

大脑原来不靠谱

http://aresy.blog.51cto.com【复制】【订阅】

博客 | 写博文 | 帮助

首页 | Javascript | Vmware | Linux运维 | 自动化运维 | Go | Docker | Kubernetes | Open vSwitch | Ceph

yangzhares 的BLOG

相关视频课程

更多



写留言

去学院学习

发消息

加友情链接

进家园 加好友

博客统计信息

用户名: yangzhares

文章数: 13

评论数: 1

访问量: 5980

微信

关注51CTO博客微信
有机会赢下载VIP会员

微信号: blog51cto

更多>>

元转

363

周年】我在

说的收获

276



从菜鸟到老鸟-教你玩
转Mac操作系统

阅读量: 326019



QT学习之路: 从入门到
精通

阅读量: 1035028

热门文章

Openswitch GRE实现Kube..

Golang标准库之Buffer

搭建私有Docker Registry

CentOS 7实战Kubernetes..

Linux性能优化和监控系列..

跟无闻学Go语言: Go
Web基础视频教程 (共12
22392人学习

跟无闻学Go语言: Go编
程基础视频教程 (共15
93326人学习

Go语言编程入门视频课
程 (共109课时)
4035人学习

博主的更多文章>>

原创 Golang标准库之Buffer

2014-04-30 20:45:17

标签: Go Golang Go标准库 Go Buffer

原创作品，允许转载，转载时请务必以超链接形式标明文章 [原始出处](#)、作者信息和本声明。否则将追究法律责
任。 <http://aresy.blog.51cto.com/5100031/1405184>

Buffer

Go标准库Buffer是一个可变大小的字节缓冲区，可以用Write和Read方法操作它，在Go标准库中, 定义了如下关于Buffer的数据结构。

```
1 type Buffer struct {
2     buf []byte // contents are the bytes buf[off : len(buf)]
3     off  int      // read at &buf[off], write at &buf[len(buf)]
4     runeBytes [utf8.UTFMax]byte // avoid allocation of slice on each WriteByte or Rune
5     bootstrap [64]byte // memory to hold first slice; helps small buffers (Printf) avoid allo
6     lastRead  readOp // last read operation, so that Unread* can work correctly.
7 }
8 // The readOp constants describe the last action performed on
9 // the buffer, so that UnreadRune and UnreadByte can
10 // check for invalid usage.
11 type readOp int
12 const (
13     opInvalid readOp = iota // Non-read operation.
14     opReadRune              // Read rune.
15     opRead                  // Any other read operation.
16 )
```

如上定义，Buffer存储的数据是在off到len(buf)区域之间，其他区域是没有数据，而且只能从&buf[off]开始读取数据和从&buf[len(buf)]写数据，同时为了避免对内存的多次操作，对于小的缓冲区，Buffer定义了bootstrap来避免多次内存的操作，runeBytes的定义也是如此目的，还有一个表示对Buffer的操作标识符lastRead。

Buffer的常见操作

1. 初始化Buffer

```
1 func NewBuffer(buf []byte) *Buffer { return &Buffer{buf: buf} }
2 func NewBufferString(s string) *Buffer {
3     return &Buffer{buf: []byte(s)}
4 }
```

方法NewBuffer使用buf作为参数初始化Buffer，Buffer既可以被读也可以被写，如果是读Buffer，buf需填充一定的数据，如果是写，buf需有一定的容量(capacity)，当然也可以通过new(Buffer)来初始化Buffer。另外一个方法NewBufferString用一个string来初始化可读Buffer，并用string的内容填充Buffer。

2. 读写操作

```
1 func (b *Buffer) Read(p []byte) (n int, err error) {
2     return b.Next(n)
3 }
4 func (b *Buffer) ReadByte() (c byte, err error) {
5     return b.ReadByteRune()
6 }
7 func (b *Buffer) ReadBytes(delim byte) (line []byte, err error) {
8     return b.ReadSlice(delim)
9 }
10 func (b *Buffer) ReadString(delim byte) (line string, err error) {
11     return b.ReadStringRune(delim)
12 }
13 func (b *Buffer) Write(p []byte) (n int, err error) {
14     return b.WriteString(s string)
15 }
```

Linux性能优化和监控系列..


Kubernetes系统架构简介

Linux性能优化和监控系列..


搜索BLOG文章

搜索


最近访客




御风少侠




andot




youxi423




自由1..




ssjmhyvi




wlq19..




一会..




shy润..




chen.j..



Kevin..



guoji..



mfsoww

最新评论

seecsea: 这么好的理论性文章居然没人顶, 不..

51CTO推荐博文

更多>>

Nginx演练(3)配置内容压缩

MySQL主从同步校验与重新同步

python实现自动监控网站并发送邮..

Centos 6.4下 MySQL配置主从服务(..

解决Eclipse java build path中We..

Lync Server 2013 _ Lync Server..

京东MySQL监控之Zabbix优化、自动化

VC客户端无法登陆都是REDO日志惹的祸

从MySQL全库备份中恢复某个库和某..

MariaDB10自动化安装部署

[实例]利用php+mysql完成shell脚..

友情链接

IT精品课程

51CTO博客开发

技术人才招聘

```
10 func (b *Buffer) ReadFrom(r io.Reader) (n int64, err error)
11 func (b *Buffer) WriteTo(w io.Writer) (n int64, err error)
12 func (b *Buffer) WriteByte(c byte) error
13 func (b *Buffer) WriteRune(r rune) (n int, err error)
```

下面对Read, ReadRune, ReadBytes方法进行分析, 对于方法Read, 其主要做三个步骤: 第一, 判断Buffer是否为空, 如果是, 则重置Buffer; 第二, 复制Buffer的buf的数据到p, 并调整off的位置标识Buffer的可读位置; 第三, 设置读标识符为opRead.

```
1 func (b *Buffer) Read(p []byte) (n int, err error) {
2     b.lastRead = opInvalid
3     if b.off >= len(b.buf) {
4         // Buffer is empty, reset to recover space.
5         b.Truncate(0)
6         if len(p) == 0 {
7             return
8         }
9         return 0, io.EOF
10    }
11    n = copy(p, b.buf[b.off:])
12    b.off += n
13    if n > 0 {
14        b.lastRead = opRead
15    }
16    return
17 }
```

方法ReadRune()定义了如何读取Buffer中UTF8编码的rune数据, 同样也需三个步骤, 第一, 判断Buffer是否为空, 若是, 重置Buffer; 第二, 设置读操作符为opReadRune; 第三, 判断可读位置off处的byte是否小于utf8.RuneSelf, 若是, 调整off位置并返回. 否则, 将Buffer的数据解码成rune, 调整off位置, 返回解码后的rune及大小.

```
1 // ReadRune reads and returns the next UTF-8-encoded
2 // Unicode code point from the buffer.
3 // If no bytes are available, the error returned is io.EOF.
4 // If the bytes are an erroneous UTF-8 encoding, it
5 // consumes one byte and returns U+FFFD, 1.
6 func (b *Buffer) ReadRune() (r rune, size int, err error) {
7     b.lastRead = opInvalid
8     if b.off >= len(b.buf) {
9         // Buffer is empty, reset to recover space.
10        b.Truncate(0)
11        return 0, 0, io.EOF
12    }
13    b.lastRead = opReadRune
14    c := b.buf[b.off]
15    if c < utf8.RuneSelf {
16        b.off++
17        return rune(c), 1, nil
18    }
19    r, n := utf8.DecodeRune(b.buf[b.off:])
20    b.off += n
21    return r, n, nil
22 }
```

方法ReadBytes(delim byte)读取Buffer中从off到第一次delim之间的数据, 并且包括delim, ReadBytes调用私有方法readSlice来实现, readSlice方法首先查找delim的位置, 如果不存在, 则返回从off到len(buf)之间的数据, 如果存在, 则返回off到off+location(delim)+1之间数据, 其中加1是为了包括delim, 最后设置操作标识符为opRead.

```
1 // ReadBytes reads until the first occurrence of delim in the input,
2 // returning a slice containing the data up to and including the delimiter.
3 // If ReadBytes encounters an error before finding a delimiter,
4 // it returns the data read before the error and the error itself (often io.EOF).
5 // ReadBytes returns err != nil if and only if the returned data does not end in
6 // delim.
7 func (b *Buffer) ReadBytes(delim byte) (line []byte, err error) {
8     slice, err := b.readSlice(delim)
9     // return a copy of slice. The buffer's backing array may
10    // be overwritten by later calls.
11    line = append(line, slice...)
12    return
13 }
14 // readSlice is like ReadBytes but returns a reference to internal buffer data.
15 func (b *Buffer) readSlice(delim byte) (line []byte, err error) {
16     i := IndexByte(b.buf[b.off:], delim)
17     end := b.off + i + 1
18     if i < 0 {
19         end = len(b.buf)
20         err = io.EOF
21     }
22     line = b.buf[b.off:end]
23     b.off = end
24     b.lastRead = opRead
25     return line, err
26 }
```

同样对相应的Write, WriteRune, ReadFrom, WriteTo写方法进行分析, 对于方法Write, 相对Read方法来说, 要简单些, 主要是扩展Buffer空间, 然后将p中的数据复制到Buffer.

```
1 // Write appends the contents of p to the buffer, growing the buffer as
2 // needed. The return value n is the length of p; err is always nil. If the
3 // buffer becomes too large, Write will panic with ErrTooLarge.
4 func (b *Buffer) Write(p []byte) (n int, err error) {
5     b.lastRead = opInvalid
6     m := b.grow(len(p))
7     return copy(b.buf[m:], p), nil
8 }
```

对于方法WriteRune, 首先判断要写的数据rune是否小于utf8.RuneSelf, 若是, 调用WriteByte将其写入Buffer, 若

不是，则将要写的数据rune编码成utf8，并调用Write将其写入Buffer。

```

1 // WriteRune appends the UTF-8 encoding of Unicode code point r to the
2 // buffer, returning its length and an error, which is always nil but is
3 // included to match bufio.Writer's WriteRune. The buffer is grown as needed;
4 // if it becomes too large, WriteRune will panic with ErrTooLarge.
5 func (b *Buffer) WriteRune(r rune) (n int, err error) {
6     if r < utf8.RuneSelf {
7         b.WriteByte(byte(r))
8         return 1, nil
9     }
10    n = utf8.EncodeRune(b.runeBytes[0:], r)
11    b.Write(b.runeBytes[0:n])
12    return n, nil
13 }

```

ReadFrom方法从io.Reader或者实现io.Reader接口的实例中读取所有数据到Buffer，默认情况下最少读取512字节，如果Buffer空间不足512，需增加Buffer空间，该方法返回读取的字节数以及错误信息。从下面可知ReadFrom首先判断Buffer是否为空，若空，则重置Buffer；其次是判断Buffer的free空间是否足够，若小于512且off+free小于512，表示Buffer从0到off之间的空间不足以存放当前Buffer中未读数据的大小，此时设置一临时缓冲区并使其空间Buffer的2倍加上MinRead(512)的空间，将原来Buffer的数据复制到临时缓冲区，然后再把临时缓冲区的数据复制到源Buffer，最后使用io.Reader的Read方法从io.Reader中读取数据，直到遇到io.EOF。

```

1 // ReadFrom reads data from r until EOF and appends it to the buffer, growing
2 // the buffer as needed. The return value n is the number of bytes read. Any
3 // error except io.EOF encountered during the read is also returned. If the
4 // buffer becomes too large, ReadFrom will panic with ErrTooLarge.
5 func (b *Buffer) ReadFrom(r io.Reader) (n int64, err error) {
6     b.lastRead = opInvalid
7     // If buffer is empty, reset to recover space.
8     if b.off >= len(b.buf) {
9         b.Truncate(0)
10    }
11    for {
12        if free := cap(b.buf) - len(b.buf); free < MinRead {
13            // not enough space at end
14            newBuf := b.buf
15            if b.off+free < MinRead {
16                // not enough space using beginning of buffer;
17                // double buffer capacity
18                newBuf = makeSlice(2*cap(b.buf) + MinRead)
19            }
20            copy(newBuf, b.buf[b.off:])
21            b.buf = newBuf[:len(b.buf)-b.off]
22            b.off = 0
23        }
24        m, e := r.Read(b.buf[len(b.buf):cap(b.buf)])
25        b.buf = b.buf[0 : len(b.buf)+m]
26        n += int64(m)
27        if e == io.EOF {
28            break
29        }
30        if e != nil {
31            return n, e
32        }
33    }
34    return n, nil // err is EOF, so return nil explicitly
35 }

```

相对ReadFrom方法，WriteTo方法比较简单，WriteTo将Buffer中的数据写到io.Writer，直到Buffer中没有数据，当Buffer为空时，重置Buffer并返回。

```

1 // WriteTo writes data to w until the buffer is drained or an error occurs.
2 // The return value n is the number of bytes written; it always fits into an
3 // int, but it is int64 to match the io.WriterTo interface. Any error
4 // encountered during the write is also returned.
5 func (b *Buffer) WriteTo(w io.Writer) (n int64, err error) {
6     b.lastRead = opInvalid
7     if b.off < len(b.buf) {
8         nBytes := b.Len()
9         m, e := w.Write(b.buf[b.off:])
10        if m > nBytes {
11            panic("bytes.Buffer.WriteTo: invalid Write count")
12        }
13        b.off += m
14        n = int64(m)
15        if e != nil {
16            return n, e
17        }
18        // all bytes should have been written, by definition of
19        // Write method in io.Writer
20        if m != nBytes {
21            return n, io.ErrShortWrite
22        }
23    }
24    // Buffer is now empty; reset.
25    b.Truncate(0)
26    return n, nil
27 }

```

3. 扩展空间和重置

Buffer的重置方法Reset()通过调用Truncate(n int)方法来实现清除Buffer的数据，Truncate丢弃了从off开始的n个未读数据之外的所有数据，如果n为0，那就重置Buffer。

```

1 // Truncate discards all but the first n unread bytes from the buffer.
2 // It panics if n is negative or greater than the length of the buffer.
3 func (b *Buffer) Truncate(n int) {
4     b.lastRead = opInvalid

```

```
5      switch {
6      case n < 0 || n > b.Len():
7          panic("bytes.Buffer: truncation out of range")
8      case n == 0:
9          // Reuse buffer space.
10         b.off = 0
11     }
12     b.buf = b.buf[0 : b.off+n]
13 }
14 // Reset resets the buffer so it has no content.
15 // b.Reset() is the same as b.Truncate(0).
16 func (b *Buffer) Reset() { b.Truncate(0) }
```

在对Buffer进行写数据时，通常需要扩展其空间来使所有的数据都能写入Buffer，Buffer用Grow(n int)方法来实现扩展Buffer空间的功能，该方法调用私有方法grow(n int)。

```
1 // grow grows the buffer to guarantee space for n more bytes.
2 // It returns the index where bytes should be written.
3 // If the buffer can't grow it will panic with ErrTooLarge.
4 func (b *Buffer) grow(n int) int {
5     m := b.Len()
6     // 如果Buffer为空，重置Buffer
7     if m == 0 && b.off != 0 {
8         b.Truncate(0)
9     }
10    //空间增加n后超过Buffer的容量
11    if len(b.buf)+n > cap(b.buf) {
12        //声明一个临时buf
13        var buf []byte
14        //Buffer的buf只是被声明，还没有初始化，如果n小于bootstrap的空间，
15        //直接将bootstrap赋值给buf避免内存的操作而增加负载。
16        if b.buf == nil && n <= len(b.bootstrap) {
17            buf = b.bootstrap[0:]
18        } //如果满足此条件，滑动b.buf的数据而不是分配一个新的slice空间，然后将b.buf的数据复制给你buf。
19        } else if m+n <= cap(b.buf)/2 {
20            copy(b.buf[:], b.buf[b.off:])
21            buf = b.buf[:m]
22        } else {
23            //空间不足，重新分配空间
24            buf = makeSlice(2*cap(b.buf) + n)
25            copy(buf, b.buf[b.off:])
26        }
27        b.buf = buf
28        b.off = 0
29    }
30    //扩展n的空间，并返回可以写数据的位置
31    b.buf = b.buf[0 : b.off+m+n]
32    return b.off + m
33 }
34 // Grow grows the buffer's capacity, if necessary, to guarantee space for
35 // another n bytes. After Grow(n), at least n bytes can be written to the
36 // buffer without another allocation.
37 // If n is negative, Grow will panic.
38 // If the buffer can't grow it will panic with ErrTooLarge.
39 func (b *Buffer) Grow(n int) {
40     if n < 0 {
41         panic("bytes.Buffer.Grow: negative count")
42     }
43     m := b.grow(n)
44     b.buf = b.buf[0:m]
45 }
```

本文出自 “大脑原来不靠谱” 博客，请务必保留此出处<http://aresy.blog.51cto.com/5100031/1405184>

分享至:

收藏 



0人

了这篇文章

类别: Go | 阅读(1070) | 评论(0) | [返回博主首页](#) | [返回博客首页](#)

[上一篇 Docker逻辑数据流](#) [下一篇 搭建私有Docker Registry](#)



相关文章

批量获取text字段的长度
Go 搭建一个Web 服务器(1):IOC工厂
使用Go Hi jack和jQuery轻松实现异步推送服务
【go学习笔记1】为什么go

职位推荐

c++后台软件开发工程师
高级PHP开发工程师
Java软件工程师
Web前端开发工程师
逆向分析破解工程师（反汇编）

文章评论

发表评论

昵 称:

验证码:

请点击后输入验证码 [博客过2级，无需填写验证码](#)

内 容:

Copyright By 51CTO.COM 版权所有

51CTO 技术博客