Go语言 标准库 Strconv包

Go语言不支持进行隐式的类型转换，只能手动去执行显性转换操作

转换数据类型的方式很简单。

```
valueOfTypeB = typeB(valueOfTypeA)
```

例如：

```
// 浮点数
a := 5.0

// 转换为int类型
b := int(a)
```

Go允许在底层结构相同的两个类型之间互转。例如：

```
// MyInt类型的底层是int类型
type MyInt int

// a的类型为MyInt，底层是int
var a MyInt = 5

// 将a(MyInt)转换为int，b现在是int类型
b := int(5)

// 将b(int)转换为MyInt，c现在是MyInt类型
c := MyInt(b)
```

但注意：

- 不是所有数据类型都能转换的，例如字母格式的string类型"baozi"转换为int肯定会失败。
- 低精度转换为高精度时是安全的，高精度的值转换为低精度时会丢失精度。例如int32转换为int16，float32转换为int。
- 这种简单的转换方式不能对int(float)和string进行互转，要跨大类型转换，可以使用strconv包提供的函数。

strconv 包

strconv包实现了基本数据类型与其字符串表示的转换，主要有以下常用函数： Atoi()、Itia()、parse系列、format系列、append系列。

string与int类型转换

这一组函数是我们平时编程中用的最多的

Atoi()

`Atoi()`函数用于将字符串类型的整数转换为int类型，函数签名如下。

```
func Atoi(s string) (i int, err error)
```

如果传入的字符串参数无法转换为int类型，就会返回错误。

```
func atoiDemo() {
    //var age ="baozi" // 转换出错
    var age = "19"
    age1, err := strconv.Atoi(age)
    if err != nil {
        fmt.Println("转换出错")
        return
    } else {
        fmt.Printf("类型是：%T，值是：%d", age1, age1)
    }
}
```

Itoa()

`Itoa()`函数用于将int类型数据转换为对应的字符串表示，具体的函数签名如下。

```
func Itoa(i int) string
```

示例：

```
func itoaDemo() {
    i := 999
    s2 := strconv.Itoa(i)
    fmt.Printf("类型是:%T 值是:%#v\n", s2, s2)
}
```

Parse系列函数

Parse类函数用于转换字符串为给定类型的值：`ParseBool()`、`ParseFloat()`、`ParseInt()`、`ParseUint()`。

这些函数都有两个返回值，第一个返回值是转换后的值，第二个返回值为转化失败的错误

ParseBool()

转换字符串为布尔型。

```
func ParseBool(str string) (value bool, err error)
```

返回字符串表示的 bool 值。它只接受值为1、0、t、f、T、F、true、false、True、False、TRUE、FALSE的字符串；否则返回错误。

```
func parseBoolDemo() {
    // i := "baozi" // 转换出错
    i := "1"
    s3, err := strconv.ParseBool(i)
    if err != nil {
        fmt.Println("转换出错")
        return
    } else {
        fmt.Printf("类型是:%T 值是:%#v\n", s3, s3)
    }
}
```

ParseInt()

转换字符串为整形。

```
func ParseInt(s string, base int, bitSize int) (i int64, err error)
```

返回字符串表示的整数值，接受正负号。

base指定进制（2到36），如果base为0，则会从字符串前置判断，”0x”是16进制，”0”是8进制，否则是10进制；

bitSize指定结果必须能无溢出赋值的整数类型，0、8、16、32、64 分别代表 int、int8、int16、int32、int64；

返回的err是*NumErr类型的，如果语法有误，err.Error = ErrSyntax；如果结果超出类型范围err.Error = ErrRange。

```
func parseIntDemo() {
    // i := "baozi" // 转换出错
    i := "1010"
    str, err := strconv.ParseInt(i, 10, 16)
    if err != nil {
        fmt.Println("转换出错")
        return
    } else {
        fmt.Printf("类型是:%T 值是:%#v\n", str, str)
    }
}
```

ParseUnit()

转换字符串为整形，但不接受正负号，用于无符号整型。

```
func ParseUint(s string, base int, bitSize int) (n uint64, err error)
```

`ParseUint` 类似 `ParseInt` 但不接受正负号，用于无符号整型。

ParseFloat()

转换字符串为浮点型。

```
func ParseFloat(s string, bitSize int) (f float64, err error)
```

解析一个表示浮点数的字符串并返回其值。

如果s合乎语法规则，函数会返回最为接近s表示值的一个浮点数

bitSize指定了期望的接收类型，32是float32（返回值可以不改变精确值的赋值给float32），64是float64；

返回值err是*NumErr类型的，语法有误的，err.Error=ErrSyntax；结果超出表示范围的，返回值f为±Inf，err.Error= ErrRange。

```
func parseFloatDemo() {
    // i := "baozi" // 转换出错
    i := "199.99"
    str, err := strconv.ParseFloat(i, 32)
    if err != nil {
        fmt.Println("转换出错")
        return
    } else {
        fmt.Printf("类型是:%T 值是:%f\n", str, str)
    }
}
```

Format系列函数

Format系列函数实现了将给定类型数据格式化为string类型数据的功能。

FormatBool()

```
func FormatBool(b bool) string
```

根据b的值返回true或false。

```
func formatBoolDemo() {
    // v := "baozi" // 报错
    v := true
    res := strconv.FormatBool(v)
    fmt.Printf("类型是:%T 值是:%#v\n", res, res)
}
```

FormatInt()

```
func FormatInt(i int64, base int) string
```

返回i的base进制的字符串表示。base 必须在2到36之间，结果中会使用小写字母'a'到'z'表示大于10的数字。

```
func formatIntDemo() {
    res := strconv.FormatInt(-2, 16)
    fmt.Printf("类型是:%T 值是:%#v\n", res, res)
}
```

FormatUint()

```
func FormatUint(i uint64, base int) string
```

是`FormatInt`的无符号整数版本。

```
func formatUintDemo() {
    res := strconv.FormatUint(2, 16)
    fmt.Printf("类型是:%T 值是:%#v\n", res, res)
}
```

FormatFloat()

```
func FormatFloat(f float64, fmt byte, prec, bitSize int) string
```

函数将浮点数表示为字符串并返回。

bitSize表示f的来源类型（32：float32、64：float64），会据此进行舍入。

fmt表示格式：'f'（-ddd.dddd）、'b'（-ddddp±ddd，指数为二进制）、'e'（-d.dddde±dd，十进制指数）、'E'（-d.ddddE±dd，十进制指数）、'g'（指数很大时用'e'格式，否则'f'格式）、'G'（指数很大时用'E'格式，否则'f'格式）。注意，第二参数为字符，并非字符串。单引号。

prec控制精度（排除指数部分）：对'f'、'e'、'E'，它表示小数点后的数字个数；对'g'、'G'，它控制总的数字个数。如果prec 为-1，则代表使用最少数量的、但又必需的数字来表示f。

```
func formatFloatDemo() {
    res := strconv.FormatFloat(3.14150, 'f', -1, 64) // prec为-1表示使用最少数量表示，最后的0就省略了，如果prec为1，表示小数点后1位
    fmt.Printf("类型是:%T 值是:%#v\n", res, res)
}
```

其他

isPrint()

```
func IsPrint(r rune) bool
```

返回一个字符是否是可打印的，和`unicode.IsPrint`一样，r必须是：字母（广义）、数字、标点、符号、ASCII空格。

```
func isPrintDemo() {
    //一个字符是否是可打印的
    res := strconv.IsPrint('n') // true 可打印
    // res := strconv.IsPrint('\n') // false  不可打印
    fmt.Printf("类型是:%T 值是:%#v\n", res, res)
}
```

CanBackquote()

```
func CanBackquote(s string) bool
```

返回字符串s是否可以不被修改的表示为一个单行的、没有空格和tab之外控制字符的反引号字符串。

```
func canBackquoteDemo() {
    //是否可以不被修改的表示为一个单行的、没有空格和tab之外控制字符的反引号字符串
    // res := strconv.CanBackquote(`lxx is Nb`) // true
    res := strconv.CanBackquote(`lxx is
    Nb`) // false ： 因为带回车
    fmt.Printf("类型是:%T 值是:%#v\n", res, res)
}
```

Append系列

Append类的函数和Format类的函数工作方式类似，只不过是将转换后的结果追加到一个slice中

将其他类型转换成字符串后append到一个slice中：`AppendBool()`、`AppendFloat()`、`AppendInt()`、`AppendUint()`

```
func appendDemo() {
    b := []byte("99的十进制是：")
    // 将转换为10进制的string，追加到slice中
    b = strconv.AppendInt(b, 99, 10)
    fmt.Println(string(b))
}
```

Quote函数族

- 将 s 转换为双引号字符串

```
func Quote(s string) string
```

- 将 r 转换为单引号字符

```
func QuoteRune(r rune) string
```

- 功能同上，非 ASCII 字符和不可打印字符会被转义

```
func QuoteRuneToASCII(r rune) string
```

- 功能同上，非图形字符会被转义

```
func QuoteRuneToGraphic(r rune) string
```

- 功能同上，非 ASCII 字符和不可打印字符会被转义

```
func QuoteToASCII(s string) string
```

- 功能同上，非图形字符会被转义

```
func QuoteToGraphic(s string) string
```

- Unquote 将“带引号的字符串” s 转换为常规的字符串（不带引号和转义字符）， s 可以是“单引号”、“双引号”或“反引号”引起来的字符串（包括引号本身），如果 s 是单引号引起来的字符串，则返回该该字符串代表的字符

```
func Unquote(s string) (string, error)
```

示例：

```
func quoteDemo() {
    s := "Hello\t世界！\n"
    fmt.Println(s)                          // Hello 世界！（换行）
    fmt.Println(strconv.Quote(s))           // "Hello\t世界！\n"
    fmt.Println(strconv.QuoteToASCII(s))    // "Hello\t\u4e16\u754c\uff01\n"
    fmt.Println(strconv.QuoteToGraphic(s)) // "Hello\t世界！\n"
}
```