



WHY NOT KOTLIN?

自我介绍

- 梁茹曦，武汉大学2018级本科生
- 一个对计算机很感兴趣的普通本科生

本次演讲的主要内容

- 本次演讲不讲的：
 - Kotlin语法（请自己看文档）
 - Kotlin的fancy特性（时间限制）
 - Java的语法（请自己看书）
- 本次演讲**重点聚焦**的
 - Kotlin相比于java的优秀设计



1:NULL SAFETY

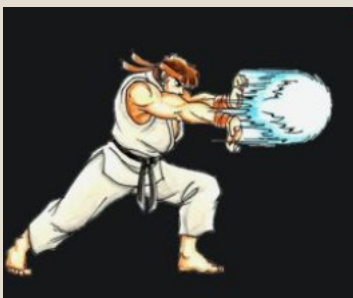
Null Safety

- Java中大家很容易被NPE困扰，而且NPE很容易就会被触发
- 比如下面的这个例子：

```
public class Main {  
    public static void main(){  
        Integer res = foo(null,null);  
    }  
    public static Integer foo(Integer lhs, Integer rhs){  
        return lhs + rhs;  
    }  
}
```

Null Safety

- 当然，你也有很多可以绕过去的办法
- 比如，多写“龟派气功”代码进行检查 🐢



```
public static Integer foo(Integer lhs, Integer rhs) {  
    if (lhs != null) {  
        if (rhs != null) {  
            return lhs + rhs;  
        }  
    }  
    return null;  
}
```

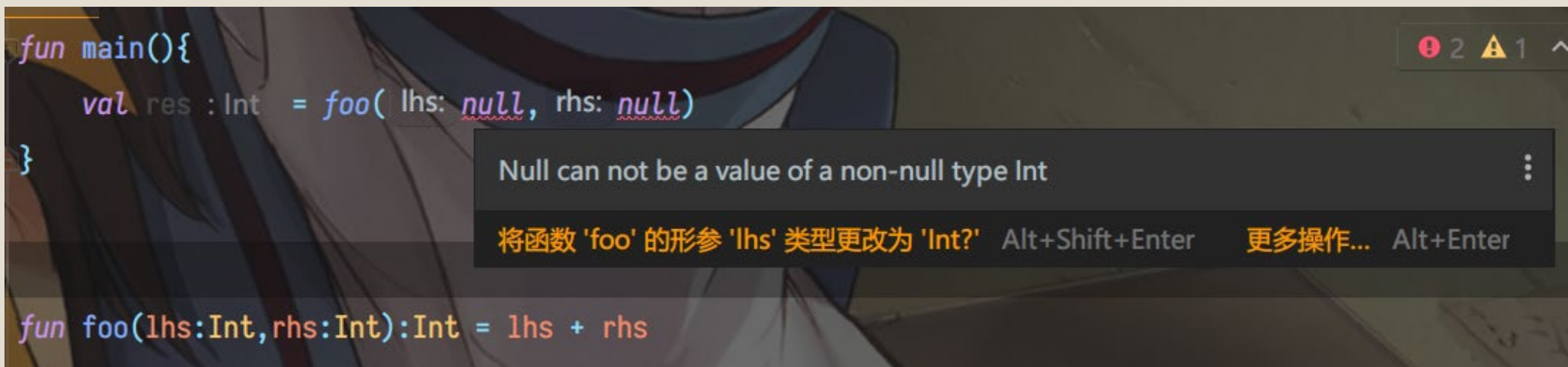
- 但很明显，龟派气功代码写起来很繁琐，而且没有从根源上解决问题

Null Safety

- 如果我们确定了我们函数的参数**不可能为空**，那该怎么办？
- 加上一些注解辅助编译器提示
 - 比如@NotNull注解
- 只可惜，这个也只是辅助提示
- 难道真的没有办法了吗.....☹

Null Safety in Kotlin

- 当然不是！我们可以转进Kotlin
- Kotlin在设计时就把可空和不可空在类型层面上区分开
 - 和裱糊匠说byebye！



```
fun main(){  
    val res : Int = foo( lhs: null, rhs: null)  
}  
  
fun foo(lhs:Int, rhs:Int):Int = lhs + rhs
```

Null can not be a value of a non-null type Int

将函数 'foo' 的形参 'lhs' 类型更改为 'Int?' Alt+Shift+Enter 更多操作... Alt+Enter

Null Safety in Kotlin

- 由于在类型层面严格区分了可空和非空，java隐式的NPE也被大量消除
- 此外，可以放心大胆的设计传入不可空参数的函数，减少了很多不必要的开销

Null Safety in Kotlin

- 此外，kotlin也优化了一些场景下空类型的使用
- 比如安全的访问可空对象的成员：

```
val lenOfNullableStr : Int? = nullableStr?.length
```

- 比如结合控制流分析，在对应的代码块里自动给可空类型转型：

```
val b:String? = "Kotlin"
if(!b.isNullOrEmpty()){
    println("String of length ${b.length}")
}else{
    println("Empty String")
}
```

智能转换为 kotlin.String

val b: String? ⋮



2:EXPR ENHANCED

Java里的表达式

- 什么是表达式（expression）？
 - Oracle给的java教程里的定义： *An expression is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language, that evaluates to a single value.*
 - 但java中的表达式屈指可数 sos

Java里的表达式

- 由于java中的表达式并不多，所以有时为了给一个逻辑上不可变变量赋值的时候我们不得不把这个变量设置成可变的
- 严重干扰了编译器分析 😬
- 而且泛滥的可变变量也会严重影响性能 🐢
- 且可变的赋值比较容易出错 ✖
- 尽管Java8以后的版本加强了switch，升级成了表达式，但众所周知，咱们基本只能用java8 🙅

Kotlin中的表达式

- 鉴于此，kotlin大幅加强了表达式
- 包括且不限于：
 - 典型的控制流结构统统升级成了表达式 ✓
 - 函数可以直接=一个表达式作为返回值，且一般的类型推导可以自动推断这个表达式，免去人工标注 ✓

Kotlin中的表达式

- 这样做获得了很多显然的好处：
 - 由于很多控制流结构升级成了表达式，因此不需要把逻辑不可变的变量声明成可变的 😊
 - 实际上kotlin中不可变变量也很常用
- 明显减少了很多可变状态，对编译器还是对程序员都减轻了负担 😊



3:FUNCS BECOMES
FIRST

Java中的函数

- 必须依附一个类存在
- 由于java设计之初强调“面向对象”，低版本中高阶函数的使用并不是很容易 ☹️
 - 需要用函数式接口+匿名类实现来模拟 ☹️

Java中的函数

- 不过可喜的是，java8中加入了lambda表达式，lambda是支持函数式接口的。
- 但java8的lambda也不是完全体，只支持引用外部的final变量☹

Java中的函数式接口

- 更严重的是，由于java一开始的设计不足，函数设计者如何在函数参数处表示一个函数又成了一个大难题(=)
- 尽管官方有给一些接口，但.....(=)

函数在Kotlin中的地位

- 很明显，这样做灵活性太低，java中使用高阶函数让api的设计者还是api的使用者都很头痛🤔
- 而Kotlin大大降低了这个问题的难度，设计的时候就考虑到了函数的地位😊
 - 通过统一函数的表示，大大降低了理解的难度
 - 设计者便于设计，以及和别人设计的HOF的统一
 - 调用者便于理解
 - 并且使用高阶函数的时候传递函数也很灵活

扩展函数

- 相比java扩展一个类需要自己写Util类，kotlin对现有类的扩展也很方便。