翻译：fd5788 校对：yankuangshi

泛型参数

本页包含内容：

本节涉及泛型类型、泛型函数以及泛型构造器的参数，包括形参和实参。声明泛型类型、函数或构造器时，须指定相应的类型参数。类型参数相当于一个占位符，当实例化泛型类型、调用泛型函数或泛型构造器时，就用具体的类型实参替代之。

关于 Swift 语言的泛型概述，见泛型(第二部分第22章)。

泛型形参语句

泛型形参语句指定泛型类型或函数的类型形参，以及这些参数的关联约束和要求。泛型形参语句用尖括号（<>）包住，并且有以下两种形式：

```
<generic parameter list>
<generic parameter list where requirements >
```

泛型形参列表中泛型形参用逗号分开，每一个采用以下形式：

```
type parameter : constrain
```

泛型形参由两部分组成：类型形参及其后的可选约束。类型形参只是占位符类型（如T，U，V，KeyType，ValueType等）的名字而已。你可以在泛型类型、函数的其余部分或者构造器声明，以及函数或构造器的签名中使用它。

约束用于指明该类型形参继承自某个类或者遵守某个协议或协议的一部分。例如，在下面的泛型中，泛型形参`T: Comparable`表示任何用于替代类型形参`T`的类型实参必须满足`Comparable`协议。

```
func simpleMin<T: COmparable>(x: T, y: T) -> T {
    if x < y {
        return y
    }
    return x
}
```

如，Int和Double均满足Comparable协议，该函数接受任何一种类型。与泛型类型相反，调用泛型函数或构造器时不需要指定泛型实参语句。类型实参由传递给函数或构造器的实参推断而出。

```
simpleMin(17, 42) // T is inferred to be Int
simpleMin(3.14159, 2.71828) // T is inferred to be
Double
```

Where语句

要想对类型形参及其关联类型指定额外要求，可以在泛型形参列表之后添加where语句。where语句由关键字where及其后的用逗号分割的多个要求组成。

where语句中的要求用于指明该类型形参继承自某个类或遵守某个协议或协议的一部分。尽管where语句有助于表达类型形参上的简单约束（如T: Comparable等同于T where T: Comparable，等等），但是依然可以用来对类型形参及其关联约束提供更复杂的约束。如，<T where T: C, T: P>表示泛型类型T继承自类C且遵守协议P。

如上所述，可以强制约束类型形参的关联类型遵守某个协议。<T: Generator where T.Element: Equatable>表示T遵守Generator协议，而且T的关联类型T.Element遵守Eauatable协议（T有关联类型是因为Generator声明了Element，而T遵守Generator协议）。

也可以用操作符==来指定两个类型等效的要求。例如，有这样一个约束：T和U遵守Generator协议，同时要求它们的关联类型等同，可以这样来表达：<T: Generator, U: Generator where T.Element == U.Element>。

当然，替代类型形参的类型实参必须满足所有类型形参所要求的约束和要

求。

泛型函数或构造器可以重载，但在泛型形参语句中的类型形参必须有不同的约束或要求，抑或二者皆不同。当调用重载的泛型函数或构造器时，编译器会用这些约束来决定调用哪个重载函数或构造器。

泛型类可以生成一个子类，但是这个子类也必须是泛型类。

Grammar of a generic parameter clause

parameter-clause → <generic-parameter-listrequirement-clause >

generic-parameter-list → generic-parameter generic-parameter,generic-parameter-list

generic-parameter → type-name

generic-parameter → type-name:type-identifier

generic-parameter → type-name:protocol-composition-type

requirement-clause → `where`requirement-list

requirement-list → requirement requirement,requirement-list

requirement → conformance-requirement same-type-requirement

conformance-requirement → type-identifier:type-identifier

conformance-requirement → type-identifier:protocol-composition-type

same-type-requirement → type-identifier==type-identifier

泛型实参语句

泛型实参语句指定泛型类型的类型实参。泛型实参语句用尖括号（<>）包住，形式如下：

```
< generic argument list >
```

泛型实参列表中类型实参有逗号分开。类型实参是实际具体类型的名字，用来替代泛型类型的泛型形参语句中的相应的类型形参。从而得到泛型类

型的一个特化版本。如，Swift标准库的泛型字典类型定义如下：

```
  struct Dictionary<KeyTypel: Hashable, ValueType>:
Collection,
DictionaryLiteralConvertible {
    /* .. */
}
```

泛型`Dictionary`类型的特化版本，`Dictionary<String, Int>`就是用具体的`String`和`Int`类型替代泛型类型`KeyType: Hashable`和`ValueType`产生的。每一个类型实参必须满足它所替代的泛型形参的所有约束，包括任何`where`语句所指定的额外的要求。上面的例子中，类型形参`KeyType`要求满足`Hashable`协议，因此`String`也必须满足`Hashable`协议。

可以用本身就是泛型类型的特化版本的类型实参替代类型形参（假设已满足合适的约束和要求）。例如，为了生成一个元素类型是整型数组的数组，可以用数组的特化版本`Array<Int>`替代泛型类型`Array<T>`的类型形参`T`来实现。

```
let arrayOfArrays: Array<Array<Int>> = [[1, 2, 3],
[4, 5, 6], [7, 8, 9]]
```

如泛型形参语句所述，不能用泛型实参语句来指定泛型函数或构造器的类型实参。

Grammar of a generic argument clause

generic-argument-clause → <generic-argument-list>

generic-argument-list → generic-argument generic-argument,generic-argument-list

generic-argument → type