

智能物联网+区块链_基础



Kotlin

什么是Kotlin?

- 靠他灵
- JetBrains发布基于JVM的编程语言 groovy
scala
- JetBrains是做编译器起家的公司,总部捷克,
有圣彼得堡和美国分部
- JetBrains <https://www.jetbrains.com/>
- IDEA



[ALL TOOLS](#) [IDEs](#) [.NET & VISUAL STUDIO](#) [TEAM TOOLS](#) [LANGUAGES](#) [STORE](#) [SUPPORT](#) [COMMUNITY](#) [COMPANY](#)



The drive to develop

Create Anything



IntelliJ IDEA

The Most Intelligent
Java IDE



CLion

A cross-platform IDE
for C and C++



DataGrip

IDE for Databases
and SQL



PhpStorm

Professional IDE for PHP
and Web Developers

Kotlin由来



Andrey Breslav

Kotlin之父，从2010年Kotlin编程语言项目开启之时就带领整个项目的设计和开发。时常在大型软件峰会上演讲，并且经常在Kotlin官方博客上发表。

- Kotlin 科特林岛

java

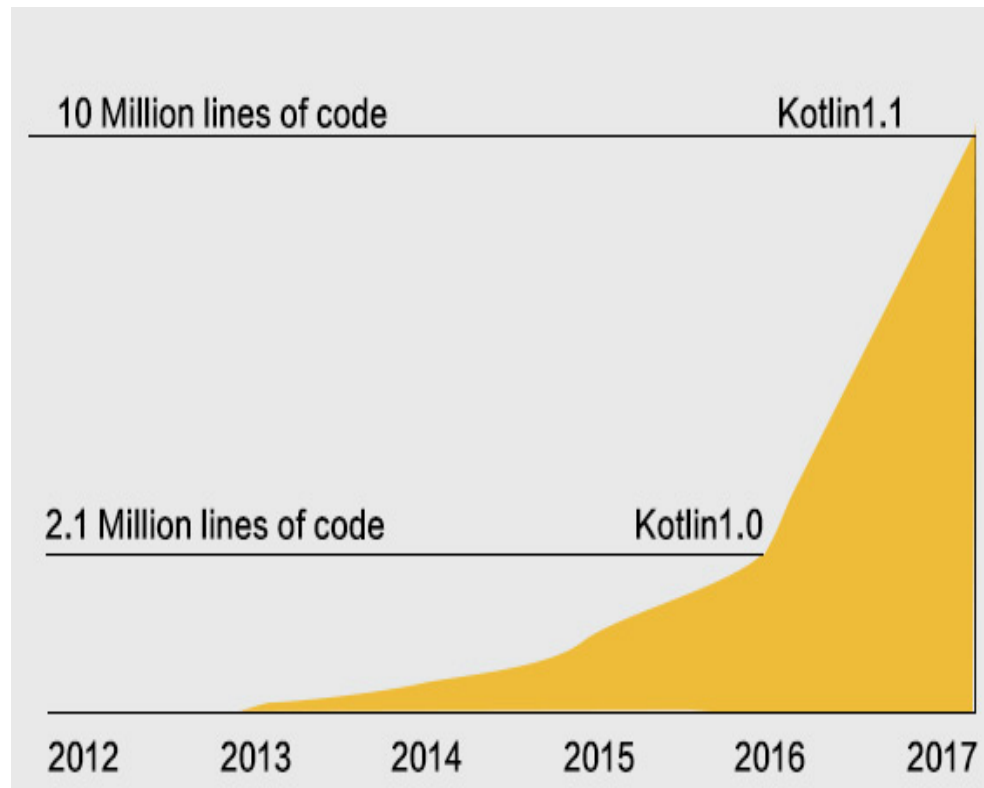
- Java 爪哇岛



Kotlin的发展

- Android第一开发语言
- Springboot2.0第一开发语言

Kotlin发展趋势



为什么使用Kotlin?

- 1.简洁(数据类 扩展方法 区间)
- 2.空值安全(针对空值处理的运算符)
- 3.100%兼容java scala
- 4.函数式编程 JDK1.8 lambda表达式
- 5.协程(thread)
- 6.DSL(领域特定语言)

Java的bean类

```
class News {  
    private String title;  
    private String desc;  
    private String content;  
  
    public String getTitle() {  
        return title;  
    }  
  
    public void setTitle(String title) {  
        this.title = title;  
    }  
  
    public String getDesc() {  
        return desc;  
    }  
  
    public void setDesc(String desc) {
```

空指针异常

```
Person person = null;  
String name = person.getName();  
System.out.println(name);
```

```
Person person = null;  
if (person != null) {  
    String name = person.getName();  
}
```

DSL

```
html {  
    head {  
        title { +"XML encoding with Kotlin" }  
    }  
    body {  
        h1 { +"XML encoding with Kotlin" }  
    }  
}
```

```
<html>  
  <head>  
    <title>  
      XML encoding...  
    </title>  
  </head>  
  <body>  
    <h1>  
      XML encoding...  
    </h1>  
  </body>  
</html>
```

Kotlin的前景?

- 1.Kotlin script(gradle) .kts
- 2.Java虚拟机应用
 - Web kotlinee
 - Javafx JDK1.8之后
- 3.前端开发 kotlinjs html+css+js
- 4.Android开发
- 5.支持开发ios oc swift
- 6.kotlin Native程序 (完全抛弃JVM虚拟机)
- 全栈工程师

参考资料

- 官方文档:
<http://kotlinlang.org/docs/reference/>
- Kotlin源码:
<http://github.com/JetBrains/kotlin>
- Kotlin官方博客:
<http://blog.jetbrains.com/kotlin>

前提

- Kotlin方向:kotlin jvm kotlin js kotlin native
- JDK下载地址

[http://www.oracle.com/technetwork/
java/javase/downloads/index.html](http://www.oracle.com/technetwork/java/javase/downloads/index.html)

选好教练车

- 学习开车要选好教练, 选好教练车
- 学习kotlin也要选好教练, 选好教练车
- 那我们选什么车上路呢, 走去4S店看看去
- Kotlin官网

<http://kotlinlang.org/>



USE
IntelliJ IDEA



USE
Eclipse



STANDALONE
Compiler



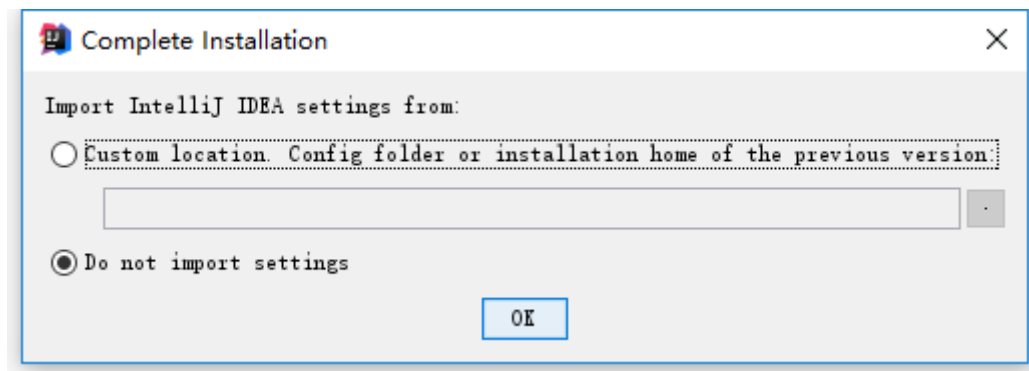
IDEA使用

- IDEA下载地址

<https://www.jetbrains.com/idea/>

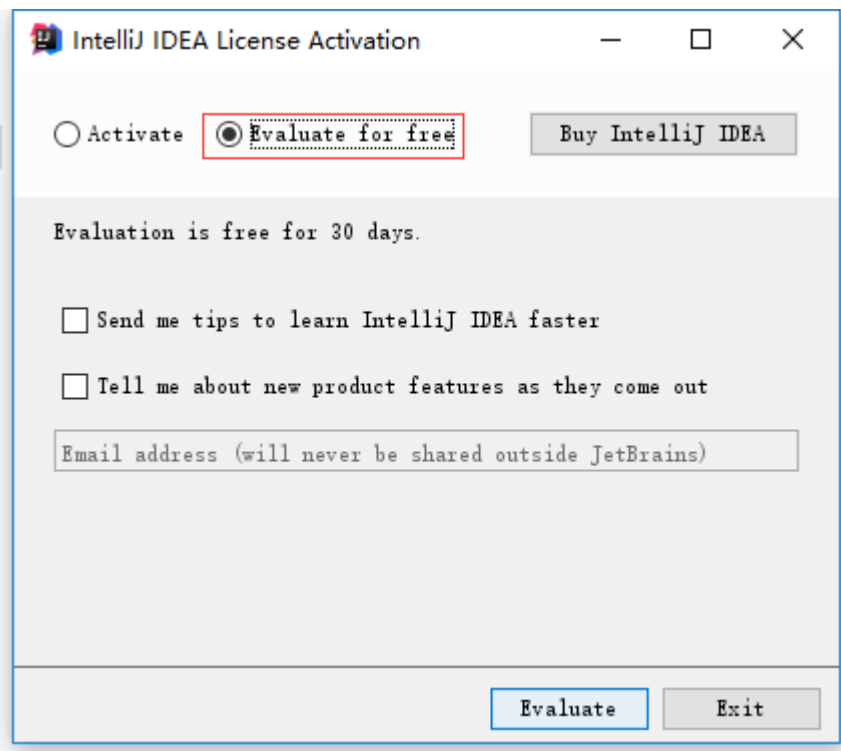
IDEA使用

- 第一步:不导入设置



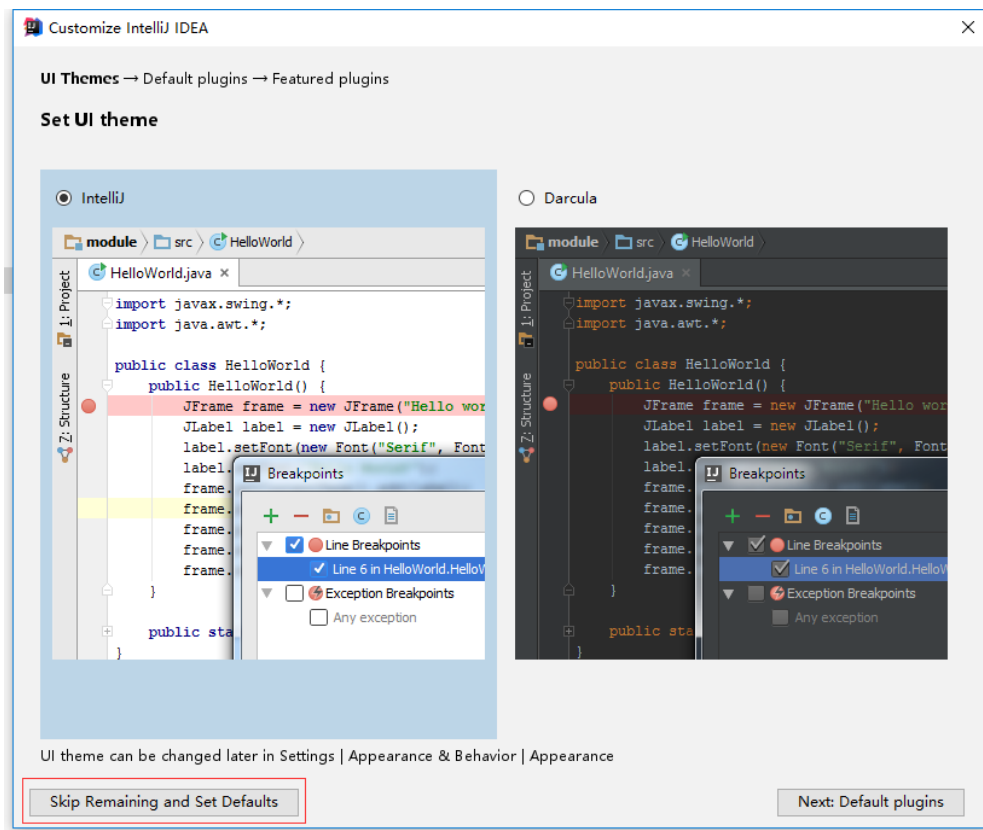
IDEA使用

- 第二步:选择免费试用



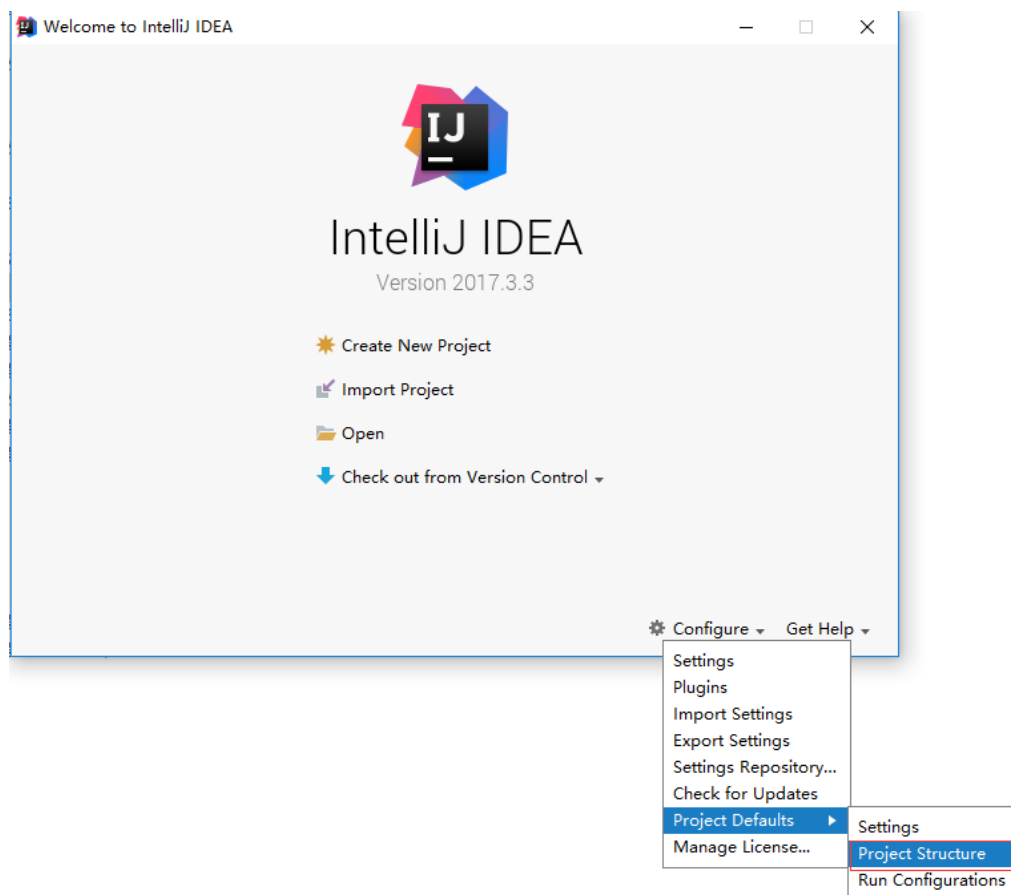
IDEA使用

- 第三步:跳过设置



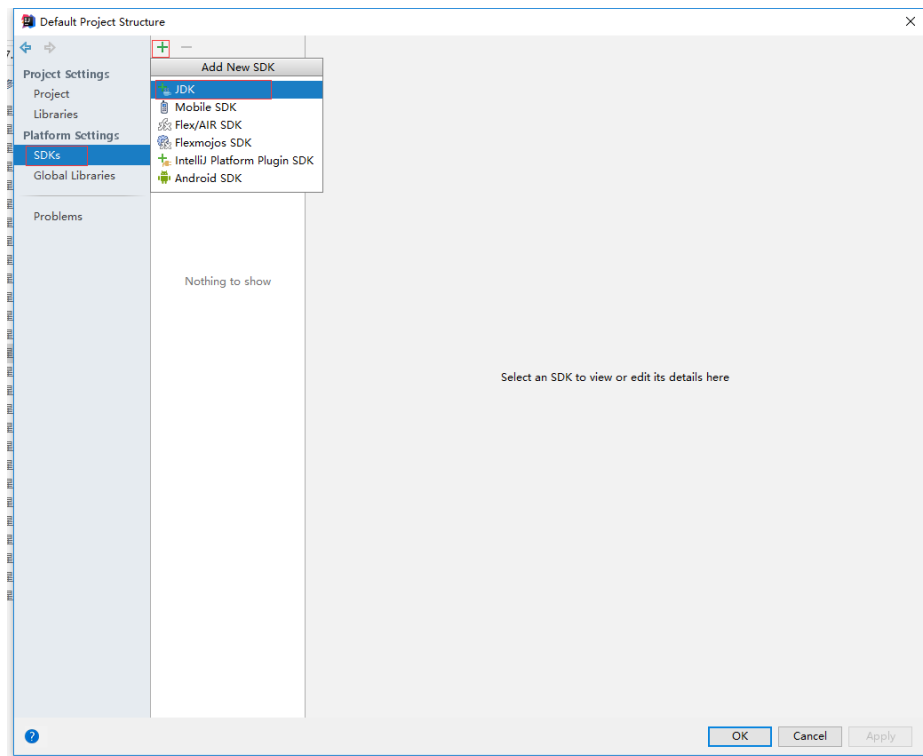
IDEA使用

- 第四步:设置SDK



IDEA使用

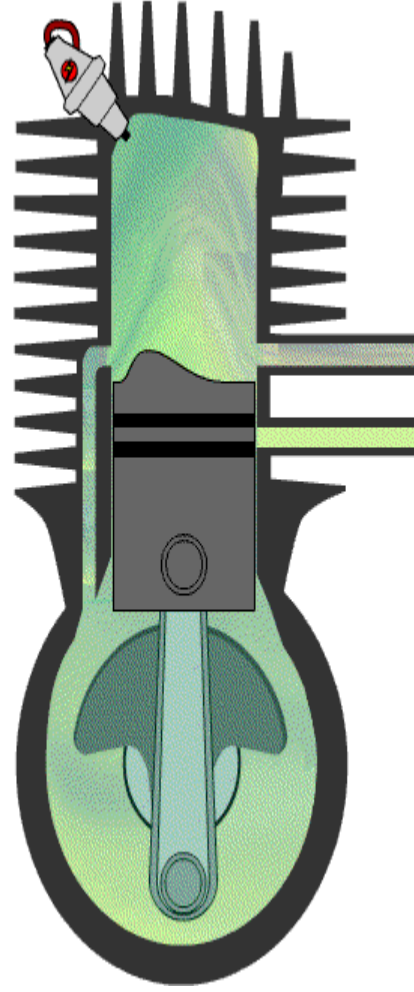
- 第五步:设置SDK

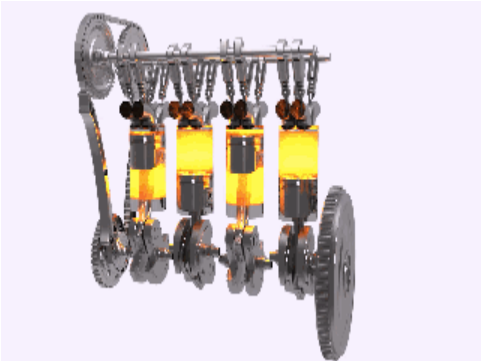


HelloWorld之前

- 学习方法

学习编程的最快方法就是, 踩油门, 走...
一直研究发动机构造, 是学不会开车的
顺便看一下发动机的原理







简单解释一下

函数的声明, 固定写法

main方法是程序的入口

接受参数名是args, 数据类型字符串

```
fun main(args:Array<String>){  
    println("hello kotlin");  
}
```

向控制台打印hello kotlin字符串

再简单解释一下

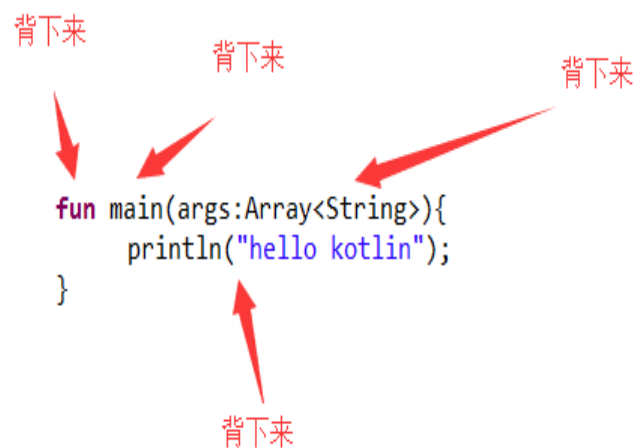
背下来

背下来

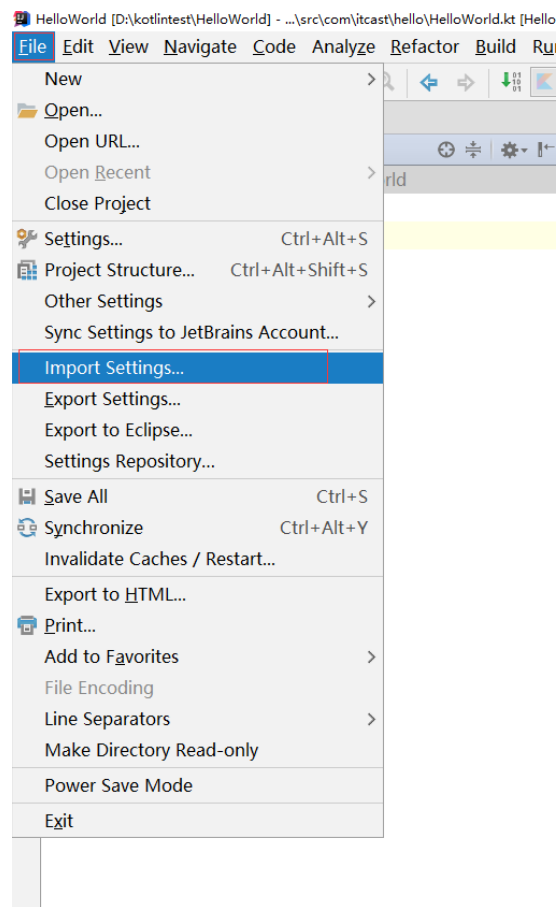
背下来

```
fun main(args:Array<String>){  
    println("hello kotlin");  
}
```

背下来



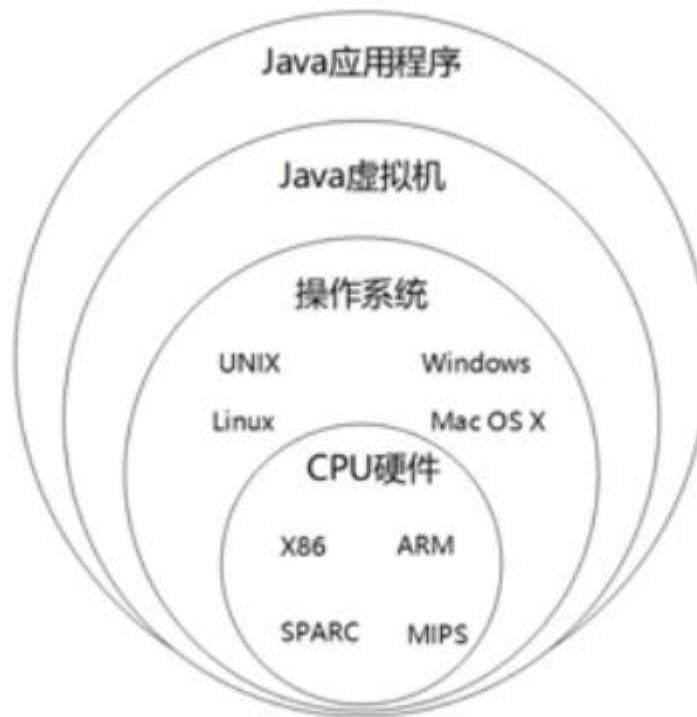
导入IDEA设置



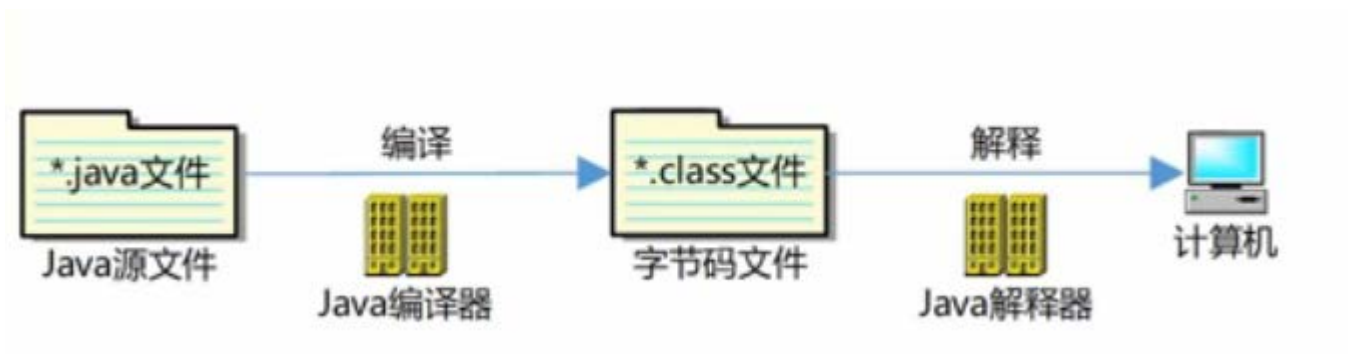
Kotlin与java虚拟机

- Kotlin使用场景:后台 android 前端js native
- 基于java虚拟机的kotlin:后台 android
- kotlinJs:前端js
- Kotlin Native:native

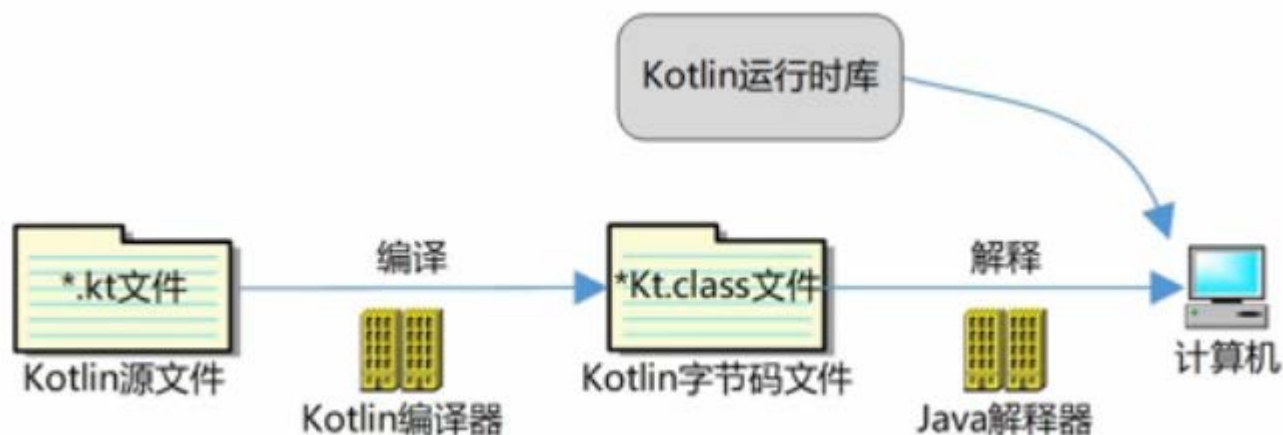
Java虚拟机



Java程序运行过程



Kotlin程序运行过程

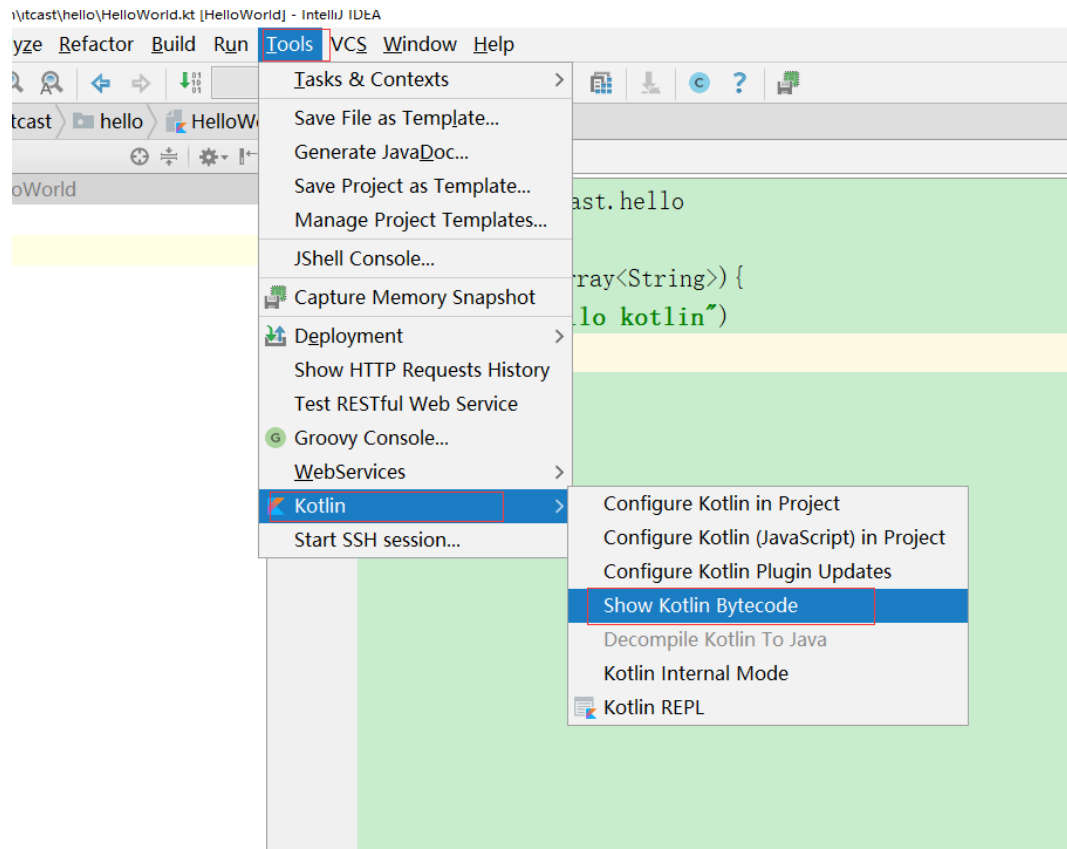


查看kotlin对应的java代码

- 第一步:找到kotlin生成字节码
- 第二步:字节码对应的java代码

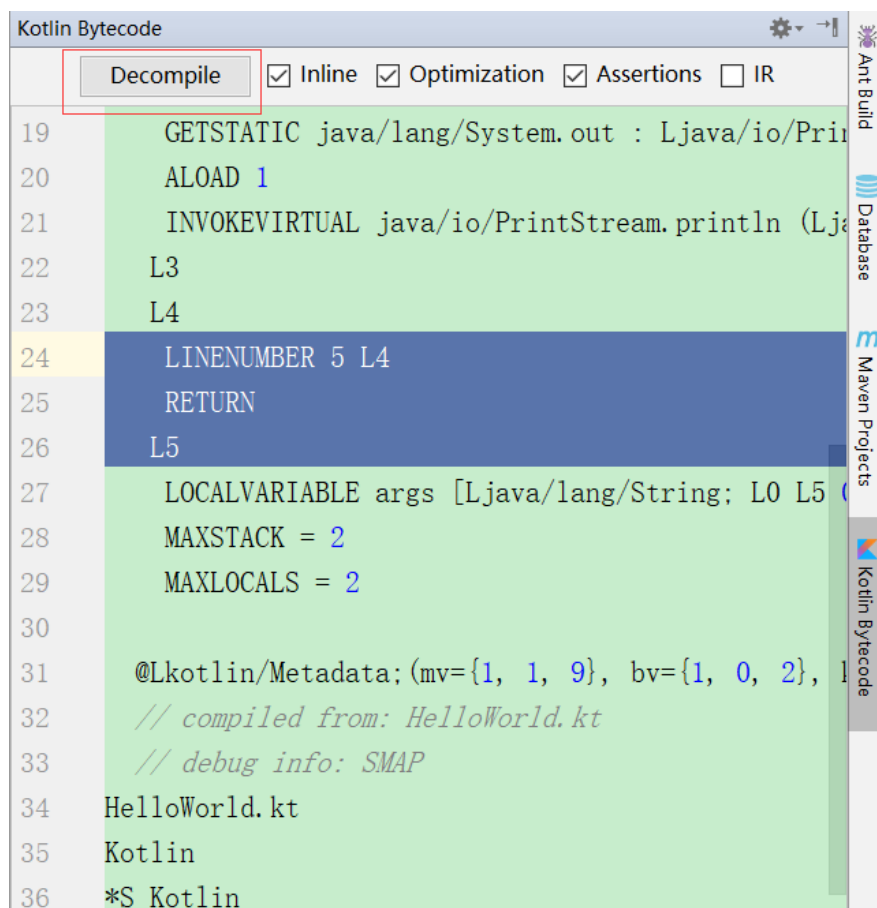
查看kotlin对应的java代码

- 第一步:Tools – Kotlin – Show Kotlin ByteCode



查看kotlin对应的java代码

- 第二步:点击Decompile按钮



```
Kotlin Bytecode
Decompile [x] Inline [x] Optimization [x] Assertions [ ] IR
19 GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
20 ALOAD 1
21 INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
22 L3
23 L4
24 LINENUMBER 5 L4
25 RETURN
26 L5
27 LOCALVARIABLE args [Ljava/lang/String; L0 L5
28 MAXSTACK = 2
29 MAXLOCALS = 2
30
31 @Lkotlin/Metadata; (mv={1, 1, 9}, bv={1, 0, 2}, 1)
32 // compiled from: HelloWorld.kt
33 // debug info: SMAP
34 HelloWorld.kt
35 Kotlin
36 *S Kotlin
```

Java基本数据类型

数据类型	占用字节	取值范围
boolean	1byte	true或false
byte	1byte	-128~127
char	2byte	
short	2byte	-32768~32767
int	4byte	-2147483648~2147483647
float	4byte	精确到小数点后6位
long	8byte	9223372036854775807~9223372036854775807
double	8byte	精确到小数点后15到16位

Kotlin基本数据类型

数据类型	占用字节	取值范围
Boolean	1byte	true或false
Byte	1byte=8bit	-128~127
Char	2byte	
Short	2byte=16bit	-32768~32767
Int	4byte	-2147483648~2147483647
Float	4byte	精确到小数点后6位
Long	8byte	9223372036854775807~9223372036854775807
Double	8byte	精确到小数点后15到16位

Java基本数据类型和包装数据类型

基本数据类型	包装数据类型
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
float	Float
long	Long
double	Double

基本数据类型取值范围

数据类型	取值范围
Boolean	true或false
Byte	-128~127
Char	
Short	-32768~32767
Int	-2147483648~2147483647
Float	精确到小数点后6位
Long	9223372036854775807~9223372036854775807
Double	精确到小数点后15到16位

计算机二进制基础

- 每个运转的事物都有适合自己的计数方式



中国发明-十进制

一 二 三 四 五 六 七 八 九 十 二十 三十 四十

1 2 3 4 5 6 7 8 9 10 20 30 40

五十 六十 七十 八十 九十 一百 二百 三百 四百 五百 六百

50 60 70 80 100 200 300 400 500 600

八百 九百 一千 二千 三千 四千 五千 八千 一萬 三萬

800 900 1000 2000 3000 4000 5000 8000 10000 30000

二进制基础

0 1 两种状态

00 01 10 11 四种状态

000 001 010 011 100 101 110 111 八种状态

0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110
1111 十六种状态

计算机存储数据用二进制存储

- 1位二进制存储2种状态
- 2位二进制存储4种状态
- 3位二进制存储8种状态
- 4位二进制存储16种状态
- ...
- N位二进制存储2的N次方种状态

Kotlin调用java

```
fun main(args: Array<String>) {  
  
    //可以省略new关键字  
    var bigDeci:BigDecimal = BigDecimal(val: "1.123456789123456789")  
    println(bigDeci)  
}
```

智能类型推断和类型转换

- 回顾一下怎么定义变量?

智能类型推断和类型转换

- 回顾一下怎么定义变量?
- 智能类型推断

```
var i = 10
```

```
var s = "100"
```

- 类型转换

```
i = s.toInt()
```


可变变量和不可变变量

- 可变变量:可以修改的
- 不可变变量:圆周率 一周7天

Val和var

- 可变量var

var a = 10;

a = 20

- 不可变量val

val b = 20 不能修改

字符串

- 一串字符:人名 地名
- 字符串定义
 - 普通字符串:"hello"
 - 原样输出字符串:""" hello"""

字符串

- 字符串删除空格`trim()`和`trimMargin()`
- 字符串比较`equals` `==` 和`===`
- 字符串切割`split`和多条件

字符串截取

/Users/yole/kotlin-book/chapter.adoc

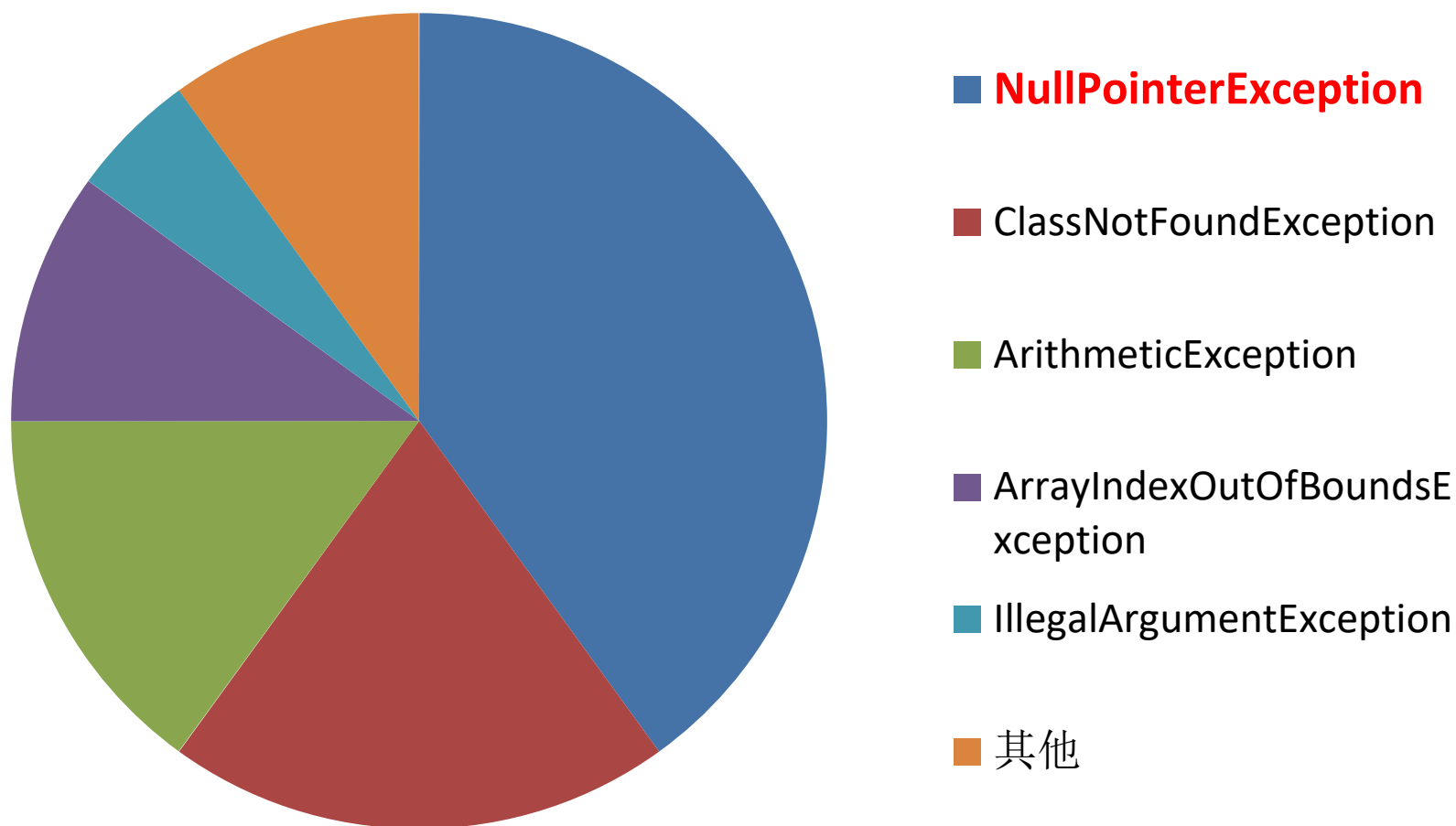
- 获取前6个字符
- 把第一个r之前的字符截取
- 把最后一个r之前的字符截取
- 把第一个r之后的字符截取
- 把最后一个r之后的字符截取

元组数据

- 给多个变量同时赋值
- 二元元组Pair
- 三元元组Triple

Kotlin空值处理

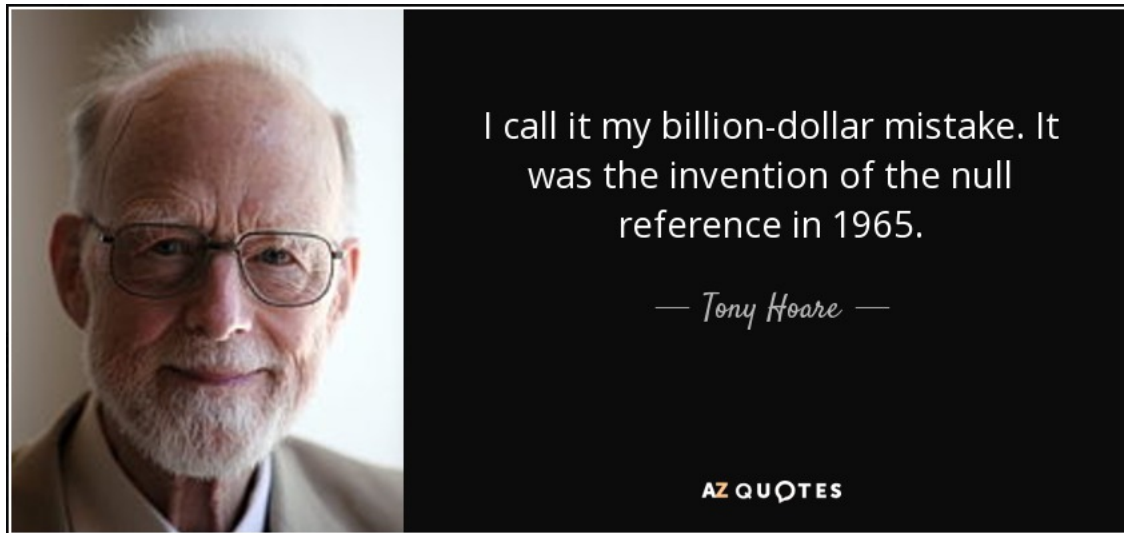
常见Exception有哪些?



根据友盟统计商业app常见Exception

来自StackOverflow 的帖子

“杯具啊！我们公司有个职工姓 Null，当用他的姓氏做查询词时，把所有员工查询应用给弄崩溃了！我该肿么办？”



托尼.霍尔

空值处理

- ?可空变量类型

val a:Int? = **null**

- !!非空断言

var age:String = **null!!**

- ?.空安全调用符

age?.toInt()

- ?:Elvis操作符(猫王)

var ageInt:Int = age?.toInt()?:10

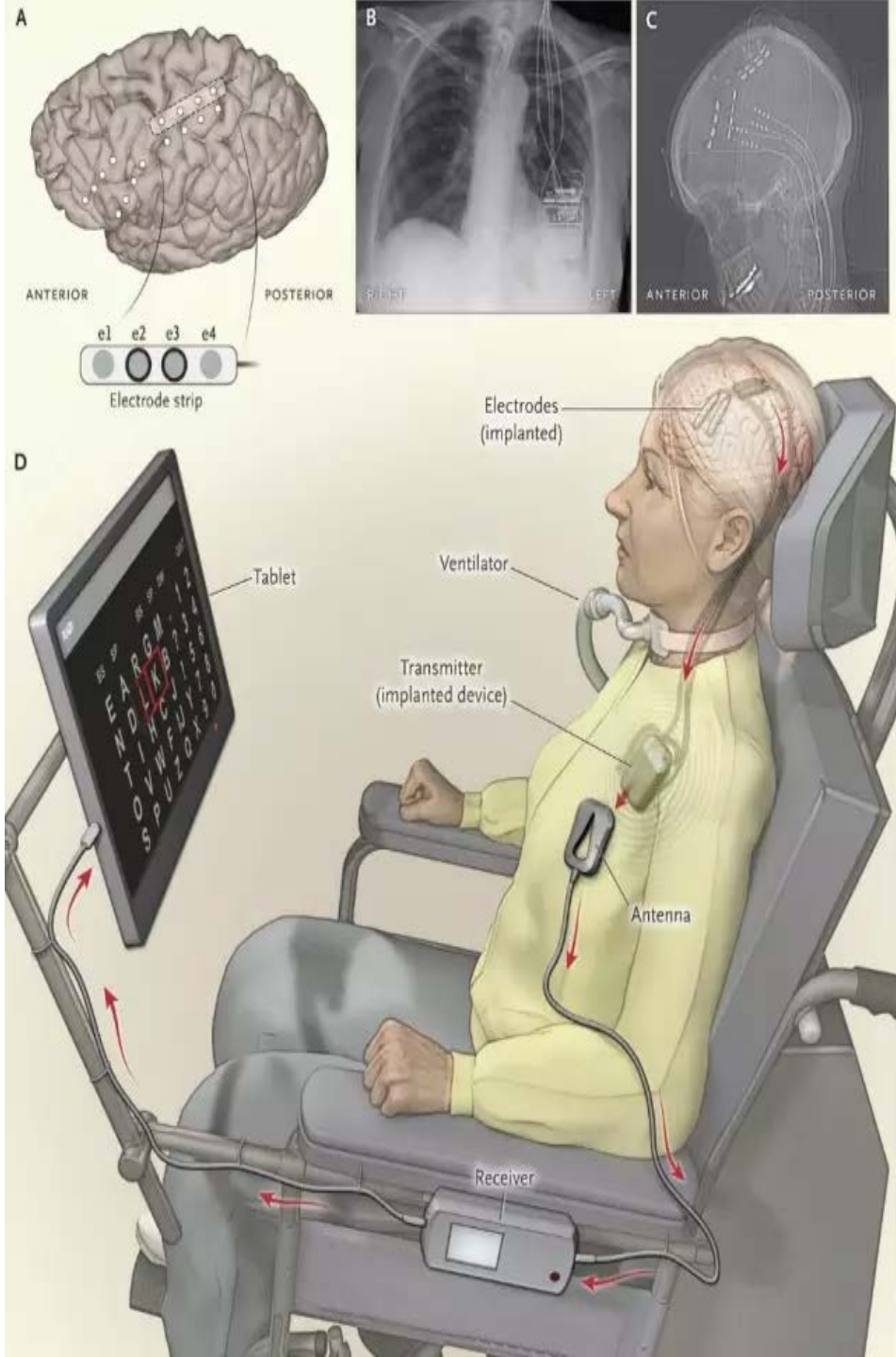


Elvis Presley

人机交互

意念交互

Brain Interface



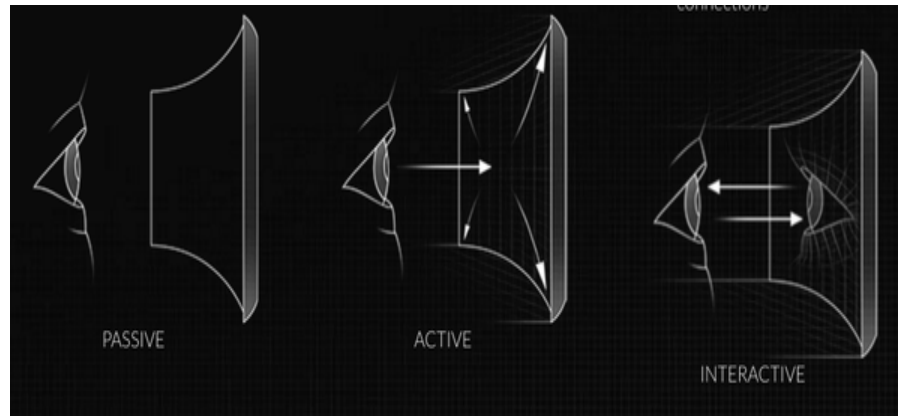
语音交互

Voice Interface



眼动跟踪

Eye Interface



体感交互

Body Interface



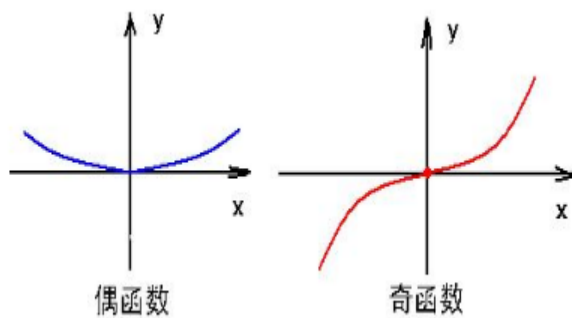
打字交互

Type Interface

- 输出函数:println()
- 输入函数:readLine()

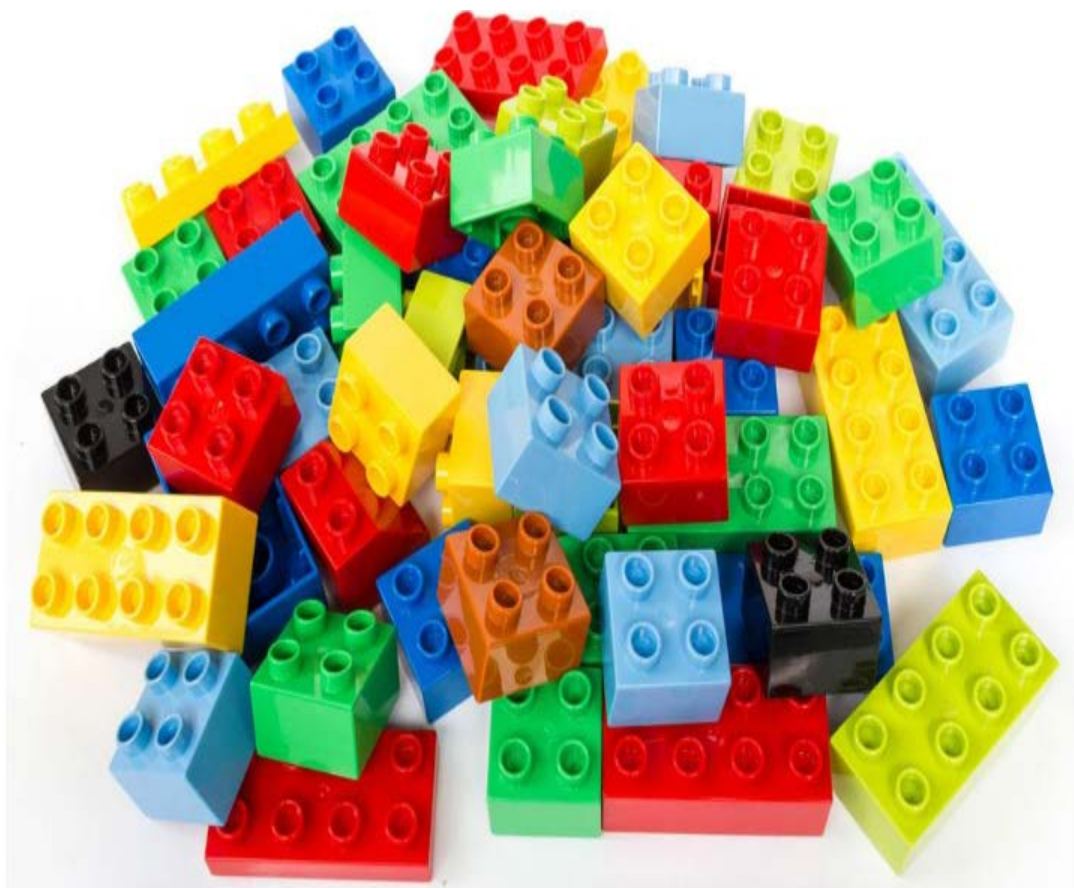
函数入门

奇函数和偶函数



程序语言中的函数

- 函数是计算机执行命令的单元





Main函数

main函数是kotlin程序的入口函数,
他是计算机运行起来第一个默认找的第一个
运行的函数.

```
fun main(args:Array<String>){  
  
}
```

输入函数和输出函数

四种函数

- 无参无返回值

```
fun myFun1() {  
    println("myFun1")  
}
```

- 无参有返回值

```
fun myFun2(): String {  
    return "myFun2"  
}
```

- 有参无返回值

```
fun myFun3(content: String) {  
    println(content)  
}
```

- 有参有返回值

```
fun myFun4(content: String): String {  
    return content  
}
```

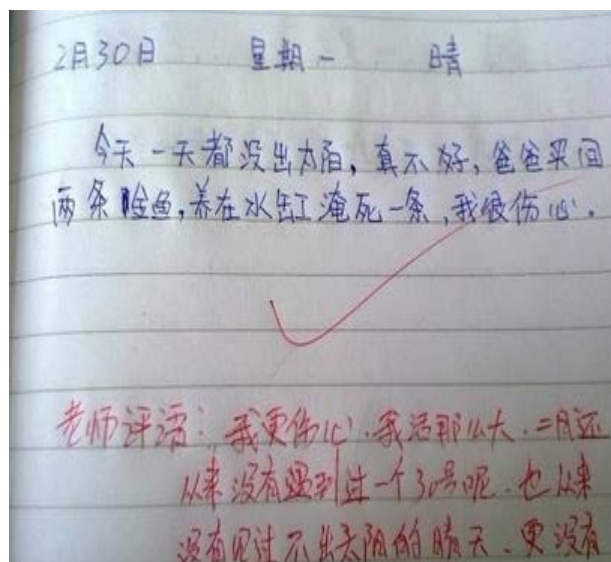
顶层函数

- Java里面函数都需要依赖于对象存在
- 不依赖于class类单独存在的函数

嵌套函数

- 函数里面定义的函数

字符串模板



今天天气晴朗，万里无云，我们去中山公园游玩，
首先映入眼帘的是，中山公园4个镏金大字。

今天天气晴朗，万里无云，我们去人民大会堂游玩，
首先映入眼帘的是，人民大会堂5个镏金大字。

今天天气晴朗，万里无云，我们去金銮殿游玩，
首先映入眼帘的是，金銮殿3个镏金大字。

今天天气晴朗，万里无云，我们去鸟巢游玩，
首先映入眼帘的是，鸟巢2个镏金大字。

条件控制语句if

- 如果...那么...
- 如果今天是周日,就可以休息了

- 普通的if/else语句

```
if(a>b){  
    return a-b  
}else{  
    return b-a  
}
```

- 去{}的if/else

```
if(a>b)  
    return a-b  
else  
    return b-a
```

条件控制语句if

- 精简if/else

if(a>b) return a-b else return b-a

- If语句返回值

return if (a > b) a - b else b - a

Statement& Expression

- Java声明式语法
- 声明 no return value
- Kotlin表达式语法
- 表达式 return value

for循环和foreach循环

- 字符串遍历 数组遍历 集合遍历

- 普通的for循环

```
for (c in s ) {  
    println("c=$c")  
}
```

- 加上index角标的for循环

```
for ((index,c) in s.withIndex()) {  
    println("index=$index c=$c")  
}
```

Foreach循环

- 普通foreach

```
s.forEach {  
    println("it=$it")  
}
```

- 加上index角标的foreach

```
s.forEachIndexed { index, c ->  
    println("$c 的角标是$index")  
}
```

Continue和break

- Continue跳出这次循环,后面还会执行

```
var newstr = "abcde"
```

```
for (c in newstr) {  
    if(c=='c') continue  
    println(c)  
}
```

输出 a b d e,不会输出c

- Break跳出循环,循环停止

```
for (c in newstr) {  
    if(c=='c') break  
    println(c)  
}
```

只输出a b 后面都不会输出

标签处返回

- 多重循环
- abc和123所有的组合

```
loop@ for (i:Char in s1) {  
    for (j:Char in s2) {  
        if (i == 'b' && j == 'f') {  
            //break到指定标签  
            break@loop  
        }  
        println("i = $i, j = $j")  
    }  
}
```

while和do while

- 打印小于100的整数
- While循环(满足条件才会执行)

```
while (x > 0){  
    println("x=$x")  
    x—  
}
```
- do while循环(不满足条件也会执行一次)

```
do {  
    println("x=$x")  
    x--  
}while (x > 0)
```


区间Range

- 任何程序的本质都是对数据的处理

区间Range

- 区间:一段实数范围 1到100
- 开区间 $(1,100)$ 不包含1和100
- 闭区间 $[1,100]$ 包含1和100
- 半开半闭区间 $[1,100)$ 包含1不包含100

Kotlin 区间

- Kotlin 区间: 1到100 'a'到'z'

区间Range

A	12345678	↑ ☆ A B C
A郑	12345678	D E F
BA郑	12345678	G H I
BA郑	12345678	J K L
陈	12345678	M N O
C	12345678	P Q R
蔡	12345678	S T U
弟	12345678	V W X
弟	12345678	Y Z #

Kotlin 区间

- 定义数据类型保存1到100的数据

区间Range

- 常见区间: IntRange CharRange LongRange

- 区间创建

IntRange(1,10)

1..100 [1,100]

1 *until* 100 [1,100)

区间遍历

- 区间遍历:for foreach

反向区间和区间的反转

- 反向区间: `10 downTo 5`
- 区间反转: `IntRange.reversed()`

数组

- 数组对于任何一门编程语言来说都是重要的数据结构之一
- 数组是相同数据类型元素的集合
- 张三 李四 王五
- 10 20 30
- 'a' 'b' 'c'
- 思考:“张三” 10 'a' 能用数组保存吗?

数组的定义

- 创建数组:`arrayOf("haha",10,20)`

- 创建基本数据类型数组

`intArrayOf(10, 20, 30)`

`booleanArrayOf(true,false,true)`

`byteArrayOf(10,20)`

...注意:没有stringArrayof

- 基本数据类型数组类型

val intArr = IntArray(10) //定义元素个数为10个的Int数组

val charArray = CharArray(10)

- 定义几根数据类型数组并初始化

val intArr = IntArray(10){

0

} //定义了一个长度为10的Int数据类型并且所有元素都为0

数组遍历

- for循环遍历

```
val array = arrayOf("张三","李四","王五")  
for (s in array) {  
    println(s)  
}
```

- for循环遍历元素以及角标

```
for ((index,s) in array.withIndex()) {  
    println("角标$index s=$s")  
}
```

数组遍历

- foreach高级for循环

```
array.forEach {  
    println(it)  
}
```

- foreach遍历元素及角标

```
array.forEachIndexed { index, s ->  
    println("角标=$index 元素=$s")  
}
```

数组元素修改

```
var newArr = arrayOf(1,2,3,4,2,3,5)
```

- 修改某一个位置上的元素

```
newArr[0] = 20 //将角标0元素值设置为20
```

```
newArr.set(4,20)//将角标4元素值设置为20
```

查找数组元素角标

`arrayOf("张三","李四","张四","王五","张三","赵六")`

- 查找第一个“张三”角标
- 查找最后一个“张三”角标
- 查找第一个姓“张”的人的角标
- 查找最后一个姓“张”的人的角标

When表达式

- `when`表示式是多分支的条件控制语句

Switch回顾

```
public void level(char grade) {  
    switch (grade) {  
        case 'A':  
            System.out.println("优秀");  
            break;  
        case 'B':  
        case 'C':  
            System.out.println("良好");  
            break;  
        case 'D':  
            System.out.println("你需要再努力努力");  
            break;  
        default:  
            System.out.println("未知等级");  
    }  
}
```


When表达式

- 最基本when表达式

```
when (age) {  
    7 -> return "开始读小学"  
    12 -> {  
        println("开始读中学")  
        return "开始读小学"  
    }  
    else -> return "读社会大学"  
}
```

when表达式加强

```
when(ele){  
    10-> println("它10岁")  
    is Int-> println("传递的是年龄")  
    in 1..10-> println("在10岁以内")  
    !in 1..10-> println("不在10岁以内")  
    else-> println("所有情况都不满足")  
}
```

when表达式不带参数

```
when{  
    ele == 10-> return "它十岁"  
    ele is Int-> return "传递的是年龄"  
    ele in 1..10->return "在10岁以内"  
    ele !in 1..10-> return "不在10岁以内"  
    else-> return "所有情况都不满足"  
}
```

When表达式返回值

```
val s = when {  
    ele == 10 -> "它十岁"  
    ele is Int -> "传递的是年龄"  
    ele in 1..10 -> "在10岁以内"  
    ele !in 1..10 -> "不在10岁以内"  
    else -> "所有情况都不满足"  
}  
return s
```

作业:打印菱形

*

*

作业

- 请编写函数, 函数名为sayHello 方法返回值是String类型, 一个参数是String类型 名称为name
- 请编写函数, 函数名为checkAge 方法的返回值是Boolean类型, 一个参数是整数类型 名称为age
- 请编写函数, 函数名为saveLog 方法没有返回值 一个参数是整数类型 名称为logLevel

作业

- 判断登录用户名是否合法(不能用正则表达式)
- 用户名是数字字母或者_,需要在3到20位,必须有2个或以上大写字母,2个或以上小写字母,3个或以上数字

hhew2383dW_fkf&E@^