链滴

# Kotlin 从入门到夺门而出 (2) 之 Kotlin 与 Java 语法区别

作者：aohanyao
原文链接：https://ld246.com/article/1537368230848
来源网站：链滴

# 打印日志

- Java

```
System.out.print("Amit Shekhar");
System.out.println("Amit Shekhar");
```

- Kotlin

```
print("Amit Shekhar")
println("Amit Shekhar")
```

---

# 常量与变量

- Java

```
String mName = "Java  string name";
final String mName = "Java  string name";
```

- Kotlin

```
var mName = "kotlin string name"
val mName = "kotlin string name"
```

---

# null声明

- Java

```
String mName;
mName = null;
```

- Kotlin

```
var mName : String?
mName = null
```

---

# 空判断

- Java

```
if (str != null) {
    int length = str.length();
}
```

- Kotlin

```
str?.let {
    val length = str.length
}
// 更简单的写法
val length = str?.length
// 为null赋予默认值
val length = str?.length?:0
```

---

# 字符串拼接

- Java

```
String firstName = "Jiang";
String lastName = "super";
String message = "My name is: " + firstName + " " + lastName;
```

- Kotlin

```
val firstName = "Jiang"
val lastName = "super"
val message = "My name is: $firstName $lastName"
```

---

# 换行

- Java

```
String text = "First Line\n" +
        "Second Line\n" +
        "Third Line";
```

- Kotlin

```
val text = """
      |First Line
      |Second Line
      |Third Line
      """.trimMargin()
```

---

# 三元表达式(三目运算符)

- Java

```
String text = x > 5 ? "x > 5" : "x <= 5";
```

- Kotlin

```
val text = if (x > 5)
        "x > 5"
      else "x <= 5"
```

---

# 操作符

- java

```
final int andResult  = a & b;
final int orResult   = a | b;
final int xorResult  = a ^ b;
final int rightShift = a >> 2;
final int leftShift  = a << 2;
final int unsignedRightShift = a >>> 2;
```

- Kotlin

```
val andResult  = a and b
val orResult   = a or b
val xorResult  = a xor b
val rightShift = a shr 2
val leftShift  = a shl 2
val unsignedRightShift = a ushr 2
```

---

# 类型判断和转换 (声明式)

- Java

```
if (object instanceof Car) {
}
Car car = (Car) object;
```

- Kotlin

```
if (object is Car) {
}
var car = object as Car
```

---

# 类型判断和转换 (隐式)

- Java

```
if (object instanceof Car) {
   Car car = (Car) object;
}
```

- Kotlin

```
if (object is Car) {
   var car = object // 智能转换
}
```

---

# 多重条件

- Java

```
if (score >= 0 && score <= 300) { }
```

- Kotlin

```
if (score in 0..300) { }
```

---

# 更灵活的case语句

- Java

```
int score = 8// 定义分数;
String grade;
switch (score) {
    case 10:
    case 9:
        grade = "Excellent";
        break;
    case 8:
    case 7:
    case 6:
        grade = "Good";
        break;
    case 5:
    case 4:
        grade = "OK";
        break;
    case 3:
    case 2:
    case 1:
        grade = "Fail";
        break;
    default:
        grade = "Fail";
}
```

- Kotlin

```
var score = 8// 定义分数
```

```
var grade = when (score) {
    9, 10 -> "Excellent"
    in 6..8 -> "Good"
    4, 5 -> "OK"
    in 1..3 -> "Fail"
    else -> "Fail"
}
```

---

# for循环

- Java

```
for (int i = 1; i <= 10 ; i++) { }

for (int i = 1; i < 10 ; i++) { }

for (int i = 10; i >= 0 ; i--) { }

for (int i = 1; i <= 10 ; i+=2) { }

for (int i = 10; i >= 0 ; i-=2) { }

for (String item : collection) { }

for (Map.Entry<String, String> entry: map.entrySet()) { }
```

- Kotlin

```
for (i in 1..10) { }

for (i in 1 until 10) { }

for (i in 10 downTo 0) { }

for (i in 1..10 step 2) { }

for (i in 10 downTo 1 step 2) { }

for (item in collection) { }

for ((key, value) in map) { }
```

---

# 更方便的集合操作

- Java

```
final List<Integer> listOfNumber = Arrays.asList(1, 2, 3, 4);

final Map<Integer, String> keyValue = new HashMap<Integer, String>();
```

```
map.put(1, "Amit");
map.put(2, "Ali");
map.put(3, "Mindorks");

// Java 9
final List<Integer> listOfNumber = List.of(1, 2, 3, 4);

final Map<Integer, String> keyValue = Map.of(1, "Amit",
                                             2, "Ali",
                                             3, "Mindorks");
```

- Kotlin

```
val listOfNumber = listOf(1, 2, 3, 4)
val keyValue = mapOf(1 to "Amit",
            2 to "Ali",
            3 to "Mindorks")
```

---

# 遍历

- Java

```
// Java 7 以及更低
for (Car car : cars) {
  System.out.println(car.speed);
}

// Java 8+
cars.forEach(car -> System.out.println(car.speed));

// Java 7 以及更低
for (Car car : cars) {
  if (car.speed > 100) {
    System.out.println(car.speed);
  }
}

// Java 8+
cars.stream().filter(car -> car.speed > 100).forEach(car -> System.out.println(car.speed));
```

- Kotlin

```
cars.forEach {
    println(it.speed)
}

cars.filter { it.speed > 100 }
      .forEach { println(it.speed)}
```

---

# 方法定义

- Java

```
void doSomething() {
   // todo here
}

void doSomething(int... numbers) {
   // todo here
}
```

- Kotlin

```
fun doSomething() {
   // logic here
}

fun doSomething(vararg numbers: Int) {
   // logic here
}
```

---

# 带返回值的方法

- Java

```
int getScore() {
   // logic here
   return score;
}
```

- Kotlin

```
fun getScore(): Int {
   // logic here
   return score
}

// 单表达式函数
fun getScore(): Int = score
```

---

# 无结束符号

- Java

```
int getScore(int value) {
    // logic here
    return 2 * value;
```

```
}
```

- Kotlin

```
fun getScore(value: Int): Int {
  // logic here
  return 2 * value
}

// 单表达式函数
fun getScore(value: Int): Int = 2 * value
```

---

## constructor 构造器

- Java

```
public class Utils {

   private Utils() {
     // 私有构造方法
   }

   public static int getScore(int value) {
       return 2 * value;
   }

}
```

- Kotlin

```
class Utils private constructor() {
   companion object {
       fun getScore(value: Int): Int {
           return 2 * value
       }

   }
}

// 第二个例子
object Utils {
   fun getScore(value: Int): Int {
       return 2 * value
   }
}
```

---

## Get Set 构造器

- Java

```java
public class Developer {

    private String name;
    private int age;

    public Developer(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Developer developer = (Developer) o;

        if (age != developer.age) return false;
        return name != null ? name.equals(developer.name) : developer.name == null;

    }

    @Override
    public int hashCode() {
        int result = name != null ? name.hashCode() : 0;
        result = 31 * result + age;
        return result;
    }

    @Override
    public String toString() {
        return "Developer{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

- Kotlin

```
data class Developer(val name: String, val age: Int)
```

---

# 原型扩展(拓展函数)

- Java

```
public class Utils {

    private Utils() {
      // 私有构造方法
    }

    public static int triple(int value) {
        return 3 * value;
    }

}

int result = Utils.triple(3);
```

- Kotlin

```
fun Int.triple(): Int {
  return this * 3
}

var result = 3.triple()
```

- Java

```
public enum Direction {
        NORTH(1),
        SOUTH(2),
        WEST(3),
        EAST(4);

        int direction;

        Direction(int direction) {
            this.direction = direction;
        }

        public int getDirection() {
            return direction;
        }
    }
```

- Kotlin

```
enum class Direction constructor(direction: Int) {
    NORTH(1),
    SOUTH(2),
    WEST(3),
    EAST(4);

    var direction: Int = 0
        private set

    init {
        this.direction = direction
    }
}
```

---