Enumerations

enumeration 定义了一系列相关的值，类似 C 中的枚举，但是 swift 的更加灵活，可以不为枚举类型提供值，如果需要值的话则可以提供 string，character，整数或是浮点类型

enumeration 的地位和 class 类似，支持计算属性和实例方法等

语法

用 `enum` 关键字定义枚举类型，用 `case` 关键字引入新的 case

```
enum CompassPoint {
    case north
    case south
    case east
    case west
}
```

每个枚举都定义一个新类型，如果对一个已经确定类型的变量再次赋值，可以用简化的语法

```
var directionToHead = CompassPoint.west
directionToHead = .east
```

与 switch 一起使用

可以用 switch 来 match 一个枚举变量，此时同样可以使用简化的语法

```
directionToHead = .south
switch directionToHead {
case .north:
    print("Lots of planets have a north")
case .south:
    print("Watch out for penguins")
case .east:
    print("Where the sun rises")
case .west:
    print("Where the skies are blue")
}
// Prints "Watch out for penguins"
```

如果使用 switch，必须为所有可能的取值提供处理语句

遍历

可以用 `allCases` 属性获取枚举类型定义中所有的 case

```
enum Beverage: CaseIterable {
    case coffee, tea, juice
}
let numberOfChoices = Beverage.allCases.count
```

`allCases` 提供一个容器，其中元素的类型就是枚举类型，可以用 for-in 循环来遍历

Associated Values

可以在 case 中存储各种类型的值

```
enum Barcode {
    case upc(Int, Int, Int, Int)
    case qrCode(String)
}
```

如上述定义中 upc 可以附带存储 4 个 Int，而 qrCode 可以存储一个 String

之后就可以创建对应的枚举变量

```
var productBarcode = Barcode.upc(8, 85909, 51226, 3)
productBarcode = .qrCode("ABCDEFGHIJKLMNOP")
```

这种用法比较类似 C 中的 union，可以用 switch 判断具体的类型，用 let 或者 var 绑定其中的值

```
switch productBarcode {
case let .upc(numberSystem, manufacturer, product, check):
    print("UPC : \(numberSystem), \(manufacturer), \(product), \(check).")
case let .qrCode(productCode):
    print("QR code: \(productCode).")
}
// Prints "QR code: ABCDEFGHIJKLMNOP."
```

Raw Values

枚举的 case 本身可以附带同一类型的值

```
enum ASCIIControlCharacter: Character {
    case tab = "\t"
    case lineFeed = "\n"
    case carriageReturn = "\r"
}
```

每个 case 的 raw value 是唯一的，使用 Int 等类型作为 raw value 时，swift 可以隐式地进行分配

```
enum Planet: Int {
    case mercury = 1, venus, earth, mars, jupiter, saturn, uranus, neptune
}
```

Int 会递增分配，如果使用 String 则默认的 raw value 为 case 名

```
enum CompassPoint: String {
    case north, south, east, west
}
```

如果定义了 raw value，则会获得一个接受 raw value 的构造器

```
let possiblePlanet = Planet(rawValue: 7)
// possiblePlanet is of type Planet? and equals Planet.uranus
```

由于不一定所有值都有对应的 case，所以该构造器返回一个 optional

递归枚举

即枚举的 case 包含了同类枚举的实例的情况，需要在 case 前加上 `indirect`

```
enum ArithmeticExpression {
    case number(Int)
    indirect case addition(ArithmeticExpression, ArithmeticExpression)
    indirect case multiplication(ArithmeticExpression, ArithmeticExpression)
}
```

或者在 `enum` 前加上 `indirect` 也可以

递归枚举适合定义天生递归的数据结构