golang 源码包阅读 bytes 包

作者：yhyddr

bytes 笔记

<a name="byte.go"></a>

byte.go

<a name="Overview"></a>

Overview

操作字节切片的函数，与字符串 strings 包类似。

<a name="db3fc4e8"></a>

核心函数

<a name="bf15136e"></a>

genSplit(s, sep []byte, sepSave, n int) [][]byte

切分切片使用的最核心的函数。<br />有四个参数，第一个是被切切片，第二个是分隔符，第三个是择包含分隔符在内往后几个字节一起作为子切片，最后一个是最多通过n个分隔符来切分

```
// Generic split: splits after each instance of sep,
// including sepSave bytes of sep in the subslices.
// 将含有 sep 的字节切片全部单独切开，最多切 n 个，同时 匹配到时候多切 sepSave 个字节一起进同一个切片
func genSplit(s, sep []byte, sepSave, n int) [][]byte {
    if n == 0 {
        return nil
    }
    if len(sep) == 0 {
        return explode(s, n)
    }
    if n < 0 {
        n = Count(s, sep) + 1
    }

    a := make([][]byte, n)
    n--
    i := 0
    for i < n {
        m := Index(s, sep)
        if m < 0 {
            break
        }
        a[i] = s[: m+sepSave : m+sepSave]
        s = s[m+len(sep):]
        i++
    }
    a[i] = s
    return a[:i+1]
}
```

<a name="b0a544c6"></a>

Fields(s []byte) [][]byte

主要是可以消除多个分隔符连续的噪声<br />这里的巧妙的地方时通过了一个 uint8 数组来实现了 ACII 编码的空格的判定，还是使用位来判定是否存在非ASCII编码加快分隔速度。<br />有一个 Fieldsunc 函数来自定义规则

```
var asciiSpace = [256]uint8{'\t': 1, '\n': 1, '\v': 1, '\f': 1, '\r': 1, ' ': 1}

// Fields interprets s as a sequence of UTF-8-encoded code points.
// It splits the slice s around each instance of one or more consecutive white space
// characters, as defined by unicode.IsSpace, returning a slice of subslices of s or an
// empty slice if s contains only white space.
func Fields(s []byte) [][]byte {
    // First count the fields.
    // This is an exact count if s is ASCII, otherwise it is an approximation.
    n := 0
    wasSpace := 1
    // setBits is used to track which bits are set in the bytes of s.
  // 意思就是通过位来判断是否所有的都可以通过字节来表示而不是需要utf-8编码
    setBits := uint8(0)
  // 这里实现了如果连续出现空格不会多次计数的除噪，通过 wasSpace
    for i := 0; i < len(s); i++ {
        r := s[i]
        setBits |= r
        isSpace := int(asciiSpace[r])
        n += wasSpace & ^isSpace
        wasSpace = isSpace
    }
    //不能通过ASCII码了就用utf-8
    if setBits >= utf8.RuneSelf {
        // Some runes in the input slice are not ASCII.
        return FieldsFunc(s, unicode.IsSpace)
    }

    // ASCII fast path 更快
    a := make([][]byte, n)
    na := 0
    fieldStart := 0
    i := 0
    // Skip spaces in the front of the input.
  // 跳过开头的空格
    for i < len(s) && asciiSpace[s[i]] != 0 {
        i++
    }
    fieldStart = i
    for i < len(s) {
        if asciiSpace[s[i]] == 0 {
            i++
            continue
        }
        a[na] = s[fieldStart:i:i]
        na++
        i++
        // Skip spaces in between fields.
        for i < len(s) && asciiSpace[s[i]] != 0 {
            i++
        }
        fieldStart = i
    }
  // 弥补上面的判断可能最后的EOF会忽略
    if fieldStart < len(s) { // Last field might end at EOF.
        a[na] = s[fieldStart:len(s):len(s)]
    }
    return a
}
```

<a name="140f0232"></a>

Join(s [][]byte, sep []byte) []byte

有分离就有连结，通过 sep 分隔符插在中间。

```
// Join concatenates the elements of s to create a new byte slice. The separator
// sep is placed between elements in the resulting slice.
func Join(s [][]byte, sep []byte) []byte {
    if len(s) == 0 {
        return []byte{}
    }
    if len(s) == 1 {
        // Just return a copy.
        return append([]byte(nil), s[0]...)
    }
  //判断需要多长的切片
    n := len(sep) * (len(s) - 1)
    for _, v := range s {
        n += len(v)
    }

    b := make([]byte, n)
    bp := copy(b, s[0])
    for _, v := range s[1:] {
        bp += copy(b[bp:], sep)
        bp += copy(b[bp:], v)
    }
    return b
}
```

<a name="9f6b8e07"></a>

Map(mapping func(r rune) rune, s []byte) []byte

通过映射函数替换切片中满足条件的字节

```
// Map returns a copy of the byte slice s with all its characters modified
// according to the mapping function. If mapping returns a negative value, the character is
// dropped from the byte slice with no replacement. The characters in s and the
// output are interpreted as UTF-8-encoded code points.

func Map(mapping func(r rune) rune, s []byte) []byte {
    // In the worst case, the slice can grow when mapped, making
    // things unpleasant. But it's so rare we barge in assuming it's
    // fine. It could also shrink but that falls out naturally.
    maxbytes := len(s) // length of b
    nbytes := 0        // number of bytes encoded in b
    b := make([]byte, maxbytes)
    for i := 0; i < len(s); {
        wid := 1
        r := rune(s[i])
        if r >= utf8.RuneSelf {
            r, wid = utf8.DecodeRune(s[i:])
        }
        r = mapping(r)
        if r >= 0 {
            rl := utf8.RuneLen(r)
            if rl < 0 {
                rl = len(string(utf8.RuneError))
            }
            if nbytes+rl > maxbytes {
                // Grow the buffer.
                maxbytes = maxbytes*2 + utf8.UTFMax
                nb := make([]byte, maxbytes)
                copy(nb, b[0:nbytes])
                b = nb
            }
            nbytes += utf8.EncodeRune(b[nbytes:maxbytes], r)
        }
        i += wid
    }
    return b[0:nbytes]
}
```

<a name="0d2d112c"></a>

indexFunc(s []byte, f func(r rune) bool, truth bool) int

返回满足条件函数的 rune 的下标，未找到就返回-1<br />条件函数可以是满足条件，可以是不满足件，看变量 truth 的使用

```
// indexFunc is the same as IndexFunc except that if
// truth==false, the sense of the predicate function is
// inverted.
func indexFunc(s []byte, f func(r rune) bool, truth bool) int {
    start := 0
    for start < len(s) {
        wid := 1
        r := rune(s[start])
    //如果是utf-8编码才能识别，则调用utf-8.DecodeRune(s[start:])
        if r >= utf8.RuneSelf {
            r, wid = utf8.DecodeRune(s[start:])
        }
        if f(r) == truth {
            return start
        }
        start += wid
    }
    return -1
}
```

<a name="8b656293"></a>

makeCutsetFunc(cutset string) func(r rune) bool

通过传入 的 string 类型变量，作为判断的条件函数，该函数判断 如果是 string 蕴含的返回真否则假

```
func makeCutsetFunc(cutset string) func(r rune) bool {
    if len(cutset) == 1 && cutset[0] < utf8.RuneSelf {
        return func(r rune) bool {
            return r == rune(cutset[0])
        }
    }
    if as, isASCII := makeASCIISet(cutset); isASCII {
        return func(r rune) bool {
            return r < utf8.RuneSelf && as.contains(byte(r))
        }
    }
    return func(r rune) bool {
        for _, c := range cutset {
            if c == r {
                return true
            }
        }
        return false
    }
}
```

<a name="142eee13"></a>

帮助实现的使用次数较多的函数

<a name="0a7566c5"></a>

DecodeRune(p []byte) (r rune, size int)

```
// DecodeRune unpacks the first UTF-8 encoding in p and returns the rune and
// its width in bytes. If p is empty it returns (RuneError, 0). Otherwise, if
// the encoding is invalid, it returns (RuneError, 1). Both are impossible
// results for correct, non-empty UTF-8.
//
// An encoding is invalid if it is incorrect UTF-8, encodes a rune that is
// out of range, or is not the shortest possible UTF-8 encoding for the
// value. No other validation is performed.
func DecodeRune(p []byte) (r rune, size int) {
    n := len(p)
    if n < 1 {
        return RuneError, 0
    }
    p0 := p[0]
    x := first[p0]
    if x >= as {
        // The following code simulates an additional check for x == xx and
        // handling the ASCII and invalid cases accordingly. This mask-and-or
        // approach prevents an additional branch.
        mask := rune(x) << 31 >> 31 // Create 0x0000 or 0xFFFF.
        return rune(p[0])&^mask | RuneError&mask, 1
    }
    sz := x & 7
    accept := acceptRanges[x>>4]
    if n < int(sz) {
        return RuneError, 1
    }
    b1 := p[1]
    if b1 < accept.lo || accept.hi < b1 {
        return RuneError, 1
    }
    if sz == 2 {
        return rune(p0&mask2)<<6 | rune(b1&maskx), 2
    }
    b2 := p[2]
    if b2 < locb || hicb < b2 {
        return RuneError, 1
    }
    if sz == 3 {
        return rune(p0&mask3)<<12 | rune(b1&maskx)<<6 | rune(b2&maskx), 3
    }
    b3 := p[3]
    if b3 < locb || hicb < b3 {
        return RuneError, 1
    }
    return rune(p0&mask4)<<18 | rune(b1&maskx)<<12 | rune(b2&maskx)<<6 | rune(b3&ma
kx), 4
}
```

<a name="0ff51bed"></a>

Equal(a, b []byte) bool

```
//go:noescape

// Equal returns a boolean reporting whether a and b
// are the same length and contain the same bytes.
// A nil argument is equivalent to an empty slice.
func Equal(a, b []byte) bool // in internal/bytealg
```

<a name="25f9c7fa"></a>

总结

实现了几乎所有能对字节切片产生的操作，基本都是基于 utf-8 编码来判定的，或者使用 ASCII 码当以使用的时候，实现了

- 分隔　各种规则的分隔符分隔（包括自定义规则）
- 裁剪　内置左右匹配的裁剪（自定义规则）和裁剪空格符
- 粘合
- 索引
- 替换
- 各种规则的替换
- 内置大小写和标题字体的替换
- 这些都是在包内分成了小函数来实现增强可自定义的性质，比如内置实现一些判断是否有前缀，是包含某些编码，就像造好了手枪和一些子弹，想要更多功能直接制造特制子弹即可。

Ps：包含有 Rabin-Karp search 的实现，被使用在 Index 这个返回索引的函数中。

原文链接：golang 源码包阅读 bytes 包