СПб АУ НОЦНТ РАН

Kotlin 04

20.11.2017

Взаимодействие с Java

- Использование Kotlin-деклараций из Java
- Работа с Java-кодом из Kotlin
- Системы сборки
  - Maven
  - Gradle
  - Ant
  - IntelliJ IDEA projects

Взаимодействие с Java

- Kotlin умеет "читать"читать Java-код
- Компилятор Java не знает про существование kt-файлы
- Сначала запускается kotlinc, генерируются класс-файлы
- Результат передается в javac в classpath

Свойства

```
class A(var name: String) // A().getName()/setName(s) in Java
```

Package level

```
// FILE: example.kt
package ru.spbau.mit

class Foo

fun bar(): String = "OK"
val property = 123
```

```
// In Java
new ru.spbau.mit.Foo();
ru.spbau.mit.ExampleKt.bar().length(); // statics
ru.spbau.mit.ExampleKt.getProperty() + 1;
```

Package level

```
// FILE: example.kt
@file:JvmName("Utils")
package ru.spbau.mit

class Foo

fun bar(): String = "OK"
val property = 123
```

```
// In Java
new ru.spbau.mit.Foo();
ru.spbau.mit.Utils.bar().length(); // statics
ru.spbau.mit.Utils.getProperty() + 1;
```

Package level

```
// FILE: example.kt
@file:JvmName("Utils")
@file:JvmMultifileClass
package ru.spbau.mit

class Foo

fun bar(): String = "OK"

// FILE: anotherExample.kt
@file:JvmName("Utils")
@file:JvmMultifileClass
package ru.spbau.mit

val property = 123
```

objects

```
object Singleton {
    fun foo() {}
}

class MyClass {
    companion object {
        fun bar() {}
    }
}
// In Java
Singleton.INSTANCE.foo() // non-static method
MyClass.Companion.bar(); // non-static method
```

@JvmStatic

```
object Singleton {
    @JvmStatic
    fun foo() {}
}

class MyClass {
    companion object {
        @JvmStatic
        fun bar() {}
    }
}

// In Java
Singleton.foo(); // static method
MyClass.bar(); // static method
```

@JvmField

```
class A {
    @JvmField
    val myField = 1
}

// In Java
A().myField + 1;
```

@JvmField

```
class Key(val value: Int) {
    companion object {
        @JvmField
        val COMPARATOR: Comparator<Key> =
            compareBy<Key> { it.value }
    }
}

// In Java
Key.COMPARATOR.compare(key1, key2);
```

const

```
object Obj {
    const val CONST = 1
}

class C {
    companion object {
        const val VERSION = 9
    }
}

const val MAX = 239
// In Java
int c = Obj.CONST;
int d = ExampleKt.MAX;
int v = C.VERSION;
```

JvmName for declarations

```
fun List<String>.filterValid(): List<String>
fun List<Int>.filterValid(): List<Int>
```

JvmName for declarations

```
fun List<String>.filterValid(): List<String>
@JvmName("filterValidInt")
fun List<Int>.filterValid(): List<Int>
```

Defaults

```
fun foo(x: Int = 1, y: String = "") {}
```

```
void foo(int x, String y) {}
void foo$default(int x, String y, int nonDefaultsMask) { ... }
```

@JvmOverloads

```
class Foo @JvmOverloads constructor(x: Int, y: Double = 0.0) {
    @JvmOverloads
    fun f(a: String, b: Int = 0, c: String = "abc") {
        ...
    }
}
// Constructors:
Foo(int x, double y)
Foo(int x)

// Methods
void f(String a, int b, String c) { }
void f(String a, int b) { }
void f(String a) { }
```

Checked exceptions

```
@Throws(IOException::class)
fun foo() {
    throw IOException()
}
```

Declaration-site variance

```
class Box<out T>(val value: T)

interface Base
class Derived : Base

fun boxDerived(value: Derived): Box<Derived> = Box(value)
fun unboxBase(box: Box<Base>): Base = box.value
```

Declaration-site variance

```
// return type - no wildcards
Box<Derived> boxDerived(Derived value) { ... }

// parameter - wildcards
Base unboxBase(Box<? extends Base> box) { ... }
// String is final
Base unboxString(Box<String> box) { ... }
```

@JvmWildcard/@JvmSuppressWildcards

```
fun boxDerived(value: Derived): Box<@JvmWildcard Derived> = Box
// is translated to
// Box<? extends Derived> boxDerived(Derived value) { ... }

fun unboxBase(box: Box<@JvmSuppressWildcards Base>): Base = bo
// is translated to
// Base unboxBase(Box<Base> box) { ... }
```

Вызываем Java из Kotlin

```
class A {
    String foo(String x) { ... }
}
```

Nullability

```
class A {
    // fun foo(x: String?): String?
    String foo(String x) { ... }
}
```

Nullability

```
class A {
    // fun foo(x: String?): String
    String foo(String x) { ... }
}
```

Nullability

```
// Using java libraries is not very seamless
System.out?.println()
```

Platform types

- T! = T..T?
- T? <: T!
- T! <: T
- T? <: T! and T <: T? => T <: T!
- T! <: T and T <: T? => T! <: T?
- X = Y <=> X <: Y and Y <: X
- T! = T
- T! = T?
- Платформенные типы невыразимы в Kotlin

Nullability

```
class A {
    // fun foo(x: String!): String!
    String foo(String x) { ... }
}
```

Nullability

```
a.foo("string")
a.foo(null)
// no "Unnecessary safe-call warning"
a.foo(null)?.hashCode()
// no errors
a.foo(null).hashCode()
```

Транзитивность эквивалентности

- String? = String! = String
- Но String? != String

Транзитивность эквивалентности

```
class A {
    static <T> T id(T t) { return t; }
}

val nullString /*: String! */ = A.id<String>(null)
nullString.hashCode() // NPE
```

Переход от платформенных типов

```
class A {
    static <T> T id(T t) { return t; }
}
fun foo(notNullString: String) {}

foo(
    // implicit !! here,
    // nulls don't spread accross Kotlin code
    A.id<String>(null)
)
```

Аннотации

```
class A {
    // fun foo(x: String): String
    @Nullable
    String foo(@NotNull String x) { ... }
}
```

Поддержанные аннотации

- Jetbrains nullability annotations
- JSR-305/Findbugs
- Android annotations
- Checker framework annotations

Используются во множестве современных Java-библиотек

Другие платформенные типы

- (Mutable)Collection<T>! = MutableCollection<T>..Collection<T>?
- Array<(out) T>! = Array<T>..Array<out T>?
- Raw-типы

Single Abstract Method

```
interface Runnable {
    void run();
}

// SAM constructor
// Works just as if Runnable had a constructor(b: () -> Unit)
val r = Runnable { println("Hello, world") }
```

SAM adapter

```
Future<?> submit(Runnable task);

// in Kotlin
executorService.submit { println("Hello, world") }
```

Волшебные свойства для Java геттеров/сеттеров

```
class Calendar {
    public int getFirstDayOfWeek() { ... }
}

// in Kotlin
calendar.firstDayOfWeek
calendar.getFirstDayOfWeek() // also possible
```