.hpp
```

```
△ 42
```

---

```
} // namespace goldfish
```

---



# 第 24 章

## 基础设施

### 24.1 持续集成

#### 24.1.1 Github平台 macOS系统

触发条件

[.github/workflows/ci-macos.yml](#)

1 ▾

---

name: Build and Test on macOS

```
on:
  push:
    branches: [ main ]
    paths:
      - 'goldfish/**'
      - 'src/**'
      - 'tests/**'
      - 'xmake/**'
      - 'xmake.lua'
      - '.github/workflows/ci-macos.yml'
  pull_request:
    branches: [ main ]
    paths:
      - 'goldfish/**'
      - 'src/**'
      - 'tests/**'
      - 'xmake/**'
      - 'xmake.lua'
      - '.github/workflows/ci-macos.yml'
```

---

具体任务

[.github/workflows/ci-macos.yml](#)

△ 2

---

```

jobs:
  macosbuild:
    runs-on: macos-13
    steps:
      - uses: actions/checkout@v2
        with:
          fetch-depth: 1

      - name: cache xmake
        uses: actions/cache@v2
        with:
          path: |
            ${github.workspace}}/build/.build_cache
            /Users/runner/.xmake
          key: ${runner.os }}-xmake-${hashFiles('**/xmake.lua')}

      - name: set XMAKE_GLOBALDIR
        run: echo "XMAKE_GLOBALDIR=${runner.workspace }}/xmake-global" >> $GITHUB_ENV

      - uses: xmake-io/github-action-setup-xmake@v1
        with:
          xmake-version: v2.8.7
          actions-cache-folder: '.xmake-cache'

      - name: xmake repo --update
        run: xmake repo --update

      - name: cache packages from xrepo
        uses: actions/cache@v3
        with:
          path: |
            ${env.XMAKE_GLOBALDIR }}/.xmake/packages
          key: ${runner.os }}-xrepo-${hashFiles('**/xmake.lua')}

      - name: config
        run: xmake config -vD --policies=build.ccache -o tmp/build -m releasedbg --yes

      - name: build
        run: xmake build --yes -vD goldfish

      - name: run tests
        run: bin/goldfish -l tests/test_all.scm

```

---

## 24.1.2 Github平台 Windows系统

触发条件

[.github/workflows/ci-windows.yml](#)

1 ▾

---

name: Build on windows

on:

  push:

    branches: [ main ]

    paths:

- 'goldfish/\*\*'
- 'src/\*\*'
- 'tests/\*\*'
- 'xmake/\*\*'
- 'xmake.lua'
- '.github/workflows/ci-windows.yml'

  pull\_request:

    branches: [ main ]

    paths:

- 'goldfish/\*\*'
- 'src/\*\*'
- 'tests/\*\*'
- 'xmake/\*\*'
- 'xmake.lua'
- '.github/workflows/ci-windows.yml'

  workflow\_dispatch:

---

## 具体任务

[.github/workflows/ci-windows.yml](#)

△ 2

jobs:

```
  windowsbuild:
    runs-on: windows-2019
    env:
      # Force xmake to a specific folder (for cache)
      XMAKE_GLOBALDIR: ${ github.workspace }/.xmake-global
    steps:
      - uses: xmake-io/github-action-setup-xmake@v1
        with:
          xmake-version: v2.8.9
      - name: update repo
        run: xmake repo -u
      - name: git crlf
        run: git config --global core.autocrlf false
      - uses: actions/checkout@v3
        with:
          fetch-depth: 1
      - name: cache xmake
        uses: actions/cache@v2
        with:
          path: |
            ${ env.XMAKE_GLOBALDIR }/.xmake/packages
            ${ github.workspace }/build/.build_cache
          key: ${ runner.os }-xmake-${ hashFiles('**/xmake.lua') }
      - name: config
        run: xmake config --yes -vD
      - name: build
        run: xmake build --yes -vD goldfish
      - name: test
        run: bin/goldfish -l tests/test_all.scn
```

### 24.1.3 Github平台Debian系统

触发条件

[.github/workflows/ci-debian.yml](#)

1 ▾

---

`name: CI on Debian bookworm``on:` `push:` `branches: [ main ]` `paths:`

- `- 'goldfish/**'`
- `- 'src/**'`
- `- 'tests/**'`
- `- 'xmake/**'`
- `- 'xmake.lua'`
- `- '.github/workflows/ci-debian.yml'`

 `pull_request:` `branches: [ main ]` `paths:`

- `- 'goldfish/**'`
  - `- 'src/**'`
  - `- 'tests/**'`
  - `- 'xmake/**'`
  - `- 'xmake.lua'`
  - `- '.github/workflows/ci-debian.yml'`
- 

## 环境变量

[.github/workflows/ci-debian.yml](#)

△ 2 ▾

---

`env:` `XMAKE_ROOT: y` `DEBIAN_FRONTEND: noninteractive`

---

## 具体任务

[.github/workflows/ci-debian.yml](#)

△ 3

---

**jobs:****build:**

container: debian:bookworm

runs-on: ubuntu-22.04

strategy:

fail-fast: true

steps:

- name: Install dependencies

run: |

apt-get update

apt-get install -y git 7zip unzip curl build-essential

- uses: actions/checkout@v3

with:

fetch-depth: 1

- name: git add safe directory

run: git config --global --add safe.directory '\*'

- name: de-gitee

run: sed -i '/gitee\.com/d' xmake/packages/s/s7/xmake.lua

- name: set XMAKE\_GLOBALDIR

run: echo "XMAKE\_GLOBALDIR=\${{ runner.workspace }}/xmake-global" &gt;&gt; \$GITHUB\_ENV

- uses: xmake-io/github-action-setup-xmake@v1

with:

xmake-version: v2.8.7

actions-cache-folder: '.xmake-cache'

- name: xmake repo --update

run: xmake repo --update

- name: cache packages from xrepo

uses: actions/cache@v3

with:

path: |

\${{ env.XMAKE\_GLOBALDIR }}/xmake/packages

key: \${{ runner.os }}-xrepo-\${{ hashFiles('\*/xmake.lua') }}

- name: config

run: xmake config -vD --policies=build.ccache -o tmp/build -m releasedbg --yes

- name: build

run: xmake build --yes -vD goldfish

- name: run tests

run: bin/goldfish -l tests/test\_all.scm

---



# 索引

|                          |     |                                |     |
|--------------------------|-----|--------------------------------|-----|
| !=                       | 40  | char-upcase                    | 186 |
| ==                       | 40  | char-upper-case?               | 186 |
| =?                       | 156 | check                          | 54  |
| ???                      | 48  | check-catch                    | 55  |
| alist-cons               | 82  | check-failed?                  | 53  |
| alist->hash-table        | 160 | check-false                    | 55  |
| and                      | 16  | check-report                   | 56  |
| and-let*                 | 18  | check-set-mode!                | 51  |
| any                      | 80  | check-true                     | 55  |
| append                   | 71  | circular-list                  | 83  |
| append-map               | 75  | circular-list?                 | 83  |
| apply                    | 35  | close-input-port               | 39  |
| arithmetic-shift         | 97  | close-output-port              | 39  |
| ash                      | 97  | close-port                     | 39  |
| assoc                    | 82  | comparator?                    | 147 |
| assq                     | 82  | comparator-check-type          | 147 |
| assv                     | 82  | comparator-equality-predicate  | 147 |
| base64-decode            | 177 | comparator-hash                | 148 |
| base64-encode            | 175 | comparator-hashable?           | 147 |
| binary-port?             | 39  | comparator-hash-function       | 147 |
| bit-count                | 95  | comparator-ordered?            | 147 |
| bitwise-and              | 93  | comparator-ordering-predicate  | 147 |
| bitwise-andc1            | 96  | comparator-test-type           | 147 |
| bitwise-andc2            | 96  | comparator-type-test-predicate | 147 |
| bitwise-nand             | 95  | compose                        | 42  |
| bitwise-nor              | 94  | cons                           | 61  |
| bitwise-not              | 93  | count                          | 71  |
| bitwise-or               | 94  | default-hash                   | 154 |
| bitwise-orc1             | 96  | define                         | 19  |
| bitwise-orc2             | 96  | define-library                 | 10  |
| bitwise-xor              | 94  | define-record-type             | 19  |
| boolean=?                | 27  | define-values                  | 19  |
| boolean-hash             | 153 | delete                         | 80  |
| boolean<?                | 153 | delete-duplicates              | 81  |
| bytevector               | 30  | delete-file                    | 9   |
| bytevector?              | 30  | digit-value                    | 187 |
| bytevector-append        | 31  | display*                       | 41  |
| bytevector-base64-decode | 176 | drop                           | 68  |
| bytevector-base64-encode | 175 | drop-right                     | 69  |
| bytevector-length        | 31  | drop-while                     | 79  |
| bytevector-u8-ref        | 31  | eighth                         | 67  |
| bytevector-u8-set!       | 31  | eof-object                     | 40  |
| caar                     | 66  | even?                          | 23  |
| call-with-port           | 38  | every                          | 80  |
| car                      | 65  | exact?                         | 21  |
| case                     | 16  | fifth                          | 67  |
| case*                    | 57  | file-error?                    | 38  |
| cdr                      | 65  | file-exists?                   | 9   |
| char?                    | 29  | file-exists-error              | 47  |
| char=?                   | 30  | file-not-found-error           | 46  |
| char-ci-hash             | 153 | filter                         | 76  |
| char-downcase            | 186 | find                           | 78  |
| char->integer            | 30  | first                          | 66  |
| char-hash                | 153 | flatmap                        | 86  |
| char-lower-case?         | 187 |                                |     |

|                            |     |                         |         |
|----------------------------|-----|-------------------------|---------|
| flatten                    | 87  | let                     | 17      |
| floor                      | 23  | let*                    | 17      |
| floor-quotient             | 25  | let1                    | 41      |
| fold                       | 73  | letrec*                 | 17      |
| fold-right                 | 73  | let-values              | 18      |
| for-each                   | 72  | list                    | 61      |
| fourth                     | 67  | list?                   | 63      |
| g_access                   | 204 | list-copy               | 62      |
| gcd                        | 26  | list->string            | 102     |
| g_chdir                    | 202 | list->vector            | 133-134 |
| g_command-line             | 199 | list-index              | 79      |
| g_current-second           | 198 | list-not-null?          | 87      |
| g_delete-file              | 197 | list-ref                | 66      |
| g_getcwd                   | 203 | list-tail               | 68      |
| g_get-environment-variable | 198 | list-view               | 85      |
| g_getlogin                 | 204 | logand                  | 93      |
| g_getpid                   | 204 | logior                  | 94      |
| g_isdir                    | 202 | lognot                  | 93-95   |
| g_isfile                   | 202 | logxor                  | 94      |
| g_listdir                  | 203 | make-bytevector         | 31      |
| g_mkdir                    | 202 | make-comparator         | 149     |
| g_os-arch                  | 200 | make-default-comparator | 155     |
| g_os-call                  | 201 | make-eq-comparator      | 150     |
| g_os-temp-dir              | 201 | make-equal-comparator   | 150     |
| g_os-type                  | 200 | make-eqv-comparator     | 150     |
| g_system                   | 201 | make-hash-table         | 159     |
| >?                         | 156 | make-list               | 61      |
| >=?                        | 156 | make-list-comparator    | 150     |
| guard                      | 37  | make-pair-comparator    | 150     |
| g_uid4                     | 206 | make-vector             | 123     |
| g_version                  | 197 | make-vector-comparator  | 152     |
| hash-table?                | 160 | map                     | 72      |
| hash-table=?               | 161 | member                  | 77      |
| hash-table-clear!          | 164 | memq                    | 77      |
| hash-table-contains?       | 160 | memv                    | 77      |
| hash-table-count           | 166 | negative?               | 23      |
| hash-table-delete!         | 163 | ninth                   | 67      |
| hash-table-empty?          | 161 | not-a-directory-error   | 46      |
| hash-table-entries         | 165 | not-null-list?          | 86      |
| hash-table-find            | 165 | null?                   | 63      |
| hash-table-foreach         | 166 | null-list?              | 65, 86  |
| hash-table->alist          | 167 | number?                 | 21      |
| hash-table-keys            | 164 | number-hash             | 154     |
| hash-table-map->list       | 167 | odd?                    | 23      |
| hash-table-ref             | 161 | open-binary-input-file  | 189     |
| hash-table-ref/default     | 162 | open-binary-output-file | 190     |
| hash-table-set!            | 162 | open-input-file         | 189     |
| hash-table-size            | 164 | open-output-file        | 189     |
| hash-table-update!         | 163 | or                      | 17      |
| hash-table-values          | 165 | os-error                | 46      |
| identity                   | 42  | output-port?            | 39      |
| import                     | 10  | output-port-open?       | 39      |
| in?                        | 41  | pair?                   | 63      |
| input-port?                | 39  | partition               | 76      |
| input-port-open?           | 39  | port?                   | 39      |
| int-vector                 | 123 | positive?               | 22      |
| iota                       | 62  | procedure?              | 35      |
| lambda*                    | 42  | queue                   | 143     |
| last                       | 71  | queue-back              | 143     |
| last-pair                  | 70  | queue-empty?            | 143     |
| length                     | 71  | queue-front             | 143     |
| length=?                   | 83  | queue->list             | 144     |
| length>?                   | 84  | queue-pop!              | 144     |
| length>=?                  | 84  | queue-push!             | 144     |
| <?                         | 156 | queue-size              | 143     |
| <=?                        | 156 | quotient                | 25      |

|                      |     |                    |     |
|----------------------|-----|--------------------|-----|
| read                 | 39  | string-take        | 109 |
| read-error?          | 38  | string-take-right  | 109 |
| receive              | 36  | string-tokenize    | 118 |
| reduce               | 74  | string-trim        | 111 |
| reduce-right         | 75  | string-trim-both   | 113 |
| remove               | 76  | string-trim-right  | 112 |
| reverse              | 71  | string-upcase      | 116 |
| second               | 66  | symbol?            | 28  |
| seventh              | 67  | symbol=?           | 28  |
| sixth                | 67  | symbol->string     | 29  |
| split-at             | 69  | symbol-hash        | 153 |
| square               | 27  | take               | 68  |
| stack                | 138 | take-right         | 69  |
| stack-empty?         | 139 | take-while         | 78  |
| stack->list          | 139 | tenth              | 68  |
| stack-pop!           | 139 | test               | 56  |
| stack-push!          | 139 | textual-port?      | 39  |
| stack-size           | 139 | third              | 67  |
| stack-top            | 139 | timeout-error      | 47  |
| string?              | 102 | typed-lambda       | 42  |
| string-any           | 106 | type-error         | 47  |
| string-append        | 117 | type-error?        | 47  |
| string-base64-decode | 177 | u8-string-length   | 31  |
| string-base64-encode | 175 | u8-substring       | 32  |
| string-ci-hash       | 153 | unless             | 17  |
| string-contains      | 115 | utf8->string       | 32  |
| string-copy          | 108 | value-error        | 47  |
| string-count         | 115 | values             | 35  |
| string-downcase      | 116 | vector             | 123 |
| string-drop          | 109 | vector?            | 125 |
| string-drop-right    | 110 | vector-any         | 129 |
| string-ends?         | 119 | vector-append      | 124 |
| string-every         | 105 | vector-copy        | 124 |
| string-for-each      | 118 | vector-copy!       | 131 |
| string->list         | 102 | vector-count       | 127 |
| string->symbol       | 29  | vector-cumulate    | 128 |
| string->utf8         | 32  | vector-empty?      | 125 |
| string->vector       | 135 | vector-every       | 129 |
| string-hash          | 153 | vector-fill!       | 131 |
| string-index         | 114 | vector-for-each    | 127 |
| string-index-right   | 115 | vector->list       | 133 |
| string-join          | 102 | vector->string     | 134 |
| string-length        | 107 | vector-index       | 129 |
| string-map           | 117 | vector-index-right | 129 |
| string-null?         | 104 | vector-length      | 126 |
| string-pad           | 110 | vector-map         | 126 |
| string-pad-right     | 111 | vector-partition   | 130 |
| string-prefix?       | 113 | vector-ref         | 126 |
| string-ref           | 107 | vector-set!        | 130 |
| string-remove-prefix | 120 | vector-skip        | 130 |
| string-remove-suffix | 120 | vector-skip-right  | 130 |
| string-reverse       | 117 | vector-swap!       | 131 |
| string-starts?       | 119 | when               | 17  |
| string-suffix?       | 114 | zero?              | 22  |