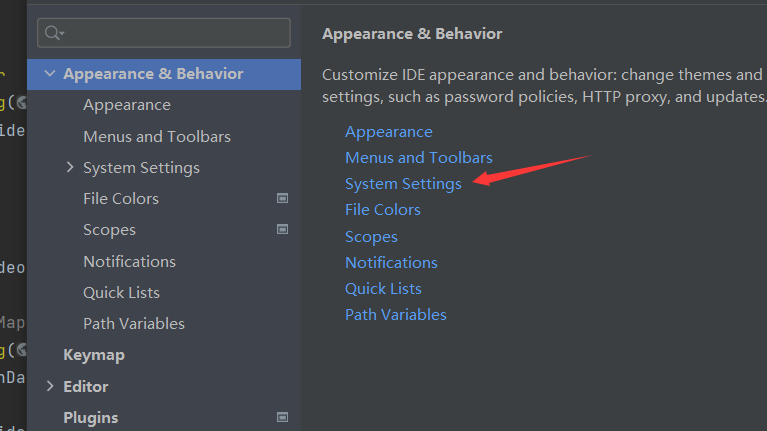
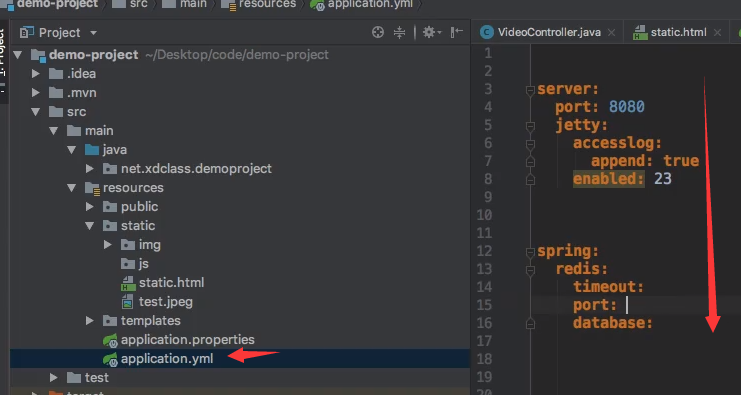
热部署：

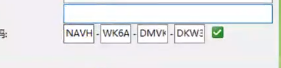


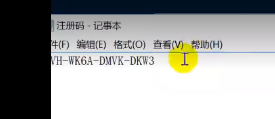


<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <optional>true</optional>  
</dependency>

<plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <configuration>  
 <fork>true</fork><!--必须添加这个配置-->  
 </configuration>  
</plugin>

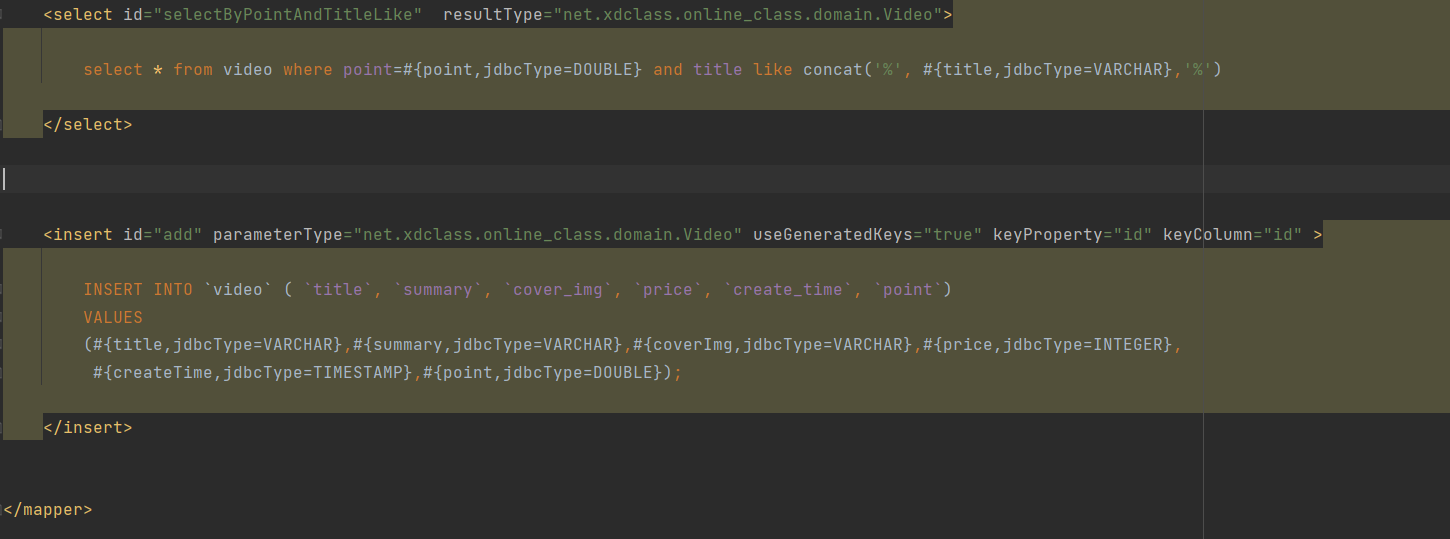


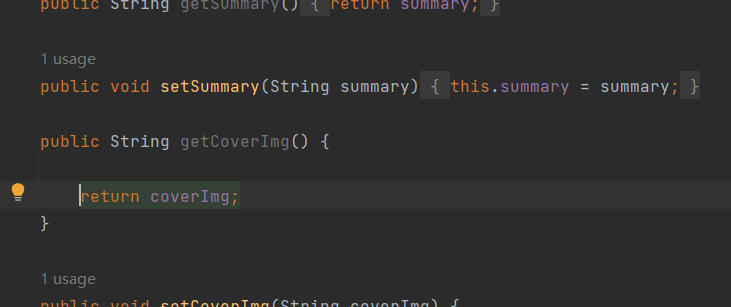


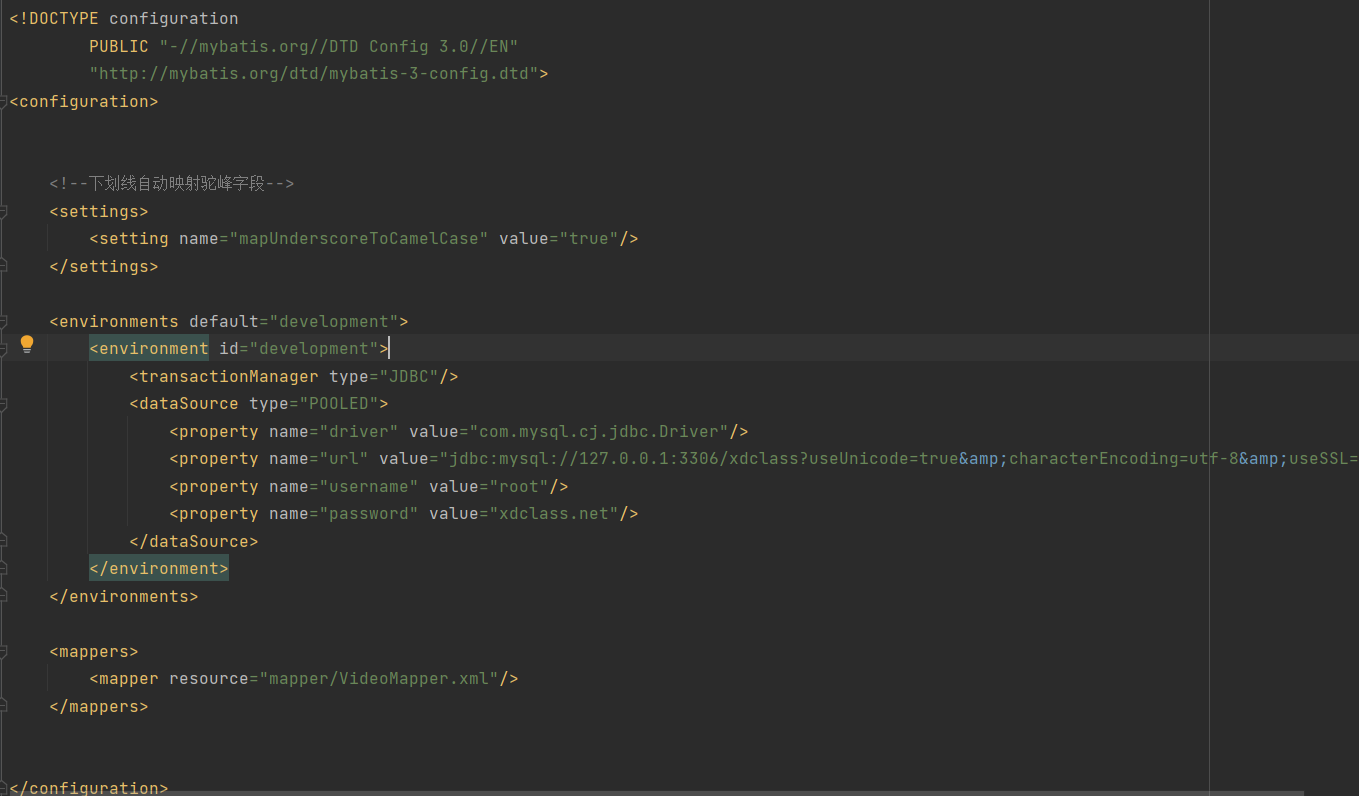


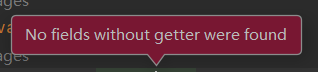
String url = "jdbc:mysql://127.0.0.1:3306/9?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimezone=Asia/Shanghai";

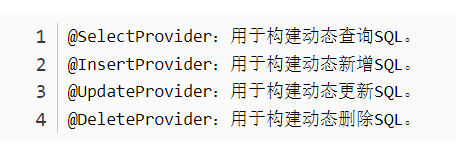
"jdbc:mysql://127.0.0.1:3306/9?  
useUnicode=true&amp;characterEncoding=utf-8&amp;useSSL=false&amp;serverTimezone=Asia/Shanghai"











如果你用了数据链接池，则在数据链接池的url一项进行修改，加上“?user=false”。

如果你是直接在方法中的conn使用jdbc，就在这个conn中加上“&useSSL=false”

//如果是在在数据连接池中（context.xml）：

<?xml version="1.0" encoding="utf-8" ?>

<Context reloadable="true">

<Resource

name="jdbc/testDS"

type="javax.sql.DataSource"

maxActive="10"

maxIdle="5"

username="root"

password="123"

maxWait="10000"

driverClassName="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost:3306/no9?useSSL=false"//这里进行了变化

/>

</Context>

//如果在直接的Conn中：

conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/jsp\_db?useUnicode=true&characterEncoding=utf-8&useSSL=false","root","123456");

//这一段其实只需要?useSSL=false即可。

//其他几段意思：useUnicode=true&characterEncoding=utf-8这一段是说编码格式为utf-8,自己可以设置eclipse的默认编码格式;

//其他的，比如spring的配置文件中：

spring:

datasource:

url: jdbc:mysql://localhost:3306/orcl?characterEncoding=utf-8&useSSL=false

所有java\_web api汇总

<http://t.zoukankan.com/ShawnYang-p-7365756.html>

<https://spring.io/projects/spring-framework>

新版SpringBoot2.X介绍

官⽹：https://spring.io/projects/spring-boot

官⽅⽂档：https://spring.io/guides/gs/spring-boot/

在线创建 ：https://start.spring.io/

https://docs.spring.io/spring-boot/docs/2.3.0.BUILD-SNAPSHOT/reference/htmlsingle/# core-properties

**讲解：SpringBoot2.x整合模板引擎thymeleaf实战**

官⽹地址：https://www.thymeleaf.org/doc/articles/thymeleaf3migration.html

简单测试代码编写和访问

注意：$表达式只能写在th标签内部

快速⼊⻔：https://www.thymeleaf.org/doc/articles/standarddialect5minutes.html

官⽅地址：https://mybatis.org/mybatis-3/

依赖地址 https://mvnrepository.com/artifact/org.mybatis/mybatis/3.5.4

核⼼流程： https://mybatis.org/mybatis-3/zh/getting-started.html

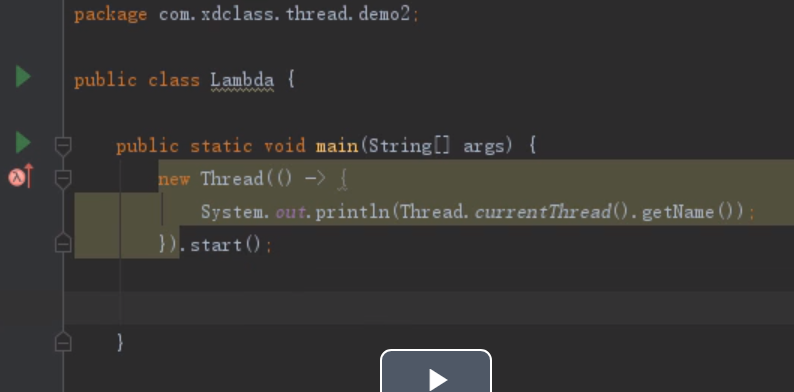
官⽅⽂档：https://mybatis.org/mybatis-3/zh/confifiguration.html#

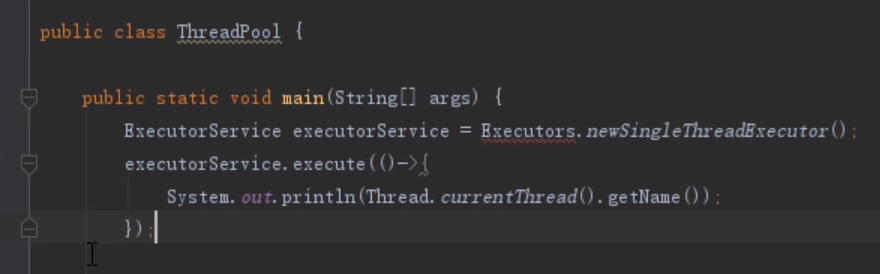
什么是Spring：轻量级的 DI / IoC 和 AOP 容器的开源框架

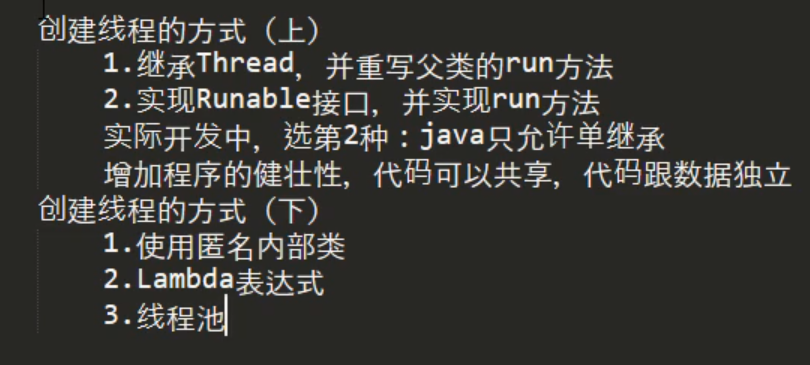
https://spring.io/projects/spring-framework

Csdn 密码：[597067358@qq.com](mailto:597067358@qq.com)

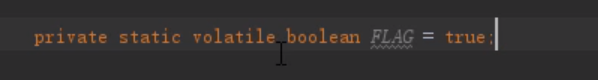
18734153812



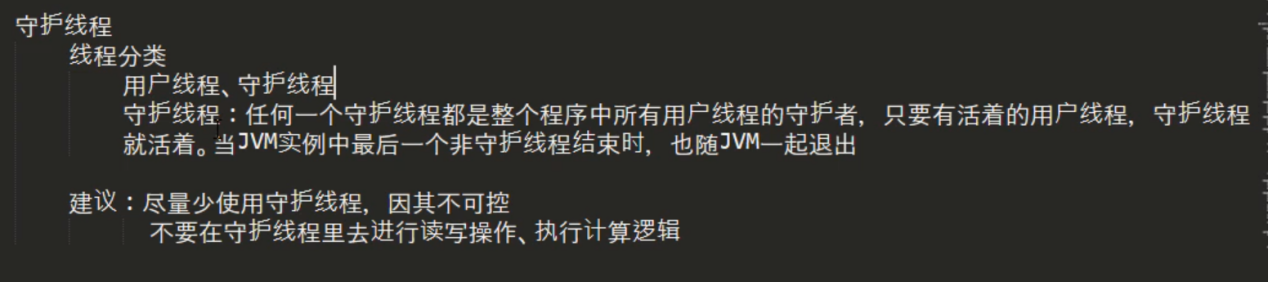


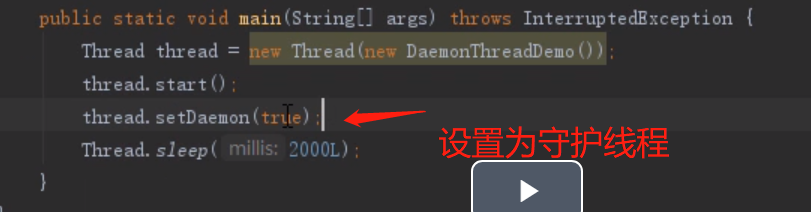


Sout

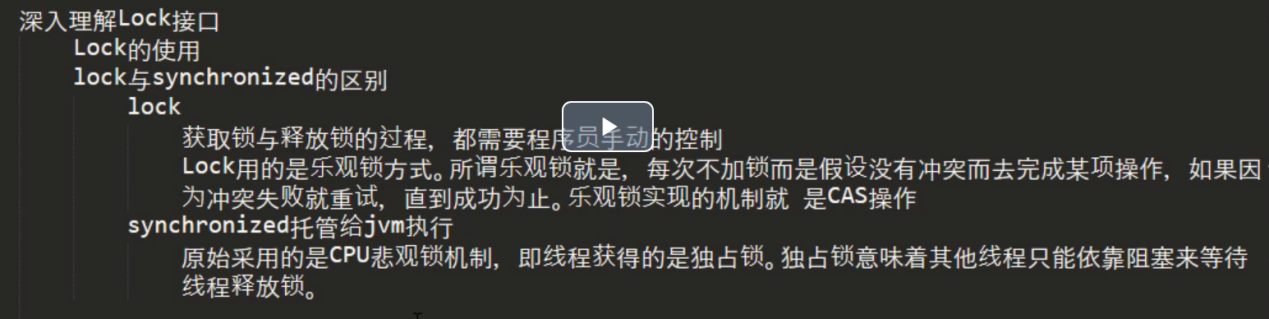


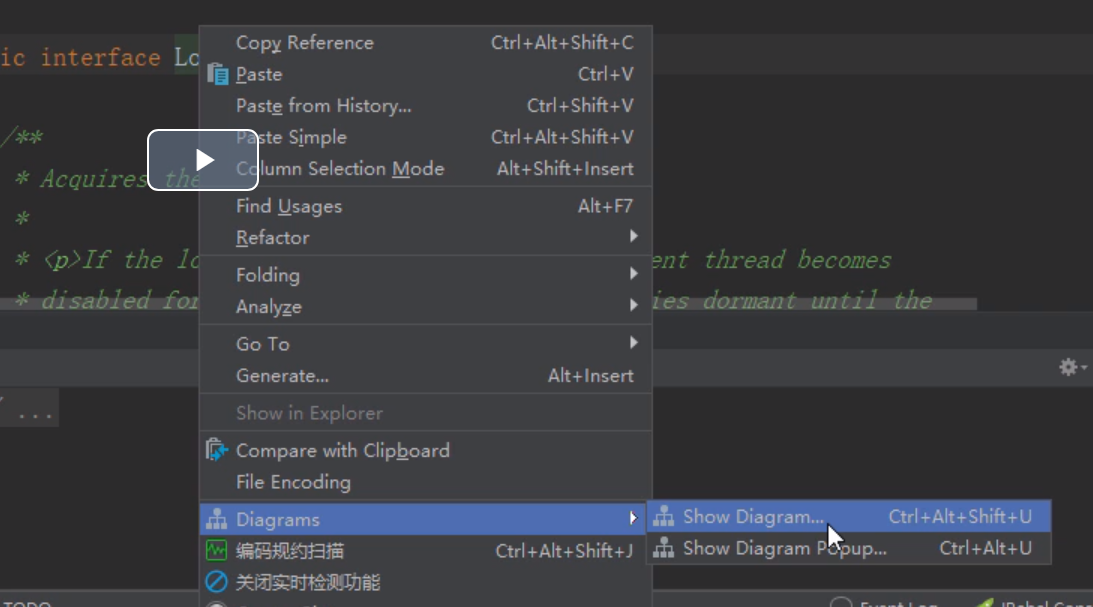
Ctrl+p查看入参

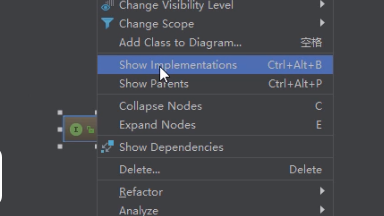


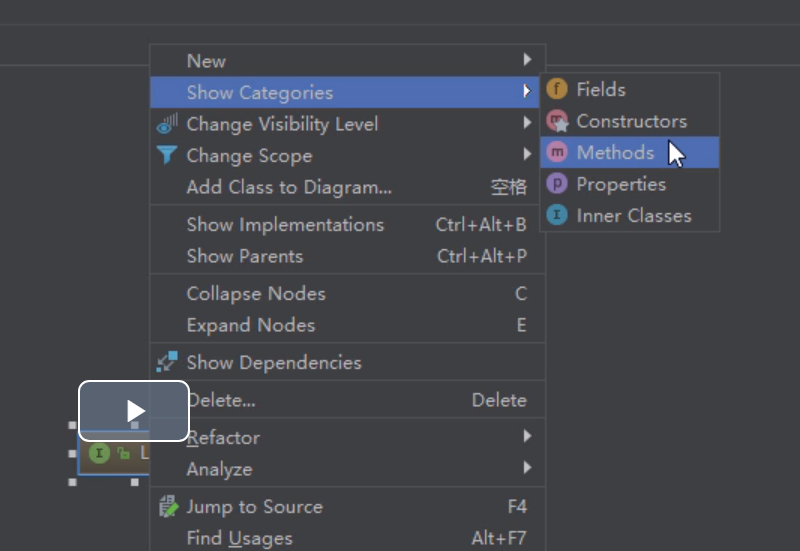


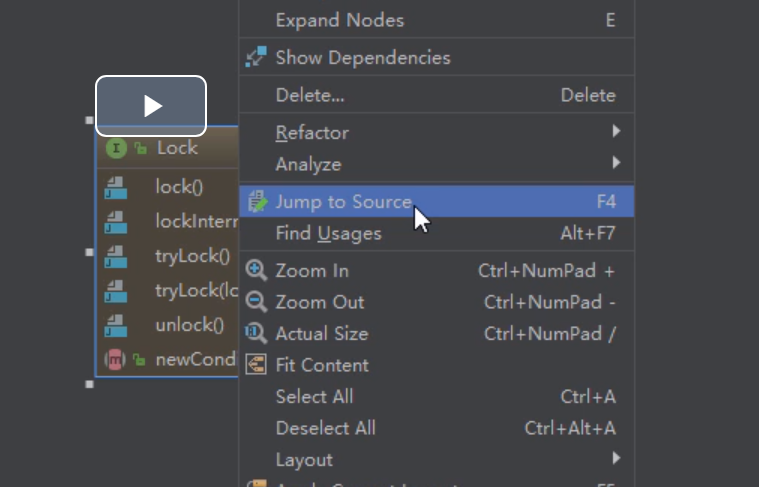


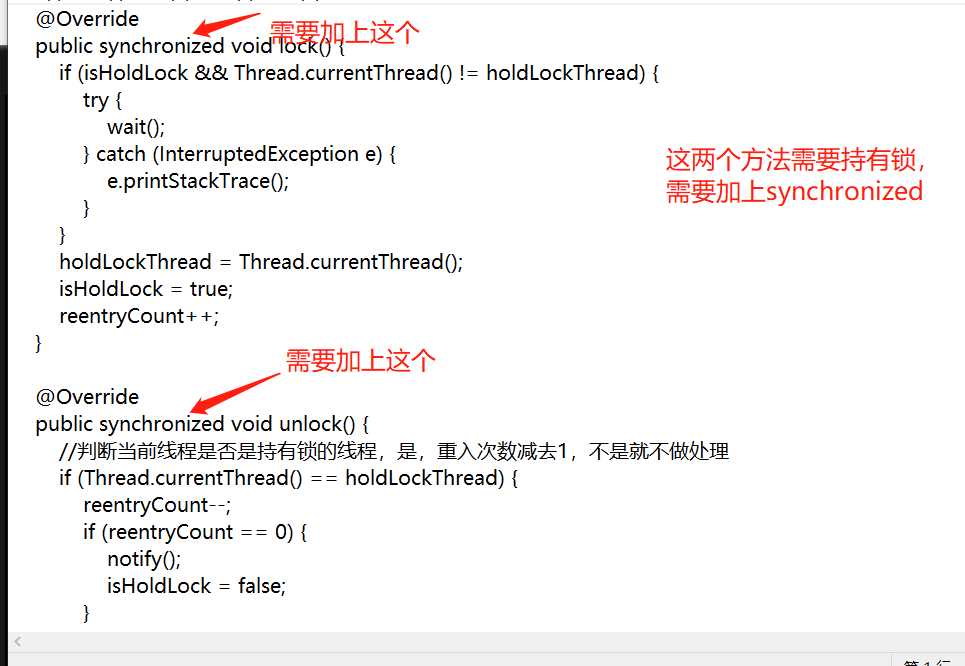












@Override

public synchronized void lock() {

if (isHoldLock && Thread.currentThread() != holdLockThread) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

holdLockThread = Thread.currentThread();

isHoldLock = true;

reentryCount++;

}

@Override

public synchronized void unlock() {

//判断当前线程是否是持有锁的线程，是，重入次数减去1，不是就不做处理

if (Thread.currentThread() == holdLockThread) {

reentryCount--;

if (reentryCount == 0) {

notify();

isHoldLock = false;

}

}

}

public class ReentryDemo {

public Lock lock = new MyLock();

public void methodA() {

lock.lock();

System.out.println("进入方法A");

methodB();

lock.unlock();

}

public void methodB() {

lock.lock();

System.out.println("进入方法B");

lock.unlock();

}

public static void main(String[] args) {

ReentryDemo reentryDemo = new ReentryDemo();

reentryDemo.methodA();

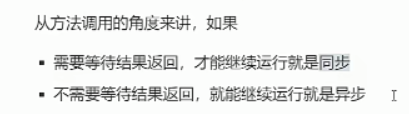
}

ctrl+alt+b看实现

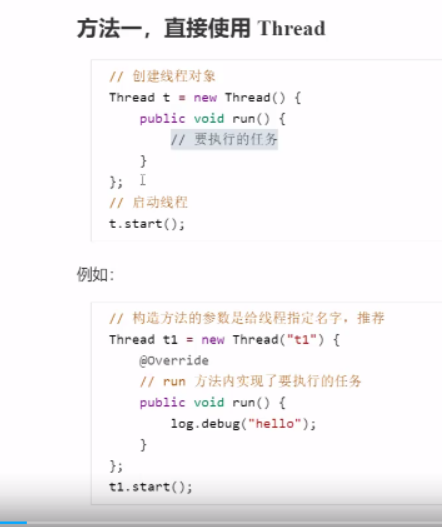
比如你去上厕所，进去之后会锁门，目的是为了保证同一时刻只有你一个人在厕所里面。

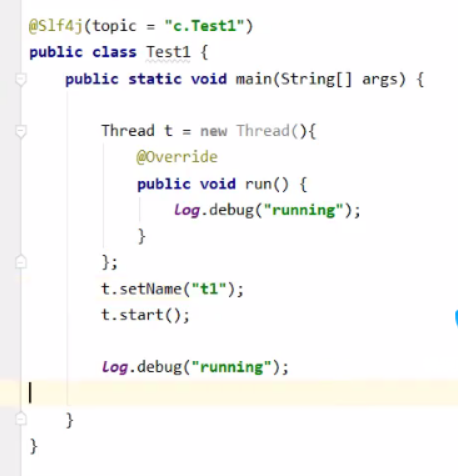
计算机中的锁目的也是这样，为了保证同一时刻只有一个线程或者进程实体能够访问某些资源。

加锁就是把并发执行的部分变成序列执行

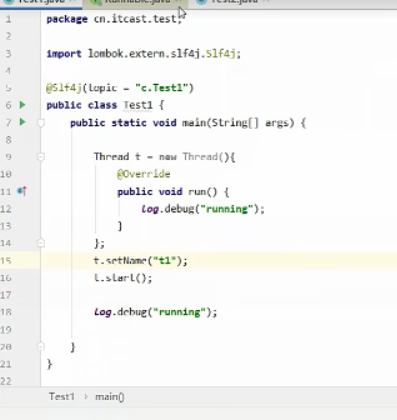


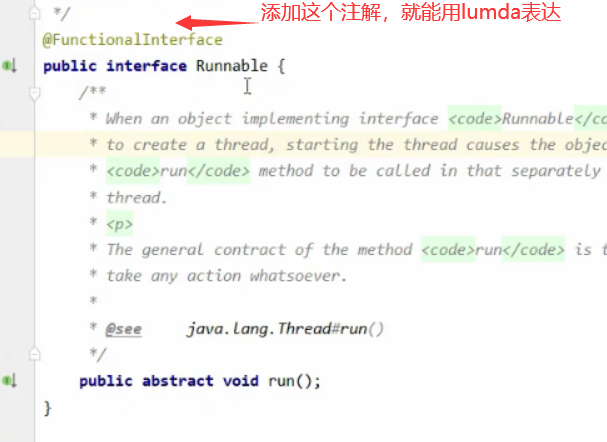


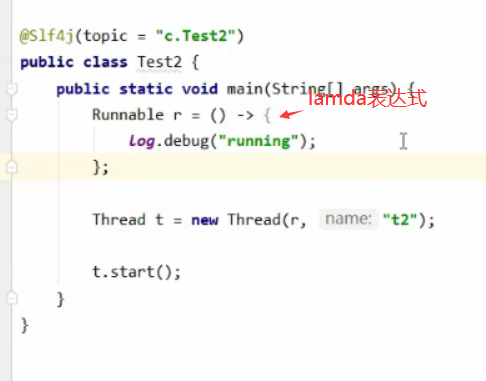


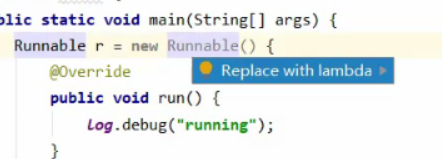




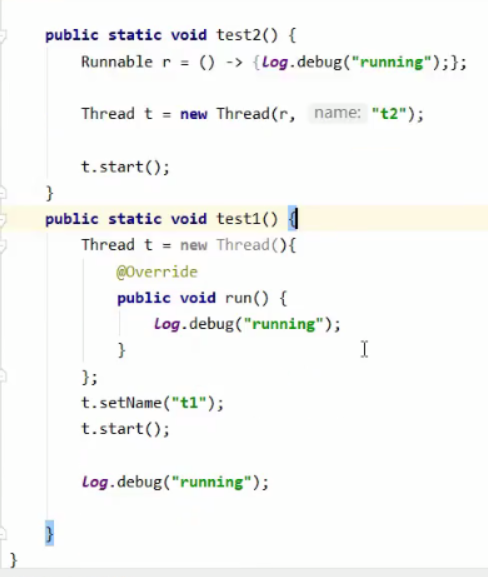




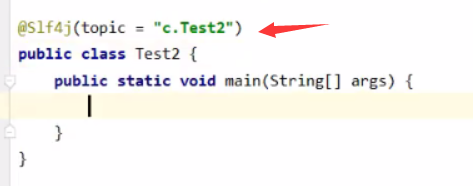


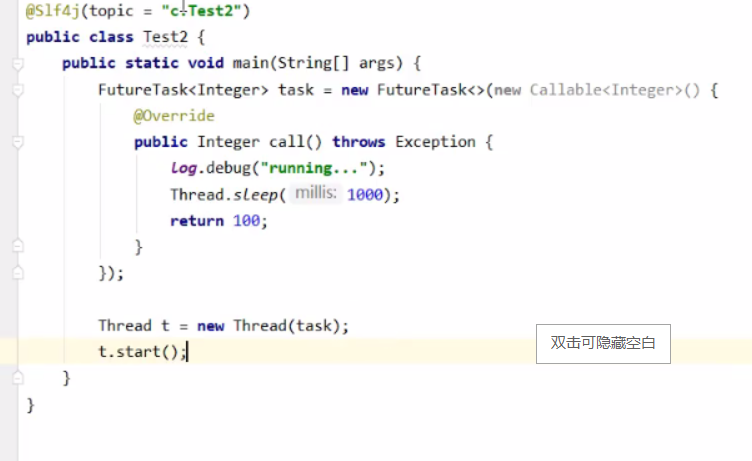


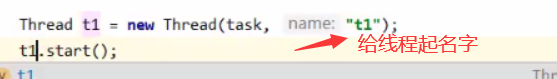


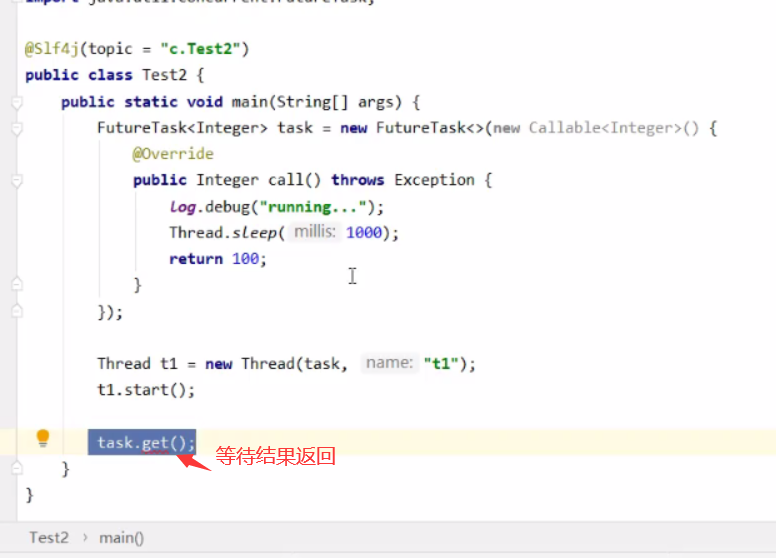




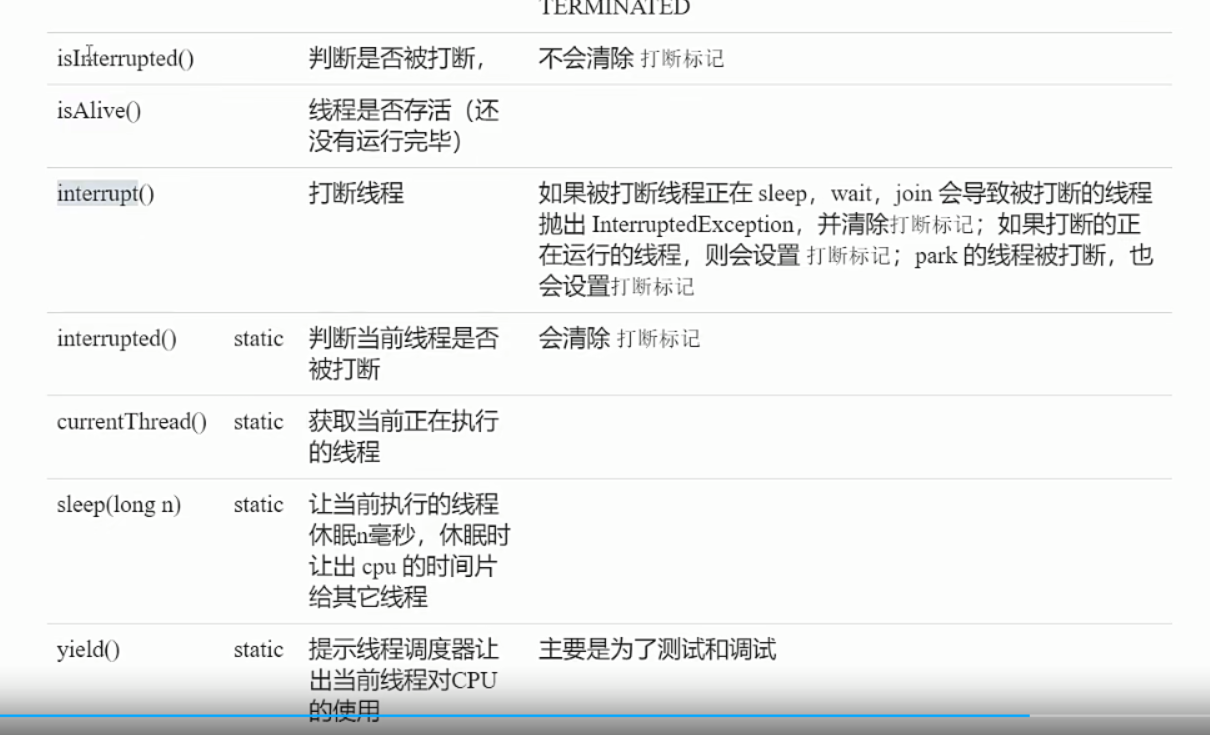








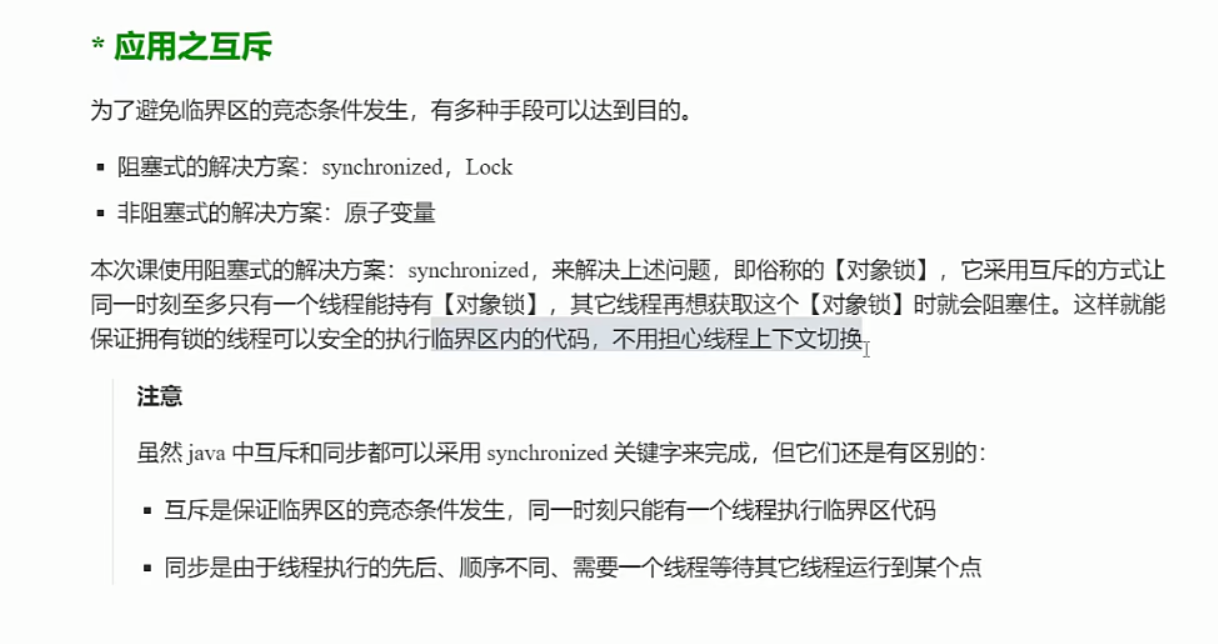


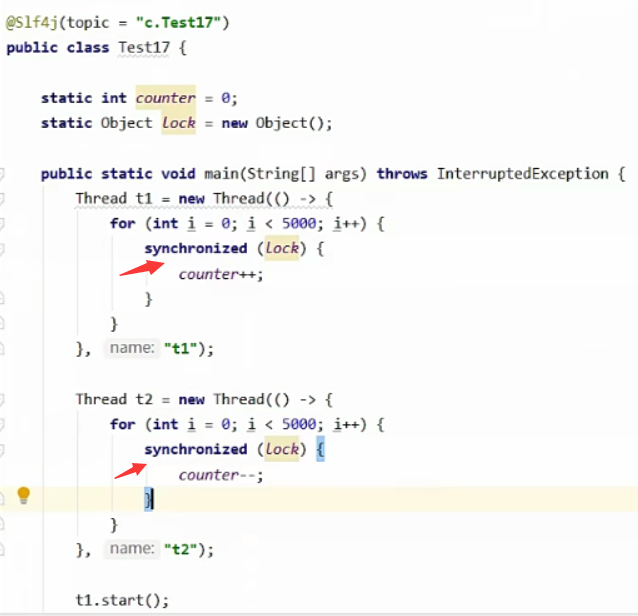








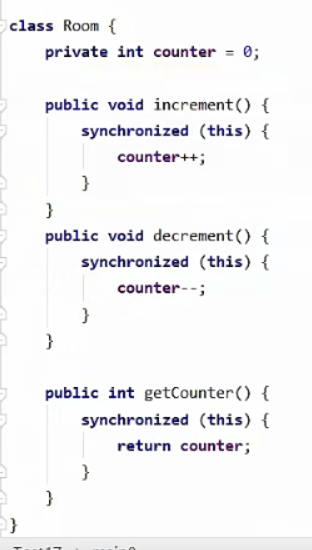






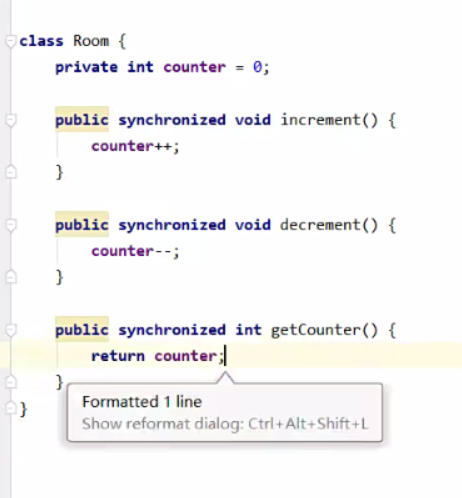


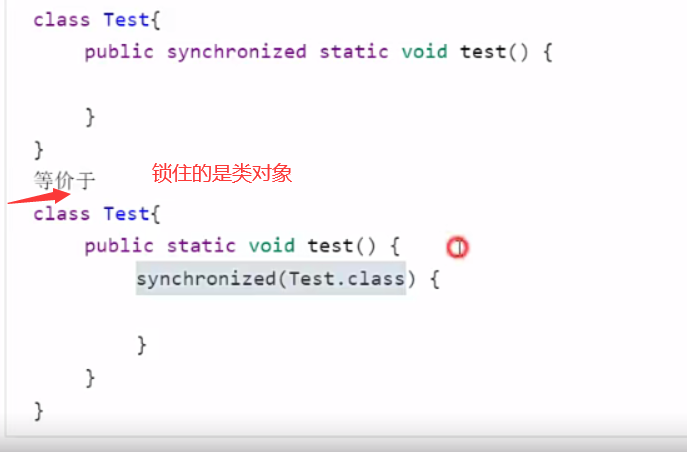




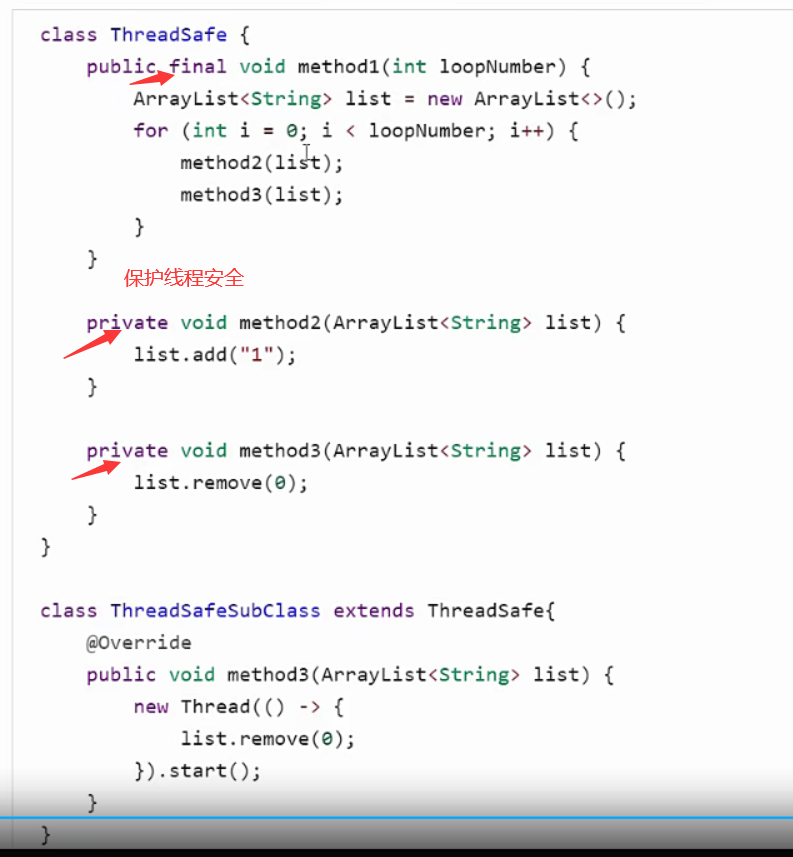


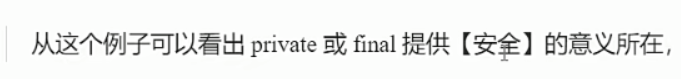


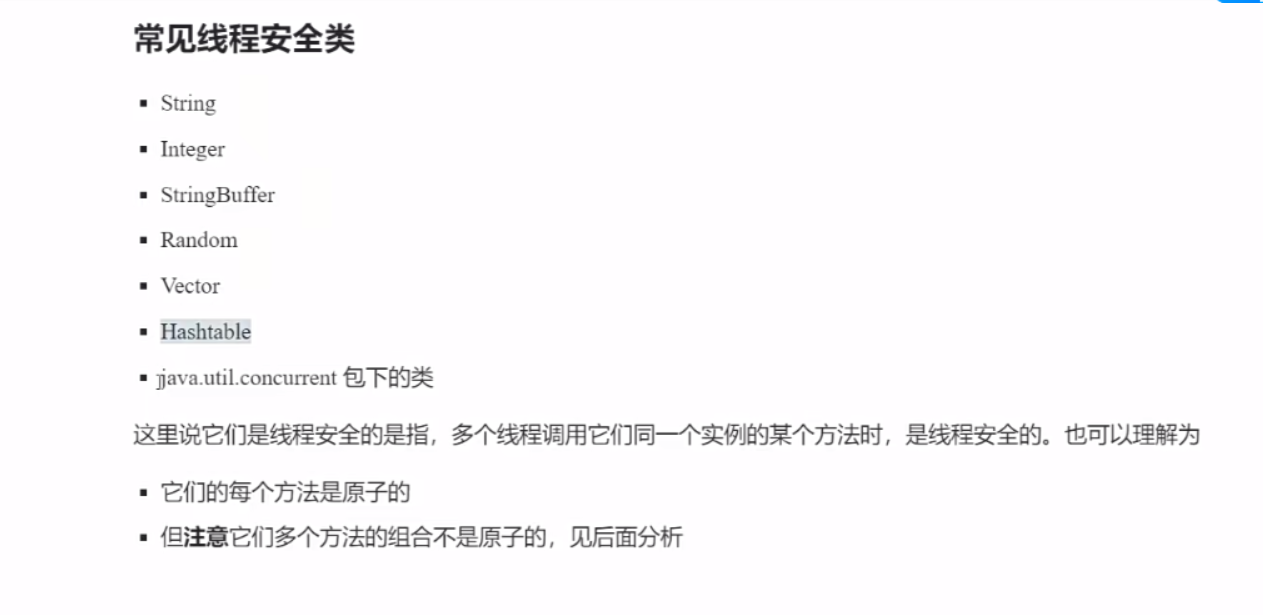
synchronized只能锁对象











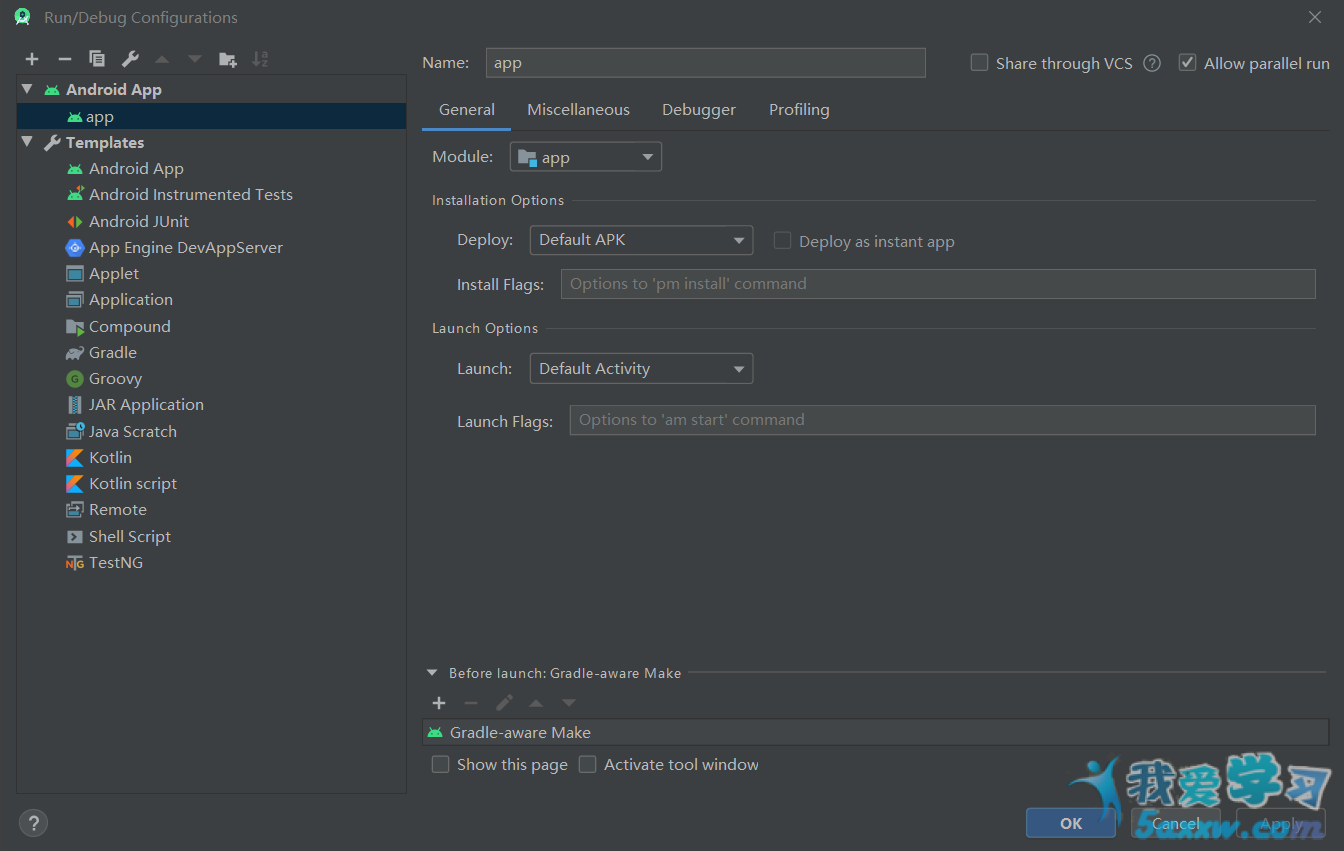
# **Android Studio 的运行配置**

前面的小节我们介绍了代码编辑和资源管理方面的知识，从本小结开始我们学习编译运行相关的知识，首先我们学习下编译运行前的配置。

## 1. 默认配置

在运行、调试我们的代码时，Android Studio 使用运行/调试配置确定如何执行操作。通常，默认配置已可满足运行或调试应用所需。在我们创建项目时，Android Studio 会根据 **Android App 模板** 为主 Activity 创建一个默认运行/调试配置。

要打开 Run/Debug Configurations 对话框，请依次选择 **Run > Edit Configurations**。此时将显示 **Run/Debug Configurations** 对话框，如下图：



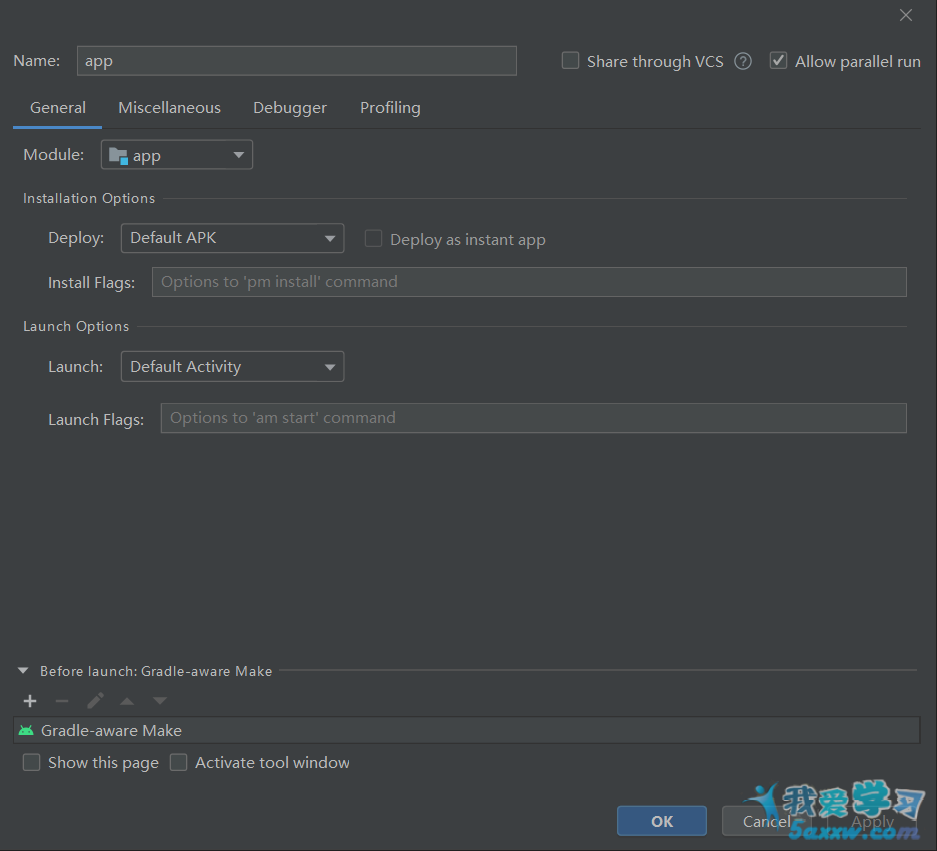
此对话框会在左侧面板中的 Templates 文件夹下方显示默认模板，并在 Templates 文件夹上方显示我们已经定义的配置。

在我们创建项目时 Android Studio 会根据 **Android App 模板** 为主 Activity 创建一个默认运行/调试配置。我们可以使用基于此模板的配置，在虚拟或硬件设备上运行或调试 Android 应用和 Activity。

## 2. Android App 模板

Android App 模板有4个标签：General，Miscellaneous，Debugger，Profiling。

### 2.1 General 标签



在 General 标签中，我们可以指定安装、启动和部署选项。

**Module**  
选择要应用此配置的模块。

**Installation Options：Deploy**

**Default APK**：为我们当前选择的变体构建和部署 APK。

**APK from app bundle**：从 Android App Bundle 构建和部署应用。也就是说，Android Studio 会先将我们的应用项目转换为 app bundle，其中包含了应用的所有经过编译的代码和资源。然后，Android Studio 仅会从该 app bundle 生成将应用部署到已连接的设备所需的 APK。通常，在测试想要上传到 Google Play 的 app bundle 时应使用此选项，因为从 app bundle 部署应用会延长总构建时间。

**Custom Artifact**：在 Android Studio 中不受支持。

**Nothing**：不在设备上安装 APK。例如，如果我们更愿意手动安装 APK，则无需使用 Android Studio 进行安装。

**Default APK**：为我们当前选择的变体构建和部署 APK。

**APK from app bundle**：从 Android App Bundle 构建和部署应用。也就是说，Android Studio 会先将我们的应用项目转换为 app bundle，其中包含了应用的所有经过编译的代码和资源。然后，Android Studio 仅会从该 app bundle 生成将应用部署到已连接的设备所需的 APK。通常，在测试想要上传到 Google Play 的 app bundle 时应使用此选项，因为从 app bundle 部署应用会延长总构建时间。

**Custom Artifact**：在 Android Studio 中不受支持。

**Nothing**：不在设备上安装 APK。例如，如果我们更愿意手动安装 APK，则无需使用 Android Studio 进行安装。

**Installation Options：Deploy as instant app**

如果我们的应用支持免安装体验（也就是说，我们在创建新项目时添加了对免安装应用的支持，或者我们创建了一项或多项支持免安装体验的动态功能），则我们可以选择部署这些支持免安装体验的模块，方法是勾选 Deploy as instant app 旁边的复选框。

**Installation Options：Dynamic features to deploy**

如果我们的应用包含动态功能模块，在部署应用时，请选中要包含的每个动态功能旁边的复选框。只有在应用包含动态功能时，才会看到此选项。

**Installation Options：Install Flags**

键入我们要使用的任何 adb pm install 选项。设置选项格式的方式与在命令行上相同，但不指定路径。

**Launch Options：Launch**

**Nothing**：当我们选择 Run 或 Debug 时，不启动任何内容。不过，如果我们的应用已在运行且我们选择了 Debug ，则 Android Studio 会将调试程序连接到我们的应用进程。

**Default Activity**：启动在清单中标记为启动项的 Activity。

**Specified Activity**：启动模块中特定的应用 Activity。选择此选项后，Activity 字段将显示在下方，我们可以在其中输入要启动的 Activity 的名称，也可以点击 More 以从列表中选择一个 Activity。

**URL**：启动与应用清单中的 intent 过滤条件匹配的网址。选择此选项后，URL 字段将显示在下方，我们可以在其中输入网址。

**Nothing**：当我们选择 Run 或 Debug 时，不启动任何内容。不过，如果我们的应用已在运行且我们选择了 Debug ，则 Android Studio 会将调试程序连接到我们的应用进程。

**Default Activity**：启动在清单中标记为启动项的 Activity。

**Specified Activity**：启动模块中特定的应用 Activity。选择此选项后，Activity 字段将显示在下方，我们可以在其中输入要启动的 Activity 的名称，也可以点击 More 以从列表中选择一个 Activity。

**URL**：启动与应用清单中的 intent 过滤条件匹配的网址。选择此选项后，URL 字段将显示在下方，我们可以在其中输入网址。

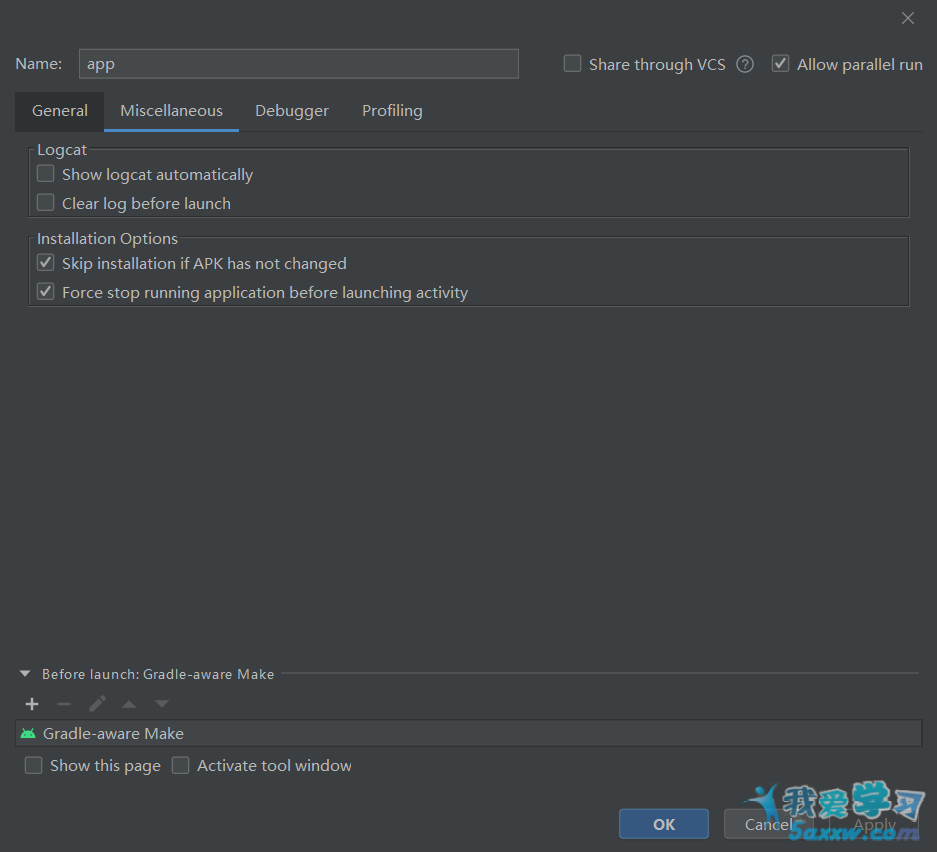
**Launch Options：Launch Flags**

键入我们要使用的任何 adb am start 选项。设置选项格式的方式与在命令行上相同，但不指定 intent。

**Before Launch**

定义启动之前的操作。

### 2.2 Miscellaneous 标签



在 Miscellaneous 标签中，我们可以指定 logcat、安装、启动和部署选项。

**Logcat：Show logcat automatically**

选中此选项后，每次使用此配置成功部署和启动应用时，系统都会打开 Logcat 窗口。

**Logcat：Clear log before launch**

如果我们希望 Android Studio 在启动应用之前从日志文件中移除之前会话的数据，请选中此选项。

**Installation Options：Skip installation if APK has not changed**

选中此选项后，如果 Android Studio 检测到我们的 APK 未更改，将不会重新部署 APK。如果我们希望 Android Studio 即使在 APK 未更改的情况下仍强行安装 APK，则取消选中此选项。

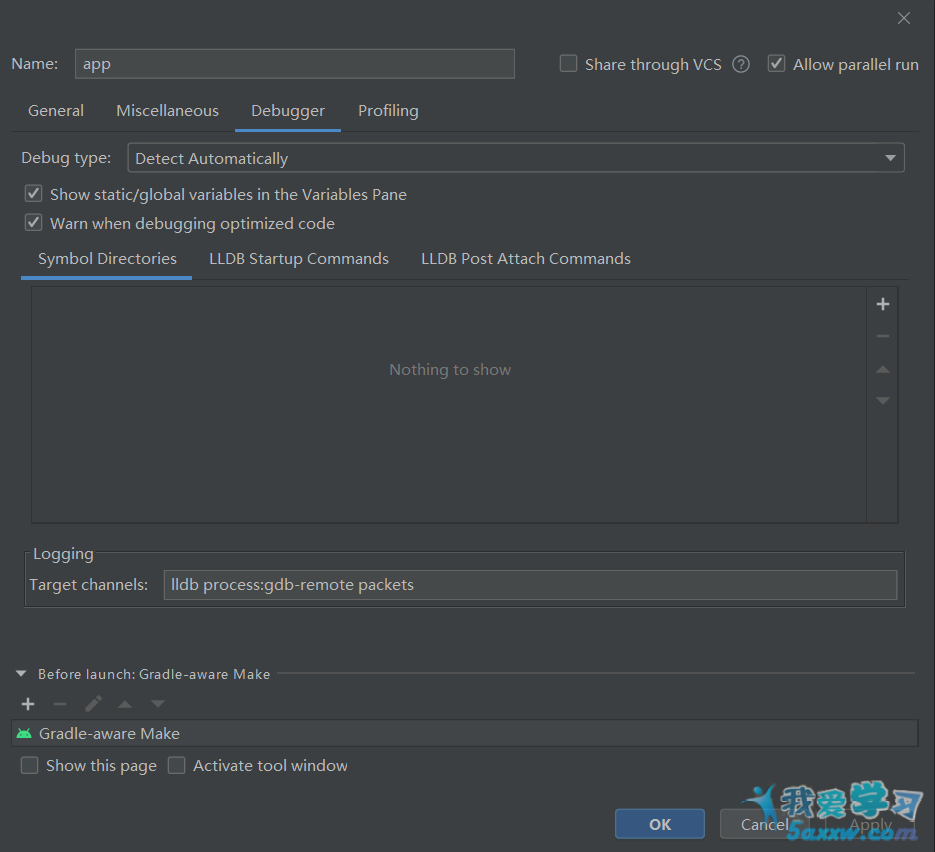
**Installation Options：Force stop running application before launching activity**

如果选中此选项，则在 Android Studio 检测到 APK 未更改从而无需重新安装 APK 时，它将强行停止应用，以便应用从默认启动器 Activity 启动。如果取消选中此选项，则 Android Studio 不会强行停止应用。

**Before Launch**

定义启动之前的操作。

### 2.3 Debugger 标签



我们可以在 Debugger 标签中指定调试选项。

对于 C 和 C++ 代码，Android Studio 会使用 LLDB 调试程序。除了普通的 Android Studio 界面，调试程序窗口还有一个 LLDB 标签，让我们可以在调试过程中输入 LLDB 命令。我们可以输入与 Android Studio 用于在调试程序界面中显示信息的命令相同的命令，还可以执行其他操作。对于 C 和 C++ 项目，我们可以在 Debugger 标签中添加符号目录，以及 LLDB 启动与连接后命令。

**Debug type**

**Java**：仅调试 Java 代码；

**Auto**：让 Android Studio 为我们的项目选择最合适的调试类型；

**Native**：调试原生 C 或 C++ 代码；

**Dual**：在两个单独的调试会话中调试 Java 和原生代码；

**Java**：仅调试 Java 代码；

**Auto**：让 Android Studio 为我们的项目选择最合适的调试类型；

**Native**：调试原生 C 或 C++ 代码；

**Dual**：在两个单独的调试会话中调试 Java 和原生代码；

**Symbol Directories**

如果要添加符号文件来为调试程序提供在 Android Studio 外部生成的 C 或 C++ 代码的信息，我们可以在此处添加一个或多个目录。

**LLDB Startup Commands**

添加要在调试程序连接到进程之前执行的 LLDB 命令。

**LLDB Post Attach Commands**

添加要在调试程序连接到进程之后立即执行的 LLDB 命令。

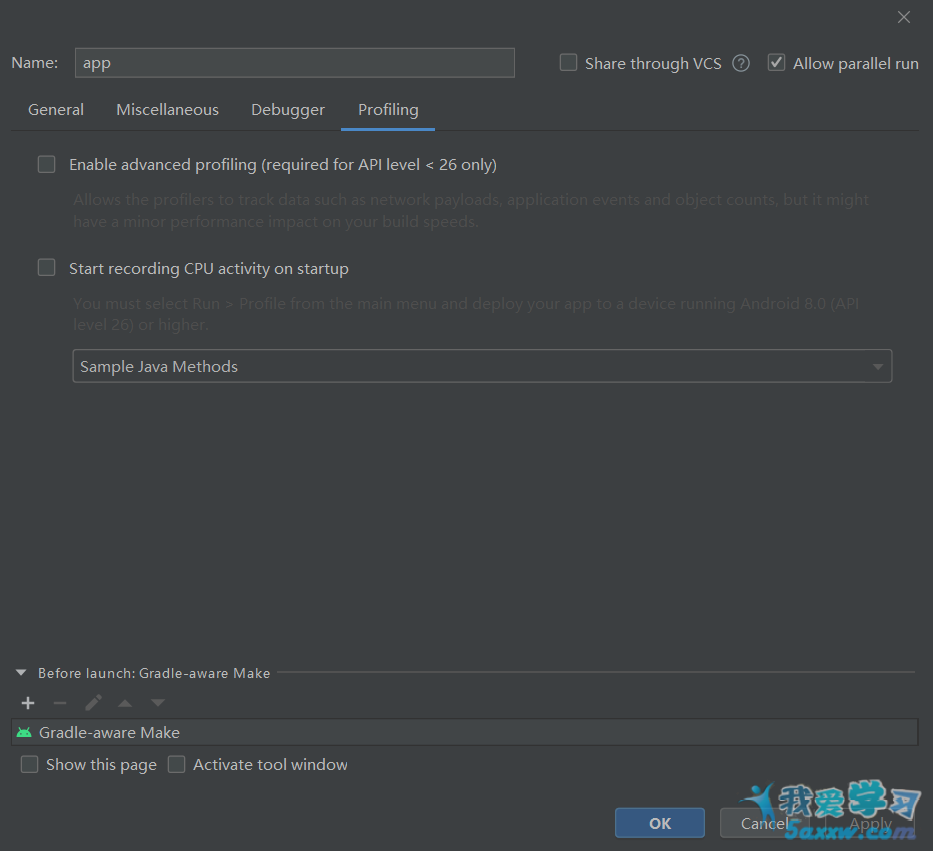
**Logging：Target channels**

指定 LLDB 日志选项。Android Studio 会根据团队的经验设置默认选项，所以速度不会太慢，但却包含问题排查所需的信息。Android Studio 错误报告经常需要请求日志。

**Before Launch**

定义启动之前的操作。

### 2.4 Profiling 标签



如果我们的设备搭载的是 Android 7.1 或更低版本，则必须勾选 Enable advanced profiling 选项才能启用 Android Profiler 中的特定功能。

## 3. 创建新的配置

Run/Debug Configurations 对话框会显示我们的运行/调试配置和可用的默认模板，我们可以直接根据模板或从其他配置的副本开始定义新配置，我们还可以在 Project 窗口中右键点击某个项，以自动创建该项专用的配置。

### 3.1 根据模板开始配置

要根据模板定义运行/调试配置，请按以下步骤操作：

打开 **Run/Debug Configurations** 对话框；

点击 **Add New Configuration** 图标 ；

选择一个默认模板；

在 **Name** 字段中输入一个名称；

根据需要修改配置；请务必更正在此对话框底部显示的所有错误；

点击 **OK**。

打开 **Run/Debug Configurations** 对话框；

点击 **Add New Configuration** 图标 ；

选择一个默认模板；

在 **Name** 字段中输入一个名称；

根据需要修改配置；请务必更正在此对话框底部显示的所有错误；

点击 **OK**。

### 3.2 使用副本开始配置

要使用其他配置的副本开始定义运行/调试配置，请按以下步骤操作：

打开 **Run/Debug Configurations** 对话框；

在左侧窗格中选择现有的运行/调试配置；

点击 **Copy Configuration** 图标 ；

在 **Name** 字段中输入一个名称；

根据需要修改配置；请务必更正在此对话框底部显示的所有错误；

点击 **OK**。

打开 **Run/Debug Configurations** 对话框；

在左侧窗格中选择现有的运行/调试配置；

点击 **Copy Configuration** 图标 ；

在 **Name** 字段中输入一个名称；

根据需要修改配置；请务必更正在此对话框底部显示的所有错误；

点击 **OK**。

### 3.3 为项目中的项定义配置

Android Studio 可以为 Project 窗口中显示的某些项创建运行/调试配置。如需为项目中的某个项创建运行/调试配置，请按以下步骤操作：

在 **Android** 或 **Project** 视图中打开一个项目；

在 **Project** 窗口中，右键点击某个可测试项，然后选择 **Run 文件名** 或 **Debug 文件名**。Android Studio 会创建一个临时的运行/调试配置并启动我们的应用；

从工具栏的 **Select Run/Debug Configuration** 下拉列表中选择 **Save**。

在 **Android** 或 **Project** 视图中打开一个项目；

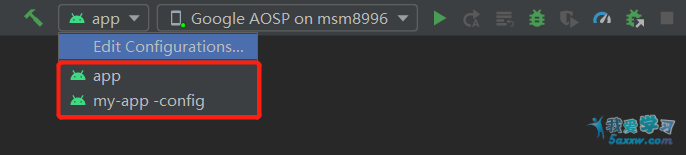
在 **Project** 窗口中，右键点击某个可测试项，然后选择 **Run 文件名** 或 **Debug 文件名**。Android Studio 会创建一个临时的运行/调试配置并启动我们的应用；

从工具栏的 **Select Run/Debug Configuration** 下拉列表中选择 **Save**。

## 4. 使用自定义的配置

如果我们保存了运行/调试配置，则可以在运行或调试应用之前选择该配置。

如需使用保存的运行/调试配置，请按以下步骤操作：

1. 从工具栏的 **Select Run/Debug Configuration** 下拉列表中选择我们自定义的配置；  
   
2. 选择 **Run > Run** 或 **Run > Debug** 运行程序。

## 5. 定义启动之前的操作

我们可以指定要在应用运行/调试配置之前执行的任务。任务会按照其在列表中显示的顺序来执行。

要创建任务列表，请按以下步骤操作：

在 **Run/Debug Configurations** 对话框底部的 **Before launch** 下方，点击 **Add** 图标 ，然后选择一种任务类型。如果对话框打开，请填充其中的字段并点击 **OK**。

根据需要添加更多任务。

要对任务进行排序，请选择一个任务并点击 **Up** 图标 或 **Down** 图标 ，以将其在列表中上移或下移。

如果我们想在应用运行/调试配置设置前显示这些设置，请选择 **Show this page**。

如果我们希望在运行或调试应用时激活 **Run** 或 **Debug** 工具窗口，请选中 **Active tool window**。

在 **Run/Debug Configurations** 对话框底部的 **Before launch** 下方，点击 **Add** 图标 ，然后选择一种任务类型。如果对话框打开，请填充其中的字段并点击 **OK**。

根据需要添加更多任务。

要对任务进行排序，请选择一个任务并点击 **Up** 图标 或 **Down** 图标 ，以将其在列表中上移或下移。

如果我们想在应用运行/调试配置设置前显示这些设置，请选择 **Show this page**。

如果我们希望在运行或调试应用时激活 **Run** 或 **Debug** 工具窗口，请选中 **Active tool window**。

可供添加的任务如下：

**Run External tool**

运行 Android Studio 外部的应用。在 **External Tools** 对话框中，选择我们要运行的一个或多个应用，然后点击 **OK**。

**Run Another Configuration**

执行某个现有的运行/调试配置。在 **Choose Configuration to Execute** 对话框中，选择要执行的配置，然后点击 **OK**。

**Make**

编译项目或模块。如果运行/调试配置指定了特定的模块，Android Studio 会执行 **Make Module** 命令；如果未指定任何模块，则会执行 **Make Project** 命令。

**Make Project**

编译项目。Android Studio 将执行 **Make Project** 命令。

**Make, no error check**

Android Studio 在执行运行/调试配置时不考虑编译结果，除此之外，此选项与 Make 相同。

**Run Gradle task**

运行 Gradle 任务。在打开的对话框中，指定详情，然后点击 **OK**。

**Gradle-aware Make**

编译项目并运行 Gradle。

**App Engine Gradle builder**

App Engine Gradle builder 任务会同步项目，然后构建模块。

## 6. 小结

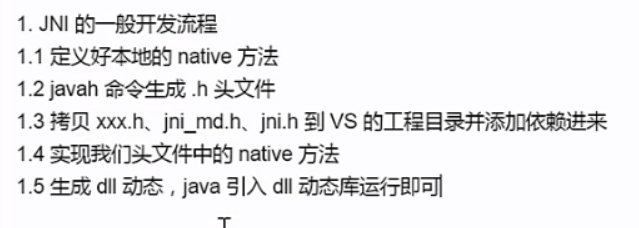
本节课程我们主要学习了运行配置相关的知识。本节课程的重点如下：

* 掌握 Android Studio 默认运行配置；
* 掌握 如何创建和使用自定义配置。

IMG_262

[法律声明](javascript:void(0);) [违版必究](javascript:void(0);) [资料版权](javascript:void(0);) [作者所有](javascript:void(0);)  [Wiki](https://www.5axxw.com/wiki/" \t "https://www.5axxw.com/wiki/content/_self) [返回首页](https://www.5axxw.com/" \t "https://www.5axxw.com/wiki/content/_blank)



#指定最低版本cmake\_minimum\_required(VERSION 3.4.1)

#设置生成的so动态库最后输出的路径set(CMAKE\_LIBRARY\_OUTPUT\_DIRECTORY ${PROJECT\_SOURCE\_DIR}/../jniLibs/${ANDROID\_ABI})

#添加动态库，如果有多个，可以有多个add\_libraryadd\_library( # 设置你的动态库名称

my-ndk-name

# 模式

SHARED

# 提供动态库的文件相对路径

src/main/jni/my-ndk.cpp )

find\_library( # Sets the name of the path variable.

log-lib

# Specifies the name of the NDK library that

# you want CMake to locate.

log )

#最终的库target\_link\_libraries( # 跟上面的动态库名称相同

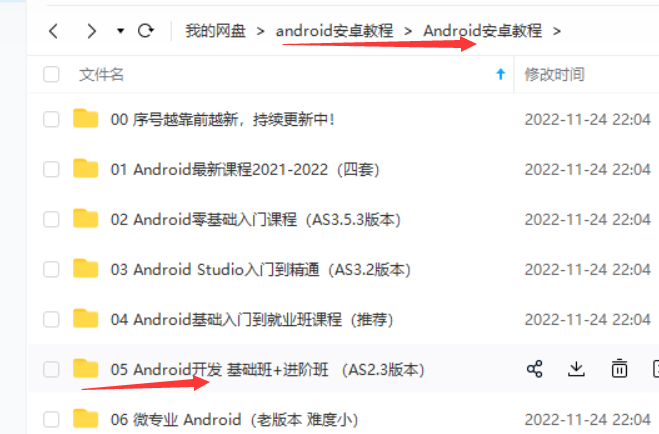
my-ndk-name

# Links the target library to the log library

# included in the NDK.

${log-lib} )

#添加相关的依赖include\_directories(src/main/jni/include/)



8ctrl+alt+c抽取

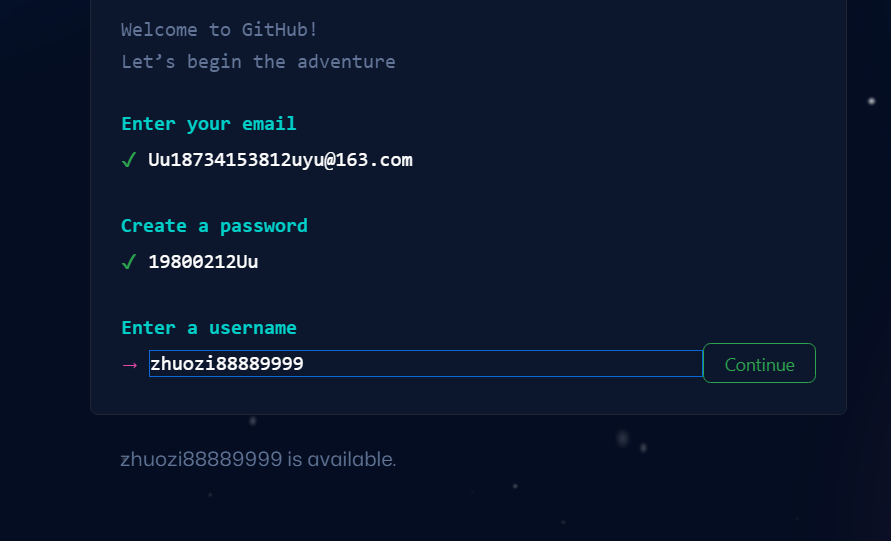
Github 账号[u18734153812@163.com](mailto:u18734153812@163.com)

密码19800212Uu





git config --global http.sslVerify "false"



19800212Uu

git config --global http.sslVerify "false"

$ git config --global http.sslVerify true

git remote rm origin

git push -u origin main

git remote add origin git@github.com:zhuozi88889999/8899.git