彻底掌握Kotlin

讲师简介

宁传奇

Jason

曾任华为高级工程师、架构师，全栈开发爱好者。

技术这条路，不和你一比高下，只伴你一同成长。

定义泛型类

定义泛型类

- 泛型类的构造函数可以接受任何类型。
- MagicBox类指定的泛型参数由放在一对<>里的字母T表示，T是个代表item类型的占位符。MagicBox类接受任何类型的item作为主构造函数值（item：T），并将item值赋给同样是T类型的subject私有属性。

```
1   class MagicBox<T>(item: T) {
2       private var subject: T = item
3   }
4   class Boy(val name: String, val age: Int)
5   class Dog(val weight: Int)
6
7   fun main() {
8       val box1: MagicBox<Boy> = MagicBox(Boy( name: "Jack",  age: 20))
9       val box2: MagicBox<Dog> = MagicBox(Dog( weight: 20))
10  }
```

注意

- 泛型参数通常用字母T（代表英文type）表示，当然，想用其他字母，甚至是英文单词都是可以的。不过，其他支持泛型的语言都在用这个约定俗成的T，所以建议你继续用它，这样写出的代码别人更容易理解。

泛型函数

泛型函数

- 泛型参数也可以用于函数。
- 定义一个函数用于获取元素，当且仅当MagicBox可用时，才能获取元素。

```
1  class MagicBox<T>(item: T) {
2      var available = false
3      private var subject: T = item
4
5      fun fetch(): T? {
6          return subject.takeIf { available }
7      }
8  }
9  class Boy(val name: String, val age: Int)
10
11 class Dog(val weight: Int)
12 fun main() {
13     val box1: MagicBox<Boy> = MagicBox(Boy( name: "Jack",  age: 20))
14     val box2: MagicBox<Dog> = MagicBox(Dog( weight: 20))
15     box1.available = true
16     box1.fetch()?.run { this: Boy
17         println("you find $name")
18     }
19 }
```

多泛型参数

- 泛型函数或泛型类也可以有多个泛型参数。

```
1  class MagicBox<T>(item: T) {
2      var available = false
3      private var subject: T = item
4
5      fun fetch(): T? {
6          return subject.takeIf { available }
7      }
8      fun <R> fetch(subjectModFunction: (T) -> R): R? {
9          return subjectModFunction(subject).takeIf { available }
10     }
11 }
12 fun main() {
13     val box1: MagicBox<Boy> = MagicBox(Boy( name: "Jack",  age: 20))
14     box1.available = true
15     //男孩变成了男人，返回
16     val man : Man? = box1.fetch() { it: Boy
17         Man(it.name, it.age.plus( other: 10))
18     }
19     man?.let { println("${it.name},${it.age}") }
20 }
```

泛型类型约束

泛型类型约束

- 如果要确保MagicBox里面只能装指定类型的物品，如Human类型，怎么办？

```
1   class MagicBox<T : Human>(item: T) {
2       var available = false
3       var subject: T = item
4
5       fun fetch(): T? {
6           return subject.takeIf { available }
7       }
8   }
9
10  fun main() {
11      val box1: MagicBox<Human> = MagicBox(Boy( name: "Jack",  age: 20))
12  }
```

vararg关键字与get函数

- MagicBox能存放任何类型的Human实例，但一次只能放一个，如果需要放入多个实例呢？

```
class MagicBox<T : Human>(vararg item: T) {
    var available = false
    var subject: Array<out T> = item

    fun fetch(index: Int): T? {
        return subject[index].takeIf { available }
    }

    fun <R> fetch(index: Int, subjectModFunction: (T) -> R): R? {
        return subjectModFunction(subject[index]).takeIf { available }
    }
}
```

[ ]操作符取值

- 想要通过[ ]操作符取值，可以重载运算符函数get函数。

```
class MagicBox<T : Human>(vararg item: T) {
    var available = false
    var subject: Array<out T> = item

    operator fun get(index: Int): T? = subject[index].takeIf { available }

    fun fetch(index: Int): T? {
        return subject[index].takeIf { available }
    }

    fun <R> fetch(index: Int, subjectModFunction: (T) -> R): R? {
        return subjectModFunction(subject[index]).takeIf { available }
    }
}
```

out

- out（协变），如果泛型类只将泛型类型作为函数的返回（输出），那么使用 out，可以称之为生产类/接口，因为它主要是用来生产（produce）指定的泛型对象。

```
1  interface Production<out T> {
2      fun product(): T
3  }
```

in

- in（逆变），如果泛型类只将泛型类型作为函数的入参（输入），那么使用 in，可以称之为消费者类/接口，因为它主要是用来消费（consume）指定的泛型对象。

```
5  interface Consumer<in T> {
6      fun consume(item: T)
7  }
```

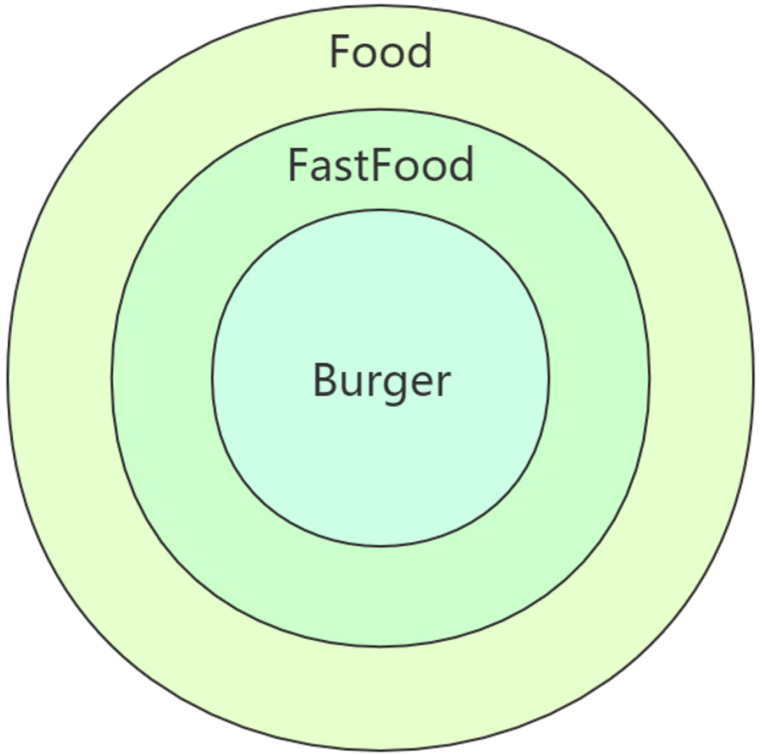
invariant（不变）

- 如果泛型类既将泛型类型作为函数参数，又将泛型类型作为函数的输出，那么既不用 out 也不用 in。

```
 9    interface ProductionConsumer<T> {
10        fun product(): T
11        fun consume(item: T)
12    }
```

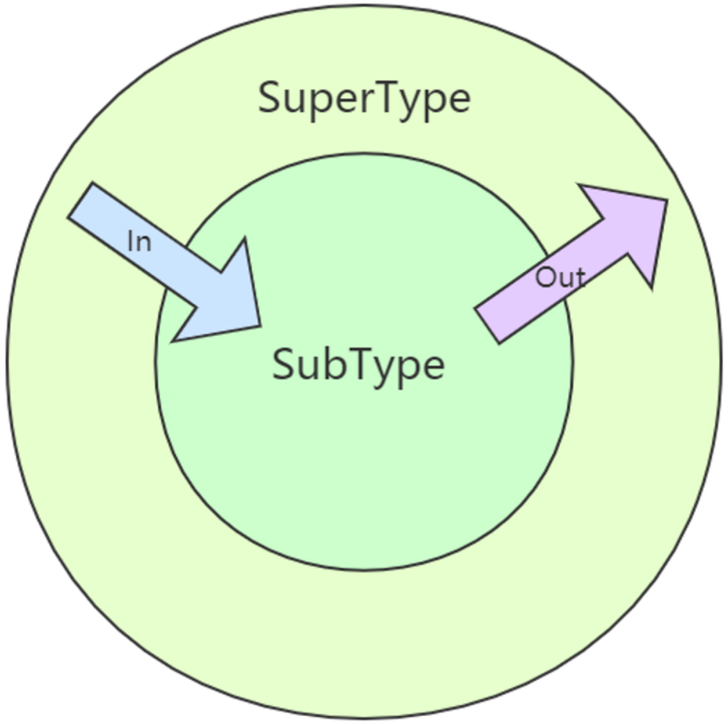
为什么使用in&out?

- 举个例子，我们定义下汉堡类对象，它是一种快餐，也是一种食物。



为什么使用in&out?

- 父类泛型对象可以赋值给子类泛型对象，用 in。
- 子类泛型对象可以赋值给父类泛型对象，用 out。



reified

- 有时候，你可能想知道某个泛型参数具体是什么类型，reified关键字能帮你检查泛型参数类型。Kotlin不允许对泛型参数T做类型检查，因为泛型参数类型会被类型擦除，也就是说，T的类型信息在运行时是不可知的，Java也有这样的规则。

```
class MagicBox<T: Human>() {
    fun <T> randomOrBackup(backup: () -> T): T {
        val items : List<Human> = listOf(
            Boy( name: "Jack", age: 20),
            Man( name: "John", age: 35))
        val random : Human = items.shuffled().first()
        return if(random is T){
            random
        }else{
            backup()
        }
    }
}
```

```
class MagicBox<T: Human>() {
    inline fun <reified T> randomOrBackup(backup: () -> T): T {
        val items : List<Human> = listOf(
            Boy( name: "Jack", age: 20),
            Man( name: "John", age: 35))
        val random:Human = items.shuffled().first()
        println("random value")
        println(random)
        return if(random is T){
            println("random is T")
            random
        }else{
            println("backup value")
            backup()
        }
    }
}
```