Kotlin语法解析

Tencent-应用宝

From TimLuo

前文概要

- 12/16 Kotlin实战分享

  介绍Kotlin基本情况以及应用宝实战中的经验总结

- Kotlin语法解析

  帮助大家了解Kotlin这门语言的使用方法

- 有一定的Java & Android 基础

- 通过Demo了解基本使用

- **探讨设计思想**

一、Hello World

二、Demo演练

三、蓝牙使用说明

四、总结

HelloWorld.kt

```
package demo

fun main(args: Array<String>) {
    println("Hello World")
}
```

- 跟Java有什么不同 ？

HelloWorld.kt

```
package demo

fun main(args: Array<String>) {
    println("Hello World")
}
```

- Kotlin源文件以.kt结尾
- 没有；号结尾
- **没有Class**

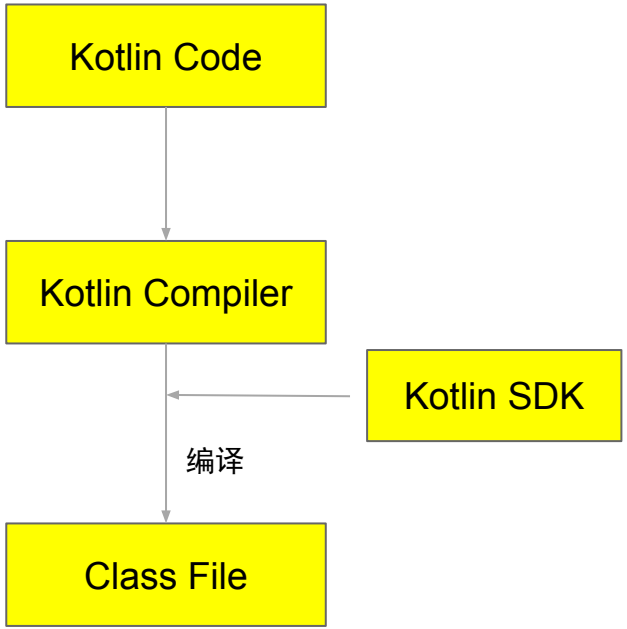
反编译结果

```
package demo;

import kotlin.jvm.internal.Intrinsics;

public final class HelloWorldKt {
    public static final void main(String[] args) {
        Intrinsics.checkParameterIsNotNull(args, "args");
        String str = "Hello World";
        System.out.println(str);
    }
}
```

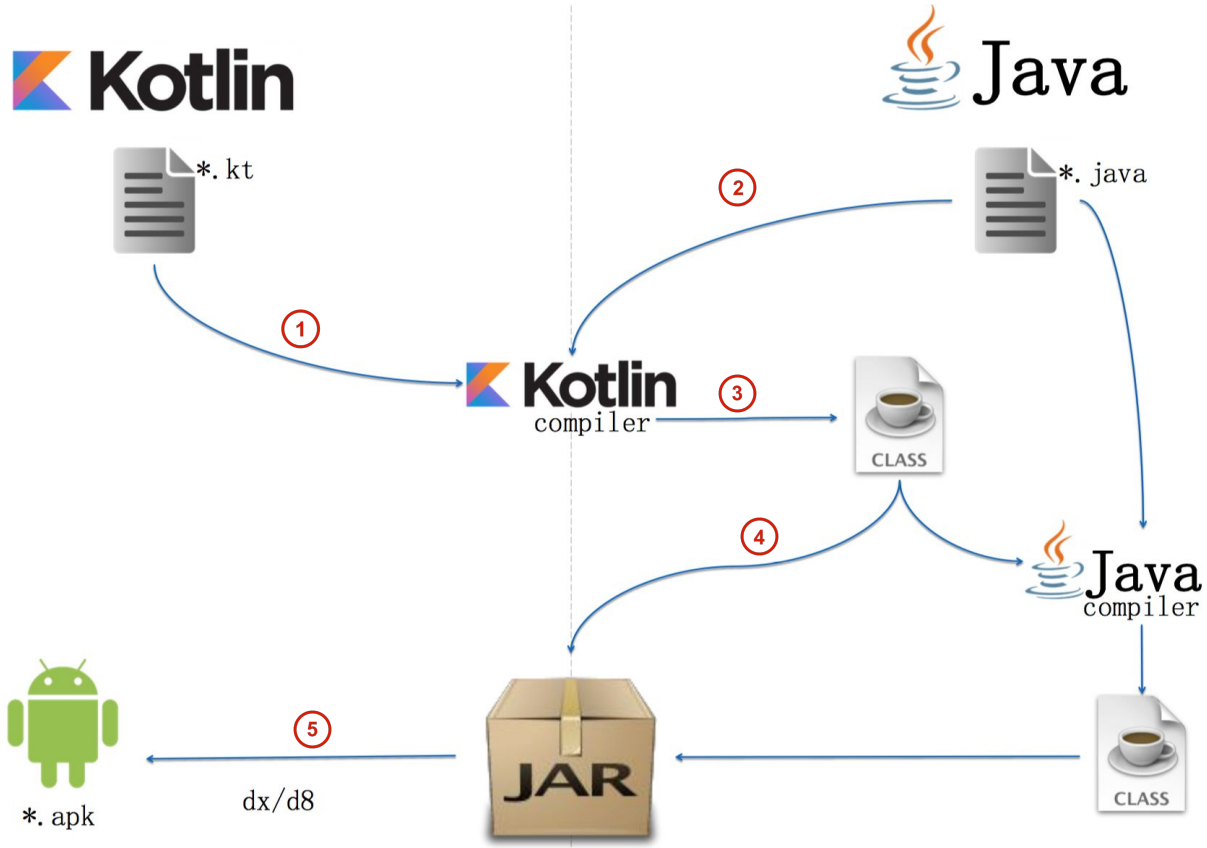
- 在编译时，自动添加MainClass
- 引入Kotlin的依赖Jar包

编译方式

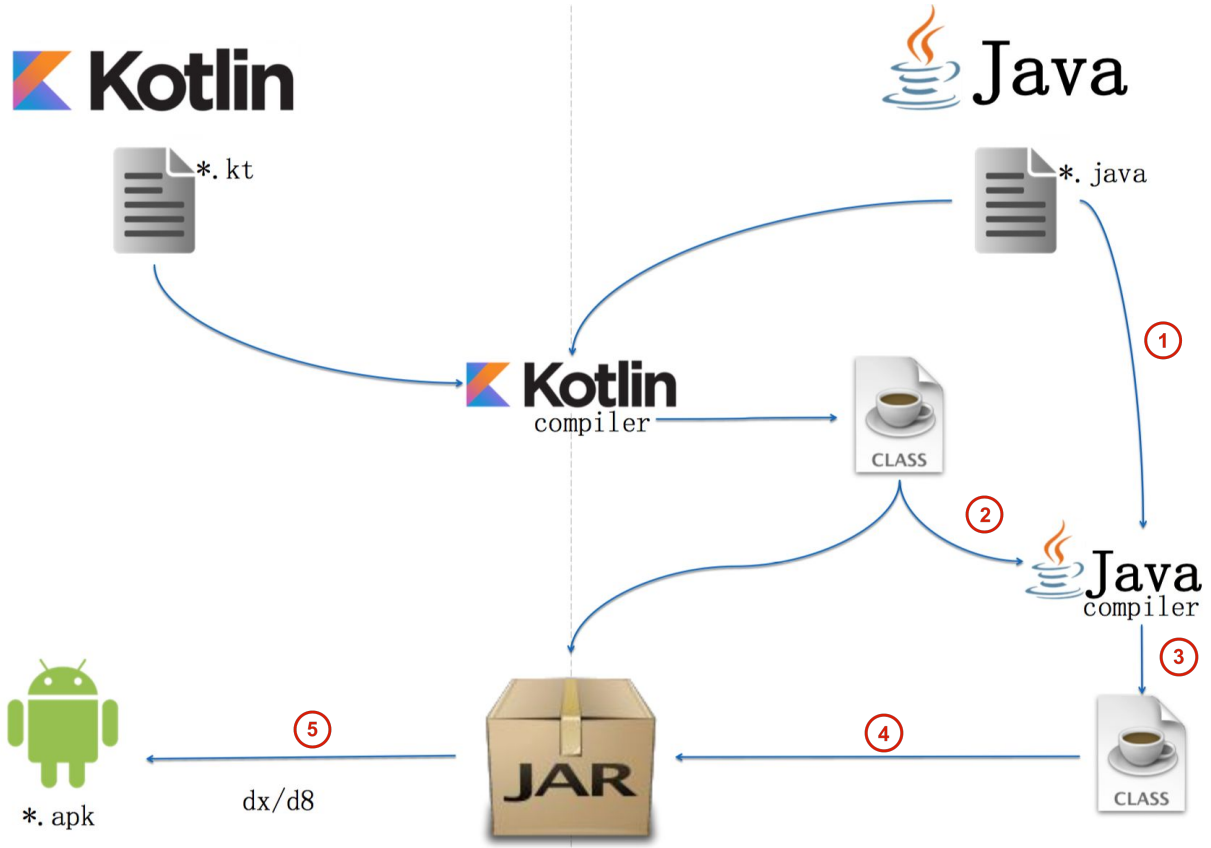
- **为什么要这样设计？**



Kotlin的本质


Kotlin
Java
*. kt
*. java
2
1
Kotlin
compiler
3
CLASS
4
Java
compiler
*. apk
5
dx/d8
JAR
CLASS

Kotlin的本质


Kotlin
Java
*. kt
*. java
Kotlin
compiler
CLASS
1
2
Java
compiler
3
*. apk
5
dx/d8
JAR
4
CLASS

一、前言

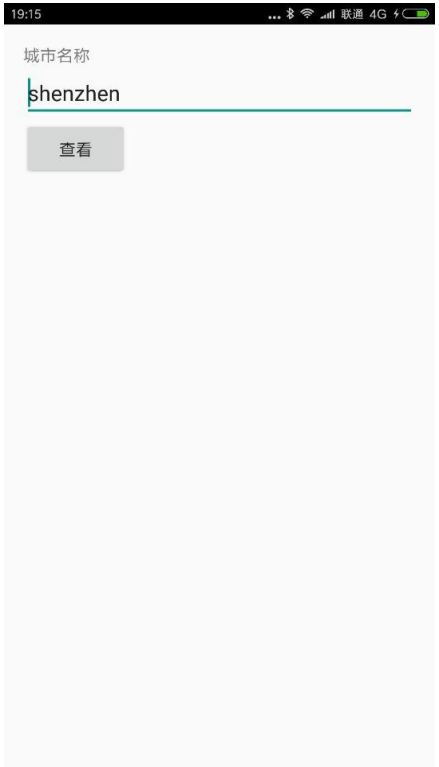
二、Demo演练

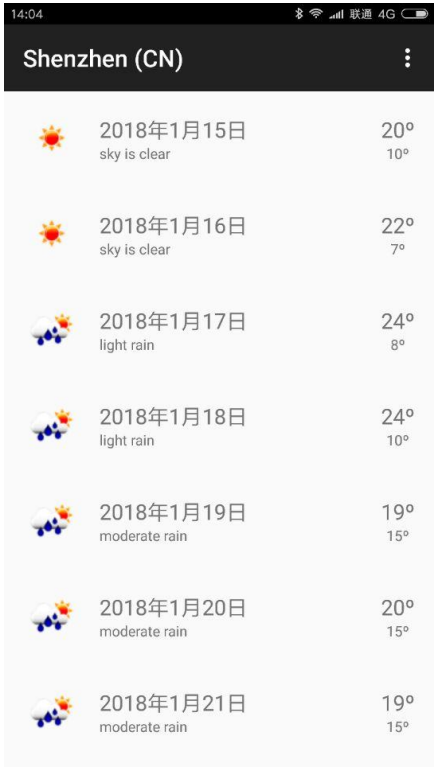
三、蓝牙使用说明

四、总结

Demo

- Demo App



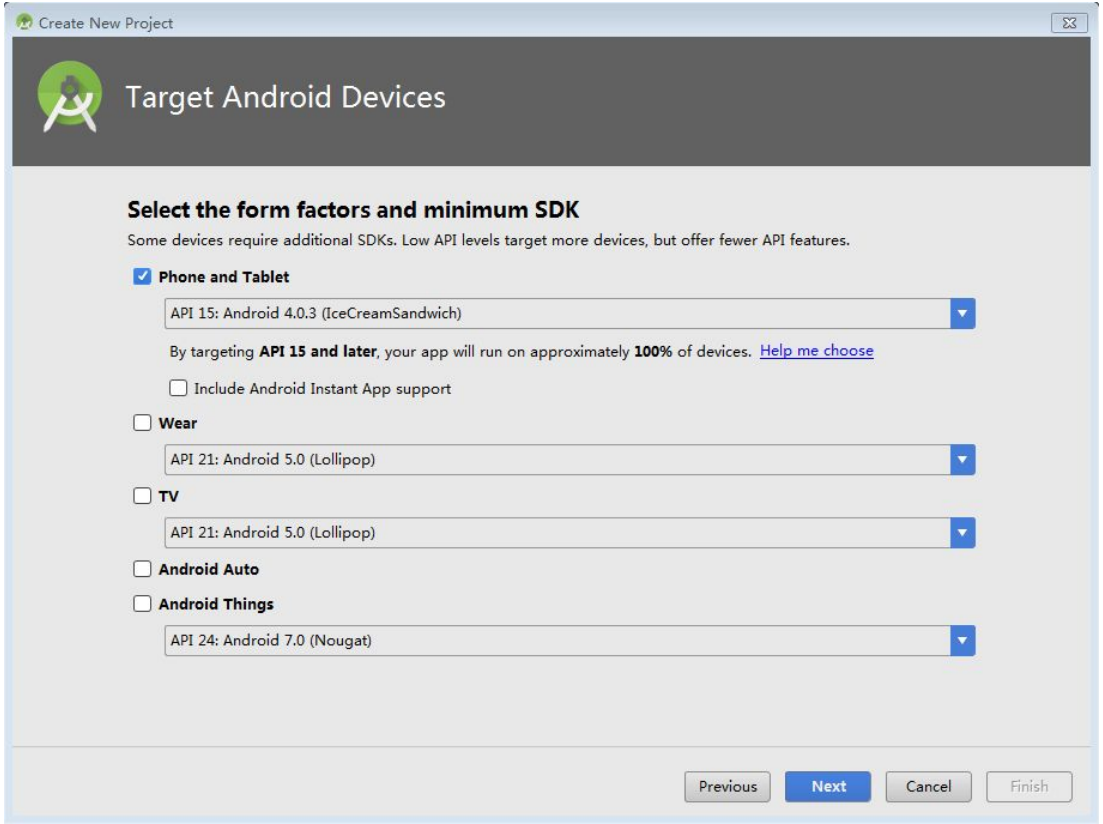


开发环境

- Android Studio 3.0
- Kotlin插件
- Stuidio 3.0以上自动集成
- Studio 3.0以下手动安装Kotlin插件

新建工程

- 最低Api：15



配置Gradle

- 主工程build.gradle

```
buildscript {
    ext.support_version = '26.1.0'
    ext.kotlin_version = '1.1.61'
    ext.anko_version = '0.10.3'
    repositories {
        jcenter()
        google()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.0.1'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    }
}

allprojects {
    repositories {
        jcenter()
        maven { url 'https://maven.google.com' }
        google()
    }
}
```

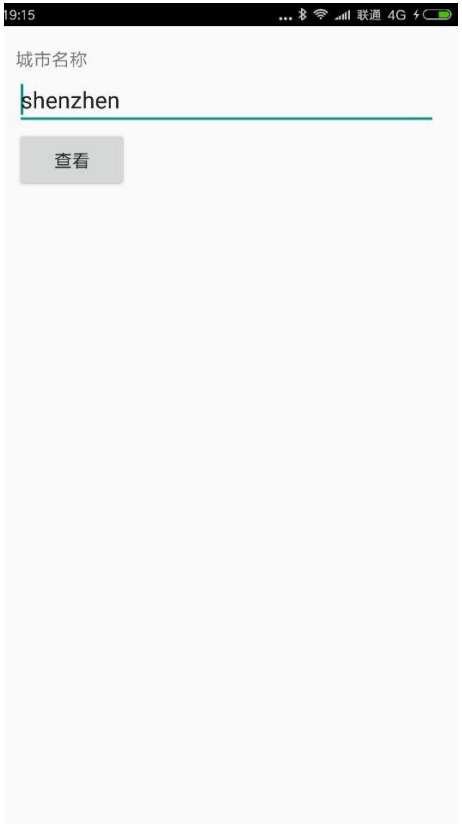
配置Gradle

- app模块build.gradle
- 引入lib库
  - Kotlin基本库
  - Anko
    - 简化布局实现
    - 丰富的语法糖
    - 协程

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
android {
    compileSdkVersion 26
    buildToolsVersion "26.0.2"

    defaultConfig {
        applicationId "com.antonioleiva.weatherapp"
        minSdkVersion 15
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
    }
}
dependencies {
    compile "com.android.support:appcompat-v7:$support_version"
    compile "com.android.support:recyclerview-v7:$support_version"
    compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    compile "org.jetbrains.anko:anko-common:$anko_version"
    compile "org.jetbrains.anko:anko-sqlite:$anko_version"
    compile "org.jetbrains.anko:anko-coroutines:$anko_version"
    compile "com.google.code.gson:gson:2.8.0"
    compile "com.squareup.picasso:picasso:2.5.2"
}
```

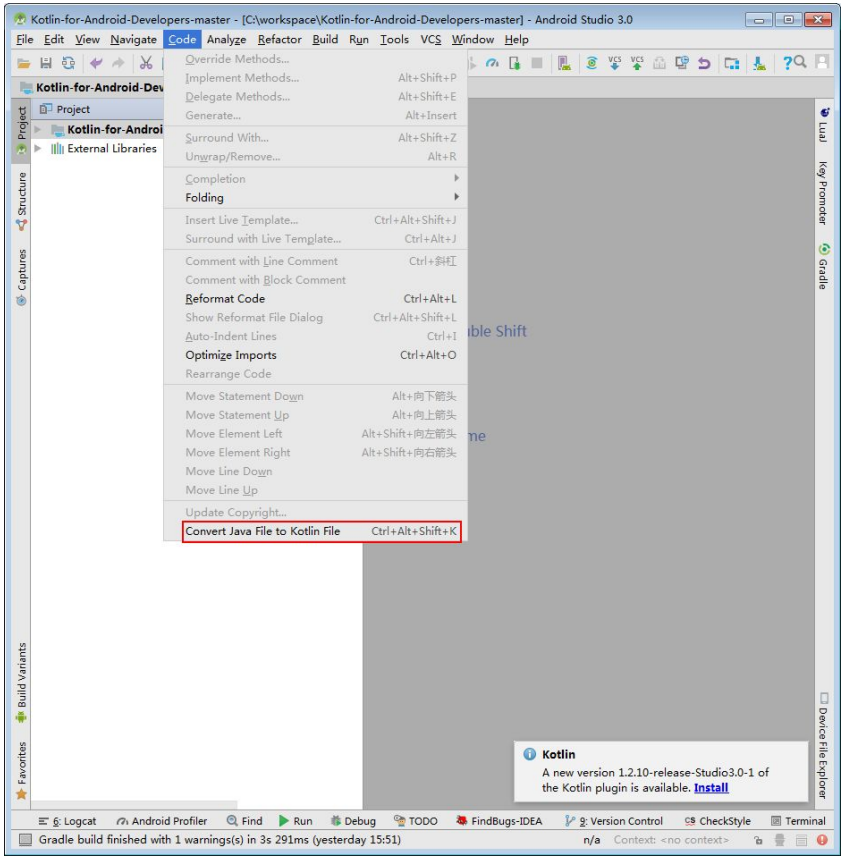
设置页

- 获取输入的城市代码
- 保存城市代码
- 点击按钮，查看天气



定义一个类

- 新增kt文件
  - File->Kotlin File/Class
  - 文件名.kt结尾
- 已有Java转Kotlin



继承

- 语法

```
class SettingsActivity : AppCompatActivity(){
    ......

}
```

- 所有类默认不可继承
    - Java：所有类默认可继承

```
open class MyClass
```

- 所有类继承自Any
    - Java：所有类默认继承自Object

构造函数

- 主构造函数
- 隐含了定义成员变量
- init为初始化块
- 次级构造函数必须收拢到主构造函数

设置布局

- activity_settings.xml

- 保存EditText中的内容到SharedPreferences

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:padding="16dp">
        <TextView...>
        <EditText
            android:id="@+id/inputText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="城市名称"/>

        <Button...>
    </LinearLayout>

</FrameLayout>
```

主页面

- Java中实现

```
public class SettingsActivity extends AppCompatActivity {
    ……
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_settings);
        final TextView inputText= (TextView) findViewById(R.id.inputText);
        inputText.setText(“shenzhen”);
        ……
    }
    ……
}
```

- 繁琐的findViewById

Kotlin实现

```
class SettingsActivity : AppCompatActivity() {
    1                          2
    val inputText: TextView by lazy {
        find<TextView>(R.id.inputText)
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_settings)
        inputText.setText(cityName.toString())
        ……
    }
}
```

基本数据类型

- **尽量用只读类型**
- 可变
- 不可变
- 标准语法

```
var/val <propertyName> : <propertyType> [=property_initializer]
[<getter>]
[<setter>]
```

- 反编译实现

```
private final String city;

private final String getCity();

private final void setCity(String cityName);
```

var	getter + setter
val	getter

基本数据类型

- 字符串拼接
- 智能类型推导
- 示例

```
fun main(args: Array<String>) {
    var bigInt: Int = Int.MAX_VALUE
    var smallInt = 0
    println("BigInt :" + bigInt)      //print "BigInt :2147483647"
    println("SmallInt : ${smallInt}") //print "SmallInt : 0"
}
```

基本数据类型	位宽
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

数组

- 数组
  - 申明
  - 常用写法

懒加载

- layz懒加载实现
  - 第一次使用时候，执行表达式
  - 只能用于不可变的变量(val)

空安全

- 空指针保护
- 解决的问题
  - 编译期，发现空安全的隐患
  - 运行期，对空指针增加保护
  - 提升代码质量

空安全

- 编译期，发现空指针错误

```
val a:String ="abc"
a = null    //编译错误

var b: String? = "abc"
b = null // 编译ok
```

空安全

- 安全调用

```
var str: String? = "testString"
val bLen: Int? = str?.length
```

```
String str = "testString";
Integer bLen = (str == null) ? null : str.length();
```

- elvis操作符

```
var str: String? = "testString"
val bLen = (str?.length) ?: -1
```

```
var str: String? = "testString"
val bLen: Int = if (str != null) str.length else -1
```

Kotlin实现

- Kotlin Android Extensions
  - 保证变量名与Xml中ViewId一致

```
class SettingsActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_settings)
        inputText.setText("shenzhen")
        ……
    }
}
```

设置按钮点击事件

- Kotlin函数写法

```
class SettingsActivity : AppCompatActivity() {
    ……
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_settings)
        inputText.setText(cityName.toString())
        confirmBtn.setOnClickListener {
            toWeatherPage(inputText.text.toString())
        }
    }

    fun toWeatherPage(cityName :String){
        this.cityName = cityName
        startActivity<MainActivity>()
    }
}
```

函数

- 函数申明

```
fun <functionName>( <propertyName> : <propertyType>) : <returnType>

fun add(x: Int, y: Int) : Int {
    return x + y
}
```

- 表达式函数

```
fun <functionName>( <paramName> : <paramType>) : <returnType> = <表达式>

fun add(x: Int, y: Int) : Int = x+y
```

函数

- 默认参数 & 命名参数

```
fun add(x: Int = 100, y: Int = 1): Int = x + y

add() // return 101
add(x = 0) // return 1
add(y = -1) // return 99
```

默

```
fun printHello(name: String?): Unit {
    ……
}
fun printHello(name: String?)  {
    ….
}
```

操作符重载

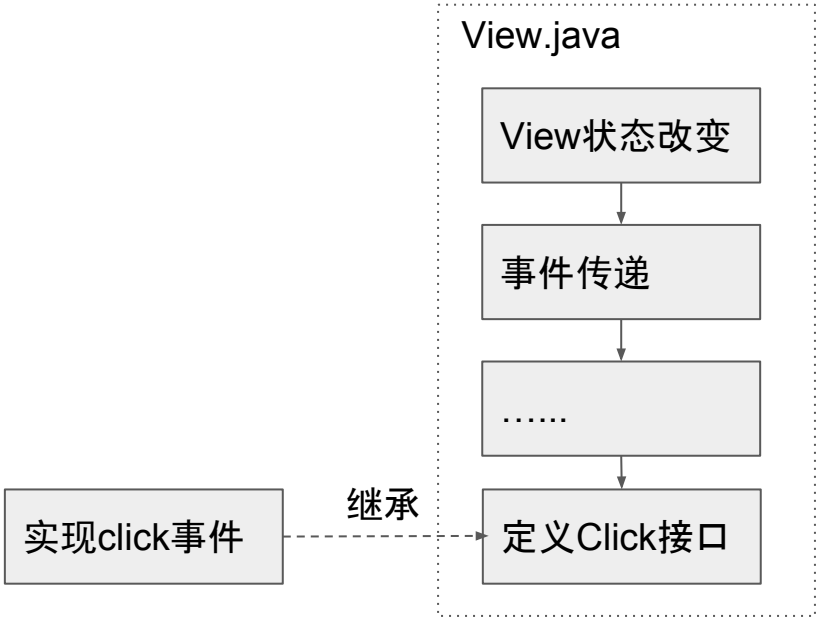
表达式	转换
a + b	a.plus(b)
a - b	a.minus(b)
a * b	a.times(b)
a / b	a.div(b)
a % b	a.mod(b)

Lambda表达式写法

- 使用

```
confirmBtn.setOnClickListener{
    startActivity<MainActivity>()
}
```

- 应用场景
  - 需要定义对象
  - 需要定义接口
  - **需要实现方法**



Lambda表达式写法

- 封装原型
  - 将函数作为入参
  - 规定函数的定义
  - 在上层进行调用

```
/** View类
 *  listener:参数名
 *  (View):函数入参
 *  Unit:函数出参，无返回 值类型
 */
fun setOnClickListener(listener : (View) -> Unit) {
    // do something.....
    clickEvent(this)
}
```

- 调用方式

```
confirmBtn.setOnClickListener( {view: View? -> Unit
    ......
})
```

Lambda表达式写法

- 单个参数可省略

```
confirmBtn.setOnClickListener({
    ......
})
```

- 单个参数用it来进行引用

```
confirmBtn.setOnClickListener({
    it.getTag()
    ......
})
```

- 闭包

```
confirmBtn.setOnClickListener{
    it.getTag()
    ......
}
```

函数式编程思想

- 函数是第一等公民
  - 可以作为参数
  - 可以赋值给变量
  - 可以作为返回值
- 纯函数
  - 函数运行不依赖外部状态
  - 函数运行不修改外部状态

保存数据到SharePreferences

```
public class SettingsActivity extends AppCompatActivity {
    private static final String CITY_NAME = "cityName";
    private static final String DEFAULT_CITY = "ShenZhen";
    private String getCityName() {
        return getApplication().getSharedPreferences("default",
        Context.MODE_PRIVATE).getString(CITY_NAME, DEFAULT_CITY);
    }
    private void setCityName(String cityName) {
        getApplication().getSharedPreferences("default",
        Context.MODE_PRIVATE).edit().putString(CITY_NAME,
        cityName).commit();
    }
    ……
}
```

Kotlin实现

```
class SettingsActivity : AppCompatActivity() {
                  1
    companion object {
        val CITY_NAME = "cityName"
        val DEFAULT_CITY = "ShenZhen"
    }                                  2
    private var cityName: String by DelegatesExt.preference(this, CITY_NAME,
    DEFAULT_CITY )
  ……
}
```

伴生对象

- 废弃static关键字
- 反编译代码

```
companion object {
    val CITY_NAME = "cityName"
    val DEFAULT_CITY = "ShenZhen"
}
```

```
class SplashActivity : Activity() {
    private static final String CITY_NAME = "cityName";
    private static final String DEFAULT_CITY = "ShenZhen";
    public static final class Companion {
        public final String getCITY_NAME() {
        return SettingsActivity.CITY_NAME ;
    }
    public final String getDEFAULT_CITY() {
        return SettingsActivity.DEFAULT_CITY ;
    }
}
```

伴生对象

- 在Kotlin中调用

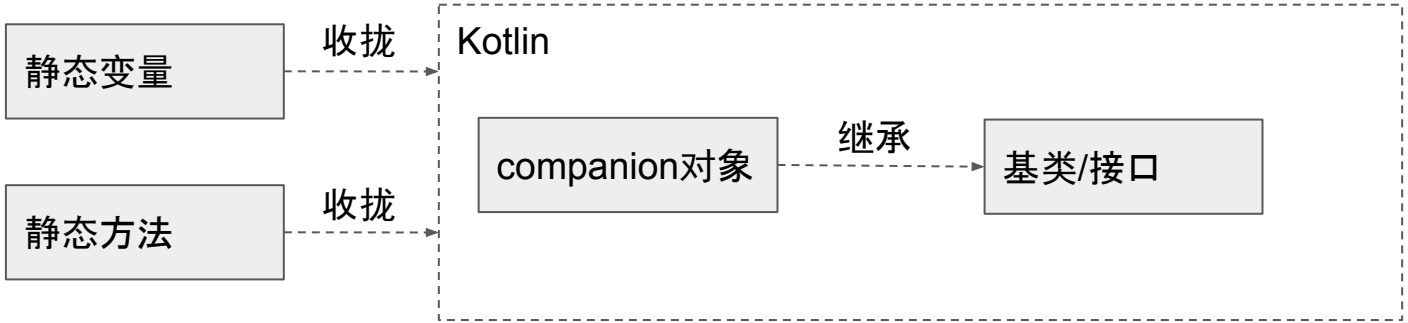
```
SettingsActivity.CITY_NAME
```

- 在Java中调用

```
SettingsActivity.Companion.getCITY_NAME()
```

伴生对象

- 万物皆对象



委托属性by

- 基本语法

```
var/var <属性名> : <类型> by <表达式>;
案例:
getCityName( ) -> DelegatesExt.preference.getValue( )
setCityName( String ) -> DelegatesExt.preference.setValue( String )
```

- 示例

```
private var cityName: String by DelegatesExt.preference(this, CITY_NAME,
    DEFAULT_CITY)
```

```
cityName.toString()
```

```
getCityName()
```

```
DelegatesExt.preference.getValue(CITY_NAME,DEFAULT_CITY)
```

```
cityName="ShenZhen"
```

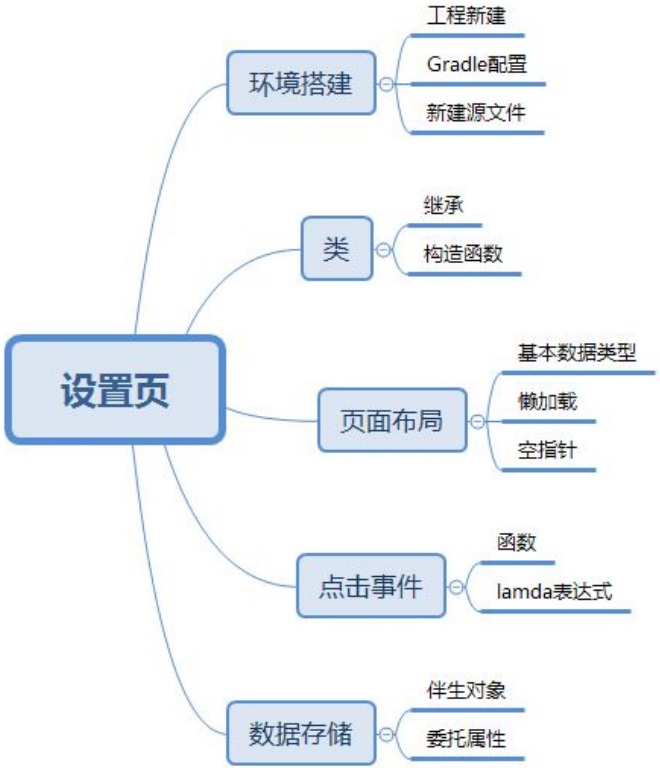
```
setCityName("ShenZhen")
```

```
DelegatesExt.preference.setValue("ShenZhen")
```

设置页完整代码

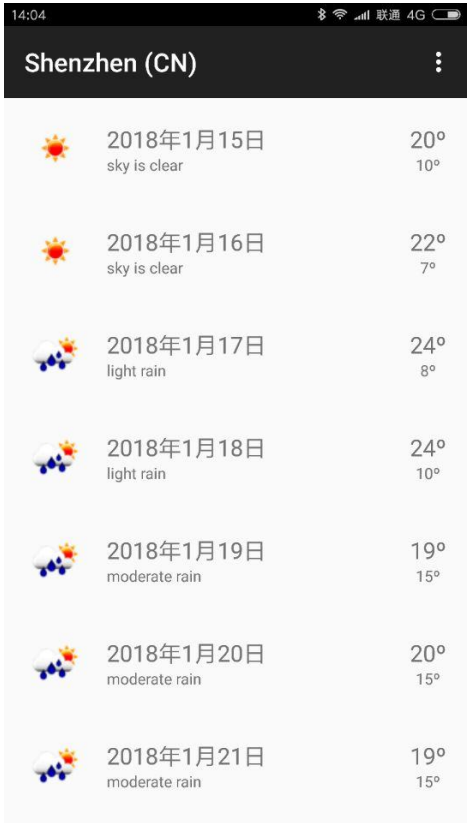
```
class SettingsActivity : AppCompatActivity() {
    companion object {
        val CITY_NAME = "city"
        val DEFAULT_CITY = "shenzhen"
    }
    private var cityName: String by DelegatesExt.preference(this, CITY_NAME, DEFAULT_CITY)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_settings)
        inputText.setText(cityName)
        confirmBtn.setOnClickListener {
            toWeatherPage(inputText.text.toString())
        }
    }
    fun toWeatherPage(cityName :String){
        this.cityName = cityName
        startActivity<MainActivity>()
    }
}
```

小结


设置页
环境搭建
工程新建
Gradle配置
新建源文件
类
继承
构造函数
页面布局
基本数据类型
懒加载
空指针
点击事件
函数
lamda表达式
数据存储
伴生对象
委托属性

天气页

- UI布局
- 请求天气数据



UI布局

- 顶部ToolBar
- RecyclerView

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/forecastList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:clipToPadding="false"
        android:paddingTop="?attr/actionBarSize"/>

    <include layout="@layout/toolbar"/>

</FrameLayout>
```

顶部ToolBar实现

```
class MainActivity : AppCompatActivity(), ToolbarManager {
    ……
    override val toolbar by lazy { find<Toolbar>(R.id.toolbar) }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        initToolbar()
        ……
    }
}
```

```
interface ToolbarManager {

    val toolbar: Toolbar

    var toolbarTitle: String
        get() = toolbar.title.toString()
        set(value) {
        toolbar.title = value
        }

    fun initToolbar() {
        toolbar.inflateMenu(R.menu.menu_main)
        toolbar.setOnMenuItemClickListener {
        when (it.itemId) {
            R.id.action_settings ->toolbar.ctx.startActivity<SettingsActivity>()
            else -> App.instance.toast("Unknown option")
        }
        true
      }
    }
}
```

接口

- 可以提供默认实现
  - 通过增加内部类来实现
  - 冲突必须重写

```
interface City {
    fun printName(): String
}

interface Country {
    fun printName(): String
}

class Rome : City, Country {
    override fun printName() = "Rome"
}
```

```
public interface ToolbarManager {

    public Toolbar getToolbar();

    public static final class DefaultImpls {

        public static void initToolbar(ToolbarManager $this) {
            ......
        }
    }
}
```

为什么加这个特性

- Java禁止多重继承

- 多个类具有相同的行为，但是分属不同类别
  - 天鹅会飞，天鹅属于鸟
  - 蝴蝶会飞，蝴蝶属于昆虫

When

- 增强型的Swith

```
val x = 10
when (x) {
    1 -> println("x == 1")
    2,3 -> println("x == 2 or 3")
    else -> { // Note the block
    println("unknown")
    }
}
```

```
val x = 10
val y = 10
when {
    x > y -> println("x > y")
    x < y -> println("x < y")
    else -> {
        println("x == y")
    }
}
```

- 只匹配最先找到的表达式

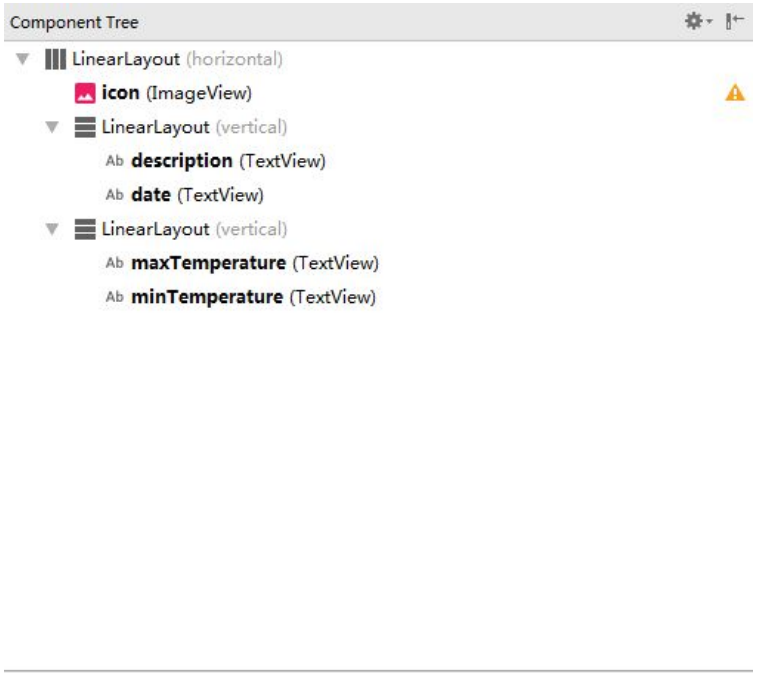
```
val x = 10
when {
    x > 2  -> println(" > 2")
    x > 3 -> println(" > 3")
    else -> println("no")
}
```

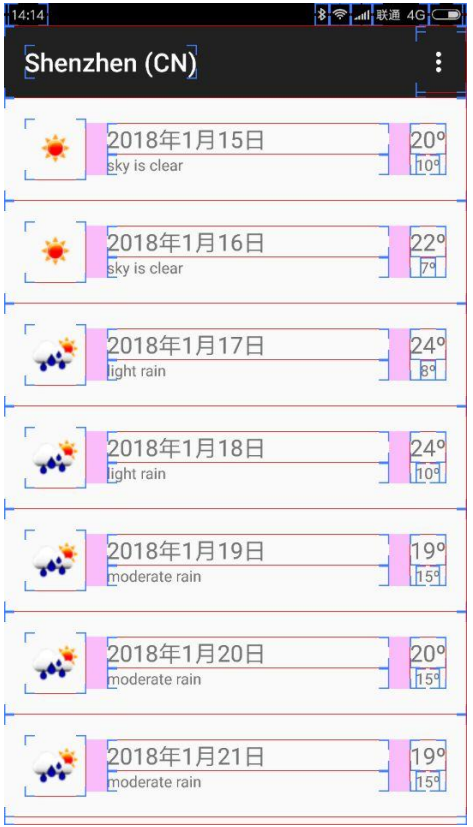
MainActivity.kt

```
class MainActivity : AppCompatActivity(), ToolbarManager {
    private val cityName: Long by DelegatesExt.preference(this, SettingsActivity.CITY_NAME,SettingsActivity.DEFAULT_CITY)
    override val toolbar by lazy { find<Toolbar>(R.id.toolbar) }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        initToolbar()
        forecastList.layoutManager = LinearLayoutManager(this)
        loadForecast()
    }
    private fun loadForecast() = async(UI) {
        val result = bg { RequestForecastCommand(cityName).execute() }
        updateUI(result.await())
    }
    private fun updateUI(weekForecast: ForecastList) {
        forecastList.adapter = ForecastListAdapter(weekForecast) {
            startActivity<DetailActivity>(DetailActivity.ID to it.id,DetailActivity.CITY_NAME to weekForecast.city)
    }
    toolbarTitle = "${weekForecast.city} (${weekForecast.country})"
    }
}
```

布局

- item_forecast.xml
- 对日期格式化





数据刷新

- 扩展函数
  - 对类进行扩展
- let闭包
  - 代码简洁

```
class ViewHolder(view: View, private val itemClick: (Forecast) -> Unit)
    : RecyclerView.ViewHolder(view) {
    fun bindForecast(forecast: Forecast) {
                3
        forecast.let { data ->
            Picasso.with(itemView.ctx).load(data.iconUrl).into(itemView.icon)
            //对日期进行格式化
                                    2
        1 itemView.date.text = data.date.toDateString()
            itemView.description.text = data.description
            itemView.maxTemperature.text = "${data.high}º"
            itemView.minTemperature.text = "${data.low}º"
            itemView.setOnClickListener { itemClick(data) }
            }
    }
}
```

扩展函数

- Java实现

```
public class TimeUtils {
    public static String toDateString(Long time) {
        SimpleDateFormat sDateFormat = new SimpleDateFormat("yyyy年MM月dd
日");
        return sDateFormat.format(time);
    }
}
```

```
public class ForecastListAdapter extends RecyclerView.Adapter {
    ……
    public void bindForecast(Forecast forecast){
        ……
        itemView.date.text = TimeUtils.toDateString(data.date)
        ……
    }
}
```

扩展函数

- 特性
  - 不修改原类
  - 增加函数
  - 简化代码

```
class ViewHolder(view: View, private val itemClick: (Forecast) -> Unit)
    : RecyclerView.ViewHolder(view) {
    fun bindForecast(forecast: Forecast) {
        forecast.let { data ->
            ……
            //data.date是一个Long
            itemView.date.text = data.date.toDateString()
            ……
        }
    }
fun Long.toDateString(dateFormat: Int = DateFormat.MEDIUM): String {
    val df = DateFormat.getDateInstance(dateFormat, Locale.getDefault())
        return df.format(this)
    }
}
```

闭包

- 使用极为普遍
- let
- apply

```
class ViewHolder(view: View, private val itemClick: (Forecast) -> Unit)
    : RecyclerView.ViewHolder(view) {
    fun bindForecast(forecast: Forecast) {
        forecast.let { data ->
            ……
        }
    }
}
```

闭包

函数名	参数	返回值
let	默认it，可自定义	闭包返回
apply	无，可用this	this

```
class LetClass {
    fun printTest() {
        val letTest = "letObject".let { it ->
            println(this.toString()) // "LetClass"
            println(it) // "letObject"
            "return letCode"
        }
        println(letTest )   //"return letCode"
    }
    override fun toString() = "LetClass"
}
```

```
class ApplyClass {
    fun printTest() {
        val applyTest = "applyObject".apply {
            println(this.toString())//"applyObject"
            "return applyCode"
        }
        println(applyTest) //"applyObject"
    }
    override fun toString() = "ApplyClass"
}
```

请求网络数据

- 通过Api访问网络

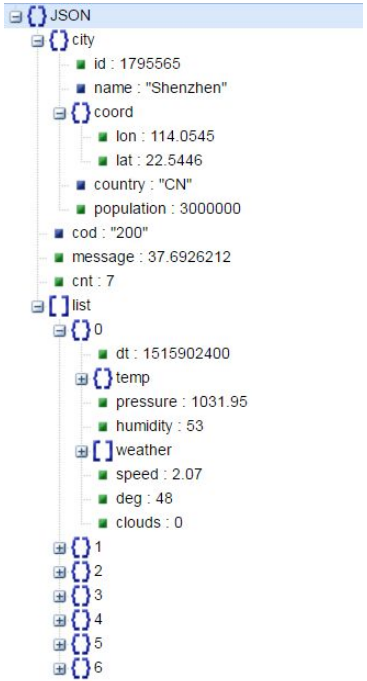
```
class ForecastByCityRequest(private val cityName: String, val gson: Gson = Gson()) {
    ……
    fun execute(): ForecastResult {
        val forecastJsonStr = URL(COMPLETE_URL + cityName).readText()
        return gson.fromJson(forecastJsonStr, ForecastResult::class.java)
    }
}
```

- 数据类

```
data class ForecastResult(val city: City, val list: List<Forecast>)
```

Json访问

- Json数据视图



- 数据类定义

```
data class ForecastResult(val city: City, val list: List<Forecast>)
data class City(val id: Long, val name: String, val coord: Coordinates,
    val country: String, val population: Int)
data class Coordinates(val lon: Float, val lat: Float)
data class Forecast(val dt: Long, val temp: Temperature, val
pressure:
    Float, val humidity: Int,val weather: List<Weather>, val speed:
    Float, val deg: Int, val clouds: Int, val rain: Float)
data class Temperature(val day: Float, val min: Float, val max: Float,
    val night: Float, val eve: Float, val morn: Float)
data class Weather(val id: Long, val main: String, val description:
    String, val icon: String)
```

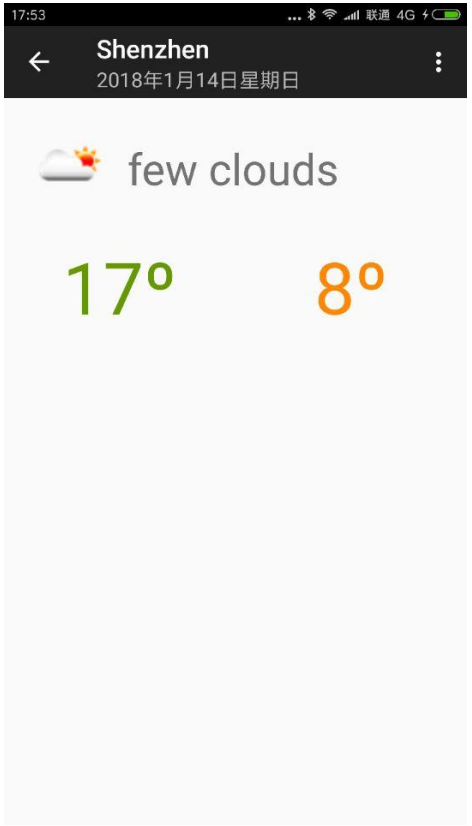
异步调用

- 协程
- 网络请求在异步线程
- 网络数据返回前，**挂起**updateUI操作
- 软件模拟
- 类同步代码

```
private fun loadForecast() = async(UI) {
    val result = bg { RequestForecastCommand(cityName).execute() }
    updateUI(result.await())
}
```

详情页

- 展示天气详情



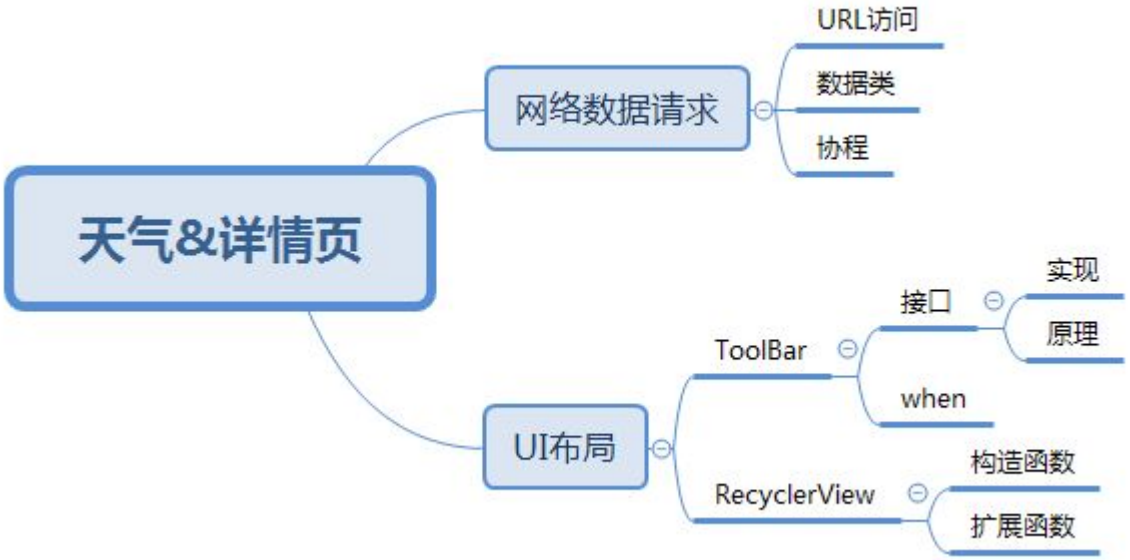
详情页

```
class DetailActivity : AppCompatActivity(), ToolbarManager {
    override val toolbar by lazy { find<Toolbar>(R.id.toolbar) }
    companion object {
        val ID = "DetailActivity:id"
        val CITY_NAME = "DetailActivity:cityName"
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_detail)
        initToolbar()
        toolbarTitle = intent.getStringExtra(CITY_NAME)
        enableHomeAsUp { onBackPressed() }
        async(UI) {
            val result = bg { RequestDayForecastCommand(intent.getLongExtra(ID, -1)).execute() }
            bindForecast(result.await())
        }
    }
    ……
}
```

详情页

```
class DetailActivity : AppCompatActivity(), ToolbarManager {
    ……
    private fun bindForecast(forecast: Forecast) {
        Picasso.with(ctx).load(forecast.iconUrl).into(icon)
        toolbar.subtitle = forecast.date.toDateString(DateFormat.FULL)
        weatherDescription.text = forecast.description
        maxTemperature.text = "${forecast.high}º"
        maxTemperature.textColor = baseContext.color(transformColor(forecast.high))
        minTemperature.text = "${forecast.low}º"
        minTemperature.textColor = baseContext.color(transformColor(forecast.low))
    }
    fun transformColor(temple: Int): Int {
        when (temple) {
            in -50..0 -> return android.R.color.holo_red_dark
            in 0..15 -> return android.R.color.holo_orange_dark
            else -> return android.R.color.holo_green_dark
        }
    }
}
```

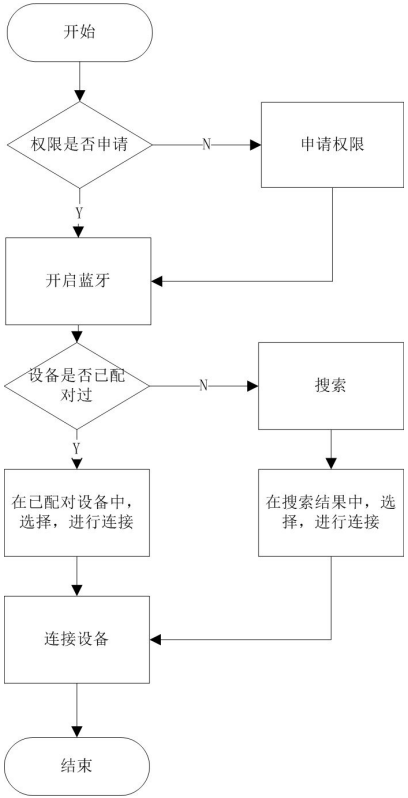
小结


天气&详情页
网络数据请求
URL访问
数据类
协程
UI布局
ToolBar
接口
实现
原理
when
RecyclerView
构造函数
扩展函数

一、Hello World
二、语法解析
三、蓝牙使用说明
四、总结

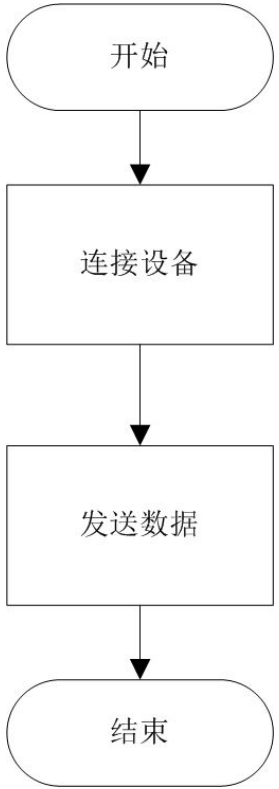
流程

- 查找设备


开始
权限是否申请
N
申请权限
Y
开启蓝牙
设备是否已配对过
N
搜索
Y
在已配对设备中，选择，进行连接
在搜索结果中，选择，进行连接
连接设备
结束

流程

- 收发数据



相关Api

- BluetoothManager

```
fun Context.getBluetoothManager(): BluetoothManager? =
    getSystemService(Context.BLUETOOTH_SERVICE) as? BluetoothManager
```

- BluetoothAdapter

```
val bluetoothAdapter = getBluetoothManager()?.adapter
```

- BluetoothDevice

```
val bluetoothAdapter = getBluetoothManager()?.adapter
adapter.let {
    val toList = it.bondedDevices.toList()
}
```

相关Api

- 获取设备

```
fun findDevice():BluetoothDevice?{
    val bluetoothManager = getSystemService(Context.BLUETOOTH_SERVICE) as?    BluetoothManager
    bluetoothManager?.let {
        val toList = it.adapter.bondedDevices.toList()
        for (i in toList) {
            if (i.name.equals("DL_BDKA"))
                return i
        }
    }
    return null
}
```

For

- 其他用法

```
// 左闭右开区间，合法值包括11，但不包括66
for (i in 11 until 66) {
    ...
}
// 每次默认递增1，这里改为每次递增4
for (i in 23..89 step 4) {
    ...
}
// for循环默认递增，这里使用downTo表示递减
for (i in 50 downTo 7) {
    ...
}
```

连接

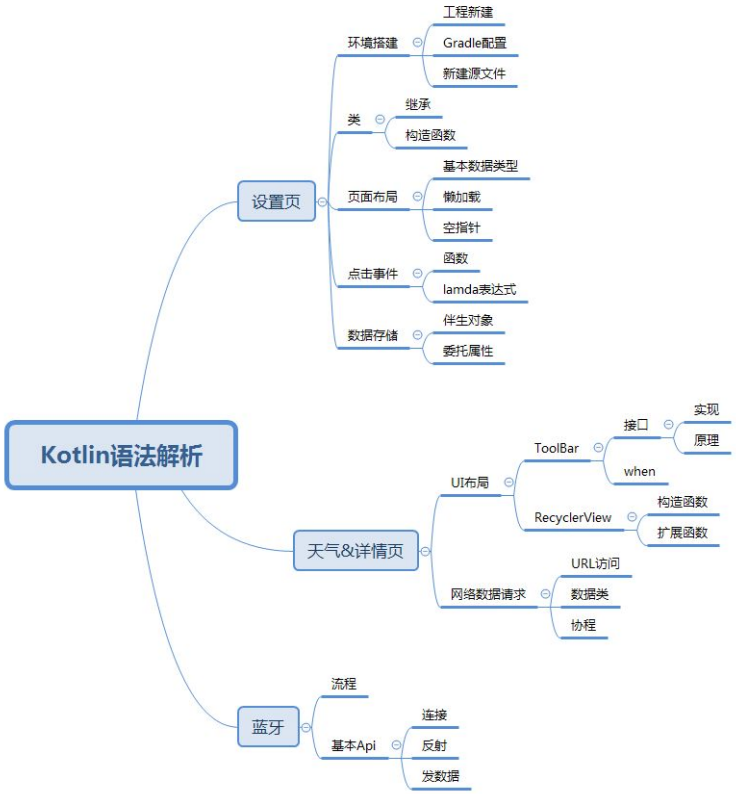
```
private fun connect(doSomething: () -> Unit) {
    async {
        try {
            val uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB")
            socket = device.createInsecureRfcommSocketToServiceRecord(uuid)
            getBluetoothManager()?.adapter?.cancelDiscovery()
            socket?.connect()
        } catch (e: IOException) {
            val clazz = BluetoothDevice::class.java
            val paramTypes = arrayOf<Class<*>>(Integer.TYPE)
            try {
                val createRfcommSocket = clazz.getMethod("createRfcommSocket", *paramTypes)
                socket = createRfcommSocket.invoke(socket?.remoteDevice, 1) as? BluetoothSocket
                socket?.connect()
            } catch (e2: Exception) {
                throw IOException(e2.message)
            }
        }
    }
}
```

发送数据

```
private fun sendMsg(msg: String = MSG_OPEN) {
    if (TextUtils.isEmpty(msg)) {
        return
    }
    val os: OutputStream? = socket?.outputStream
    try {
        os?.let {
            it.write("$msg".toByteArray())
            it.flush()
            it.close()
        }
    } catch (e: IOException) {
        Log.e(TAG, "error on sending message", e)
    } finally {
        socket?.close()
        socket = null
    }
}
```

一、Hello World
二、语法解析
三、蓝牙使用说明
四、总结

内容总结


Kotlin语法解析
设置页
环境搭建
工程新建
Gradle配置
新建源文件
类
继承
构造函数
页面布局
基本数据类型
懒加载
空指针
点击事件
函数
lamda表达式
数据存储
伴生对象
委托属性
天气&详情页
UI布局
ToolBar
接口
实现
原理
when
RecyclerView
构造函数
扩展函数
网络数据请求
URL访问
数据类
协程
蓝牙
流程
基本Api
连接
反射
发数据

资料汇总

- 源码
  - https://github.com/JetBrains/kotlin
- 文档
  - http://www.kotlincn.net/docs/reference/
- 训练
  - https://try.kotlinlang.org

Kolin中文站

欢迎加入！

中文站
https://kotliner.cn/
中文博客
https://blog.kotliner.cn/
微信公众号
Kotlin
GitHub
https://github.com/enzowyf/Okhttp4k
https://github.com/enbandari/QCloudImageUploaderForMarkDown

QQ群：162452394

微信公众号：Kotlin

应用宝技术历程

- 性能优化
- 模块化
- 动态化
- 代码优化

用户体验

快速触达用户

研发效率、维护成本