

效率的抉择：用 Kotlin 做Android开发

Bennyhuo

个人简介

- 霍丙乾 , Bennyhuo ,
- 就职于腾讯地图
- Github: <https://github.com/enbandari>
- Kotlin 微信公众号
- Kotlin 社区 : <https://kotliner.cn>
- Kotlin 博客 : <https://blog.kotliner.cn>

Kotlin 简介

Kotlin 的基本情况



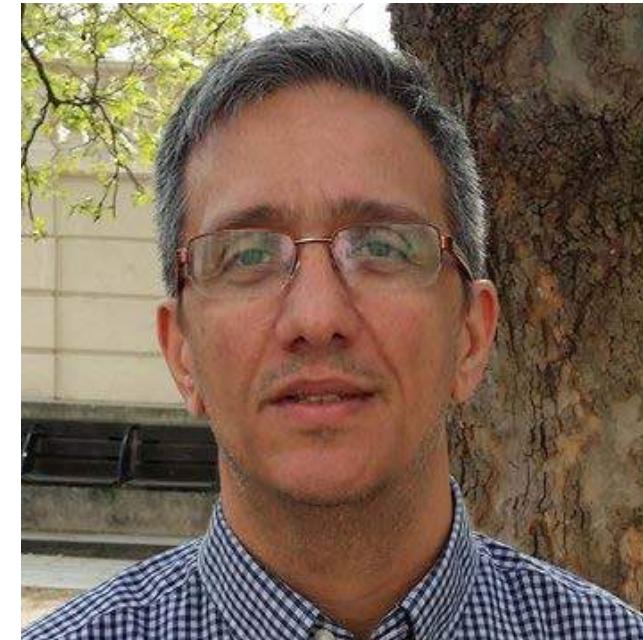
Kotlin 的基本情况



Kotlin 的基本情况



[Andrey Breslav](#)

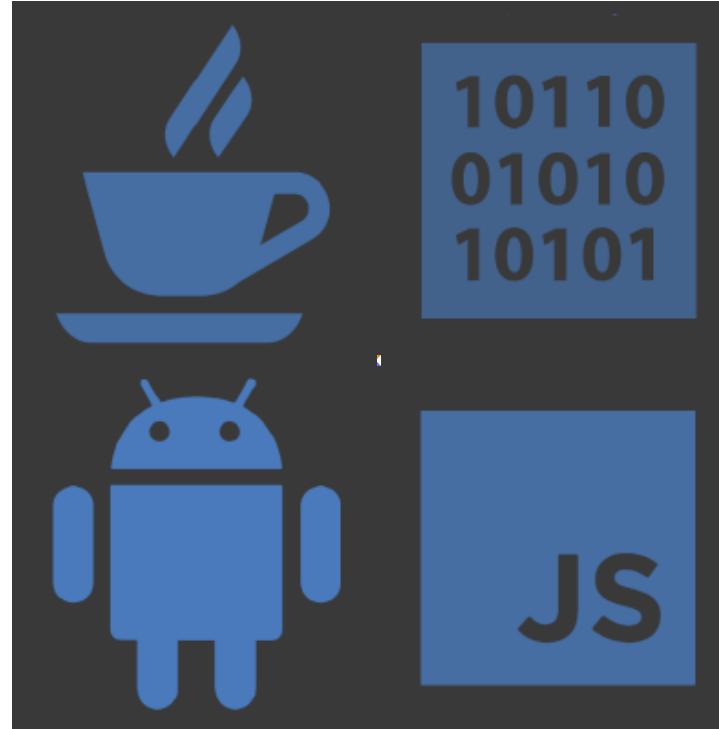


[Hadi Hariri](#)

Kotlin 的基本情况



Kotlin 的基本情况



Kotlin 的基本情况



100% 兼容 Java

Kotlin 的基本情况



Kotlin 的基本情况



Kotlin 的基本情况



Android 官方开发语言

Kotlin 的基本情况



Kotlin 的基本情况



Kotlin 的基本情况



正式支持 JavaScript Target

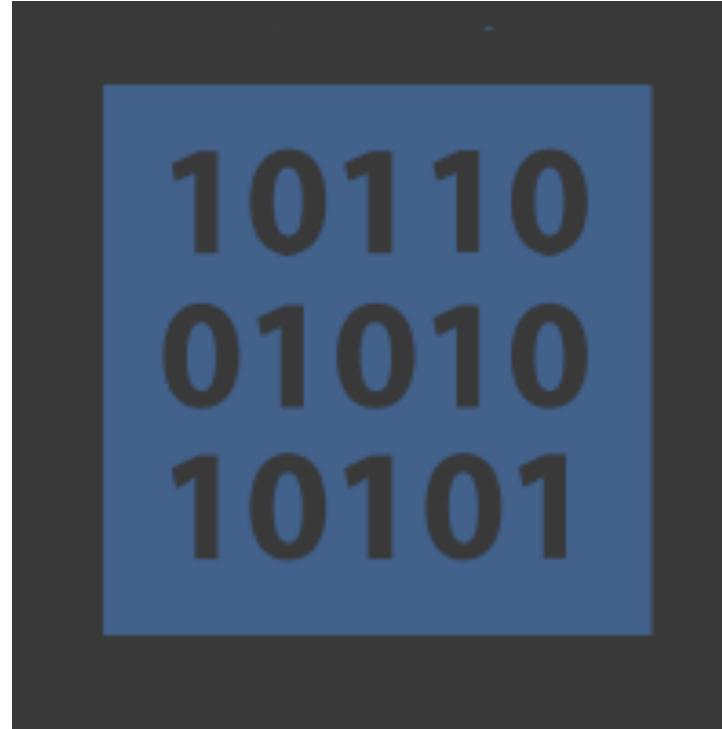
Kotlin 的基本情况



Kotlin 的基本情况



Kotlin 的基本情况

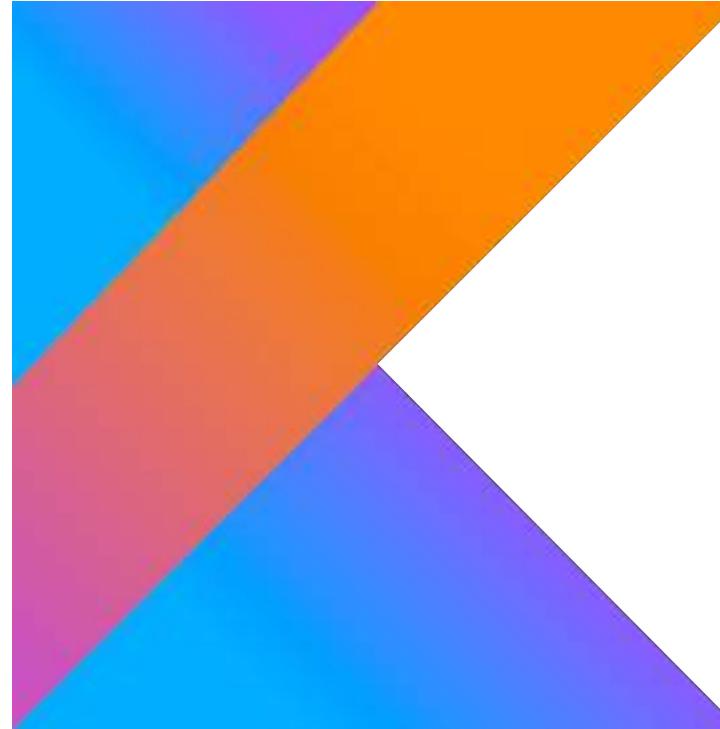


Native Target 预览版 0.4

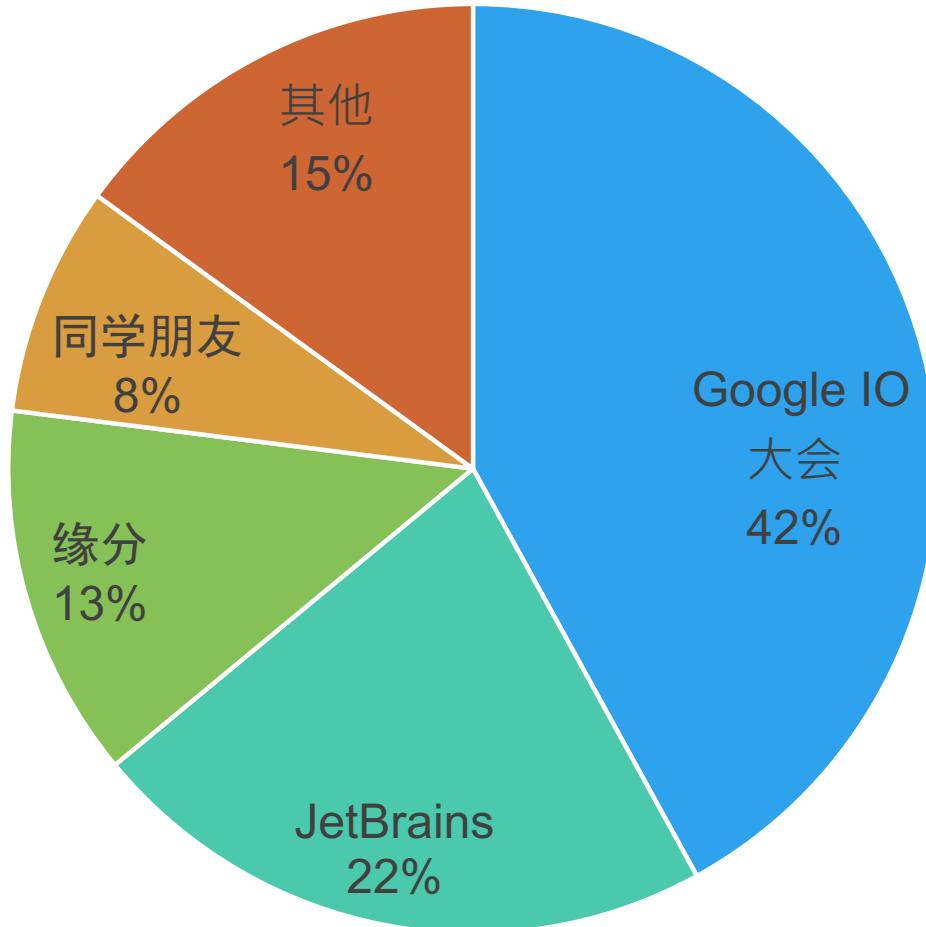
Kotlin 的基本情况



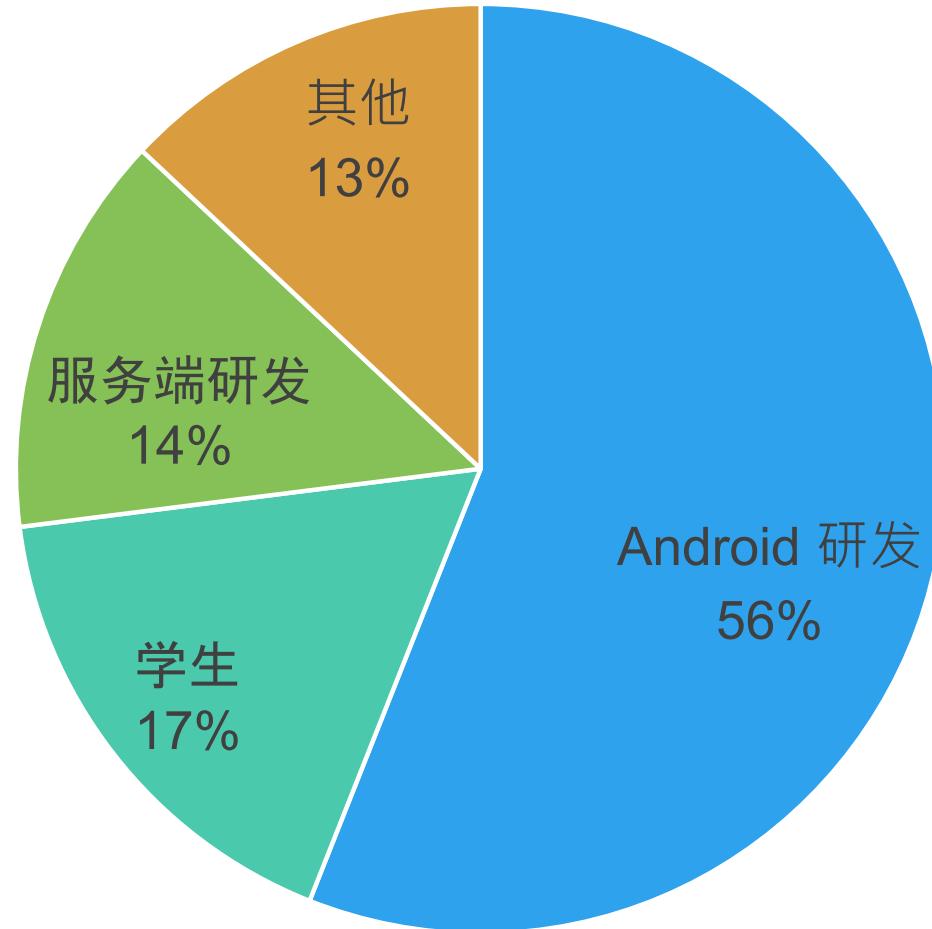
Kotlin 的基本情况



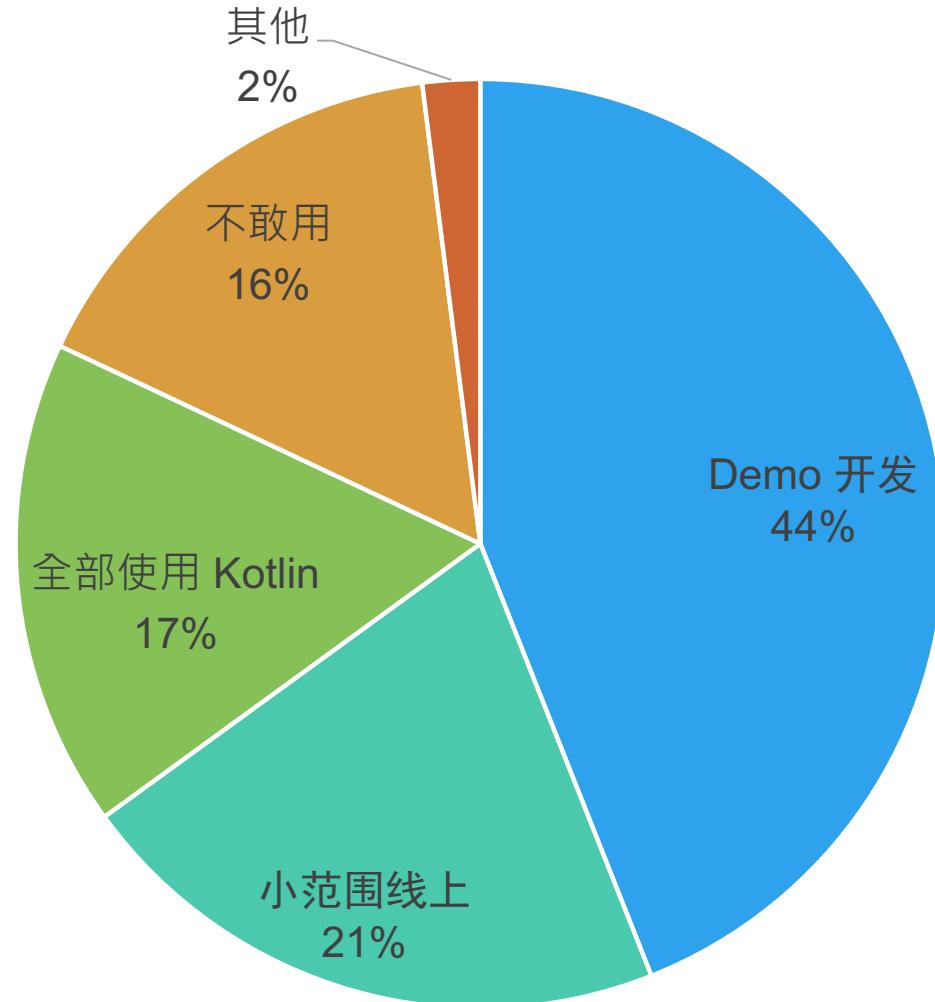
你是如何认识 Kotlin 的？



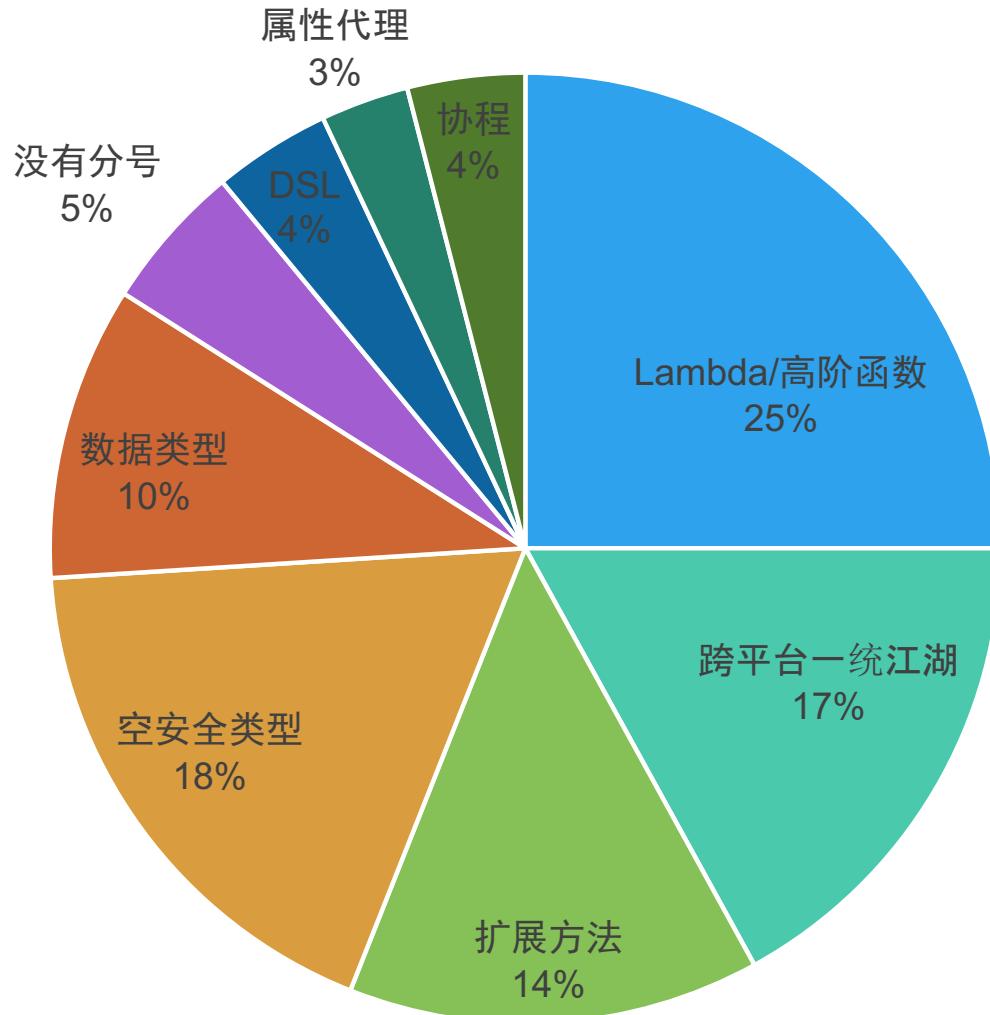
你从事何种职业？



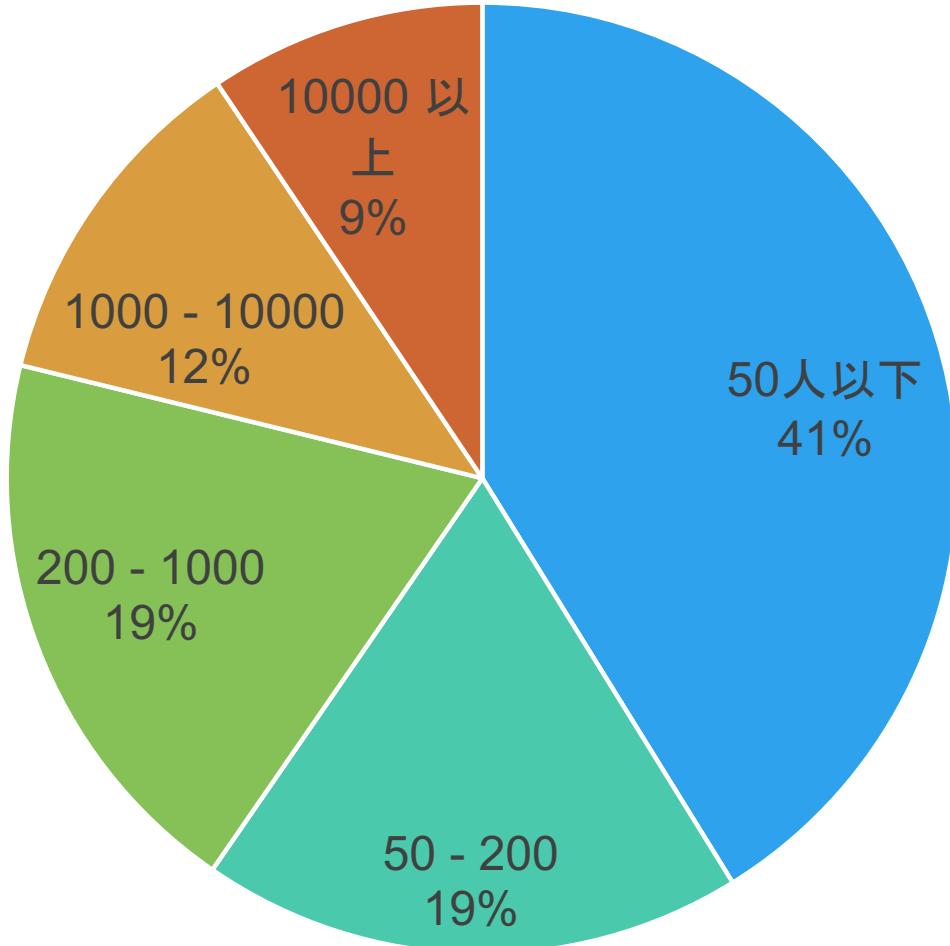
你或者你所在的项目如何使用 Kotlin 的？



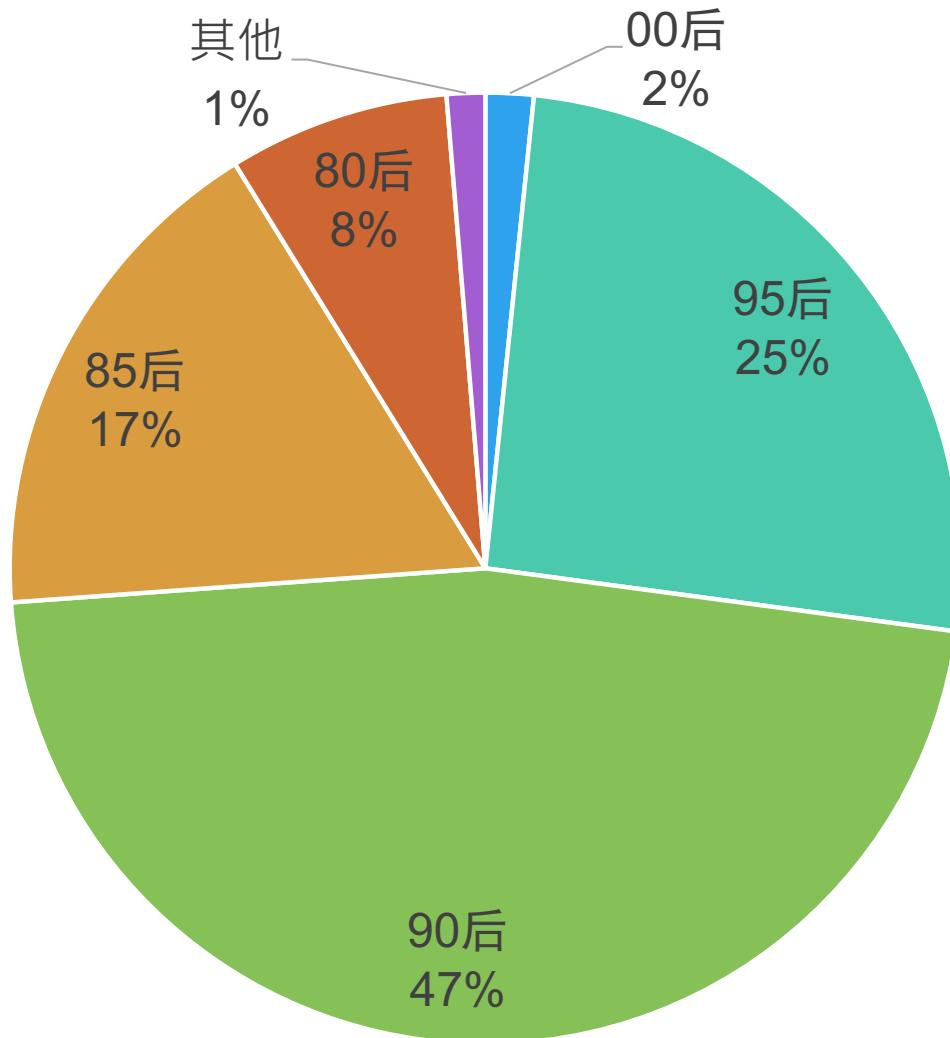
你觉得 Kotlin 最吸引你的特性是什么？

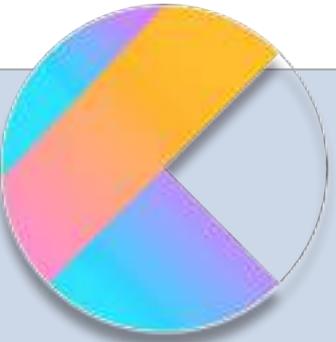


你所在的公司人数？



你的年龄范围？





Kotlin

高端大气上档次



Gender

7

Age



Tiger



Scorpio

11 18

Birthday

年轻

有实力

有潜力

爸爸有钱

宗里有地

相关资料

Android developer

- 地址：<https://developer.android.com/kotlin/index.html>

Kotlin and Android

Kotlin is now an official language on Android. It's expressive, concise, and powerful. Best of all, it's interoperable with our existing Android languages and runtime.

GET STARTED



Modern. Expressive. Safe.

Kotlin is concise while being expressive. It contains safety features for nullability and immutability, to make your Android apps healthy and performant by default.

Kotlin 官网

- 地址：<http://kotlinlang.org/>



Kotlin 官网（中文版）

- 地址：<https://www.kotlincn.net/>



Kotlin 论坛

- 地址：<https://discuss.kotlinlang.org/>

The screenshot shows the homepage of the Kotlin Discussions forum. At the top, there is a dark blue header bar with the "Kotlin" logo on the left. Below the header, there is a navigation bar with several tabs: "all categories" (disabled), "Latest" (highlighted in red), "New (4)", "Unread (4)", "Top", and "Categories".

The main content area displays a list of recent topics. Each topic is represented by a row with three columns: "Topic", "Category", and "Users".

- Welcome to the Kotlin Discussions powered by Discourse** (Category: General, User icons: B, C, D)
- When with "less than"** (Category: General, User icons: A, B, C, D, E)
- Kapt "IllegalStateException: failed to analyze" when encountering a large amount of errors** (Category: Android, User icons: A, B, C, D, E)
- Companion Functions • new** (Category: Language Design, User icons: A, B, C, D)

Kotlin 论坛（中文版）

- 地址：<https://kotliner.cn/>

The screenshot shows the homepage of the Kotlin CHINA forum. At the top, there is a navigation bar with links for 'Kotlin CHINA' (highlighted in blue), '论坛' (Forum), 'Wiki', '博客' (Blog), and '中文站' (Chinese Station). On the right side of the nav bar are '注册' (Register) and '登录' (Login) buttons. A large blue banner in the center says 'KOTLIN CHINA 上线了!' (Kotlin CHINA is live!). Below the banner are four cards: '【视频】Kotlin 入门到进阶' (Video: Kotlin from Beginner to Advanced) with a puzzle piece icon, 'Kotlin for Android Dev' with an airplane icon, '社区博客' (Community Blog) with a speech bubble icon, and '中文站' (Chinese Station) with a globe icon. Below these cards is a navigation bar with tabs: '精品' (Premium), '最新发布' (Latest Release), '问答' (Questions & Answers), '原创' (Original), '转载' (Reposting), '站务' (Station Affairs), and '翻译' (Translation). A blue button labeled '发布新话题' (Post New Topic) is located on the right. A post by 'admin' is shown, dated 4 months ago. The post title is '【专题】Kotlin for Android Developer' and it includes a '转载' (Reposting) button and a '#Kotlin-for-Android-Dev' tag. The post has 0 comments and was last updated 4 months ago by 'admin'. To the right, there is a '网站通告' (Website Announcement) section with a box containing 'Kotlin-CN'.

Kotlin 博客

- 地址：<https://blog.jetbrains.com/kotlin>

The screenshot shows the header of the Kotlin Blog. It features the Kotlin logo (a stylized 'K') and the text "KOTLIN BLOG". On the right side of the header, there are links for "CONFERENCE", "KOTLIN HOME PAGE", "TWITTER", and "SLACK". Below the header, the main content area has a large title "Kotlin 1.1.60 is out". To the right of the title is a search bar with the placeholder "Search". Below the title, a post by Dmitry Jemerov is listed with the date "Posted on November 13, 2017". The post announces the release of Kotlin 1.1.60, highlighting bugfixes and tooling updates. A bulleted list details these changes, including experimental support for Kotlin/JS incremental compilation, new JSR-305 annotations support, bug fixes in Parcelable implementation generation, Gradle incremental builds, and new inspections in the IntelliJ plugin. At the bottom of the post, it states compatibility with IntelliJ IDEA 2016.3-2017.3 and Android Studio 2.3-3.1 Canary. To the right of the post, there is a sidebar titled "Tweets about kotlin" showing three tweets from users @yoshihi, @DoRuby, and @yoshihi, along with their profile pictures and tweet text.

Kotlin 1.1.60 is out

Posted on November 13, 2017 by Dmitry Jemerov

We're happy to announce the release of Kotlin 1.1.60, a new bugfix and tooling update for Kotlin 1.1. This update:

- Adds experimental support for Kotlin/JS incremental compilation
- Adds new features to JSR-305 custom nullability annotations support
- Brings a lot of bug fixes in the automatic `Parcelable` implementation generator and provides it with IDE support
- Improves Gradle incremental builds
- Introduces new inspections, performance improvements and bug fixes in the IntelliJ plugin

The update is compatible with all versions of IntelliJ IDEA from 2016.3 until 2017.3, as well as with Android Studio 2.3, 3.0 and 3.1 Canary.

CONFERENCE KOTLIN HOME PAGE TWITTER SLACK

Tweets about kotlin

Yoshihi (@yoshihi) "RealmでKotlinのモデルクラスを扱うときの注意点！スマホ開発！DoRuby" tinyurl.com/DomzdbKu3X

DoRuby (@DoRuby) "RealmでKotlinのモデルクラスを扱うときの注意点！スマホ開発！DoRuby" tinyurl.com/DomzdbKu3X

Yoshihi (@yoshihi) "Kotlin便利なプラグインが2つ登

Kotlin 博客（中文版）

- 地址：<https://blog.kotliner.cn/>

2017-09-19 · 编程语言

Kotlin 的 val list: ArrayList<String>= ArrayList() 居然报错！

1 语法错误？也许看了我们的题目，大家还没有明白过来到底发生了什么，那么我请大家再仔细看看：12val list: ArrayList<String>= ArrayList()
什么地方报错呢？就是泛型参数后面的 > 处。这就让人不理解了，看上去并没有什么问题啊。我们再来看看错误提示：...

[阅读全文...](#)

Kotlin 公众号

- 微信公众号 : Kotlin



Kotlin & Geek

发消息

正确地使用 Kotlin 的 internal

2017年11月13日



11月, Kotlin 之月

2017年11月6日 (周四)

简单说说 Android Studio3.0的更新

2017年10月30日 (周三)



说说你和 Kotlin 的故事

2017年10月21日 (周四)

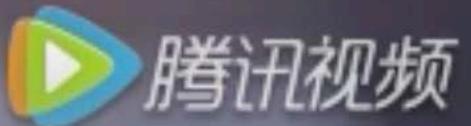


Kotlin 教学视频

- O'Reilly **Introduction to Kotlin Programming** : <http://hadihariri.com/2016/11/01/oreilly-kotlin-course/>
- O'Reilly **Advanced Kotlin Programming** : <http://hadihariri.com/2016/11/01/oreilly-kotlin-course/>
- Github **Kotlin 入门到“放弃”** : <https://github.com/enbandari/Kotlin-Tutorials>
- 慕课网 **Kotlin 入门到进阶** : <http://coding.imooc.com/class/108.html>

培训目标

- 认识 Kotlin
- 了解 Kotlin 的基本语法
- 熟悉 Kotlin 特性的一些应用场景
- 最后把培训的内容都忘了



不坏不坏 忘得真快呀

下面开始讲代码

Kotlin: So You Don't
Need A Billion Lines
Of Code To Get
Your Shit Done!

in: So You Don't
Need A Billion Lines
Of Code To Get
Your Shit Done!

KotlinConf

“

简洁，就要少写废话

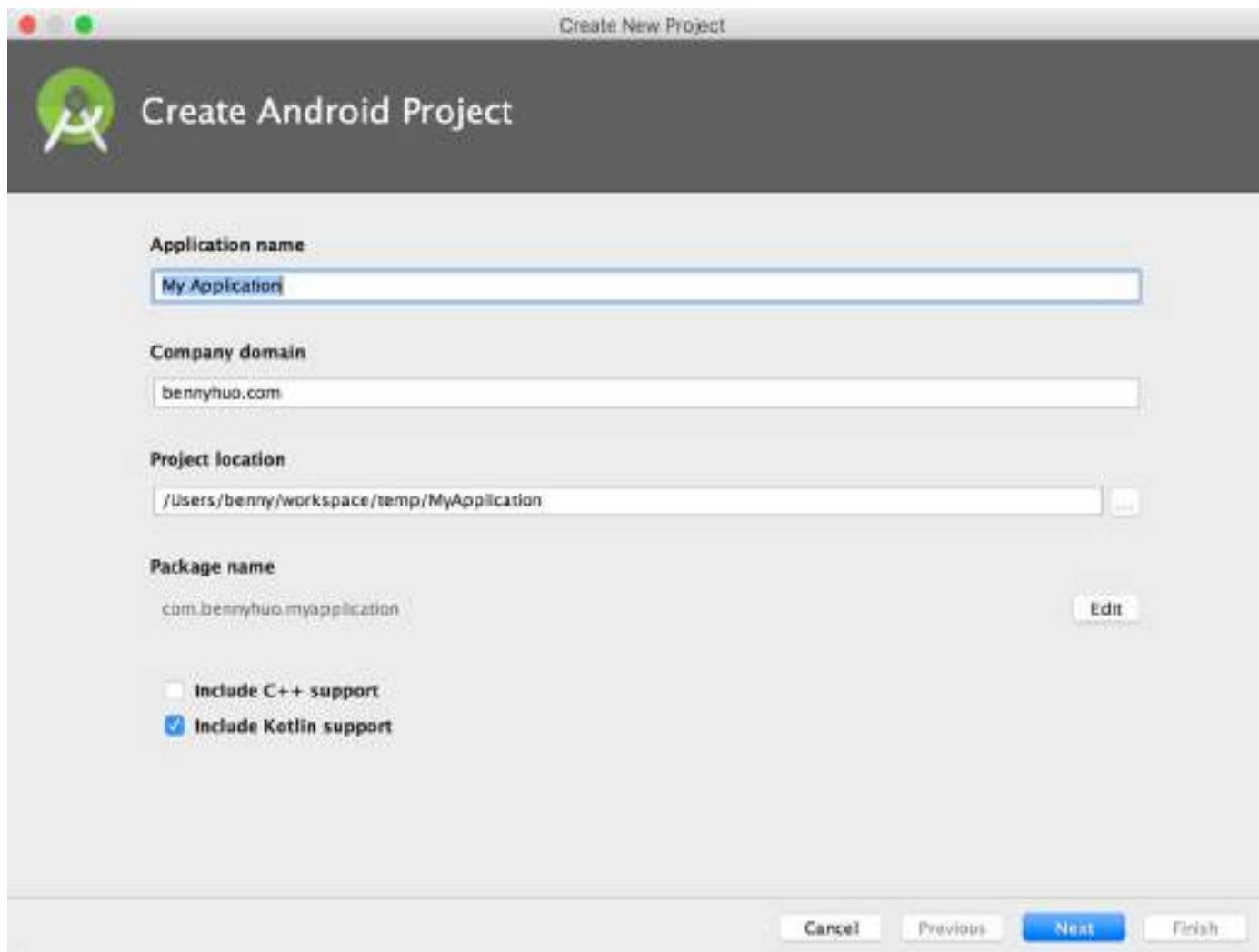
”

“

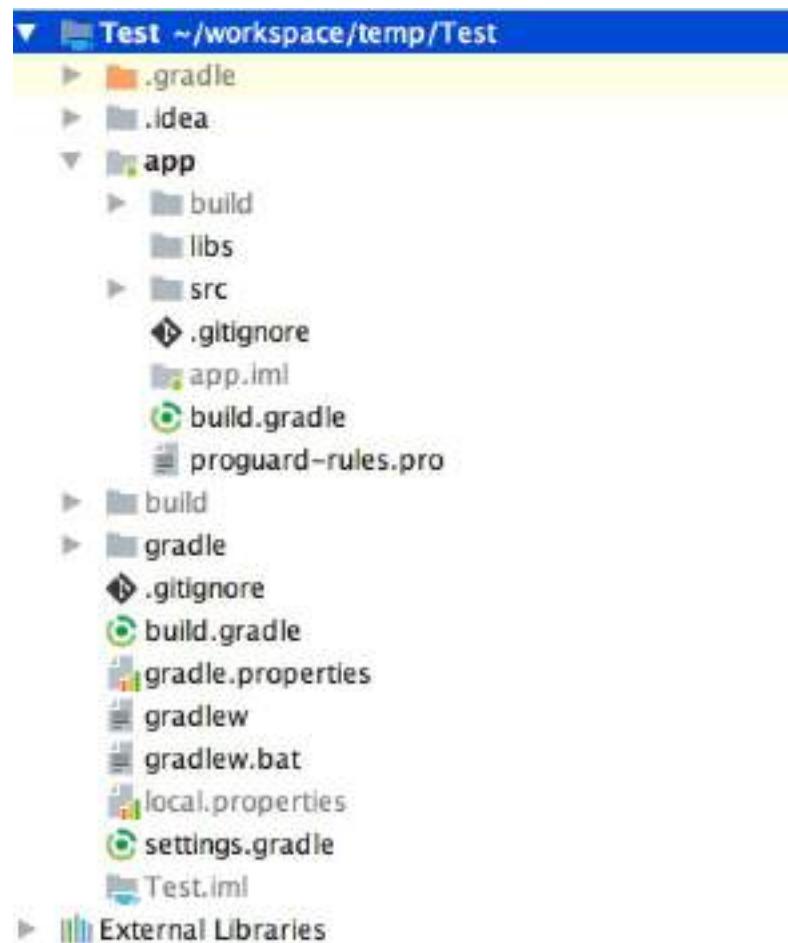
高效，就要少犯错误

”

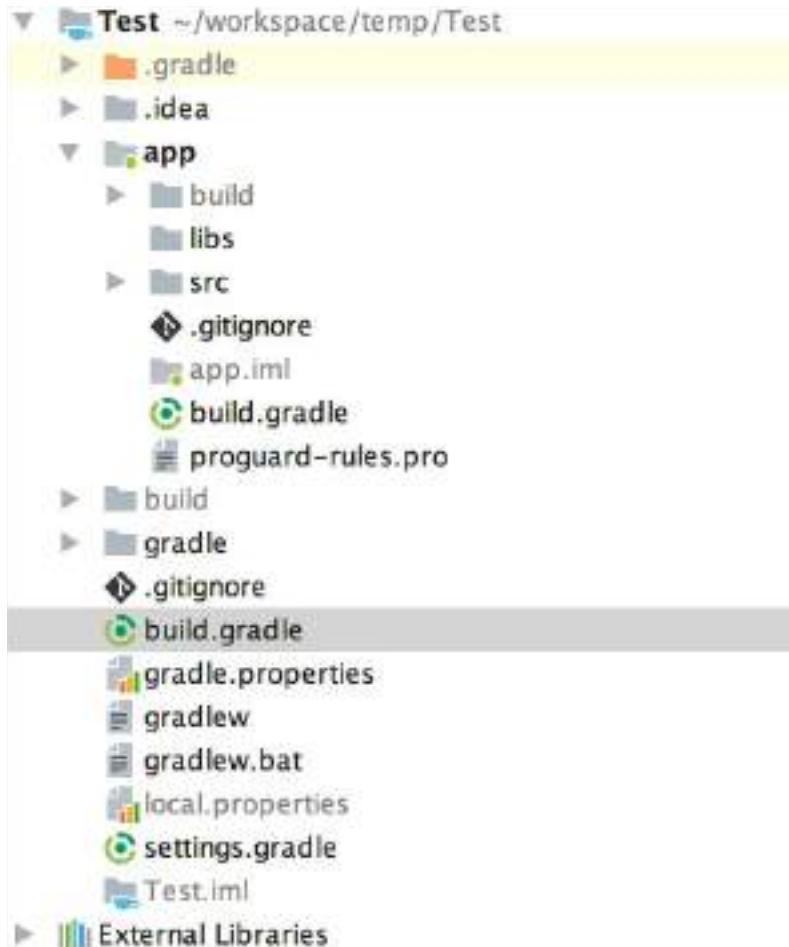
创建工程



创建工程

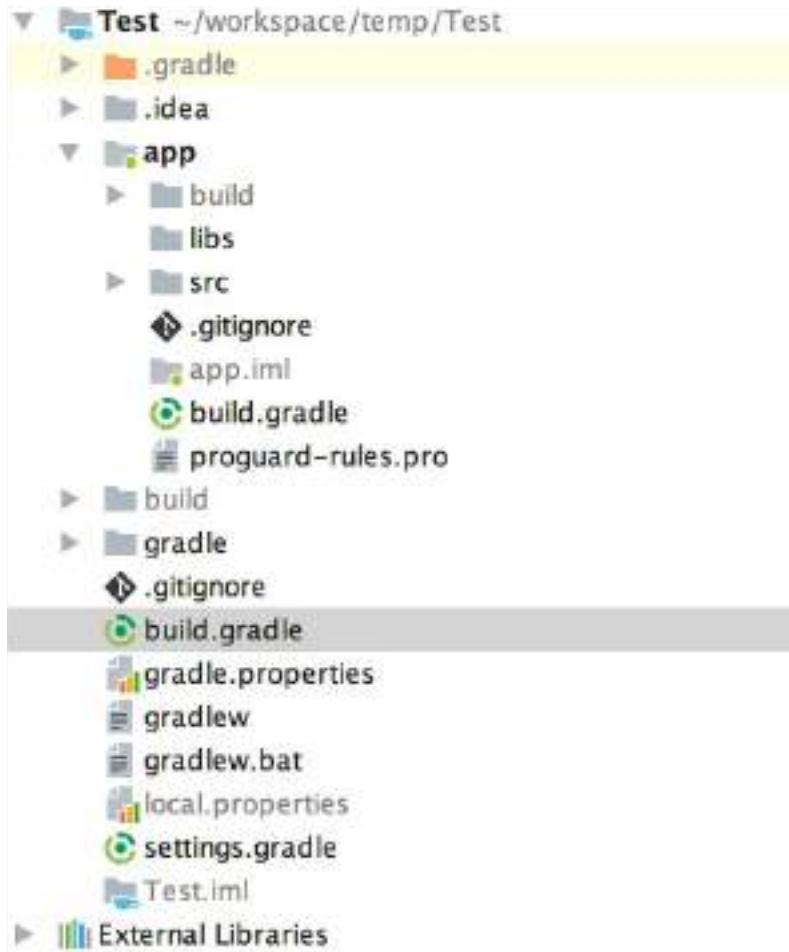


配置工程



```
buildscript {  
    ext.kotlin_version = '1.1.51'  
    repositories {  
        google()  
        jcenter()  
    }  
    dependencies {  
        classpath 'com.android.tools.build:gradle:3.0.0'  
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"  
  
        // NOTE: Do not place your application dependencies here; they belong  
        // in the individual module build.gradle files  
    }  
}
```

配置工程



```
buildscript {  
    ext.kotlin_version = '1.1.51'  
    repositories {  
        google()  
        jcenter()  
    }  
    dependencies {  
        classpath 'com.android.tools.build:gradle:3.0.0'  
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"  
    }  
}  
  
// NOTE: Do not place your application dependencies here; they belong  
// in the individual module build.gradle files
```

配置工程



```
apply plugin: 'com.android.application'
```

```
apply plugin: 'kotlin-android'
```

```
android {
```

```
...
```

```
}
```

```
dependencies {
```

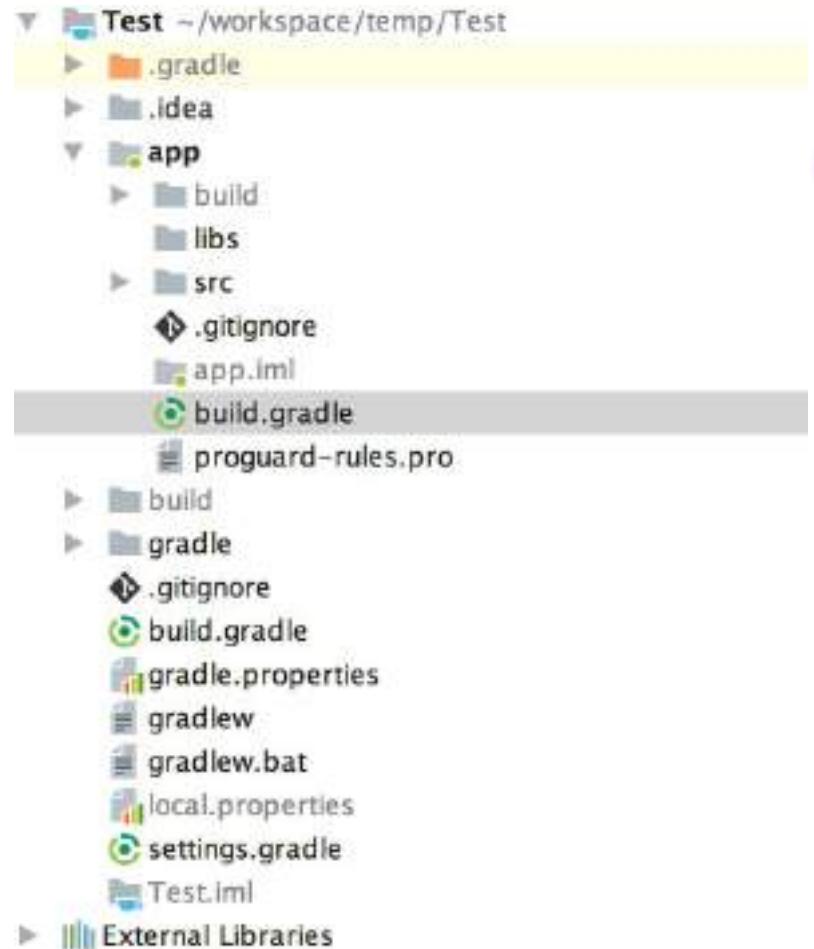
```
    implementation fileTree(dir: 'libs', include: ["*.jar"])
```

```
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jre7:$kotlin_version"
```

```
...
```

```
}
```

配置工程



```
apply plugin: 'com.android.application'
```

```
apply plugin: 'kotlin-android'
```

```
android {
```

```
...
```

```
}
```

```
dependencies {
```

```
    implementation fileTree(dir: 'libs', include: ["*.jar"])
```

```
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jre7:$kotlin_version"
```

```
...
```

```
}
```

如何定义变量

```
int anInt = 2;
```

如何定义变量

```
val anInt: Int = 2
```

如何定义变量

```
val anInt: Int = 2
```

如何定义变量

```
val anInt = 2
```

如何定义变量

```
val anInt = 2  
~~~~~  
anInt = 3
```

如何定义变量

```
val anInt = 2
```

```
anInt = 3
```

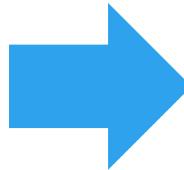
Val cannot be reassigned

如何定义变量

```
var anInt = 2  
anInt = 3
```

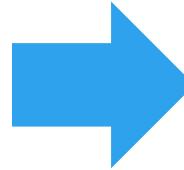
如何定义变量

val anInt = 2



final int anInt = 2;

var anInt = 2



int anInt = 2;

Kotlin

Java

如何定义函数

```
void hello(int anInt){  
    System.out.println(anInt);  
}
```

如何定义函数

```
fun hello(anInt: Int): Unit{  
    println(anInt)  
}
```

如何定义函数

```
fun hello(anInt: Int): Unit{  
    println(anInt)  
}
```

如何定义函数

```
fun hello(anInt: Int) {  
    println(anInt)  
}
```

如何定义函数

```
fun hello(anInt: Int) {  
    println(anInt)  
}
```

如何定义函数

```
fun hello(anInt: Int) {  
    println(anInt)  
}
```

如何定义函数

```
fun hello(anInt: Int) {  
    println(anInt)  
}
```

如何定义函数

```
fun hello(anInt: Int) {  
    println(anInt)  
}
```

如何定义函数

```
fun hello(anInt: Int): Unit{  
    println(anInt)  
}
```

如何定义数组

```
int[] intArray = new int[5];
```

```
int[] anotherIntArray = {1,2, 3,4,5};
```

如何定义数组

```
val intArray = IntArray(5)
```

```
val anotherIntArray = intArrayOf(1, 2, 3, 4, 5)
```

如何定义数组

```
int[] intArray = new int[5];
```

如何定义数组

```
int[] intArray = new int[5];
```

```
for (int i = 0; i < intArray.length; i++) {  
    intArray[i] = i;  
}
```

如何定义数组

```
val intArray = IntArray(5){ it }
```

如何定义数组

```
val intArray = IntArray(5){ it * 2 }
```

如何操作数组

```
intArray[1] = 2;
```

```
int element3 = intArray[3];
```

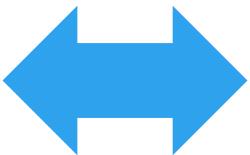
如何操作数组

```
intArray[1] = 2
```

```
val element3 = intArray[3]
```

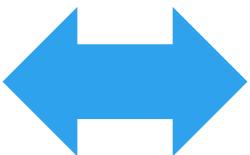
运算符

`intArray[1] = 2`



`intArray.set(1, 2)`

`val element3 = intArray[3]`



`val element3 = intArray.get(3)`

如果是 List、Set 呢？

运算符

```
class 逗你玩{
```

```
    operator fun set(int: Int, value: Any){}
```

```
    operator fun get(int: Int) = Unit
```

```
}
```

运算符

```
class 逗你玩{  
    operator fun set(int: Int, value: Any){}
    operator fun get(int: Int) = Unit
}
```

val 逗你玩的实例 = 逗你玩()

逗你玩的实例[**2**] = 3

val x = 逗你玩的实例[**1**]

WARNING

请不要用中文编程，
不然你可能找不到工作。

数组的类型

- 基本类型的数组
 - `ShortArray/IntArray/LongArray/FloatArray/DoubleArray/CharArray/BooleanArray`
- 其他类型数组
 - `val stringArray = Array(5){ it.toString() } // Array<String>`
 - `val anotherStringArray = arrayOf("hello", "world")`
 - `val strings = arrayOfNulls<String>(5)`

数组的类型

- 基本类型的数组
 - ShortArray/IntArray/LongArray/FloatArray/DoubleArray/CharArray/BooleanArray
- 其他类型数组
 - `val stringArray = Array(5){ it.toString() } // Array<String>`
 - `val anotherStringArray = arrayOf("hello", "world")`
 - `val strings = arrayOfNulls<String>(5)`

数组的类型

- 基本类型的数组
 - ShortArray/IntArray/LongArray/FloatArray/DoubleArray/CharArray/BooleanArray
- 其他类型数组
 - **val** stringArray = Array(5){ it.toString() } // *Array<String>*
 - **val** anotherStringArray = *arrayOf("hello", "world")*
 - **val** strings = *arrayOfNulls<String>(5)*

如何定义类

```
interface AnInterface{}
```

```
class SuperClass{
    public SuperClass() {
    }
}
```

```
class SubClass extends SuperClass implements AnInterface{
    public SubClass() {
        super();
    }
}
```

如何定义类

```
interface AnInterface
```

```
class SuperClass()
```

```
class SubClass : SuperClass(), AnInterface{}
```

如何定义类

interface AnInterface

class SuperClass{}

class SubClass : SuperClass(), AnInterface{}

如何定义类

interface AnInterface

class SuperClass

class SubClass : SuperClass(), AnInterface

如何定义类

interface AnInterface

class SuperClass

class SubClass : SuperClass(), AnInterface

This type is final, so it cannot be inherited from

如何定义类

interface AnInterface

open class SuperClass

class SubClass : SuperClass(), AnInterface

如何定义类

interface AnInterface

open class SuperClass

class SubClass : SuperClass(), AnInterface

如何实例化类

```
SubClass subClass = new SubClass();
```

如何实例化类

```
val subClass = new SubClass()
```

如何实例化类

```
val subClass = new SubClass()
```

如何实例化类

```
val subClass = SubClass()
```

如何实例化类

```
var subClass = SubClass()
```

构造方法

interface AnInterface

open class SuperClass

class SubClass : SuperClass(), AnInterface

构造方法

interface AnInterface

open class SuperClass(aParam: Int)

class SubClass : SuperClass(10), AnInterface

构造方法

```
interface AnInterface

open class SuperClass(aParam: Int) {
    init {
        println(aParam)
    }

    val aProperty = aParam
}

class SubClass : SuperClass(10), AnInterface
```

构造方法

```
interface AnInterface

open class SuperClass(aParam: Int) {
    init {
        println(aParam)
    }

    val aProperty = aParam
}

class SubClass ( aParamForSuper: Int )
    : SuperClass( aParamForSuper ), AnInterface
```

构造方法

```
class SubClass( val aPropertyInSub: String, aParamForSuper: Int )  
    : SuperClass( aParamForSuper ), AnInterface
```

构造方法

```
class SubClass( val aPropertyInSub: String, aParamForSuper: Int )  
    : SuperClass( aParamForSuper ), AnInterface
```

val subClass = SubClass("Hello", 10)

subClass.

v	aPropertyInSub	String
v	aProperty	Int
m	toString()	String("Hello Kotlin".toByteArray())
λ	to(that: B) for A in kotlin	Pair<SubClass, B>
m	hashCode()	Int
m	equals(other: Any?)	Boolean
v	javaClass	for T in kotlin.jvm
λ	let {...} (block: (SubClass) -> R) for T in kotlin	R
λ	apply {...} (block: SubClass.() -> Unit) for T in kotlin	SubClass
λ	also {...} (block: (SubClass) -> Unit) for T in kotlin	SubClass
λ	run {...} (block: SubClass.() -> R) for T in kotlin	R
Use ⇧ to syntactically correct your code after completing (balance parentheses etc.)		» π

构造方法

```
public class Call {  
    private String callId;  
}
```

构造方法

```
public class Call {  
  
    private String callId;  
  
    public Call(String callId) {  
        this.callId = callId;  
    }  
  
}
```

构造方法

```
public class Call {  
  
    private String callId;  
  
    public Call(){  
        // 稍后再初始化 callId  
    }  
  
    public Call(String callId) {  
        this.callId = callId;  
    }  
}
```

构造方法

```
public class Call {  
    private final String callId;  
  
    public Call(String callId) {  
        this.callId = callId;  
    }  
}
```

构造方法

```
class Call(val callId: String)
```

构造方法

```
class Call(val callId: String){  
    constructor()  
}
```

构造方法

```
class Call(val callId: String){
```

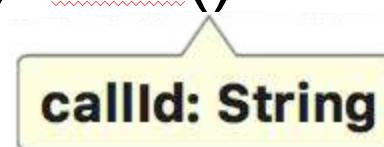
constructor()

Primary constructor call expected

}

构造方法

```
class Call(val callId: String){  
    constructor(): this()  
}
```



构造方法

```
public ViewGroup(Context context) {  
    this(context, null);  
}  
  
public ViewGroup(Context context, AttributeSet attrs) {  
    this(context, attrs, 0);  
}  
  
public ViewGroup(Context context, AttributeSet attrs, int defStyleAttr) {  
    this(context, attrs, defStyleAttr, 0);  
}  
  
public ViewGroup(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes) {  
    super(context, attrs, defStyleAttr, defStyleRes);  
    ...  
}
```



失败

DEFEAT

构造方法

```
public SomeViewGroup(Context context) {  
    super(context);  
    init();  
}
```

```
public SomeViewGroup(Context context, AttributeSet attrs) {  
    super(context, attrs);  
    init();  
}
```

构造方法

```
public SomeViewGroup(Context context) {  
    super(context);  
    init();  
    initData();  
}
```

```
public SomeViewGroup(Context context, AttributeSet attrs) {  
    super(context, attrs);  
    init();  
}
```

构造方法

```
public SomeViewGroup(Context context) {  
    super(context);  
    init();  
    initData();  
}
```

```
public SomeViewGroup(Context context, AttributeSet attrs) {  
    super(context, attrs);  
    init();  
    initData();  
}
```

构造方法

```
481     public SomeViewGroup(Context context) {  
482         super(context);  
485         init();  
486         initData();  
    }
```

```
547     public SomeViewGroup(Context context, AttributeSet attrs) {  
548         super(context, attrs);  
549         init();  
550         initData();  
    }
```

方法重载

```
public interface List<E> extends Collection<E> {  
    ...  
    public E remove(int location);  
  
    public boolean remove(Object object);  
    ...  
}
```

方法重载

```
public interface List<E> extends Collection<E> {  
    ...  
    public E remove(int location);  
  
    public boolean remove(Object object);  
    ...  
}
```

```
List<Integer> integers = new ArrayList<>();
```

```
...
```

```
integers.remove(2);
```

方法重载

```
public interface List<E> extends Collection<E> {  
    ...  
    public E remove(int location);  
  
    public boolean remove(Object object);  
    ...  
}
```

val integers = ArrayList<Int>()

...

integers.removeAt(2)

方法重载

```
public interface List<E> extends Collection<E> {  
    ...  
    public E remove(int location);  
  
    public boolean remove(Object object);  
    ...  
}
```

val integers = ArrayList<Int>()

...

integers.removeAt(2)

默认参数

```
class SomeViewGroup(context: Context,  
                    attrs: AttributeSet?,  
                    defStyleAttr: Int,  
                    defStyleRes: Int)  
    : ViewGroup(context, attrs, defStyleAttr, defStyleRes) {  
  
    init {  
        ...  
    }  
}
```

默认参数

```
class SomeViewGroup(context: Context,  
                    attrs: AttributeSet? = null,  
                    defStyleAttr: Int = 0,  
                    defStyleRes: Int = 0)  
    : ViewGroup(context, attrs, defStyleAttr, defStyleRes) {  
  
    init {  
        ...  
    }  
}
```

覆写父类成员

```
@Override  
protected void onResume() {  
    super.onResume();  
    tencentMapView.onResume();  
}
```

```
@Override  
protected void onPause() {  
    super.onPause();  
    tencentMapView.onPause();  
}
```

覆写父类成员

```
public class MyMapView extends MapView {  
    ...  
    public void onResume() {  
        doSomethingOnResume();  
    }  
  
    @Override  
    public void onPause() {  
        super.onPause();  
        doSomethingOnPause();  
    }  
    ...  
}
```

覆写父类成员

```
class MyMapView(context: Context) : MapView(context) {
```

```
    fun onResume() {  
        doSomethingOnResume()  
    }
```

'onResume' hides member of supertype 'MapView' and needs 'override' modifier

```
    override fun onPause() {  
        super.onPause()  
        doSomethingOnPause()  
    }  
}
```

覆写父类成员

```
class MyMapView(context: Context) : MapView(context) {  
  
    override fun onResume() {  
        doSomethingOnResume()  
    }  
  
    override fun onPause() {  
        super.onPause()  
        doSomething.onPause()  
    }  
}
```

属性代理

```
public class Link {  
    private double length = -1;  
  
    private List<LatLng> points;  
  
    ...  
}
```

属性代理

```
public void update(LatLng latLng){
```

```
...
```

```
if(length == -1){
```

```
    length = resolveLength();
```

```
}
```

```
...
```

```
}
```

```
private double resolveLength(){
```

```
    return ...;
```

```
}
```

属性代理

```
public void update(LatLng latLng){  
    ...  
    double length = getLength();  
    ...  
}  
  
public double getLength(){  
    if(length == -1){  
        length = resolveLength();  
    }  
    return length;  
}  
  
private double resolveLength(){  
    return ...;  
}
```

属性代理

```
private val length by lazy {  
    resolveLength()  
}
```

```
private fun resolveLength(): Double {  
    return ...  
}
```

属性代理

```
private val length by lazy {
```

...

```
}
```

属性代理

```
private val length by lazy( SYNCHRONIZED ) {  
    ...  
}
```

属性代理

```
enum class LazyThreadSafetyMode {  
    SYNCHRONIZED,  
    PUBLICATION,  
    NONE,  
}
```

属性代理

```
inline operator fun <T> Lazy<T>.getValue(  
    thisRef: Any?,  
    property: KProperty<*>  
): T = value
```

属性代理

```
private var _value: Any? = UNINITIALIZED_VALUE  
  
override val value: T  
  
    get() {  
        if (_value === UNINITIALIZED_VALUE) {  
            _value = initializer!!()  
            initializer = null  
        }  
        return _value as T  
    }
```

属性代理

```
private var _value: Any? = UNINITIALIZED_VALUE  
override val value: T  
  
    get() {  
        if (_value === UNINITIALIZED_VALUE) {  
            _value = initializer!!()  
            initializer = null  
        }  
        return _value as T  
    }
```

```
private val length by lazy {  
    ...  
}
```

属性代理

```
public interface ReadWriteProperty<in R, T> {  
    public operator fun getValue(thisRef: R, property: KProperty<*>): T  
    public operator fun setValue(thisRef: R, property: KProperty<*>, value: T)  
}
```

属性代理

```
class PropertiesDelegate <T> (val path: String): ReadWriteProperty <Any, T> {

    val properties: Properties by lazy {
        ... // 读属性
    }

    override operator fun getValue(thisRef: Any, property: KProperty<*>): T {
        val value = properties[property.name]
        val classOfT = property.returnType.classifier as KClass<*>
        return if (Number::class.isSuperclassOf(classOfT)) {
            ... // 如果是数值需要做一些转换
        } else {
            value
        } as T
    }

    override operator fun setValue(thisRef: Any, property: KProperty<*>, value: T) {
        properties[property.name] = value
        ... // 存属性
    }
}
```

属性代理

```
class PropertiesDelegate (val path: String) {

    val properties: Properties by lazy {
        ... // 读属性
    }

    operator fun <T> getValue(thisRef: Any, property: KProperty<*>): T {
        val value = properties[property.name]
        val classOfT = property.returnType.classifier as KClass<*>
        return if (Number::class.isSuperclassOf(classOfT)) {
            ... // 如果是数值需要做一些转换
        } else {
            value
        } as T
    }

    operator fun <T> setValue(thisRef: Any, property: KProperty<*>, value: T) {
        properties[property.name] = value
        ... // 存属性
    }
}
```

属性代理

```
abstract class AbsProperties(path: String) {  
    protected val prop = PropertiesDelegate(path)  
}
```

属性代理

```
abstract class AbsProperties(path: String) {  
    protected val prop = PropertiesDelegate(path)  
}  
  
object MetalInfo: AbsProperties("/meta.properties"){  
    val version: String by prop  
    val author: String by prop  
    val name: String by prop  
    val desc: String by prop  
}
```

属性代理

```
object MetalInfo: AbsProperties("/meta.properties"){
    val version: String by prop
    val author: String by prop
    val name: String by prop
    val desc: String by prop
}
```

version=1.0-SNAPSHOT

author=bennyhuo@kotliner.cn

name=QCloudImageUploader

desc=方便地批量上传指定目录的图片到腾讯云的免费 50G 图床

属性代理

link: <<https://api.github.com/user/repos?page=2>>; rel="next",
<<https://api.github.com/user/repos?page=5>>; rel="last"

属性代理

```
link: <https://api.github.com/user/repos?page=2>; rel="next",
<https://api.github.com/user/repos?page=5>; rel="last"
```

属性代理

```
map["next"] = "https://api.github.com/user/repos?page=2"  
map["last"] = "https://api.github.com/user/repos?page=5"
```

属性代理

```
map["next"] = "https://api.github.com/user/repos?page=2"  
map["last"] = "https://api.github.com/user/repos?page=5"  
map["first"] = "https://api.github.com/user/repos?page=1"  
map["prev"] = "https://api.github.com/user/repos?page=1"
```

属性代理

```
class GitHubPaging{  
    val isLast: Boolean = ?  
    val isFirst: Boolean = ?  
    val hasPrev: Boolean = ?  
    val hasNext: Boolean = ?  
    operator fun get(key: String): String?{  
        return ?  
    }  
}
```

属性代理

```
class GitHubPaging{  
    val first: String? = ?  
    val last: String? = ?  
    val next: String? = ?  
    val prev: String? = ?  
    val isLast: Boolean = ?  
    val isFirst: Boolean = ?  
    val hasPrev: Boolean = ?  
    val hasNext: Boolean = ?  
}
```

属性代理

```
class GitHubPaging{  
    val first by map  
    ...  
    val isFirst  
        get() = first == null  
    ...  
}
```

参考 Kotlin 公众号文章：[用 Map 为你的属性做代理](#)

属性代理

```
object Settings {  
    var lastPage by pref(0)  
    var dayNightMode by pref(false)  
    var enablePush by pref(false)  
}
```

参考 Kotlin 公众号文章：[用 Map 为你的属性做代理](#)

接口代理

```
interface PageManager {  
    fun showPage(clazz: KClass<out Page>)  
  
    fun goBack()  
}
```

接口代理

```
class PageManagerImpl: PageManager{
    override fun showPage(clazz: KClass<out Page>) {
        ...
    }

    override fun goBack() {
        ...
    }

    override fun dismiss() {
        ..
    }
}
```

接口代理

```
abstract class AbstractPage(val pageManager: PageManager)
    : Page, PageManager {
    override fun showPage(clazz: KClass<out Page>) {
        pageManager.showPage(clazz)
    }

    override fun goBack() {
        pageManager.goBack()
    }

    override fun dismiss() {
        pageManager.dismiss()
    }
}
```

接口代理

```
abstract class AbstractPage(val pageManager: PageManager)  
  : Page, PageManager by pageManager{  
}
```

接口代理

```
abstract class AbstractPage(val pageManager: PageManager)  
  : Page, PageManager by pageManager
```

扩展成员

```
inline operator fun <T> Lazy<T>.getValue(  
    thisRef: Any?,  
    property: KProperty<*>  
): T = value
```

扩展成员

```
inline operator fun <T> Lazy<T>.getValue(  
    thisRef: Any?,  
    property: KProperty<*>  
): T = value
```

扩展成员

```
inline operator fun <T> Lazy<T>.getValue(  
    thisRef: Any?,  
    property: KProperty<*>  
): T = value
```

扩展成员

```
inline operator fun <T> Lazy<T>.getValue(  
    thisRef: Any?,  
    property: KProperty<*>  
): T = value
```

扩展成员

```
public class DensityUtil {  
  
    /**  
     * 根据手机的分辨率从 dp 的单位 转成为 px(像素)  
     */  
    public static int dip2px(float dpValue) {  
        final float scale = ContextHolder.getInstance().getContext().getResources().getDisplayMetrics().density;  
        return (int) (dpValue * scale + 0.5f);  
    }  
  
    /**  
     * 根据手机的分辨率从 px(像素) 的单位 转成为 dp  
     */  
    public static int px2dip(float pxValue) {  
        final float scale = ContextHolder.getInstance().getContext().getResources().getDisplayMetrics().density;  
        return (int) (pxValue / scale + 0.5f);  
    }  
}
```

扩展成员

//returns dip(dp) dimension value in pixels

```
fun Context.dip(value: Int) = (value * resources.displayMetrics.density).toInt()  
fun Context.dip(value: Float) = (value * resources.displayMetrics.density).toInt()
```

//return sp dimension value in pixels

```
fun Context.sp(value: Int) = (value * resources.displayMetrics.scaledDensity).toInt()  
fun Context.sp(value: Float) = (value * resources.displayMetrics.scaledDensity).toInt()
```

//converts px value into dip or sp

```
fun Context.px2dip(px: Int) = px.toFloat() / resources.displayMetrics.density  
fun Context.px2sp(px: Int) = px.toFloat() / resources.displayMetrics.scaledDensity
```

扩展成员

listView.scrollX = dip(2)

paint.*strokeWidth* = *dip(2).toFloat()*

paint.*textSize* = *sp(12).toFloat()*

扩展成员

```
logger.error("*****")  
logger.error("+++++")
```

扩展成员

```
fun String.times(other: Int): String{
    return (1..other).fold(StringBuilder()){ acc, i ->
        acc.append(this)
        acc
    }.toString()
}
```

扩展成员

```
logger.error("*****")  
logger.error("+++++")
```

扩展成员

logger.error("" .times(8))*

logger.error("+" .times(8))

扩展成员

```
fun String.times(other: Int): String{
    return (1..other).fold(StringBuilder()){ acc, i ->
        acc.append(this)
        acc
    }.toString()
}
```

扩展成员

```
operator fun String.times(other: Int): String{  
    return (1..other).fold(StringBuilder()){ acc, i ->  
        acc.append(this)  
        acc  
    }.toString()  
}
```

扩展成员

logger.error("" .times(8))*

logger.error("+" .times(8))

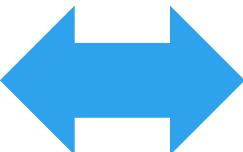
扩展成员

```
logger.error("*" * 8 )
```

```
logger.error"+" * 8 )
```

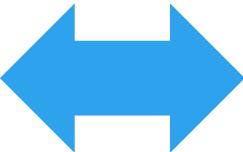
运算符

$a + b$



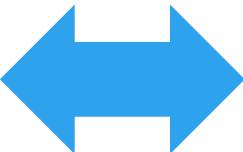
`a.plus(b)`

$a - b$



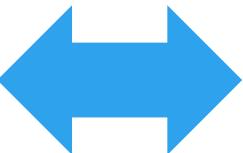
`a.minus(b)`

$a * b$



`a.times(b)`

a / b



`a.div(b)`

扩展成员

```
String formattedDate  
= new SimpleDateFormat("yyyy-MM-dd")  
.format(new Date());
```

Java

扩展成员

```
new Date().format("yyyy-MM-dd")
```

Groovy

扩展成员

```
val formattedDate  
    = SimpleDateFormat("yyyy-MM-dd").format(Date())
```

Kotlin

扩展成员

```
fun Date.format(format: String) : String{  
    return SimpleDateFormat(format).format(this)  
}
```

扩展成员

```
fun Date.format(format: String) : String  
= SimpleDateFormat(format).format(this)
```

扩展成员

```
fun Date.format(format: String)  
    = SimpleDateFormat(format).format(this)
```

扩展成员

```
val formattedDate  
    = SimpleDateFormat("yyyy-MM-dd").format(Date())
```

Kotlin

扩展成员

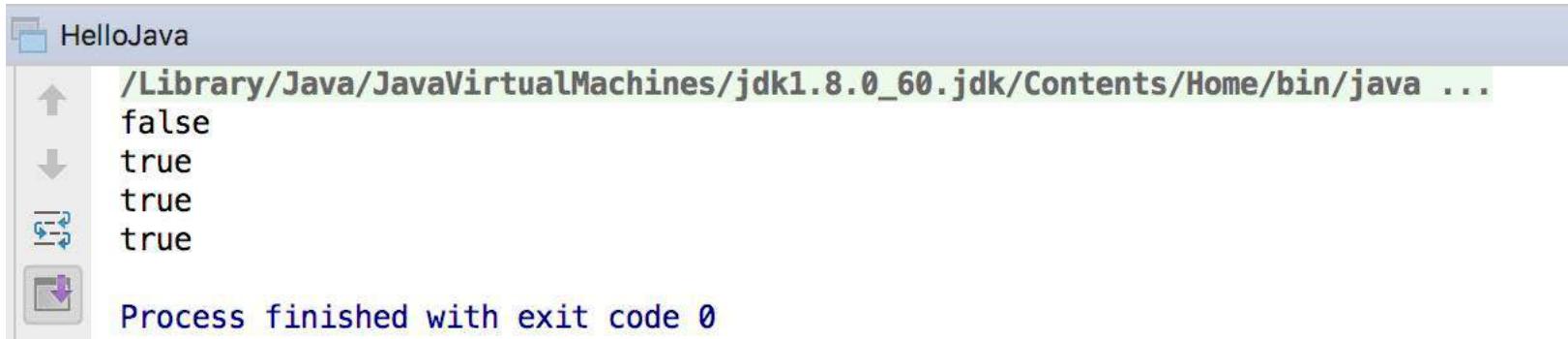
```
val formattedDate = Date().format("yyyy-MM-dd")
```

如何比较对象

```
String aString = "Hello Kotliner";  
String anotherString = new String("Hello Kotliner");  
System.out.println(aString == anotherString);  
System.out.println(aString.equals(anotherString));  
System.out.println(aString == anotherString.intern());  
System.out.println(aString.equals(anotherString.intern()));
```

如何比较对象

```
String aString = "Hello Kotliner";
String anotherString = new String("Hello Kotliner");
System.out.println(aString == anotherString);
System.out.println(aString.equals(anotherString));
System.out.println(aString == anotherString.intern());
System.out.println(aString.equals(anotherString.intern()));
```



The screenshot shows a terminal window titled "HelloJava". The command entered is the Java executable followed by the class name. The output consists of five lines of text: "false", "true", "true", "true", and "true". At the bottom, it says "Process finished with exit code 0".

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java HelloJava
false
true
true
true
true
Process finished with exit code 0
```

如何比较对象

```
val aString = "Hello Kotliner"
```

```
val anotherString = String("Hello Kotliner")
```

如何比较对象

```
val aString = "Hello Kotliner"
```

```
val anotherString = String("Hello Kotliner")
```

None of the following functions can be called with the arguments supplied.

- `String(StringBuffer)` defined in `kotlin.text`
- `String(StringBuilder)` defined in `kotlin.text`
- `String(ByteArray)` defined in `kotlin.text`
- `String(CharArray)` defined in `kotlin.text`

如何比较对象

```
val aString = "Hello Kotliner"  
val anotherString = String("Hello Kotliner".toByteArray())  
println(aString === anotherString)  
println(aString == anotherString)  
println(aString === anotherString.intern())  
println(aString == anotherString.intern())
```

如何比较对象

```
val aString = "Hello Kotliner"  
val anotherString = String("Hello Kotliner".toByteArray())  
println(aString === anotherString)  
println(aString == anotherString)  
println(aString === anotherString.intern())  
println(aString == anotherString.intern())
```

如何比较对象

```
val aString = "Hello Kotliner"  
val anotherString = String("Hello Kotliner".toByteArray())  
println(aString === anotherString)  
println(aString == anotherString)  
println(aString === anotherString.intern())  
println(aString == anotherString.intern())
```

如何比较对象

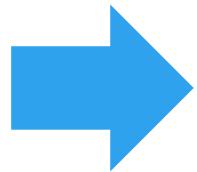
```
val aString = "Hello Kotliner"  
val anotherString = String("Hello Kotliner".toByteArray())  
println(aString === anotherString)  
println(aString == anotherString)  
println(aString === anotherString.intern())  
println(aString == anotherString.intern())
```

如何比较对象

```
val aString = "Hello Kotliner"  
val anotherString = String("Hello Kotliner".toByteArray())  
println(aString === anotherString)  
println(aString.equals(anotherString))  
println(aString === anotherString.intern())  
println(aString.equals(anotherString.intern()))
```

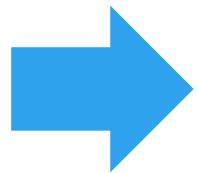
如何比较对象

`a == b`



`a.equals(b)`

`a === b`



`a == b`

Kotlin

Java

如何比较对象

```
var anInt = ...
```

```
var anotherInt = ...
```

```
if(anInt < anotherInt){
```

```
    println("anInt < anotherInt")
```

```
} else {
```

```
    println("anInt > anotherInt")
```

```
}
```

如何比较对象

```
var anInt = ...
```

```
var anotherInt = ...
```

```
if(anInt .compareTo( anotherInt ) < 0){
```

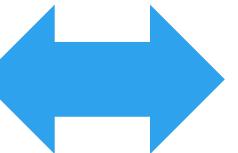
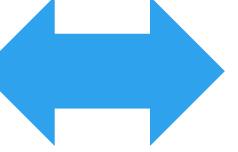
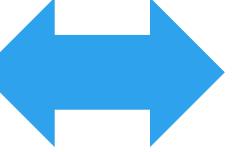
```
    println("anInt < anotherInt")
```

```
} else {
```

```
    println("anInt > anotherInt")
```

```
}
```

运算符

$a == b$		$a.equals(b)$
$a < b$		$a.compareTo(b) < 0$
$a > b$		$a.compareTo(b) > 0$

$a \geq b$ 和 $a \leq b$ 呢？

数据类

```
public class Link {  
    private String id;  
    private double length;  
    private String name;  
    private String attributes;  
  
    public String getId() {  
        return id;  
    }  
  
    public void setId(String id) {  
        this.id = id;  
    }  
  
    ... // Getters/Setters  
}
```