Strings and Characters

string 是一系列 character 的集合，兼容 Unicode

字面量

string 的字面量由双引号包围

多行的 string 用三个双引号前后包围（同 python），第一行的缩进确定了之后的缩进

字面量里可以有 `\n` 这样的转义符，也可以有 Unicode，形如 `\u{xxxx}`

如果需要在多行字符串里出现连续的三个双引号，需要至少转义其中一个

可以在字面量两端加上 `#` 来避免转义，如果想要使转义符发挥作用，需要加上 `#`，如 `\#"Line 1\#nLine 2"#`

初始化

初始化一个 String 变量可以给其赋值空串或者使用构造函数 `String()`

判断字符串是否为空则检查 `str.isEmpty` 属性

string 是否可修改取决于声明为变量还是常量

string 是 value type，即传参或赋值时传值

这保证了收到 string 参数时可以随意修改而不影响原值

编译器会优化，只有真的需要复制并传值时才会复制

Character

用 for-in 循环可以遍历 String 里的 Character

也可以通过一个字符的 string 字面量以及显式的类型声明创建一个 character

```
let exclamationMark: Character = "!"
```

string 可以通过一个 character 的数组来初始化

```
let catCharacters: [Character] = ["C", "a", "t", "!", "🐱"]
let catString = String(catCharacters)
```

字符串拼接

可以通过 `+` 拼接 string

也可以通过 `append` 方法在 string 后添加 character

```
let exclamationMark: Character = "!"
welcome.append(exclamationMark)
```

String Interpolation

可以在字符串字面量里加入常量/变量/表达式，只要用括号包起来并用 backslash 来 escape 左括号，如下

```
let multiplier = 3
let message = "\(multiplier) times 2.5 is \(Double(multiplier) * 2.5)"
// message is "3 times 2.5 is 7.5"
```

这部分的避免转义和恢复转义同上

```
print(#"6 times 7 is \#(6 * 7)."#)
// Prints "6 times 7 is 42."
```

Unicode

string 和 character 都是兼容 Unicode 的

string 由 21 bit 的 Unicode 标量值组成，每个值都唯一代表一个字符

character 则是一个扩展字素簇，即一个 Unicode 标量值的序列，用于代表单个字符

> \u00E9 可以表示为 \u0065\u0301
>
> ```
> let eAcute: Character = "\u{E9}"                         // é
> let combinedEAcute: Character = "\u{65}\u{301}"          // e followed by ´
> // eAcute is é, combinedEAcute is é
> ```

扩展字素簇可以使多个 Unicode 标量组成一个 Character

获取 string 中的 character 数可以用 `count` 属性（因为扩展字素簇的原因，增加 Unicode 标量不总是改变 character 数）

```swift
var word = "cafe"
print("the number of characters in \(word) is \(word.count)")
// Prints "the number of characters in cafe is 4"

word += "\u{301}"    // COMBINING ACUTE ACCENT, U+0301

print("the number of characters in \(word) is \(word.count)")
// Prints "the number of characters in café is 4"
```

操作 String

下标

由于 Unicode 的缘故，string 不能通过整数来索引，其索引类型为 `String.Index`

`startIndex` 指向第一个 character，而 `endIndex` 指向最后一个 character 的后面

可以用 String 的 `index()` 方法获取下标，然后访问对应位置的 character

```swift
let greeting = "Guten Tag!"
greeting[greeting.startIndex]
// G
greeting[greeting.index(before: greeting.endIndex)]
// !
greeting[greeting.index(after: greeting.startIndex)]
// u
let index = greeting.index(greeting.startIndex, offsetBy: 7)
greeting[index]
// a
```

试图获取超过界限的下标或者下标处的 character 会导致 runtime error

使用 `indices` 属性可以获得所有的下标

```swift
for index in greeting.indices {
    print("\(greeting[index]) ", terminator: "")
}
```

插入与删除

插入可以插入单个 character 或者一个 string，使用 `insert(_:at:)/insert(contentsOf:at:)`

删除也可以删除单个 character 或者一个子串，使用 `remove(at:)/removeSubrange(_:)`

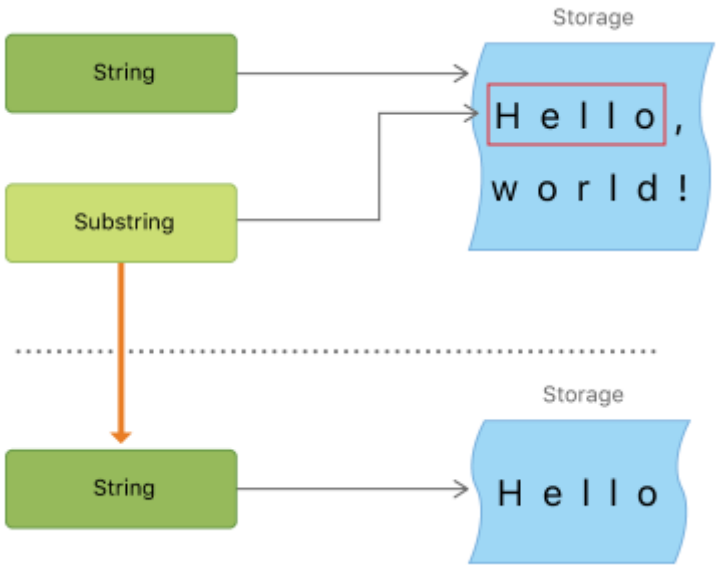
删除子串时的 range 用 `..<` 或 `...` 定义

这些操作均适用于实现了 `RangeReplaceableColletion` 的类型

Substring

通过切片等方法获得的是 Substring，大部分适用于 String 的方法同样适用于 Substring，但是如果想长久保存结果，就需要将 Substring 转换成 String

substring 在被修改之前都和 string 共享内存，而 string 在内存中有自己的空间（只有两个 string 相等时才会共享一片内存）



比较字符串

swift 提供三种比较字符串的方法

- string/character equality
- prefix equality
- suffix equality

String and Character Equality

通过 `==` 和 `!=` 检查 string 或是 character 是否相等

相等的条件是扩展字素簇正则等价，即有相同的语义和形态，而组成的 Unicode 标量可以不一样

```
1 "\u{E9}" == "\u{65}\u{301}" // true
```

Prefix and Suffix Equality

通过 `hasPrefix(_:)/hasSuffix(_:)` 检查字符串是否有特定的前缀/后缀

比较是通过逐字符进行正则等价比较实现的

Unicode Representations of String

在存储时，Unicode string 会被存储为某种具体的形式（utf-8，utf-16，……），每种形式都将 string 编码成连续的 code unit（对 utf-8 来说就是 8 bit 的 code unit）

除了按照扩展字素簇访问每个 character，也可以访问其 code unit

```
for codeUnit in dogString.utf8 {
    print("\(codeUnit) ", terminator: "")
}
print("")
// Prints "68 111 103 226 128 188 240 159 144 182 "
```

也可以按 Unicode 标量来访问