Collection Types

swift 提供了三种基本的 collection

- Array
- Set
- Dictionary

这三种容器都是泛型的

声明为变量的容器是可变的，可以在创建后增加或删除或修改其中的元素，而声明为常量的容器大小和内容均不可变

考虑到可能发生的 bug 和对性能的优化，只有确定容器内容会变化的时候声明为变量

Array

Array 将同种数据按序存储，其中的内容可重复

初始化

其类型写作 `Array<T>` ，也可以简写为 `[T]`

```
var someInts = [Int]()
```

当上下文信息足够推断出类型时，也可以用 `[]` 直接初始化空数组

也可以为初始化提供参数

```
var threeDoubles = Array(repeating: 0.0, count: 3)
// threeDoubles is of type [Double], and equals [0.0, 0.0, 0.0]
```

使用 `+` 可以将两个数组连起来，形成新数组

也可以用字面量来初始化，如 `[1, 2, 3]`

访问和修改

Array 有属性 `count` 和 `isEmpty` ，可以通过 `append` 在数组后增加元素，或是用 `+=` 来增加数组

使用 `[index]` 可以访问 index 位置的元素，也可以修改对应元素，如果下标超出范围会导致 runtime error

也可以用 range 修改一个范围内的元素（即使长度不匹配）

```
shoppingList[4...6] = ["Bananas", "Apples"]
```

要在数组中插入元素使用 `insert(_:at:)`，删除则使用 `remove(at:)`，remove 返回被移除的元素

删除最后一个元素有专门的函数 `removeLast()`

使用 for-in 循环可以遍历数组中的元素，如果需要下标则使用 `enumerated()`

```
for (index, value) in shoppingList.enumerated() {
    print("Item \(index + 1): \(value)")
}
```

Set

Set 存储同类型的互不相同的数据，Set 是无序的

Hash

集合中的元素必须是 `hashable`，即能够提供 hash 值，满足如果 `a == b` 则 `a.hashValue == b.hashValue`

所有基本类型都是 hashable 的，可以用作 set 中的元素或者 dictionary 中的 key

自定类型需要实现 `Hashable` 的接口，返回的 hash 值不要求在多次执行中都相同，且由于 Hashable 实现了 Equatable，所以也必须实现 `==` 操作符，满足等价关系（自反，对称，传递）

初始化

Set 的类型为 `Set<T>`

可以用构造函数初始化空集合

```
var letters = Set<Character>()
```

如果上下文提供了足够的信息，也可以用 `[]` 来初始化

可以用数组的字面量初始化 set，此时需要显式写出类型信息，如果字面量中的值类型均相同，则可以不写具体的类型，由 swift 推断

```swift
var favoriteGenres: Set<String> = ["Rock", "Classical", "Hip hop"]
var favoriteGenres: Set = ["Rock", "Classical", "Hip hop"]
```

访问和修改

Set 同样有属性 `count` 和 `isEmpty`

插入元素使用 `insert(_:)`，删除则使用 `remove(_:)`，如果集合中有该元素则返回被删除元素，否则返回 `nil`

检查元素是否在集合使用 `contains(_:)`

同样可以用 for-in 循环遍历集合，由于集合是无序的，如果想按序访问需要 `sorted()`

```swift
for genre in favoriteGenres.sorted() {
    print("\(genre)")
}
```

集合操作

集合支持交，并，差，对称差等操作

```swift
let oddDigits: Set = [1, 3, 5, 7, 9]
let evenDigits: Set = [0, 2, 4, 6, 8]
let singleDigitPrimeNumbers: Set = [2, 3, 5, 7]

oddDigits.union(evenDigits).sorted()
// [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
oddDigits.intersection(evenDigits).sorted()
// []
oddDigits.subtracting(singleDigitPrimeNumbers).sorted()
// [1, 9]
oddDigits.symmetricDifference(singleDigitPrimeNumbers).sorted()
// [1, 2, 9]
```

同样可以检查集合的包含关系

- `==` 判断是否相等
- `isSubset(of:)` 判断是否是子集
- `isSuperset(of:)` 判断是否是超集
- `isStrictSubset(of:)/isStrictSuperset(of:)` 判断是否是真子集/超集

- `isDisjoint(with:)` 判断是否是 disjoint

Dictionary

Dictionary 存储键值对，是无序的。每个 value 关联一个唯一的 key

初始化

Dictionary 的类型为 `Dictionary<Key, Element>`，可以简写为 `[Key:Element]`

可以使用构造函数初始化

当上下文信息足够时可以用 `[:]` 初始化空字典，也可以用字面量初始化

```
var airports: [String: String] = ["YYZ": "Toronto Pearson", "DUB": "Dublin"]
```

同 set，在字面量中类型无歧义时，可以省略显式的声明交给 swift 推断

访问和修改

Dictionary 同样有属性 `count` 和 `isEmpty`

可以用 Key 访问字典

```
airports["LHR"] = "London"
```

如果 key 不存在则新增，否则修改

也可以用 `updateValue(_:forKey:)` 来修改 value 的值，该函数会返回修改之前的旧值（一个 optional）

使用方括号的形式读取时会返回 optional 值

```
if let airportName = airports["DUB"] {
    print("The name of the airport is \(airportName).")
} else {
    print("That airport is not in the airports dictionary.")
}
```

如果要删去某个键值对，只需要将其 value 赋为 `nil`，或是使用 `removeValue(forKey:)`，区别在于会返回被移除的值

遍历字典可以使用 for-in 循环，返回一个 tuple，如果只需要 key 或者 value，用 for-in 遍历 `values/keys` 属性，也可以用 `values/keys` 初始化数组

由于字典没有序，需要有序遍历时使用 `sorted()`