# Dart编程

## Dart特点

Dart 文件以 .dart 结尾。

Dart 属于动态的编译型语言，与大部分的编译型语言一样以 main 函数作为程序入口：

|  |
| --- |
| void main() {  print("Hello World!!!");  } |

在 Dart 中，所有东西都是对象，比如num、null、Function 等，它们都继承自Object，如果一个变量没有进行初始化，则默认值全部为 null；由于所有东西都是对象，因此数字、字符串都可以调用各种方法。

## Dart注释

多种注释格式：

|  |
| --- |
| main() {  // 单行注释  /\*  \* 多行注释  \* \*/  /// 单行文档注释  /\*\*  \* 多行文档注释  \* \*/  /// 编写高质量Dart程序-文档注释参考  /// https://blog.csdn.net/marswill/article/details/81781928  ///  } |

## Dart变量

### 变量格式

在 Dart 中，定义变量的标准格式为：

|  |
| --- |
| [const|final] [var|type] 变量名 = 变量值 |
| 如果使用 [const|final] 则 [var|type] 可以省略； |
| 如果没用 [const|final] 则 [var|type] 不可省略； |
| 如果使用 var 则程序在运行时，会自动推导变量的数据类型，比如：  var name = "Orange";  程序在运行时，会自动推断name变量的数据类型为 String 类型。 |

### final 与 const 的异同点

|  |  |
| --- | --- |
| 相同点 | const 和 final 都是定义常量的，值不能被修改，且在声明时就必须初始化； |
| 不同点 | const 定义的是编译时常量，且必须是编译时就能确定值的常量，因此只能用编译时常量来初始化；  final 定义的是运行时常量，因此可以用变量来初始化； |

代码示例：

|  |
| --- |
| //new DateTime.now() 不是 const 常量  //const time\_const = new DateTime.now(); //这个定义有问题  final time\_final = new DateTime.now(); //这个定义没问题 |

另外，const 不管是在 = 的左边还是右边，都必须是常量，以下代码无法通过编译：

|  |
| --- |
| //The values in a const list literal must be constants.  //在 const 修饰的列表里面的列表项必须是常量  //final f\_list\_3 = const [new DateTime.now(), 2, 3]; |

## Dart数据类型

|  |  |
| --- | --- |
| num | 整型 |
| double | 双精度实型 |
| bool | 布尔型 |
| String | 字符串 |
| List | 列表 |
| Map | 字典 |

### String字符串

字符串的几种表示方式：

|  |
| --- |
| String str = 'Hello World'; // 单引号 |
| String str = "Hello World"; // 双引号 |
| //使用三个单引号或者双引号可以多行字符串赋值  String str = """Dart Lang Hello World!!!"""; |

如果使用的是双引号，可以内嵌单引号；当然，如果使用的是单引号，可以内嵌双引号，否则需要“\”转义，如下示例：

|  |
| --- |
| String str\_1 = 'It\'s ok!';  String str\_2 = "It's OK!"; |

字符串的连接方式：

|  |
| --- |
| //字符串连接方式一  //在 Dart 中，相邻的字符串在编译的时候会自动连接  //如果多个字符串相邻，中间的字符串不能为空，否则报错  //但是如果单引号和双引号相邻，即使是空值也不会报错  //var name = 'Wang''''Jianfei'; 报错  String name = 'Wang'' ''JianFei';  var name\_2 = 'Hello'"World";  var name\_1 = "Hello" "World"; //正确  var name\_2 = 'Hello' 'World'; //正确  //字符串连接方式二  print("Name: $name");  //字符串连接方式三  var str = str\_1 + str\_2;  print(str);  //字符串连接方式四 ${}  var hex = 0xDEADBEEF;  print("整型转换为16进制：$hex —> 0x${hex.toRadixString(16).toUpperCase()}");  //声明原始字符串，直接在字符串前加字符 r 可避免 \ 的转义作用，在正则里特别有用 print(r"换行符：\n"); |

## Dart运算符

以下列出 Dart 的运算符，从高到低按照优先级排列：

|  |  |
| --- | --- |
| 描述 | 运算符 |
| 一元运算符（后置） | expr++ expr-- () [] . ?. |
| 一元运算符（前置） | -expr !expr ~expr ++expr --expr |
| 乘法运算符 | \*  /  %  ~/ |
| 加法运算符 | +  - |
| 位移运算符 | <<  >> |
| 按位与 | & |
| 按位异或 | ^ |
| 按位或 | | |
| 关系与类型检测 | >=  >  <=  <  as  is  is!  in  not in |
| 平等 | ==  != |
| 逻辑与 | && |
| 逻辑或 | || |
| 是否为null | ?? |
| 三目运算符 | expr1 ? expr2 : expr3 |
| 级联运算符 | .. |
| 赋值运算符 | =  \*=  /=  ~=  %=  +=  -=  <<=  >>=  &=  ^=  |=  ??= |

### 比较运算符

|  |
| --- |
| main() {  /\*\*  \* 比较运算符  \*/  print(2 == 2);  print(2 != 2);  print(2 > 2);  print(2 < 2);  print(2 >= 2);  print(2 <= 2);  } |

### 算术运算符

|  |
| --- |
| main() {  /\*\*  \* 算术运算符  \*/  print(5 + 2); // 计算 5 加 2  print(5 - 2); // 计算 5 减 2  print(5 \* 2); // 计算 5 乘 2  print(5 / 2); // 计算 5 除 2  print(5 ~/ 2); // 计算 5 整除 2  print(5 % 2); // 计算 5 模以 2  } |

### 自增自减运算符

|  |
| --- |
| main() {  /\*\*  \* 自增运算符 ++  \* 自减运算符 --  \*/  int age = 20;  int num\_1 = ++age;  print(num\_1);  print(age);  int num\_2 = age++;  print(num\_2);  print(age);  int num\_3 = --age;  print(num\_3);  print(age);  int num\_4 = age--;  print(num\_4);  print(age);  } |

## 流程控制

流程控制语句基本上跟其它编程语言类似，在 Dart 中的流程控制语句包括：

|  |  |
| --- | --- |
|  |  |
| 条件语句 | if else  switch case |
| 循环语句 | for  while  do-while |
| 跳转语句 | break  continue  return |
| 断言语句 | assert |
| 异常语句 | try-catch |

### 条件语句

#### if和else

示例代码：

|  |
| --- |
| main() {  bool is\_raining = true;  bool is\_snowing = false;  if (is\_raining) {  print("请带雨伞吧！");  } else if (is\_snowing) {  print("今天，你可以穿雪地靴了！");  } else {  print("今天是个好晴天。");  }  } |

在 Dart 中的条件必须使用布尔值或者 null。

### 循环语句

#### for循环

##### 普通for 循环

示例代码：

|  |
| --- |
| main() {  var message = StringBuffer('Dart 是一门有趣的编程语言！');  print(message);  for (var i = 0; i < 5; i++) {  message.write('!');  }  print(message);  } |

##### forEach 循环

闭包在 Dart 的 for 循环中会捕获循环的 index 索引值，如下代码所示：

示例代码：

|  |
| --- |
| main() {  var callbacks = [];  for (var i = 0; i < 10; i++) {  callbacks.add(() => print(i));  }  callbacks.forEach((c) => c());  } |

##### for in 循环

示例代码：

|  |
| --- |
| main() {  var collection = [0, 1, 2, 3, 4, 5, 6, 7, 8];  for (var x in collection) {  print(x); // 0 1 2  }  } |

## Dart函数

### 普通函数

|  |
| --- |
| String sayHello(String name) {  return "Hello ${name}";  } |

### 函数别名

|  |
| --- |
| typedef int Add(int a, int b);  int Subtract(int a, int b) => a - b;  //我们在 main 函数中写如下代码：  void main() {  print(Subtract is Function); //true  print(Subtract is Add); //true  } |

### 匿名函数

|  |
| --- |
| void main() {  var sayHellos = (name) => "Hello $name";  print(sayHellos("Orange"));  } |

### 闭包函数

|  |
| --- |
| Function makeSubstract(num n) {  return (num i) => n - i;  } |

代码演示：

|  |
| --- |
| var x = makeSubstract(5);  print(x(2)); |

闭包函数的一个有趣的例子：

|  |
| --- |
| void main() {  var call\_backs = [];  for (var i = 1; i <= 10; i++) {  call\_backs.add(() => print("Save $i"));  }  call\_backs.forEach((j) => j());  } |

## Dart函数参数

### 形式参数

### 可选参数

### 位置参数

### 实际参数

## Dart异常处理

基本格式：

|  |
| --- |
| void mian() {  try {    } catch (e) {    } finally {    }  } |

## Dart安装依赖库

在项目根目录下创建 pubspec.yaml，然后在该文件填写如下格式的代码：

|  |
| --- |
| {  name: cnDartLang,  description: 这个是项目依赖描述信息,  dependencies:  {  http: ^0.12.0+4,  date\_format: ^1.0.6,  yaml: ^2.2.0,  }  } |

然后就会自动安装依赖库了。

# Flutter编程

## Flutter资料

SDK 下载地址：

<https://flutter.dev/docs/development/tools/sdk/releases>

Flutter入门实战：从0到1仿写web版掘金App：

<https://juejin.im/post/5c910bd55188252da05f3f05>

Flutter实战 <https://book.flutterchina.club/>

《Flutter实战》随书源码：

<https://github.com/wendux/flutter_in_action_source_code>

掘金 Flutter <https://juejin.im/tag/Flutter>

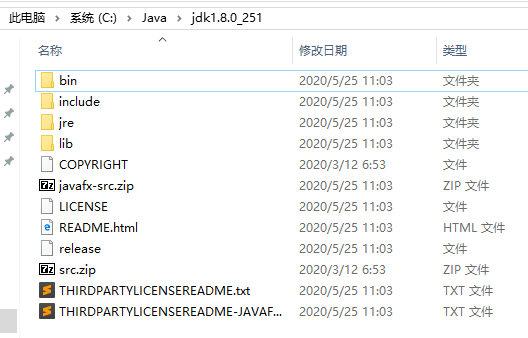
Flutter中文网 <https://flutterchina.club/get-started/install/>

Flutter从入门到精通 收费 <https://ke.qq.com/course/package/16132>

## Windows系统搭建Flutter开发环境

### 安装和配置JDK

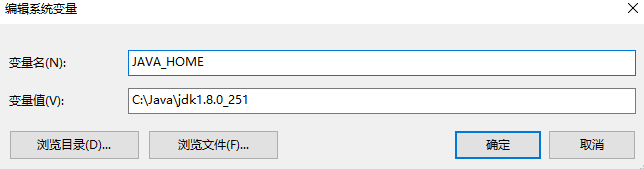
这里安装和配置的是C:\Java\jdk1.8.0\_251版本，如下图所示：



**新建环境变量**：

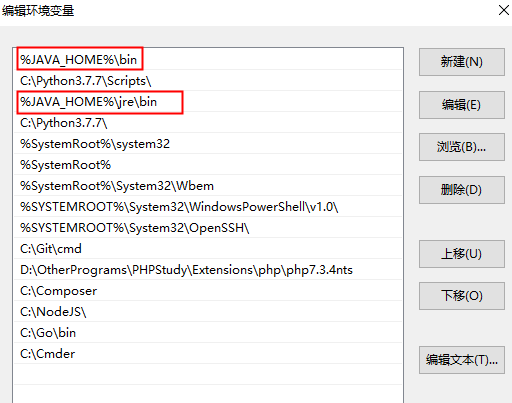
|  |  |
| --- | --- |
| 变量名 | JAVA\_HOME |
| 变量值 | C:\Java\jdk1.8.0\_251 |

如下图所示：

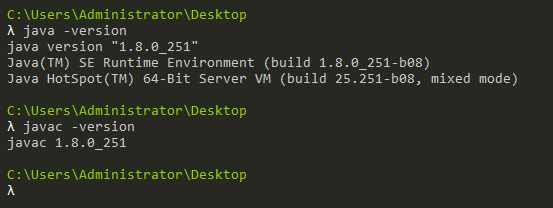


**配置Path环境变量**：

在Path环境变量中添加 %JAVA\_HOME%\bin 和 %JAVA\_HOME%\jre\bin 并删除环境变量中的C:\Program Files (x86)\Common Files\Oracle\Java\javapath 变量，如下图所示：



在命令行输入 java 和 javac 命令来验证 JDK 配置是否成功，如下所示：



### 安装和配置 Flutter

#### 下载和安装 Flutter

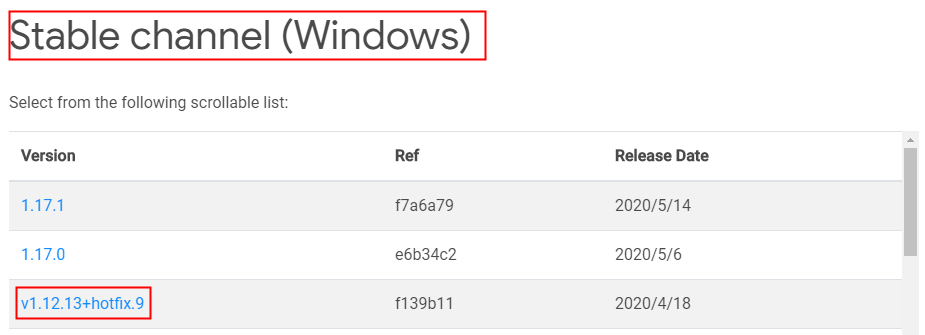
Flutter SDK 下载地址：

<https://flutter.dev/docs/development/tools/sdk/releases>

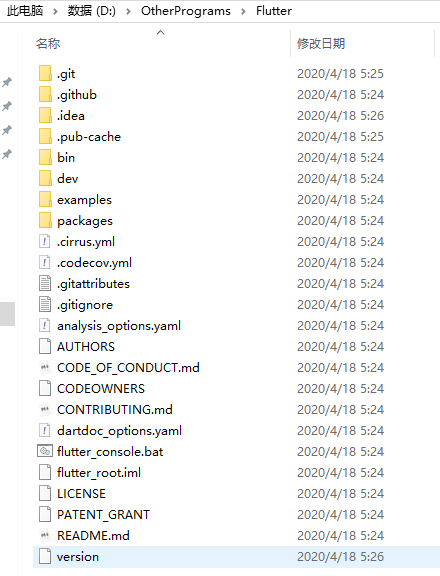
我这里下载Stable channel (Windows)版本，我具体下载的版本是v1.12.13 + hotfix.9版本，具体下载链接为：

<https://storage.googleapis.com/flutter_infra/releases/stable/windows/flutter_windows_v1.12.13+hotfix.9-stable.zip>

如下图所示：



下载下来之后，解压缩到D:\OtherPrograms\Flutter目录下，如下图所示：

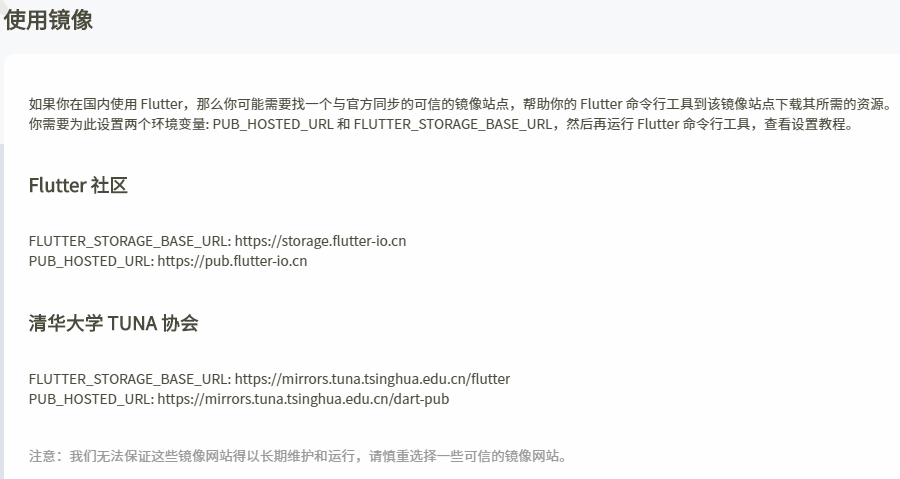


#### 配置 Flutter 环境变量

##### 配置 Flutter 镜像变量

Flutter 中文站：<https://flutter.cn/>

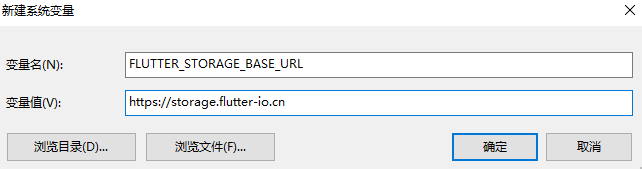
我们打开中文站地址，把页面拉到最下面，看到下图所示：

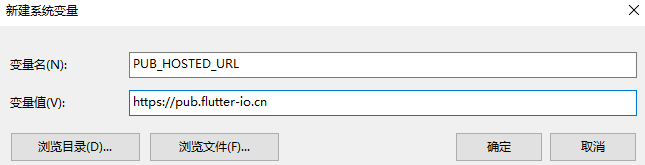


根据上图所示，我们新建如下两个变量：

|  |  |
| --- | --- |
| 变量名 | 变量值 |
| FLUTTER\_STORAGE\_BASE\_URL | https://storage.flutter-io.cn |
| PUB\_HOSTED\_URL | https://pub.flutter-io.cn |

如下图所示：

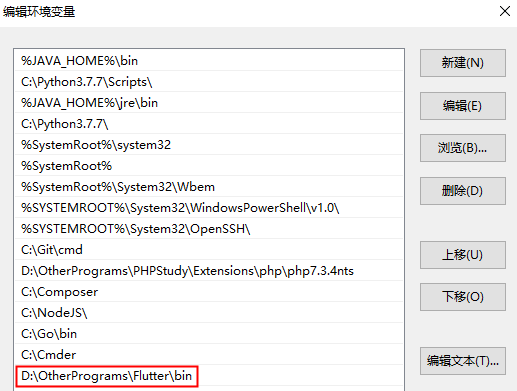




这样中文镜像站环境配置就完成了。

##### 修改 Path 环境变量

把D:\OtherPrograms\Flutter\bin 加入到系统的 Path 环境变量。

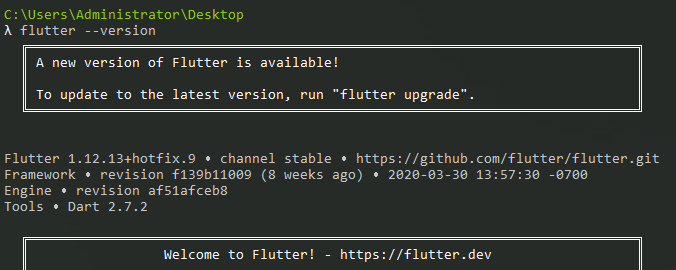


##### 检验 Flutter 环境配置

在命令行输入如下命令来验证我们的环境变量是否配置成功：

|  |
| --- |
| flutter --version |

结果如下图所示：

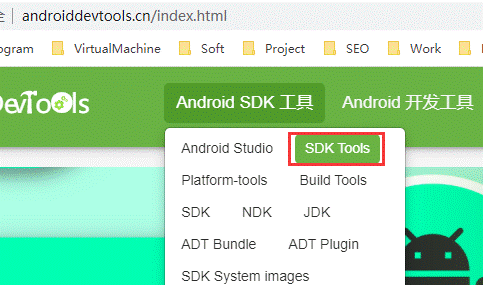


### 安装和配置 Android SDK

安装路径应该是D盘，实际截图时，不小心安装到了C盘。

#### 下载和安装 Android SDK

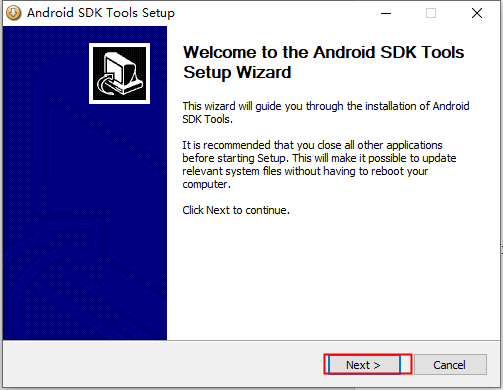
打开<https://www.androiddevtools.cn/index.html>网址，然后如下图操作：

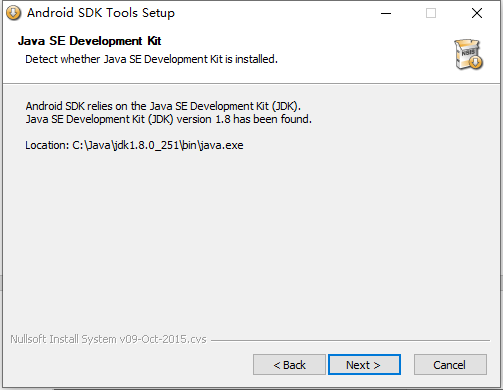


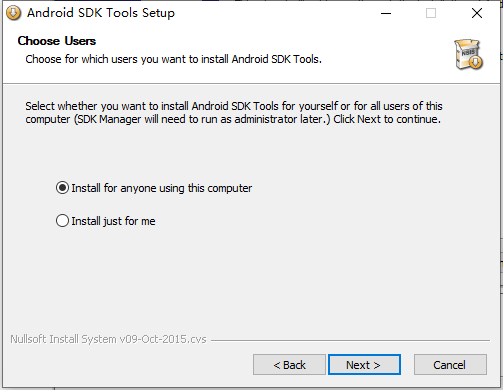
然后选择exe安装文件进行下载，如下图所示：



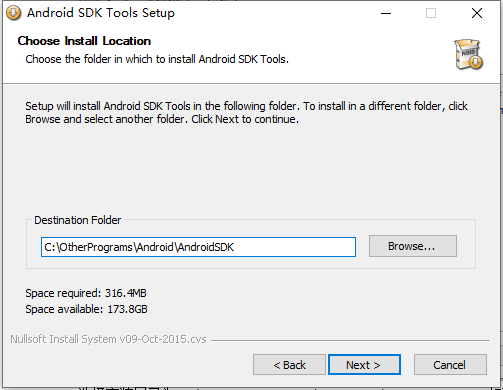
安装过程如下：

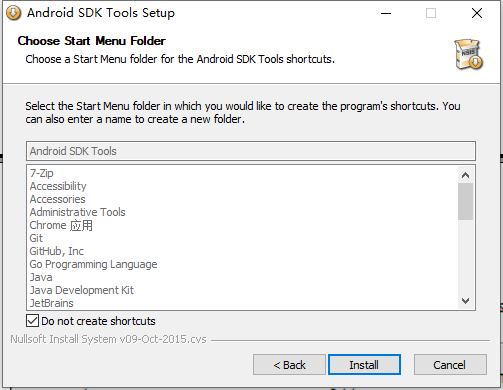


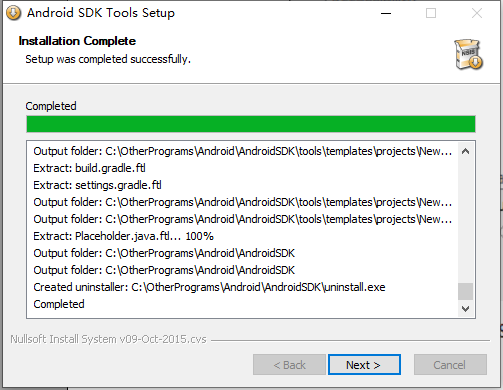




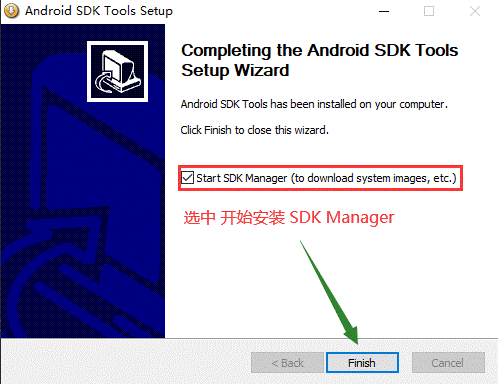
选择安装目录为D:\OtherPrograms\Android\AndroidSDK，如下图所示：



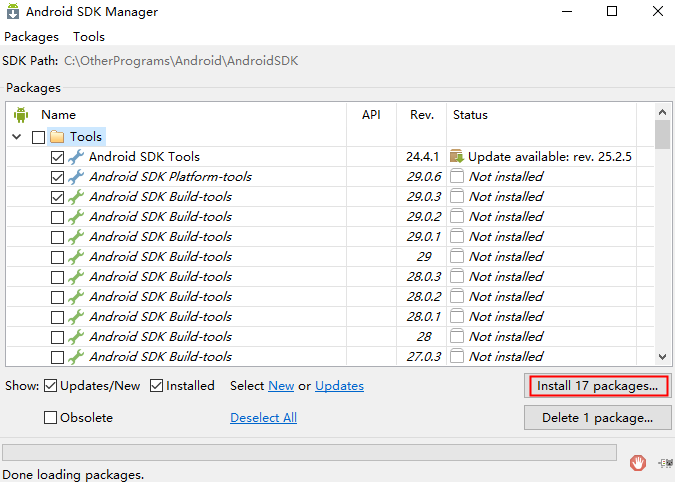




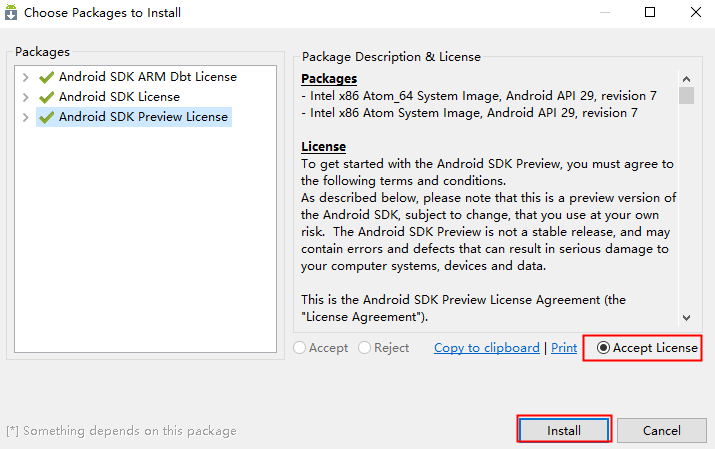
安装完成之后，选择：



#### 安装 SDK Manager

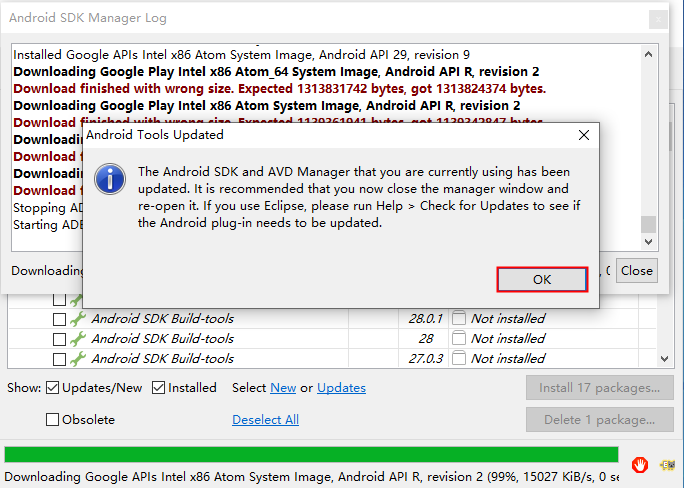


默认选中即可，然后点击 Install 进行安装：



三个部分的包都选择 Accept License，选择完成之后，点击 Install 进行安装即可。

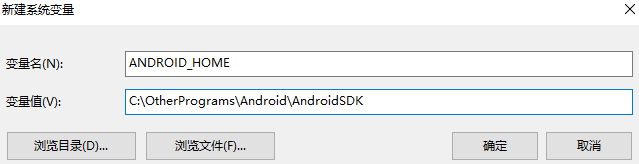
安装过程比较长，需要耐心等待安装完成，安装完成如下图所示：



在安装完成之后，我们需要做环境配置：

|  |  |
| --- | --- |
| 变量名 | 变量值 |
| ANDROID\_HOME | C:\OtherPrograms\Android\AndroidSDK |

如下图所示：

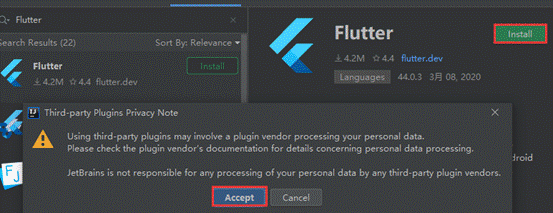


### 安装和配置 IntelliJ IDEA

#### 安装相关插件

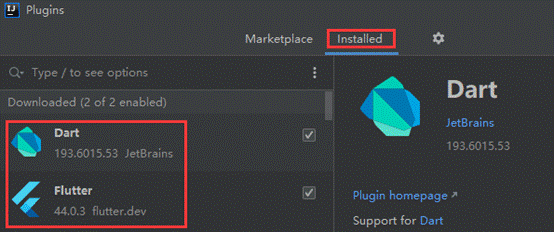
启动IntelliJ IDEA → File → Settings → Plugins → Marketplace选项卡 → 搜索 Dart 和 Flutter 两个插件进行安装。

在安装 Flutter 时，跳出如下对话框：



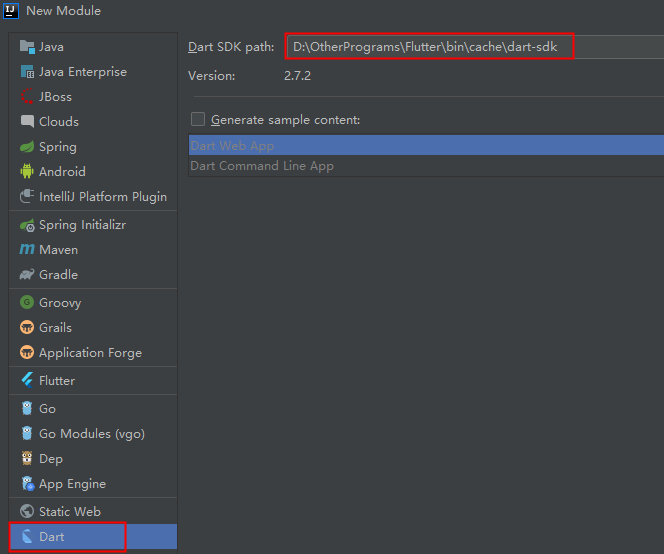
选择 Accept 即可。安装完成之后，重启 IDE。

重启之后，选择File → Settings → Plugins → Installed 选项卡，可以看到 Dart 和 Flutter 插件已经安装成功，如下图所示：



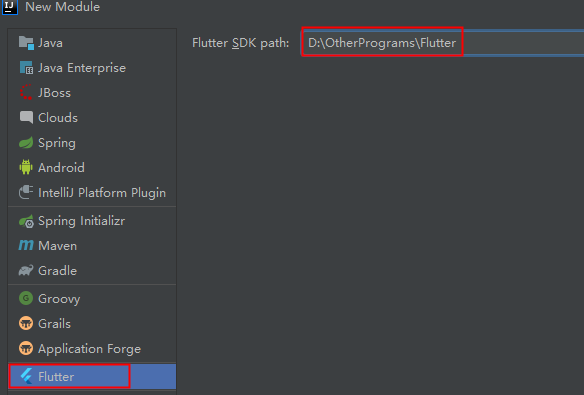
#### 配置 Dart SDK

启动IDE → File → New → Module → Dart，如下图所示：



#### 配置 Flutter SDK

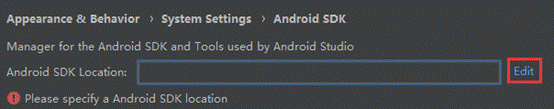
启动IDE → File → New → Module → Flutter，如下图所示：

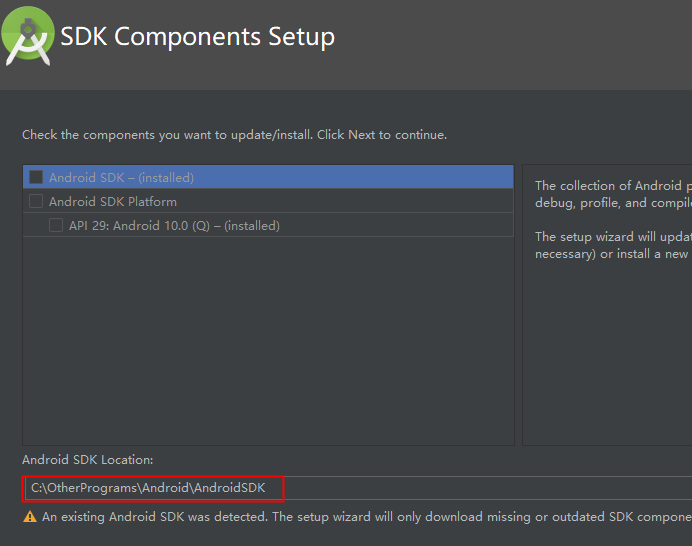


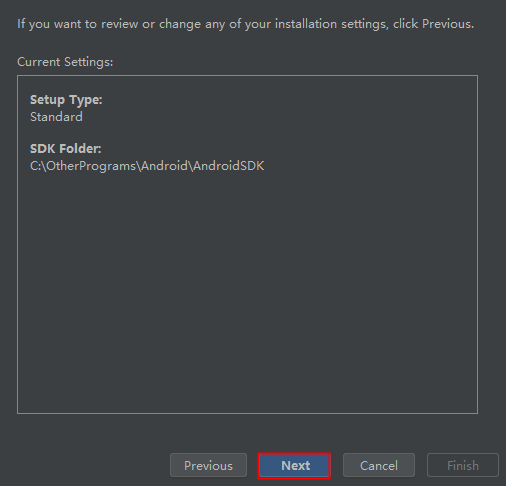
然后点击Next，最后Finish。

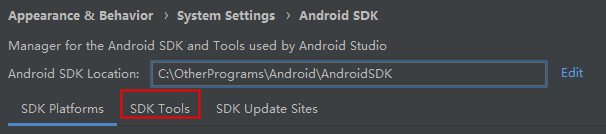
#### 配置 Android SDK

IDE → File → Settings → Appearance & Behavior → System Settings → Android SDK，如下图所示：

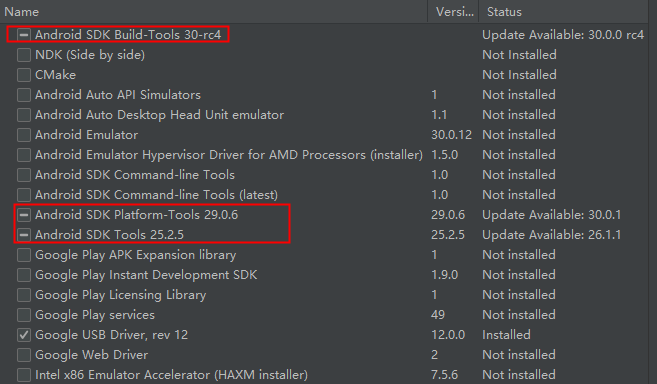




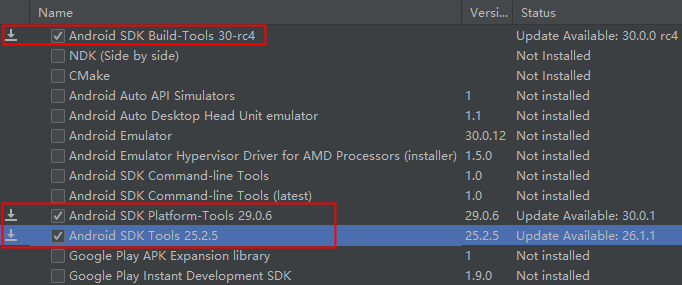




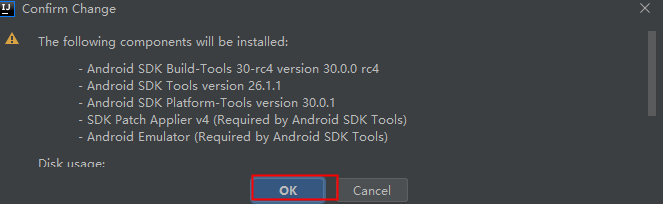
指定好 Android SDK 路径之后，选择 SDK Tools 选项卡，选中如下选项进行安装：

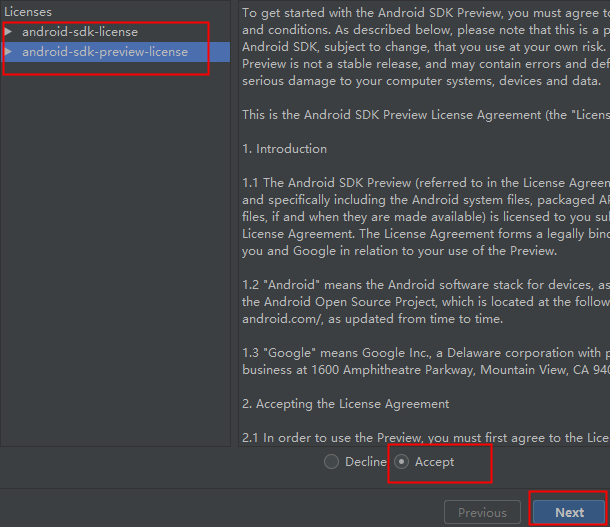


把带 - 的选中，并进行安装，如下图：

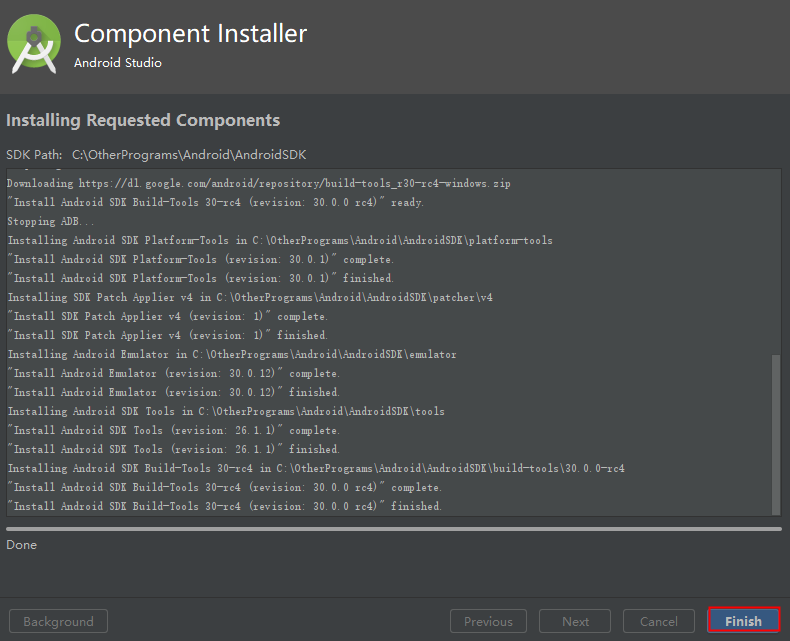


然后点击OK进行安装。在确认框点击OK即可，如图：





然后进入下载安装，安装完成后如下图所示：

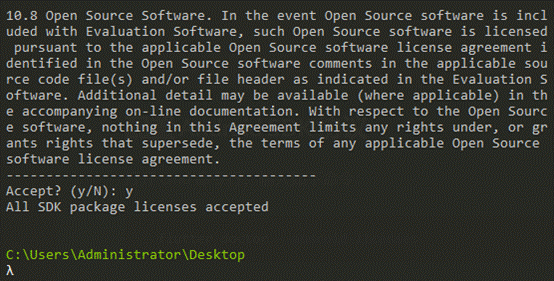


### 配置 Android SDK 协议

打开 cmd 命令行，输入如下命令：

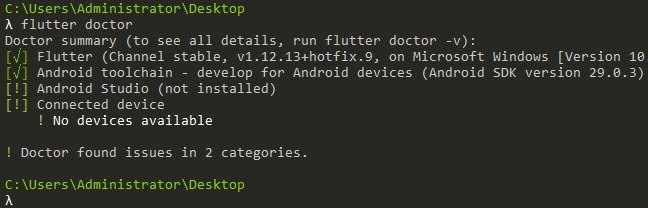
|  |
| --- |
| flutter doctor --android-licenses |

然后一直输入y即可，直到安装完成，如下图所示：



然后输入如下命令：

|  |
| --- |
| flutter doctor |



没有任何错误，说明我们的开发环境已经成功配置。

### Flutter其它命令

升级 Flutter 版本

|  |
| --- |
| flutter upgrade |

切换 Flutter 版本

|  |
| --- |
| flutter channel dev/stable/beta/master |
| 说明：  dev/stable/beta/master 分别对应 开发版/稳定版/测试版/未经测试的最新版。 |

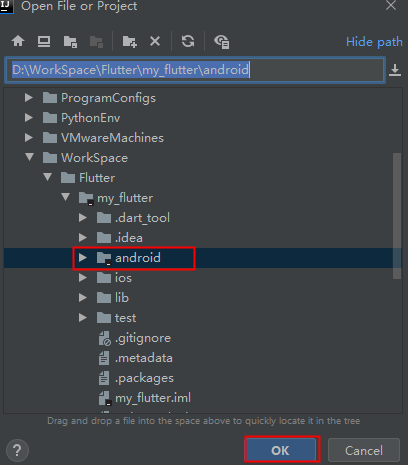
### 配置夜神手机模拟器

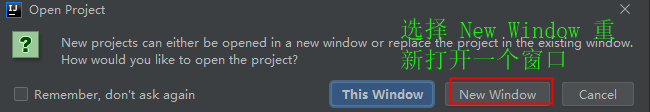
设置 → 关于平板电脑 → 版本号 → 连续点击 5 次，即可开启开发者选项 → 进入开发者选项 → 开启 USB 调试功能。

### 下载和安装Gradle插件

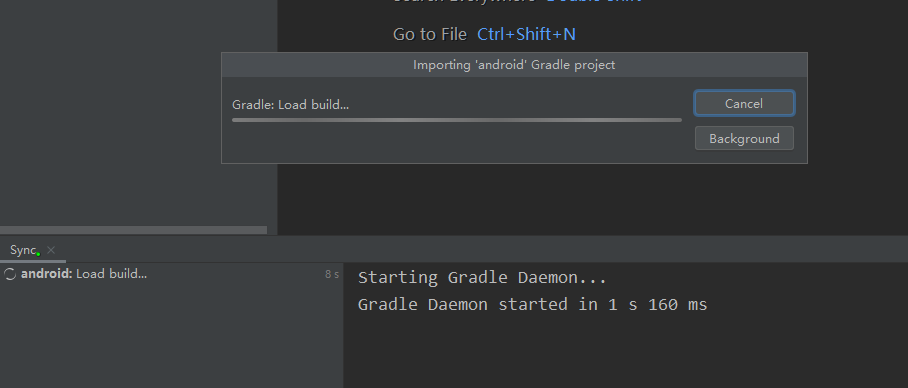
打开 flutter 项目下的 android目录，过程如下：

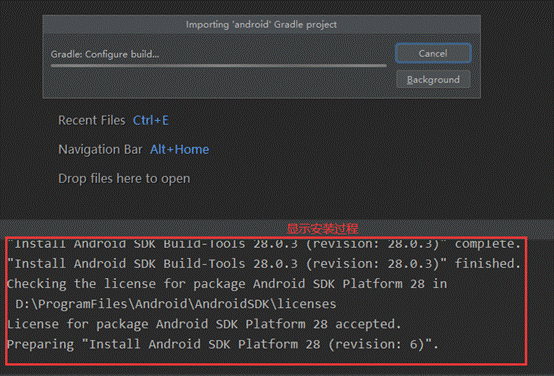
在当前IDE中 → File → Open → 选择android目录，如下图所示：

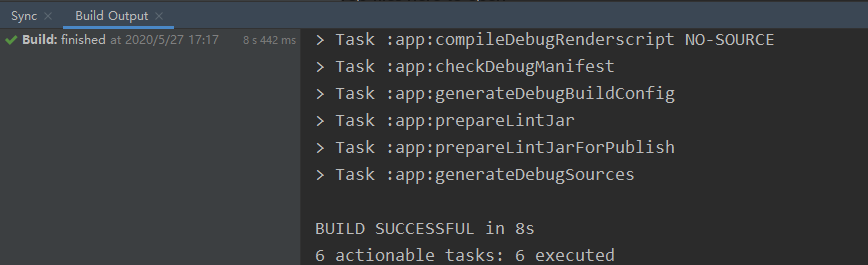




在IDE的新打开窗口中，Gradle会自动下载和安装，过程如下图所示：







下载和安装过程比较慢，需要耐心等待。

如果 Gradle 下载失败，如何进入重新下载？

IDE → File → Sync Project with Gradle Files 会重新进入下载。

### IDE 无法识别虚拟设备

使用 IntilliJ 开发 Flutter 时，发现无法自动识别虚拟设备，怎么办？

首先把手机模拟器比如夜神的执行路径配置到 Path 环境变量：

|  |
| --- |
| C:\Nox\bin |

在 IDE 的终端执行如下命令：

|  |
| --- |
| adb.exe connect 127.0.0.1:62001 |
| 或者 |
| nox\_adb.exe connect 127.0.0.1:62001 |

即可显示可连接的虚拟设备。

### 相关操作快捷键

在使用 IntelliJ 运行 Flutter 项目时，不要使用 Console 因为无法使用快捷键，应该打开 Terminal 终端，然后在项目目录下运行：

|  |
| --- |
| flutter run |

然后在终端就可以使用以下的快捷键了：

|  |  |
| --- | --- |
| r | 热重载 |
| R | 热重启 |
| h | 帮助信息 |
| p | 显示网格，方便预览布局情况 |
| o | 切换Android和IOS的预览模式 |
| q | 退出调试预览模式 |

另外，IntelliJ IDEA 可以自动重新加载修改过的代码。

### 项目创建注意事项

我们使用 IntelliJ IDEA 做为 Flutter 的开发工具。在创建 Flutter 项目时，项目名称必须全部小写，否则无法创建项目。

## Flutter 布局

## Flutter 插件

## Flutter 动画

## Flutter 路由

## Flutter 组件