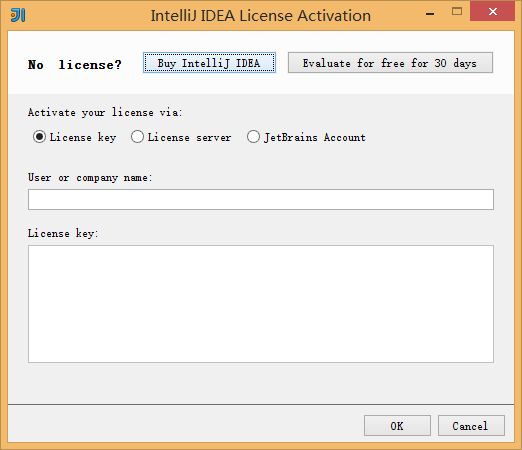
IntelliJ IDEA使用说明

# 下载以及破解

首先去官网下载最新版的 IntelliJ IDEA（[点击进入官网](http://www.jetbrains.com/idea/download/)）并安装。在安装完毕之后，选择运行IntelliJ，当出现需要输入注册码的时候，选择试用30天如下所示：



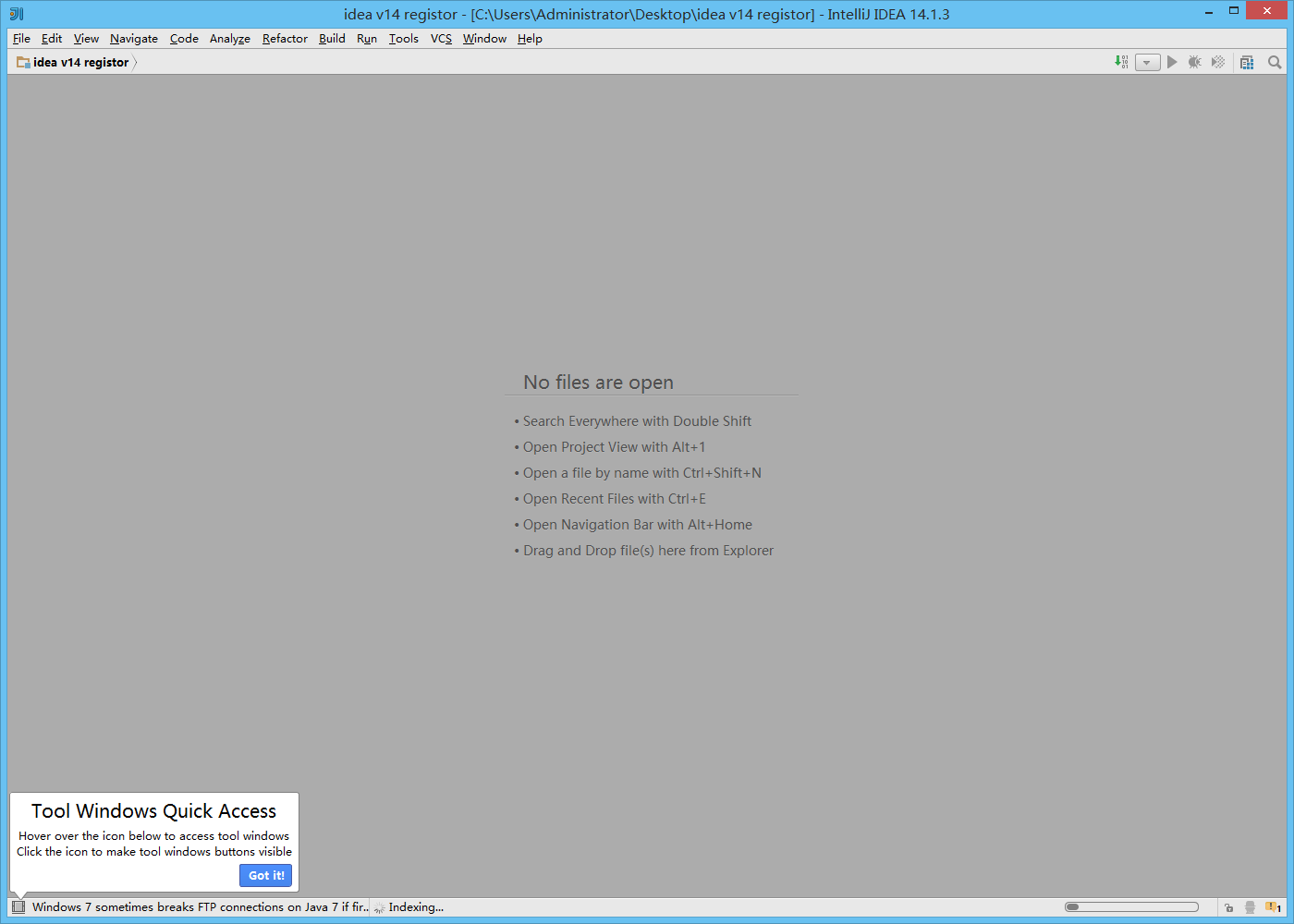
点击Evaluate for free for 30 days

启动成功后，选择导入工程，将我们的Crack工程导入到IntelliJ中来。

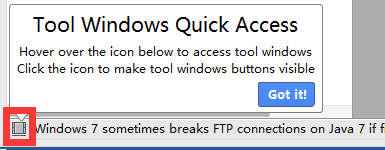


此处选择Import Project

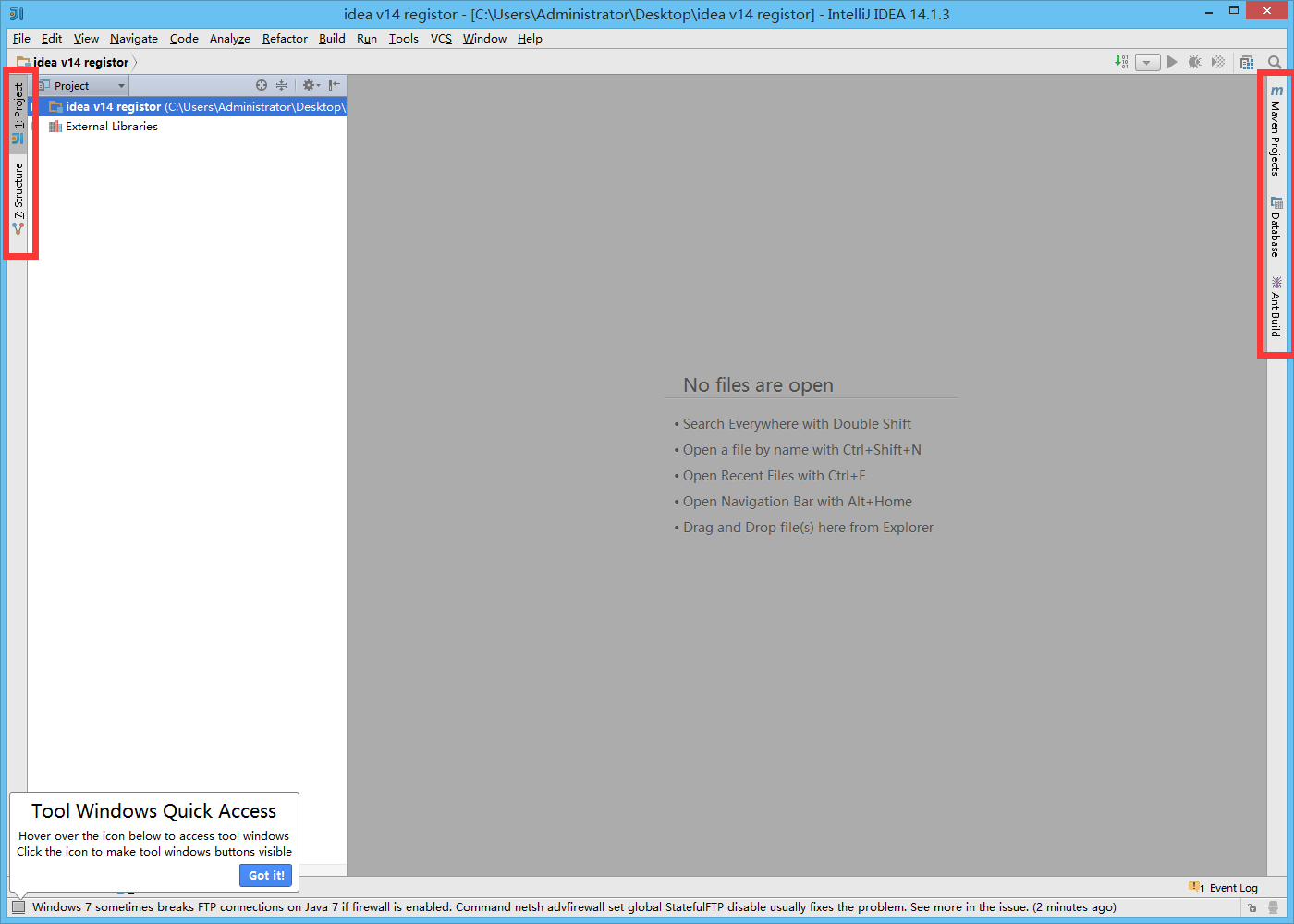
选择Crack工程，然后点击下一步直到打开整个环境如下所示（期间可能会让你手动选择JDK）：



按照提示，点击左下角的那个小图标，将侧边栏全部显示出来。侧边栏包含了很多有用的工具。

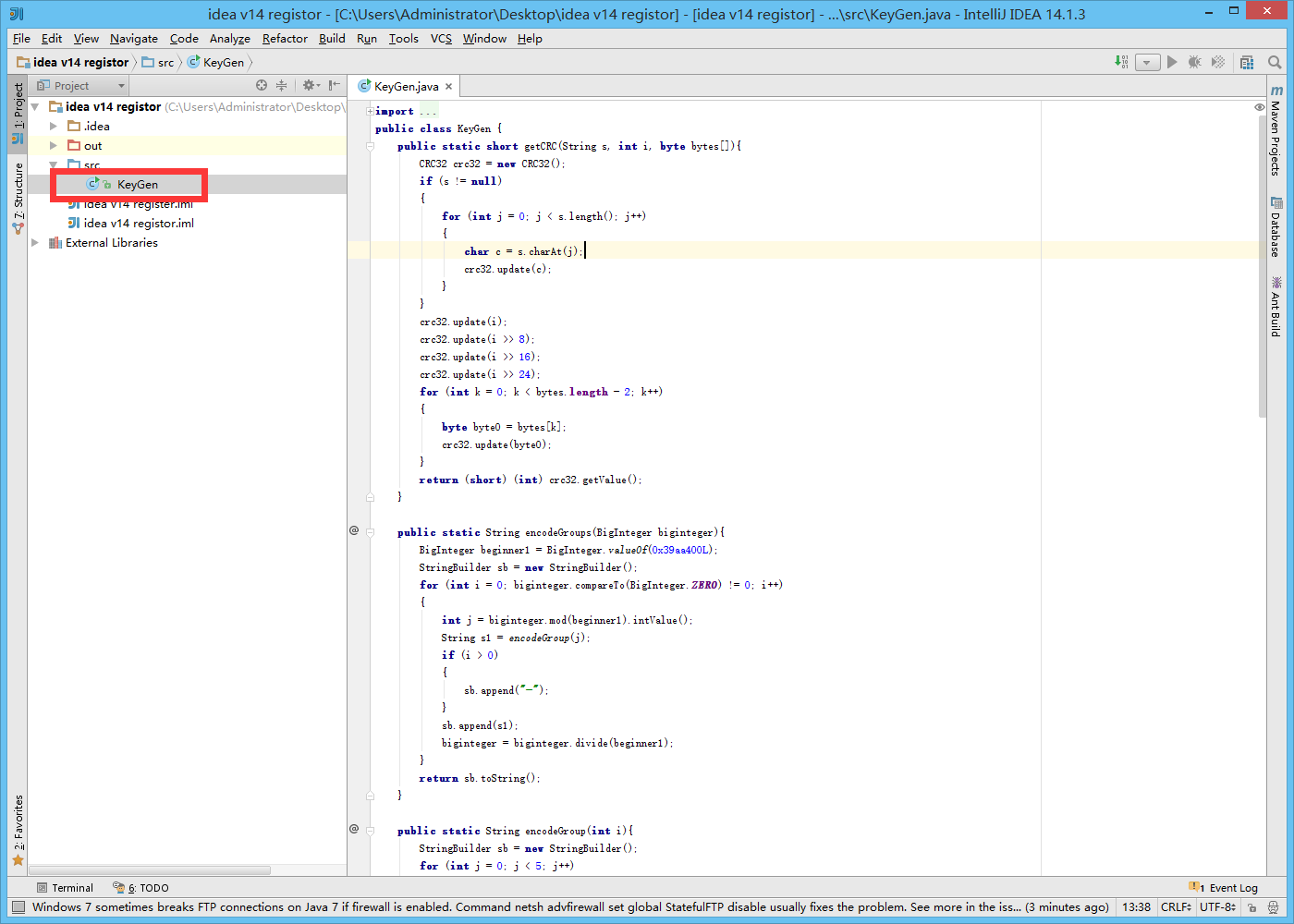


点击标红的小图标

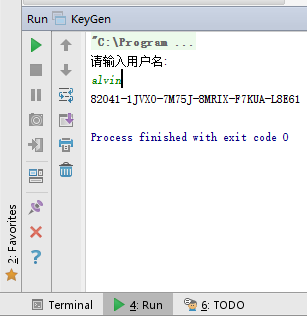


侧边栏全都显示出来了

打开我们的Crack工程，然后在src目录下找到KeyGen 类，双击打开看到代码：

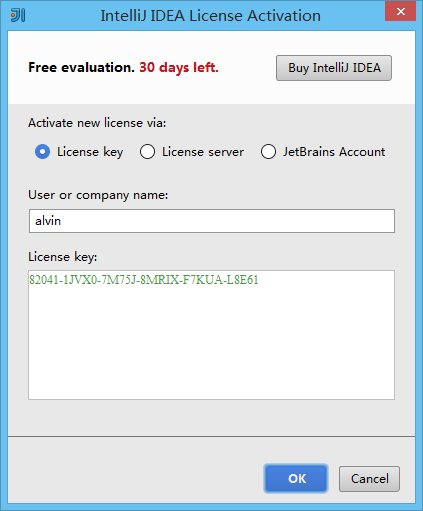


然后右键选择Run KeyGen.main() 或按下 Ctrl+shift+F10 快捷键执行这个代码。



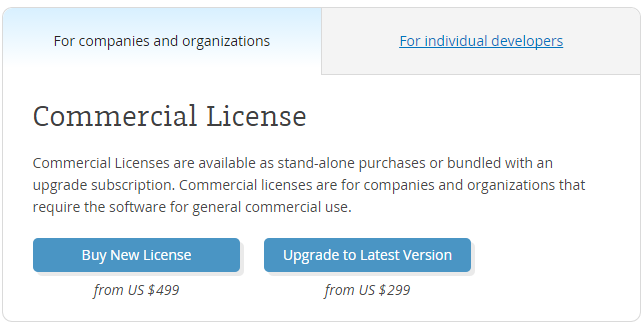
输入名字得到注册码

得到注册码之后，复制下来，然后打开Help中的 Register 输入生成的用户名、注册码即可。

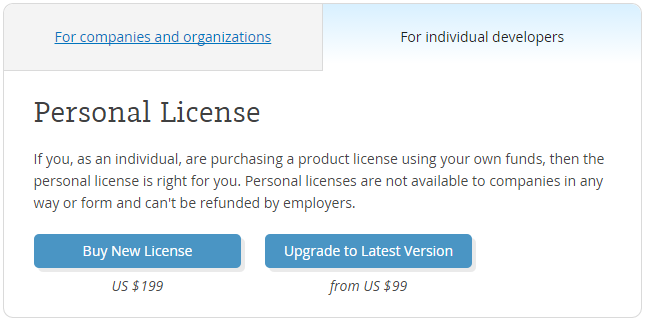


输入我们的刚才生成的注册码

这样我们就破解完毕了。当然，如果想购买的话，这个好像也不便宜……，公司购买是499刀，注册码是公司的。个人购买是199刀，不过个人版的经常会有打折促销，有兴趣高大上一把的可以去官网看看最新的消息。



公司License $499，升级一次 $299

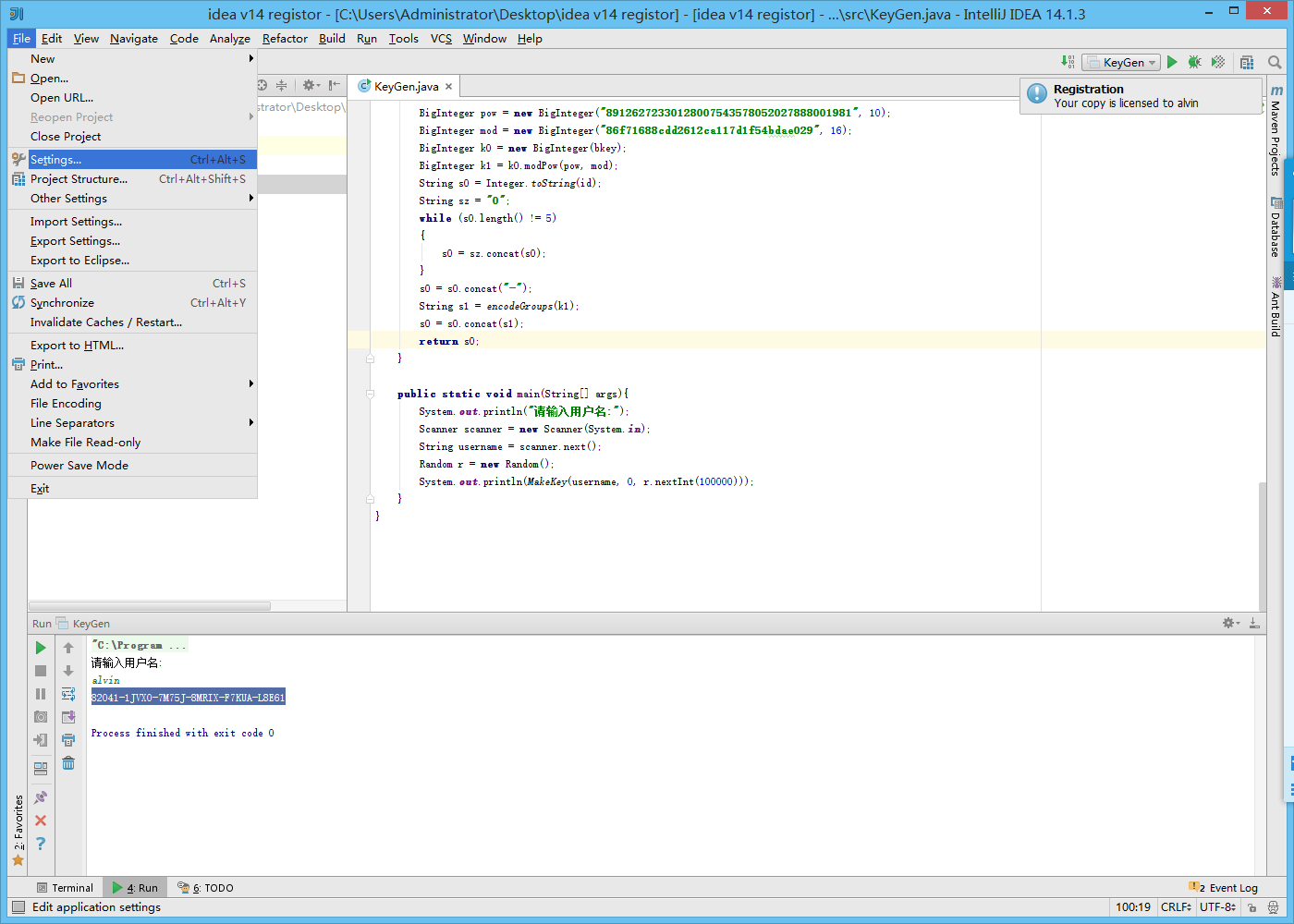


个人License $199，升级一次 $99

至于社区版，还是不要用来正式开发了，我一般用社区版开发IntelliJ插件用的。有兴趣的可以看看IntelliJ的插件是怎么编写的（用的是Java Swing）。

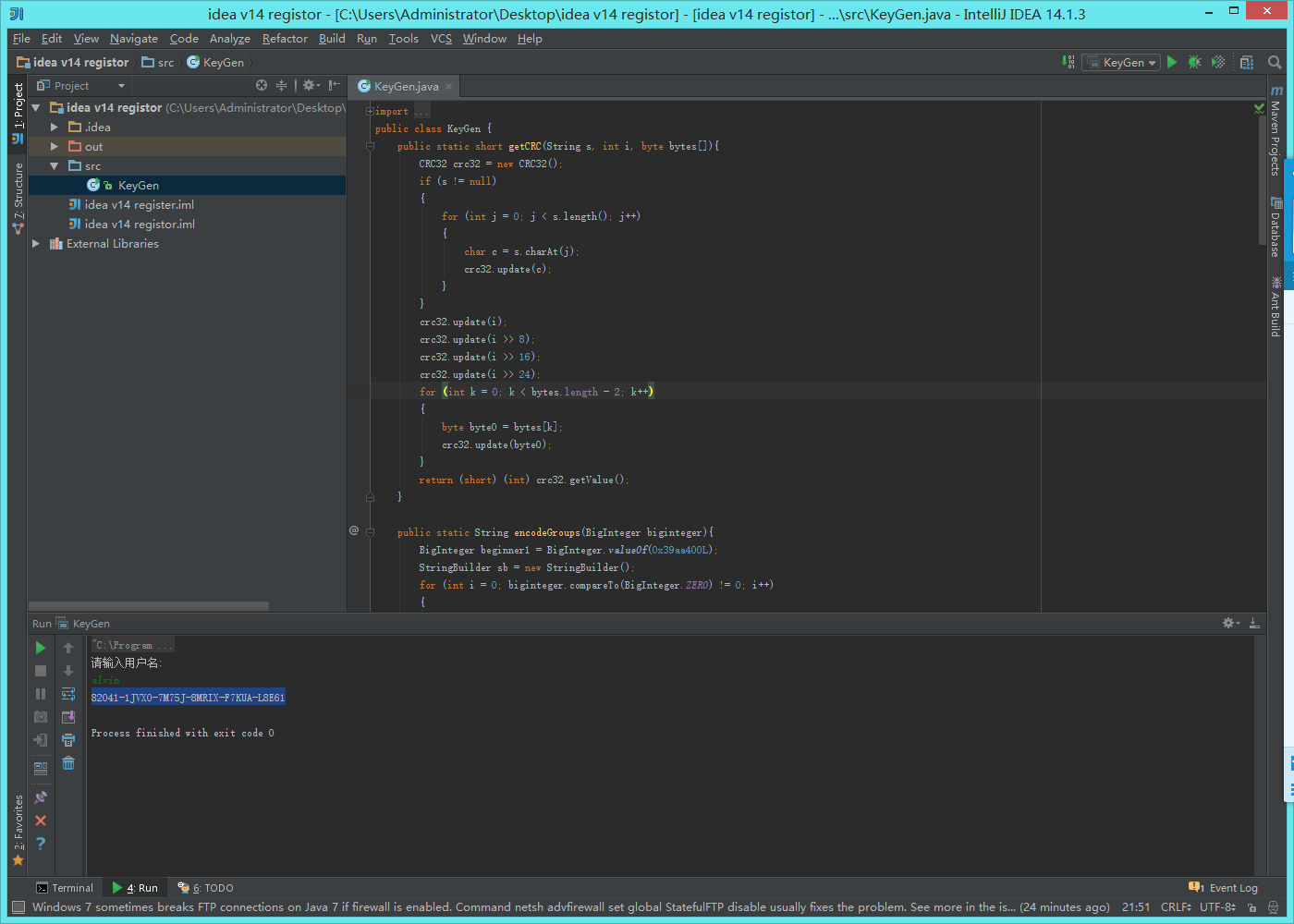
# 基本设置

在第一次打开的时候会让你选择主题样式，我这里是采用默认的，如果想采用黑底白字的Darcula，可以手动进行设置，通过File->Settings打开设置。



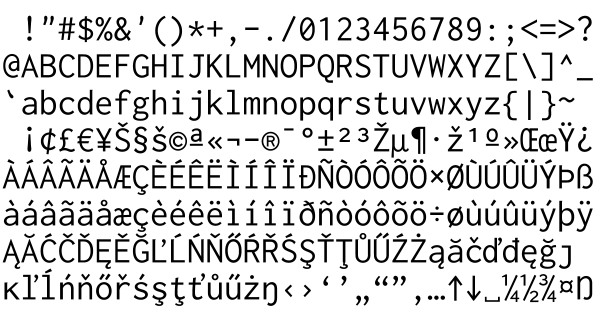
File->Settings(Ctrl+shift+S)

然后选择Appearance下的Theme，选中你中意的界面，点击确定即可（老版本的 IntelliJ 可能需要重启）。例如我常用的主题Darcula：



Darcula

当然，还有号称最佳编程字体的Monaco/Inconsolata字体。这两个字体都能很好的区分0和O，以及I和l还有1。所以为了尽量减少这种低级错误，还是设置一下字体吧。

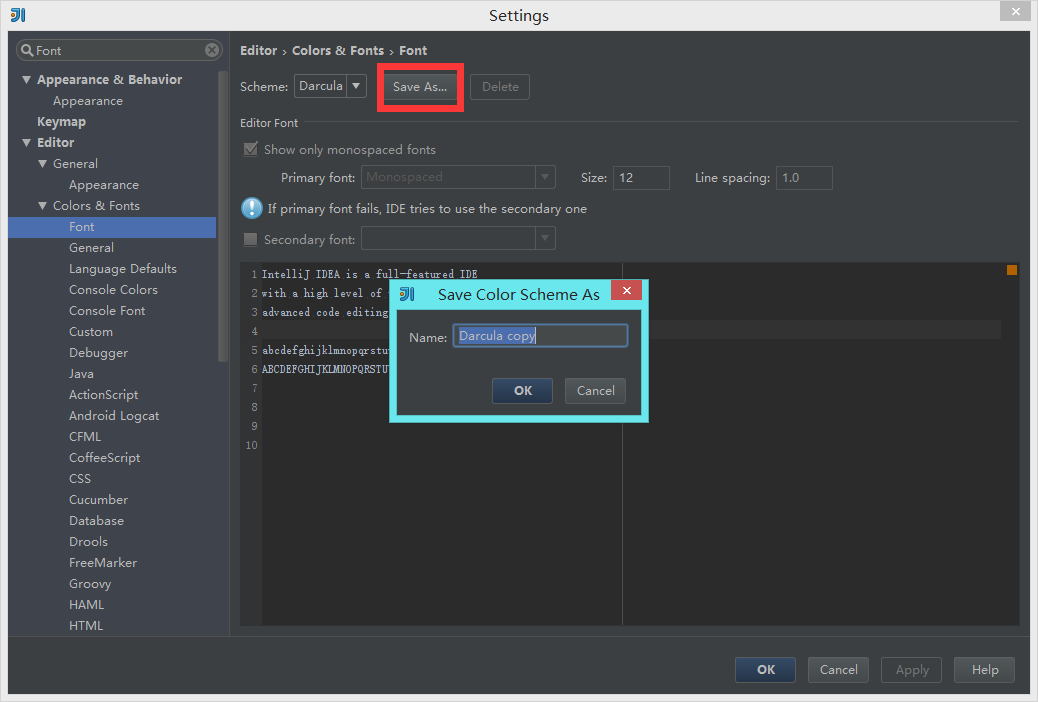


Inconsolata 字体

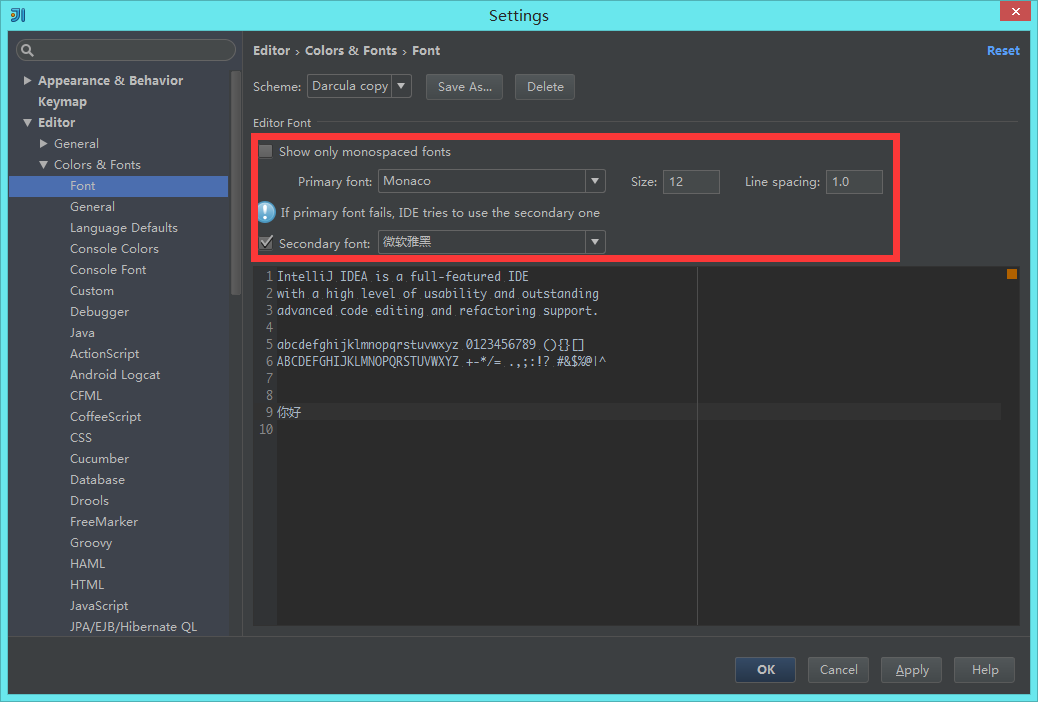


Monaco字体

首先下载字体（附件中有包含），然后双击打开，点击安装即可。我们这里以Monaco为例。在安装完毕之后，重启 IDEA ，打开设置（Ctrl+Shift+S）在顶部搜索Font，选择Editor下Colors & Fonts下的Font，然后选择Scheme后面的Save As，新建一个主题样式如下：



然后勾掉Show Only monospacked fonts，然后如下设置：



Primary font 选择 Monaco，字体大小为14号

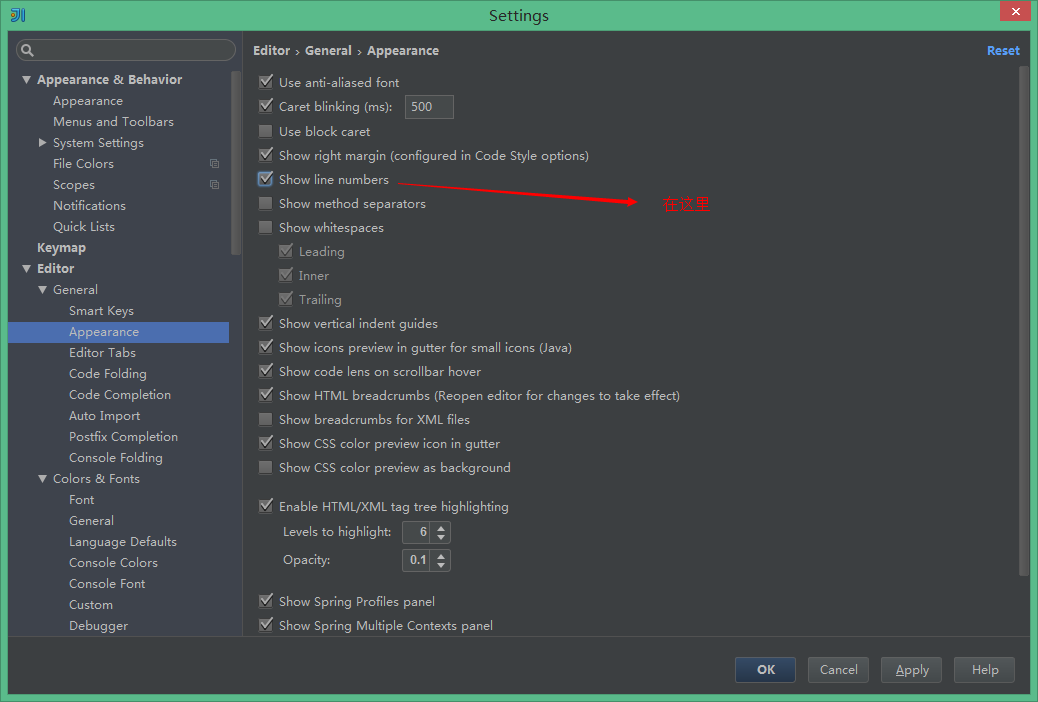
Secondary font 选择微软雅黑（用于显示中文）

其实，我认为 Monaco 要比 Inconsolata 更漂亮。并且Inconsolata在IDEA中显示是有问题的。实际的编码效果如下所示：



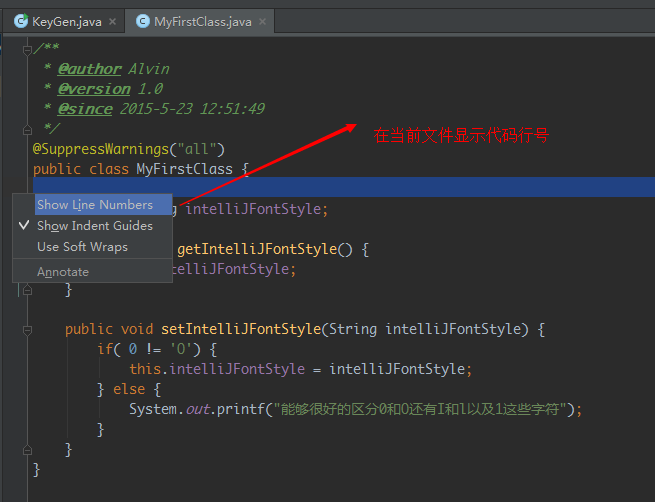
实际编码效果

IDEA默认是不显示行号的，如果需要显示行号，需要自己手动设置。设置方法如下：File->Settings->Editor->Appearence->Show Line Number



默认显示行号

如果不想默认显示行号，但是又想查看行号，可以通过右键来临时打开行号，如下图所示：



# 插件

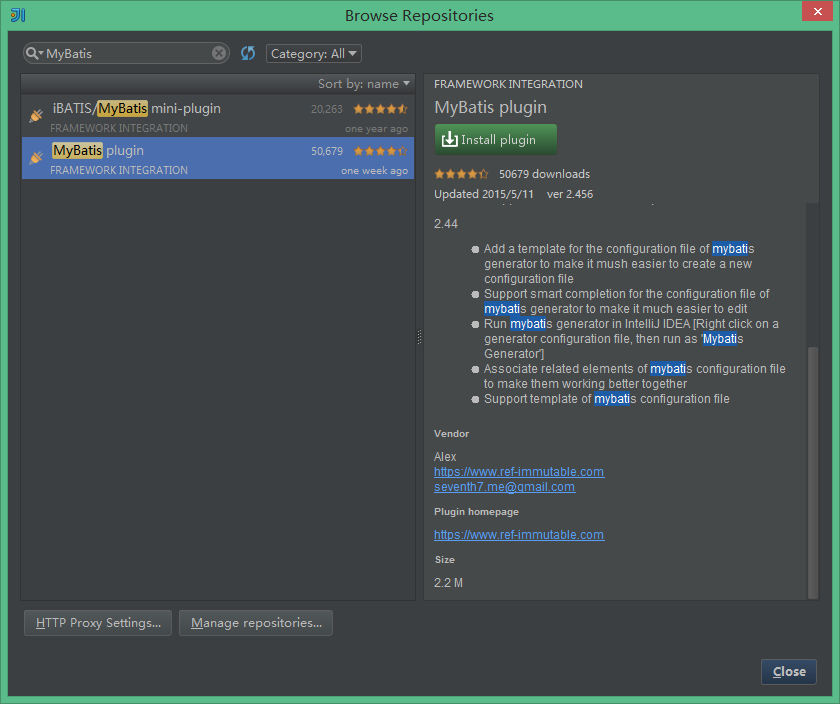
工欲善其事必先利其器，好的插件能够帮你事半功倍的去完成一些工作。那么这里就介绍一下 IntelliJ 最常用的几个插件。

默认情况下，IntelliJ 已经集成了大部分常用的工具，例如 Tomcat、SVN、CVS、Git、GitHub、Maven、Ant以及Spring、Hibernate等框架的自动化检测功能，但是为了更快速的进行开发，我们还需要下载以下两个插件：MyBatis和JRebel。

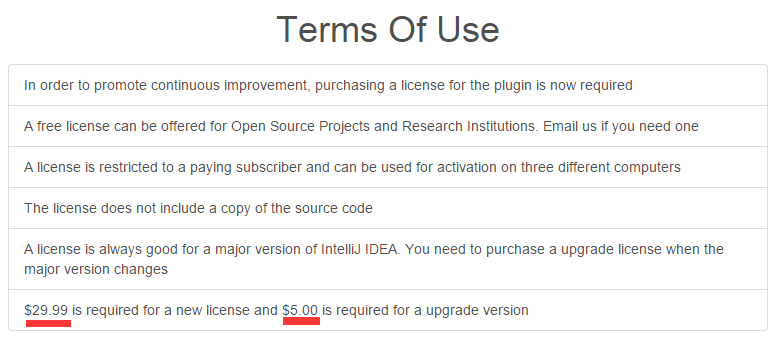
虽然过程可能会比较罗嗦，但是请放心，你现在所做的一切都是值得的。而且是相当值得……

其中所有的相关文件都在文章最后面。

首先，咱们安装 MyBatis的插件。打开Plugins（Ctrl+Shift+S打开设置，搜索Plugins），在插件管理中搜索MyBatis并点击Browse去服务器上进行搜索。

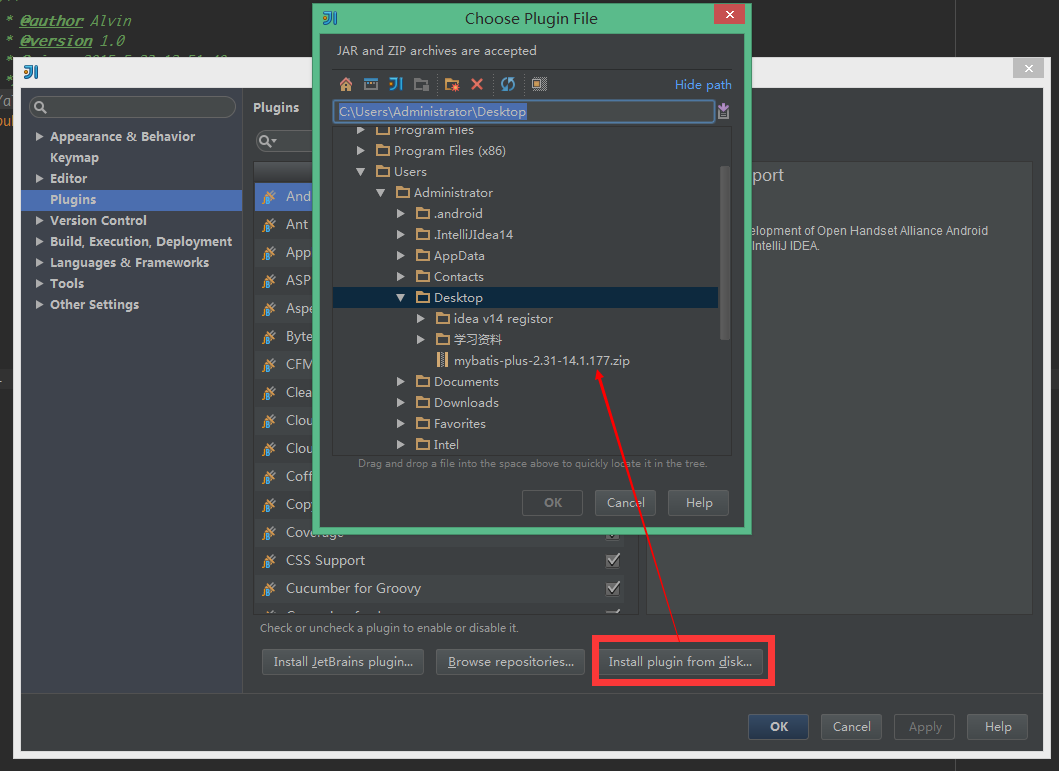


但是很不幸的是，开发这个插件的家伙在两个月前开始收费了，而且还TMD一点都不便宜，如果要是$9.9，或许我还会支持一下，结果给老子的报价居然有$29.9，这还不算完，每次升级还得额外给$5，看来作者深谙互联网精神啊！呵呵……我寻思这东西虽然挺牛逼的，但是真的不值这个价。



亲，是$29.99哦，不是$29.9……

但是英勇的中国人民是不可能被打败的，那就是下载上个版本的MyBatis插件，并通过硬盘安装的方式安装到IDEA中去。想高大上一把的兄弟可以去他的[官方网站](https://ref-immutable.com/terms)购买一个。

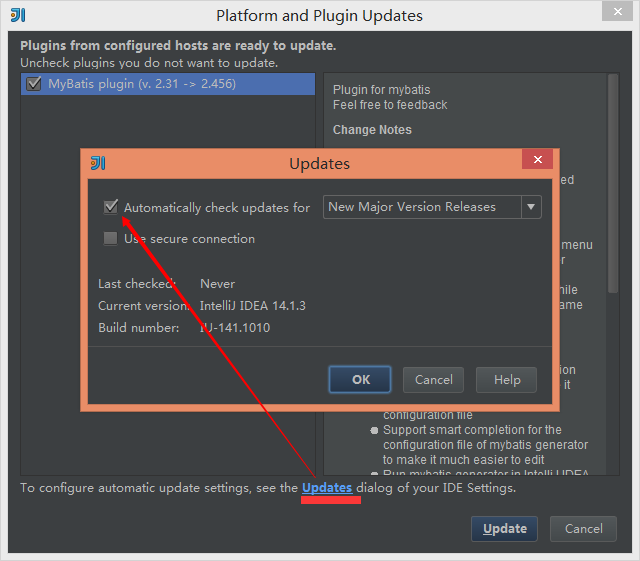


点击Install plugin from disk，选中我们老版本的插件（是个zip），点击Ok，重启Idea就搞定了，以后提示升级不理会即可（“呵呵”）。



不要理会它

如果嫌烦的话，就点击Update，进入到更新窗口，再点击下方的Updates，进入到更新设置，不让他自动检测就Okay了。如下图所示：



去掉自动检查更新

这个插件怎么用的，我们暂且不用去管，我们先把另外一个插件装好，那就是狂拽叼炸天的JRebel。

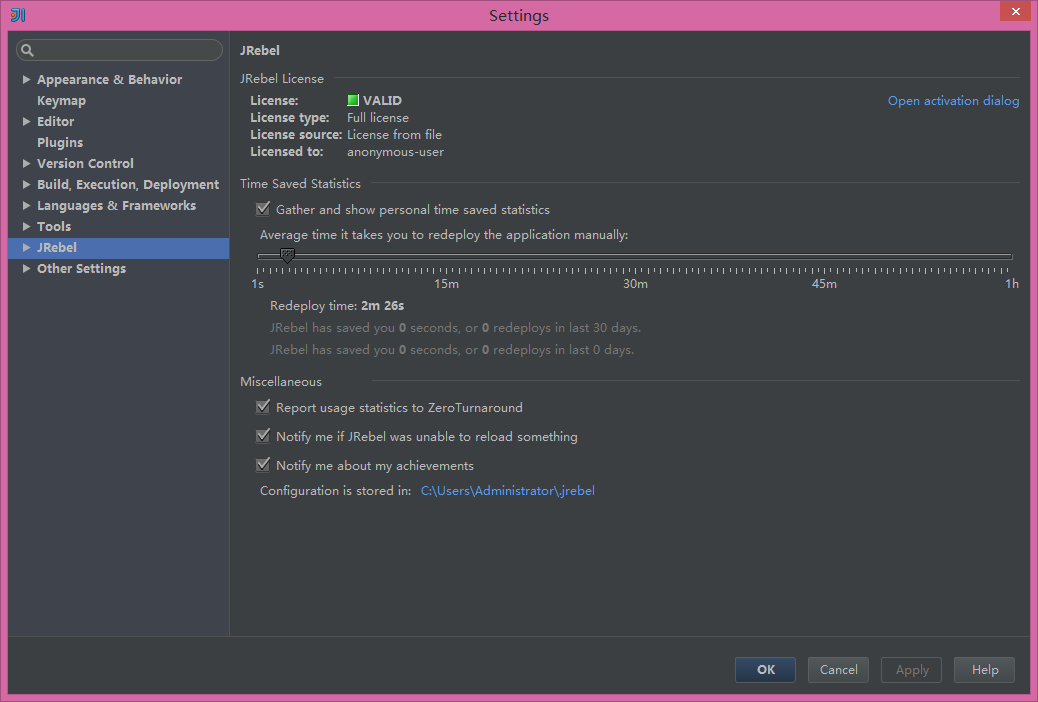
同样，我们先通过硬盘安装JRebel插件，不过与 MyBatis 插件不同的是，JRebel这个插件是正版的，需要后续的一些操作来破解它，安装并重启IntelliJ。

此时JRebel会提示你现在是不是需要激活，或者获得试用期，我们不用管他。再次关闭IntelliJ，因为重启IntelliJ是让他把安装的插件解压出来放到正确的目录下而已。

现在我们打开破解文件，将lib里面的jrebel.jar和jrebel.lic拷贝，粘贴到以下两个目录下覆盖原文件即可：

|  |
| --- |
| C:\Users\{你本机的用户名}\.IntelliJIdea14\config\plugins\jr-ide-idea\lib\jrebel6  C:\Users\{你本机的用户名}\.IntelliJIdea14\config\plugins\jr-ide-idea\lib\jrebel |

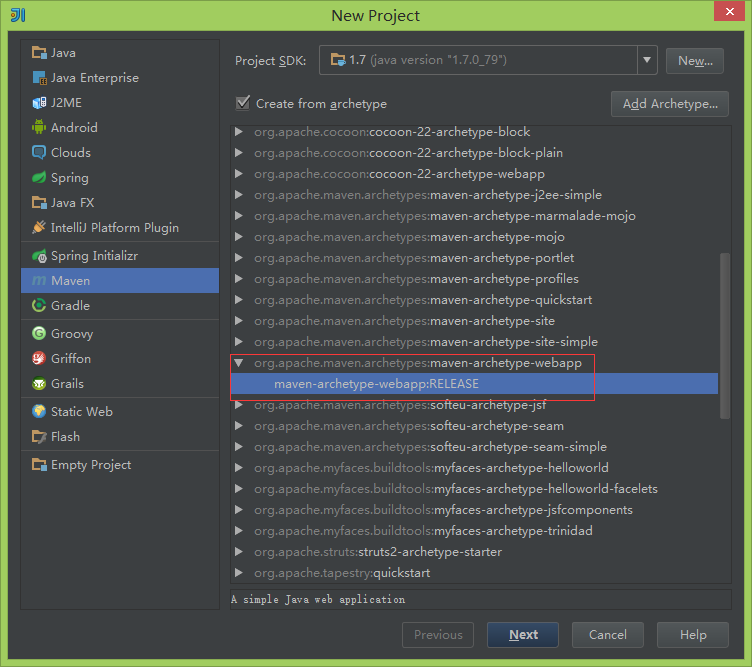
再次打开IntelliJ，打开设置，查看JRebel插件，已经VALID了……

  
 至此，两大开发利器已经安装完毕，其他的一些零碎插件，我们可以在需要的时候再装。

# 新建项目

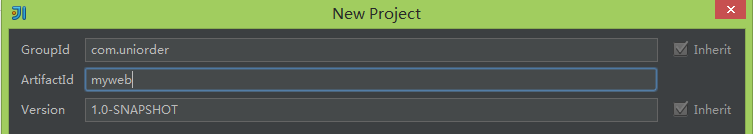
已经准备好了所有的环境问题，那么我们实际新建一个项目来看一下到底怎么玩。

选择 File-New-Project 打开新建项目窗口，我们选择 Maven，并基于WEB模板。



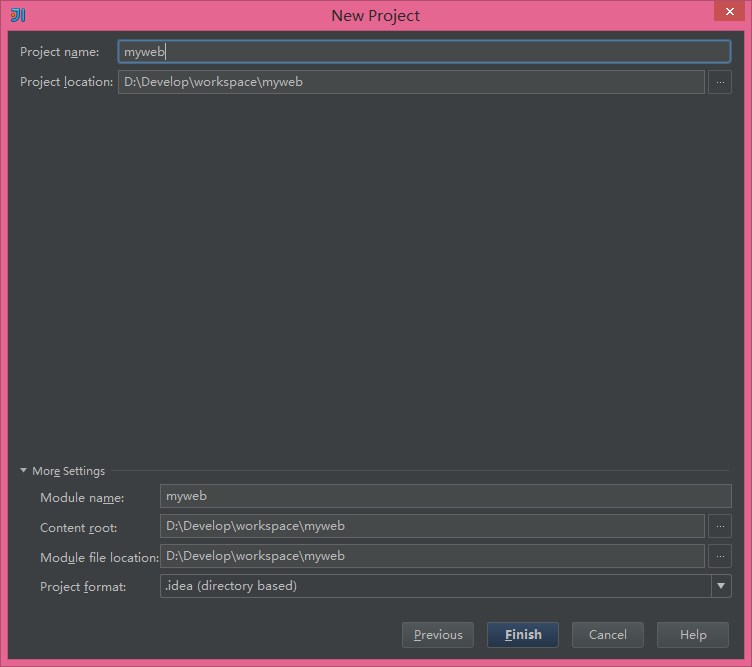
基于 WEBAPP的Maven 项目

点击下一步为 Maven 添加一个Group ID 和 Artifact ID，如下图所示：



设置 Group和 Artifact

然后配置Maven，我这里使用的是自己的 Maven 而不是 IntelliJ自带的。如果嫌麻烦，可以使用它自带的 Maven，然后指定项目所在的文件夹目录，如下图所示：



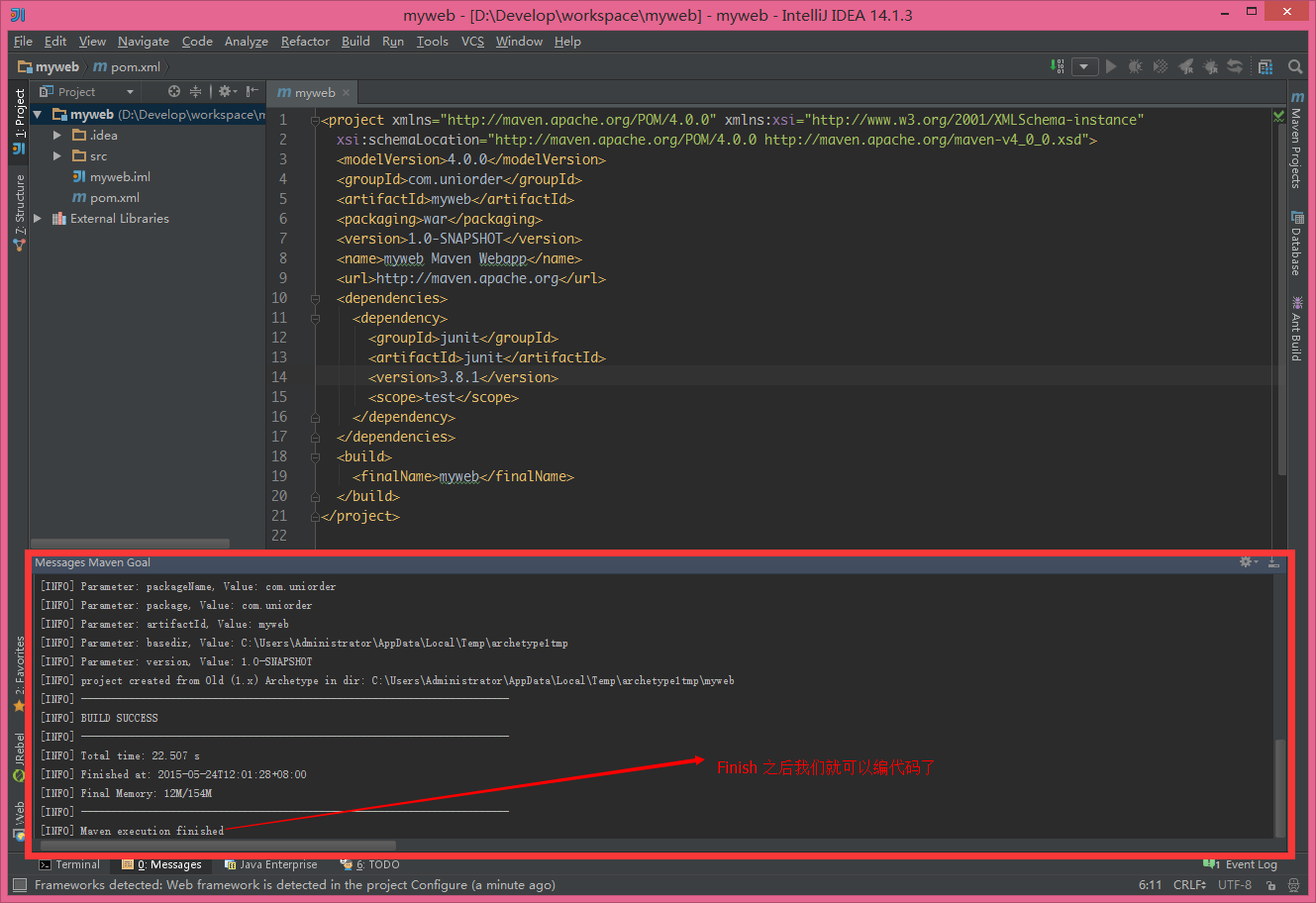
制定项目所在的文件夹目录

点击Finish之后，等待 Maven 为我们创建 WEB 工程，此时右上角会提示你是否启用自动导入依赖，我们选择自动导入即可。



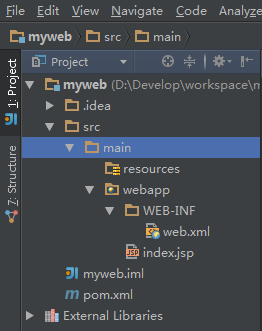
选择 Enable Auto-Import

此时，IntelliJ 会自动去创建WEB 工程，在下方的输出中我们可以看到当前的进度：



等待Maven构建工程

等待Maven 构建完毕之后，我们看一看当前的项目，此时已经是一个标准的基于 Maven 的 WEB 工程了，可以放在任何IDE下进行开发，例如 Eclipse 、NetBeans等的，目录结构如下：

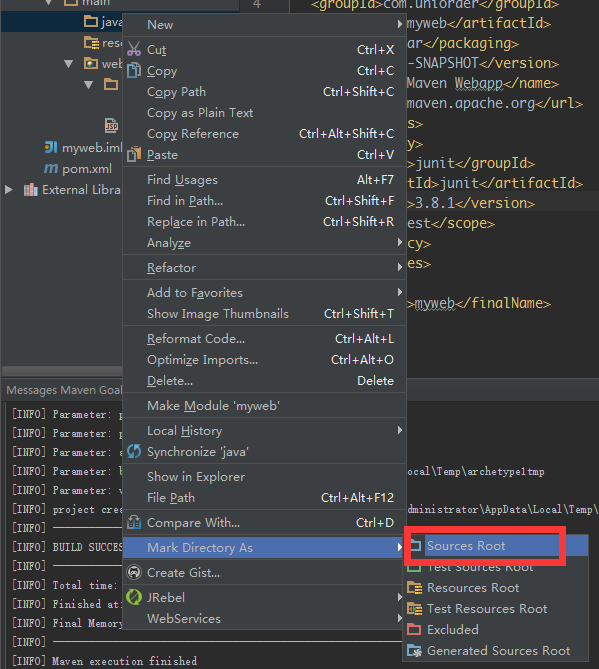


自动生成的目录结构

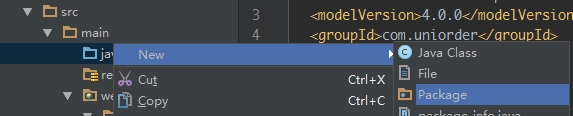
现在我们发现没有Java目录，所以我们需要在main目录下新建一个java目录，并让其成为Source root。



新建目录

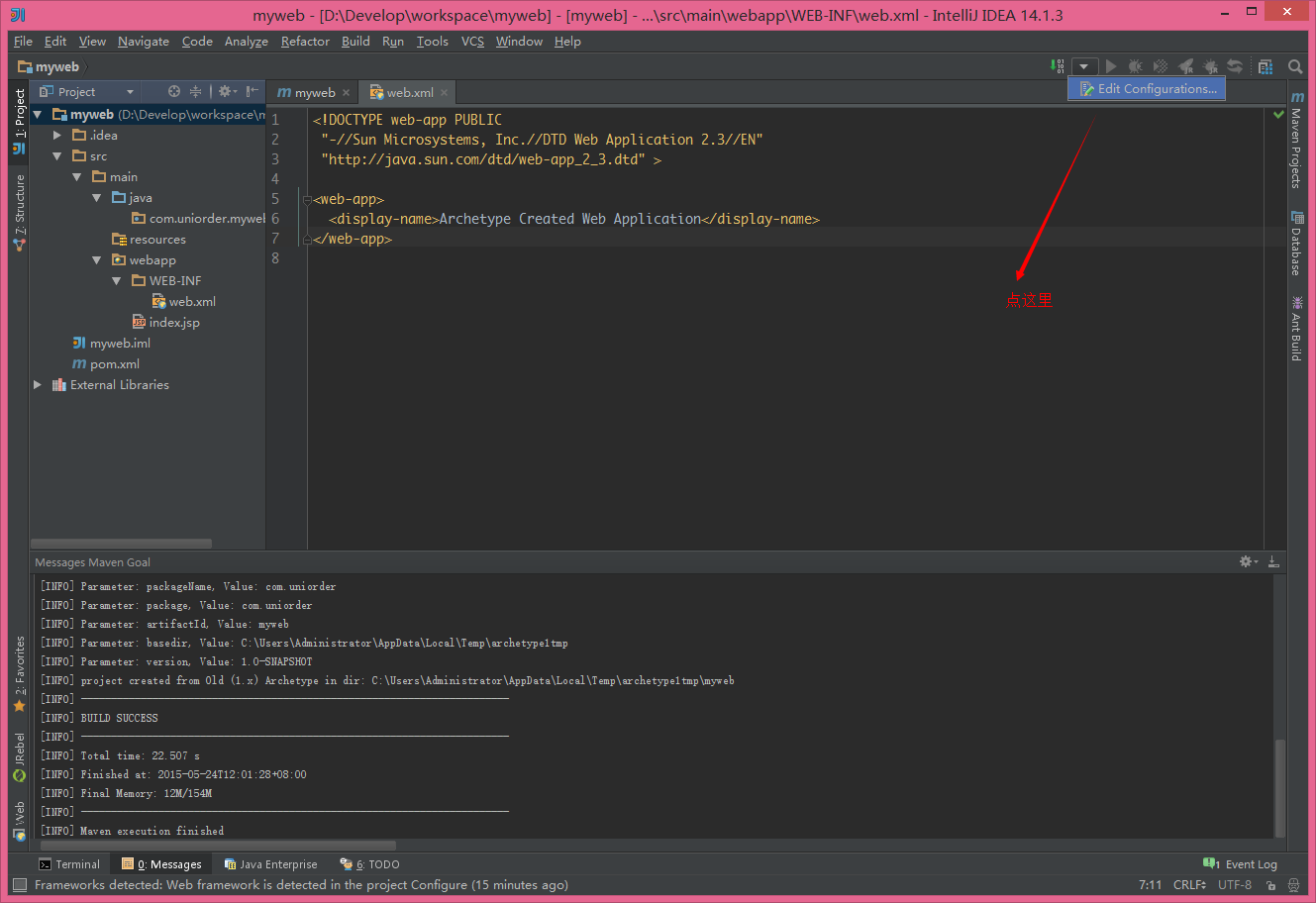


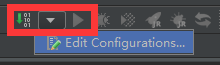
让他成为Sources Root



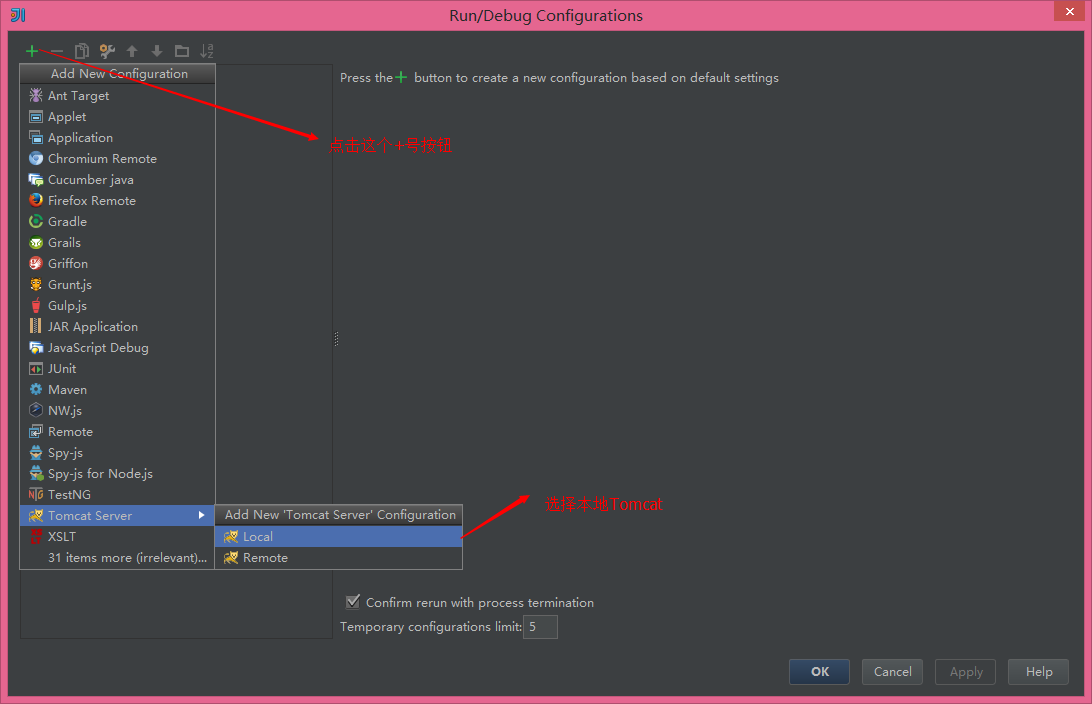
现在就可以在里面新建包了

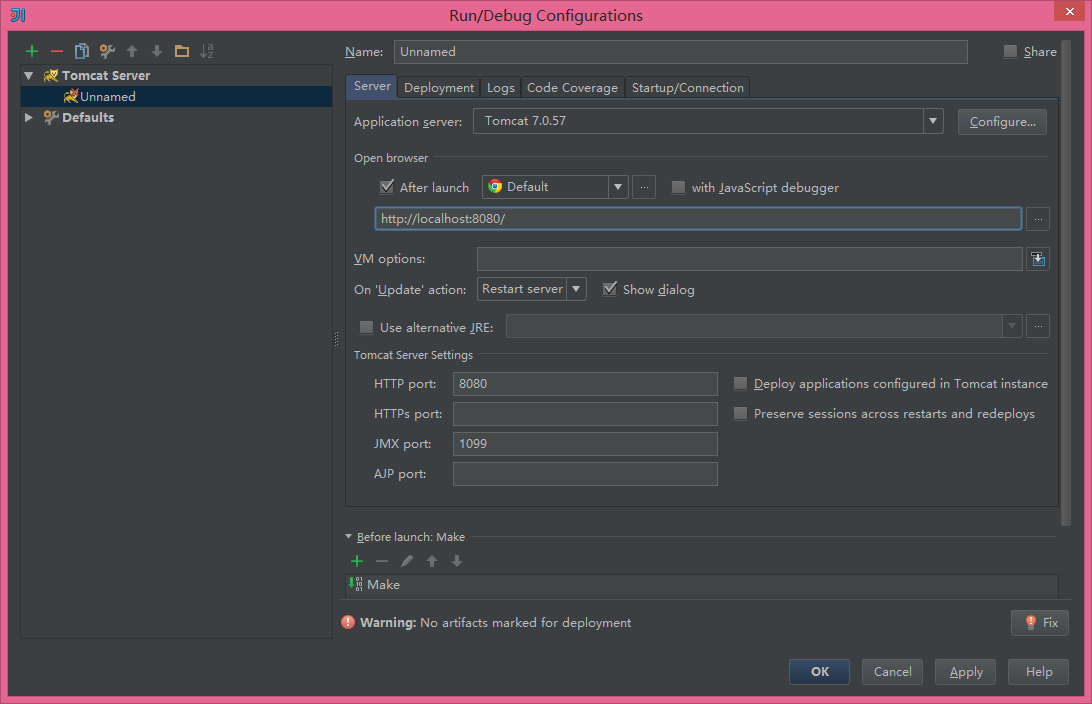
我们暂且不在里面加入代码，而是先让工程在 Tomcat 下面跑起来。下面看看如何配置 Tomcat。



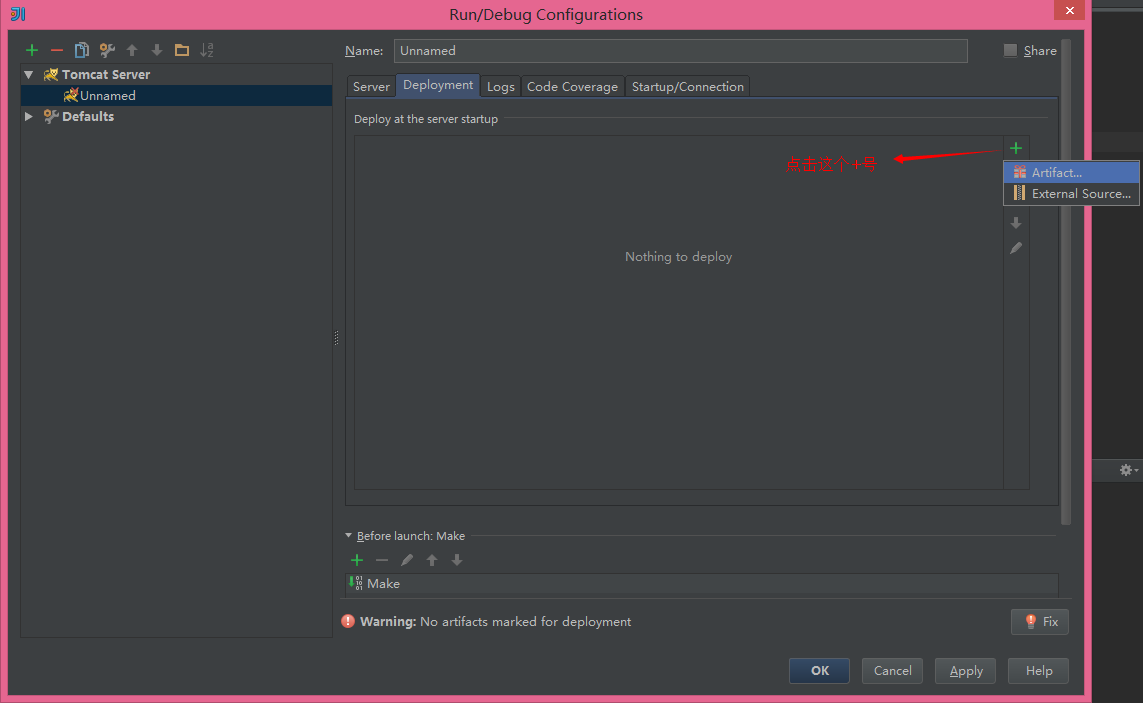


首先在右上角的下拉中选择 Edit Configurations，然后点击添加，选择 Tomcat并使用本地Tomcat。（如果找不到Tomcat，就点击More……）



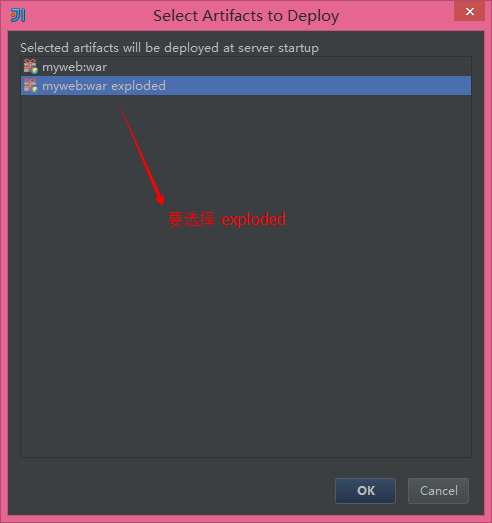


我们在这个界面需要设置Application Server（你Tomcat的安装目录）。然后点击 Deployment 后添加部署包。

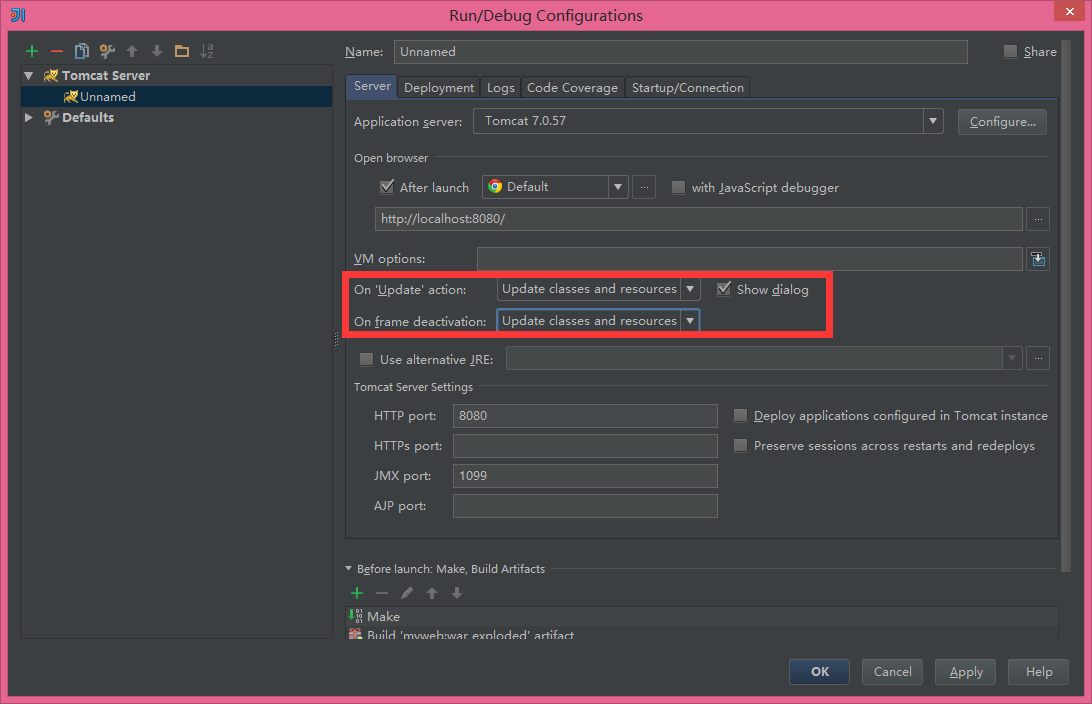


选择 Artifact

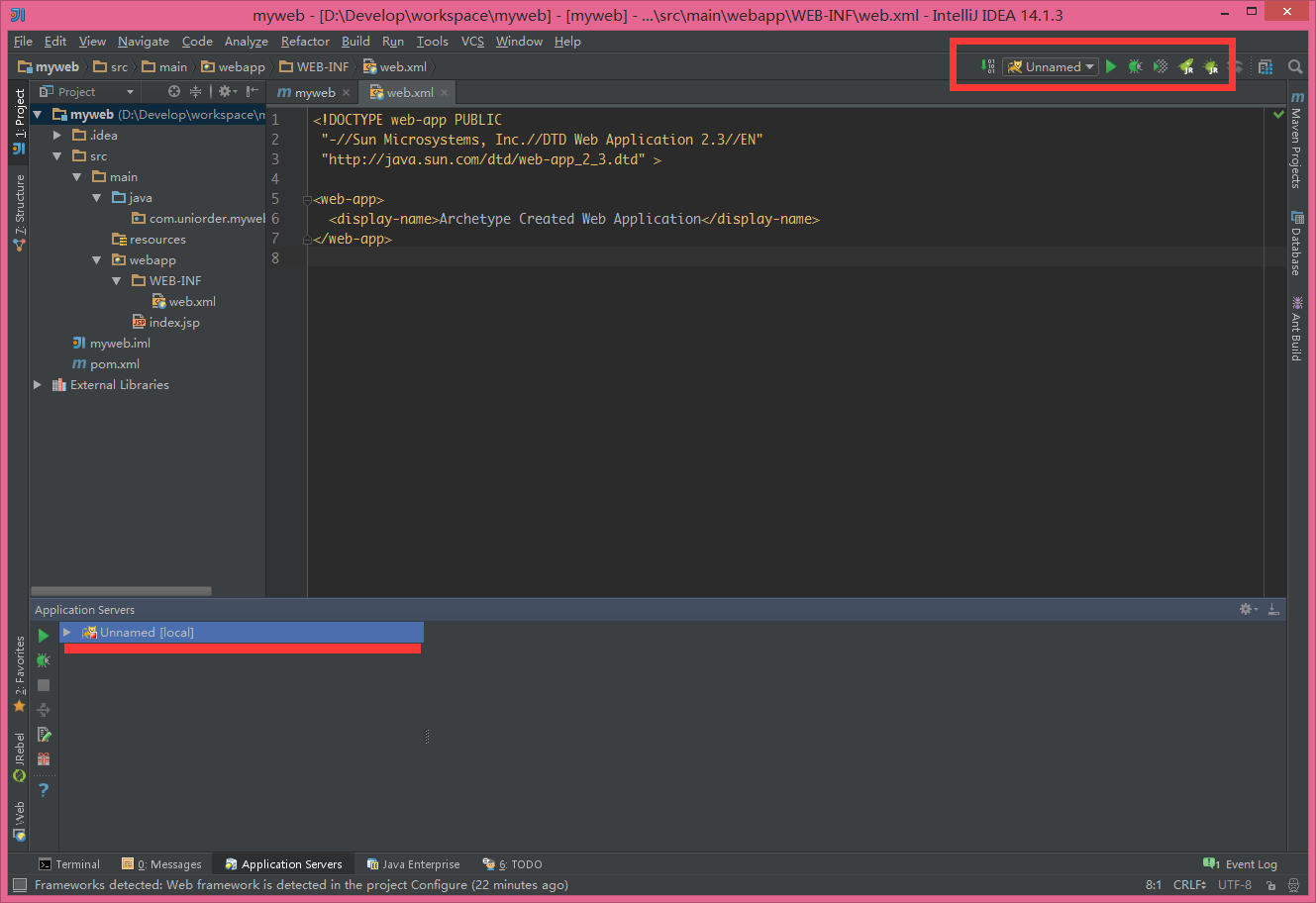
然后将我们的工程部署进去如下图所示：



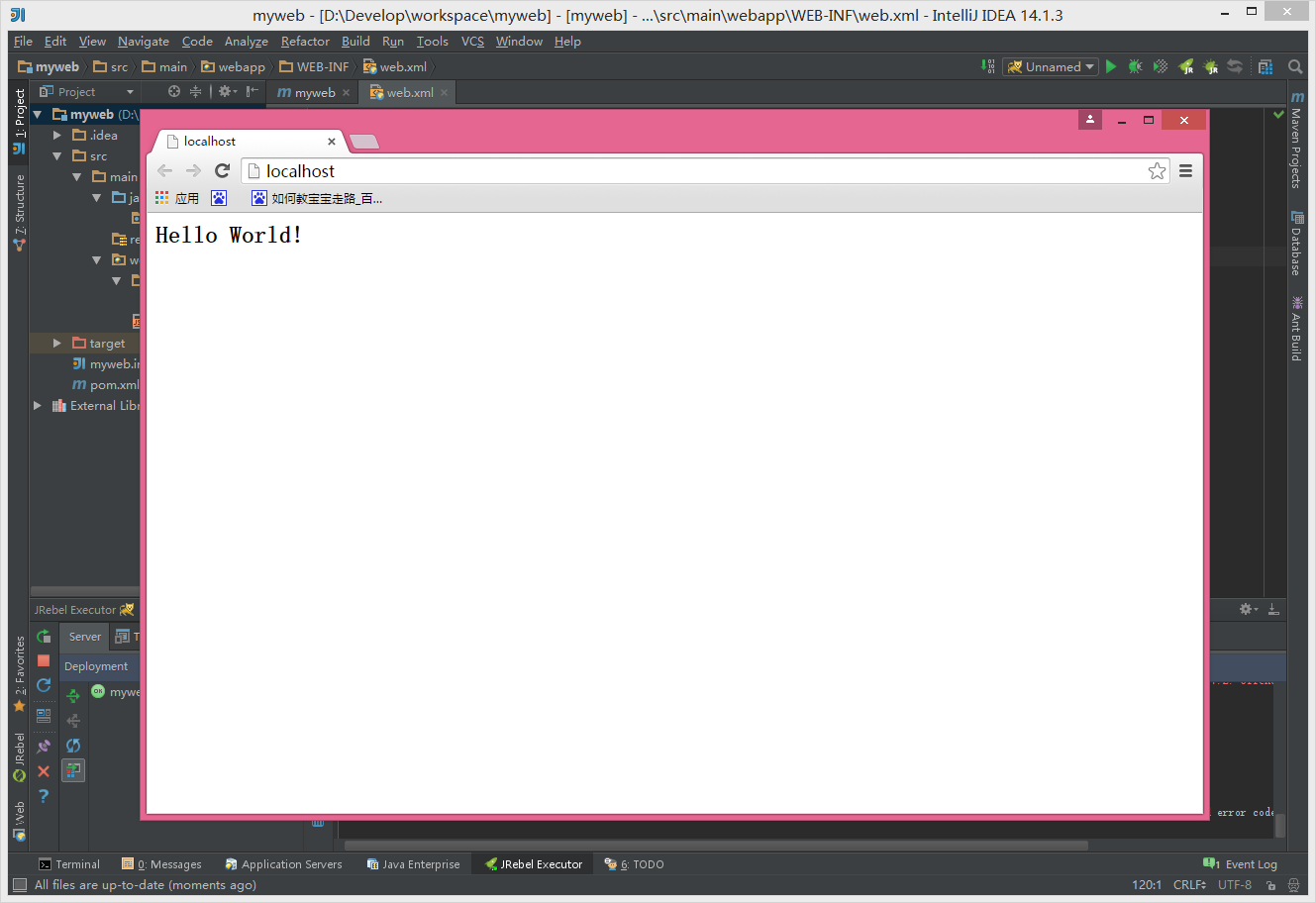
点击Ok之后，再回到Server界面，将On Update Action和 On frame deactivation设置成Update Class and Resource。如下图所示：



然后点击 Ok。我们可以看到我们的 Web 工程已经蓄势待发，准备启动了。如下图所示：



我们现在看到，在右上角有五个按钮，其中前面两个是通过IDEA启动 Tomcat 和通过IDEA以Debug模式启动Tomcat。后面两个是通过酷炫狂拽叼炸天的 JRebel 启动和通过同样酷炫狂拽叼炸天的 JRebel Debug 模式启动。既然我们已经装了 JRebel，那么我们肯定首选使用 JReble 启动 Tomcat。

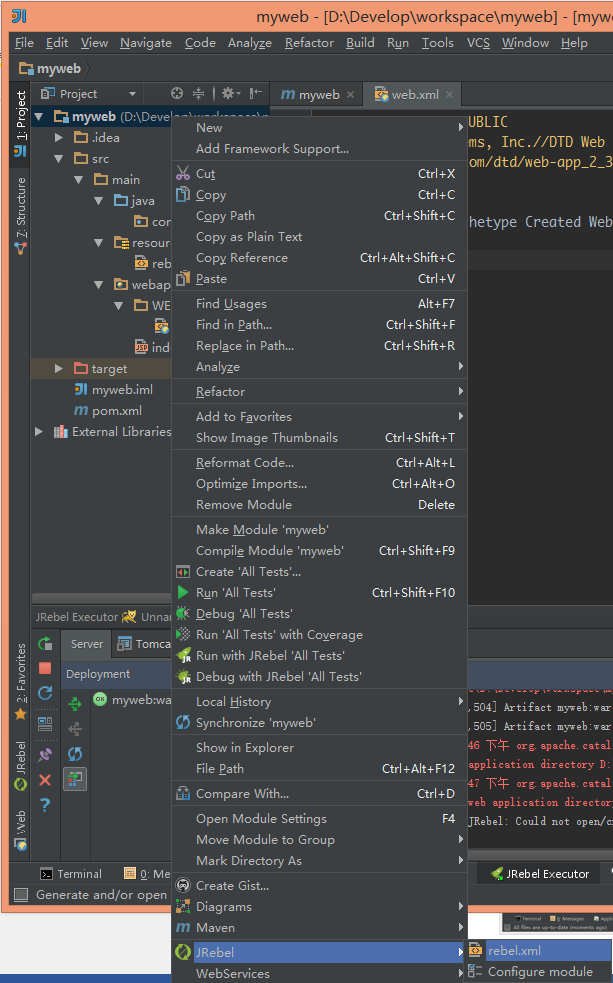


启动成功后会自动打开浏览器

既然说到 JRebel 狂拽吊炸天，那么就介绍一下 JRebel。JRebel的主要作用是热部署，就是当你修改了Java类、Spring 配置文件之后，不需要重启 Tomcat 就能看到效果，这样能够节省很多很多的开发时间。在新版本中，甚至连修改静态的Java类、方法都能热部署上去，相当的彪悍。

但是美中不足的是，修改 Velocity 的宏文件，还是需要重启 Tomcat 的。这一点不知道为什么。

完全使用 JReble 的功能，需要为项目添加一个 JReble的配置文件。不过这个配置文件不需要我们去编写，只需要导入即可：



加入 JReble 的配置文件

这个配置文件会存在于你的 resources目录下。在提交SVN的时候请ignore掉。

下面是工程文件的目录说明：

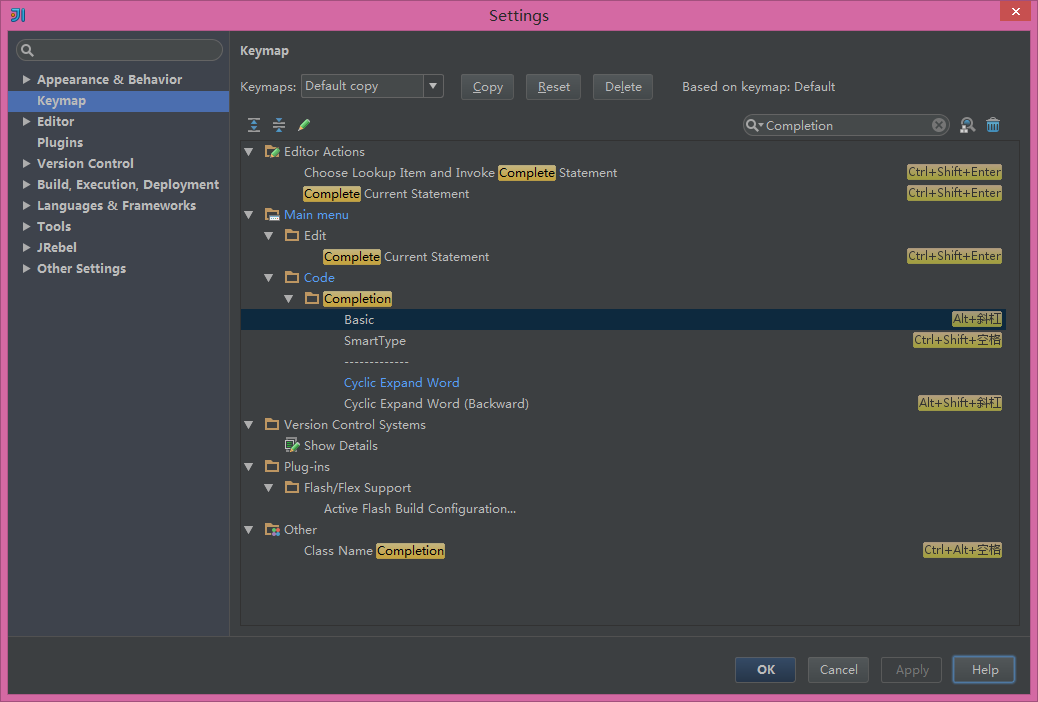
|  |  |
| --- | --- |
| .idea | IntelliJ 工程描述 |
| src | 项目代码 |
| test | 测试文件夹 |
| pom.xml | Maven 的POM文件 |
| .iml | IntelliJ工程描述 |

其中.idea和 \*.iml在提交SVN的时候需要ignore掉。

|  |
| --- |
| *题外话：*  *如果我们想要要手动指定一下我们的Maven目录，（虽然IDEA自导了内嵌式的Maven，但是我们还是习惯使用自己的Maven，毕竟修改 Settings之类的东西更加有把握一些）需要通过Ctrl+Shift+S打开设置窗口，搜索Maven，并选择Maven的安装目录，如下图所示：*    *然后在POM中加入一个实验性质的依赖——Spring-core，在第一次输入的时候，不会有任何提示，不过只需要更新本地索引（在报错的地方按下Alt+Insert快捷键，选择Update Indices……），只要更新下来所有的索引，以后所有的依赖代码都会有提示。只是因为国内的缘故，要想实现这一点，真的不太容易。*    *总共177.25兆，你很难有耐心……不过它是后台运行的，你可以不用管它，总有一天会下载完毕的。* |

# 快捷键

IntelliJ IDEA 有很多很多的快捷键，与 Eclipse 不同的是，IDEA的快捷键很少有两个的组合，通常情况下都是三个按键进行组合，这一点不如 Eclipse。但是，在 IDEA中，我们常用的快捷键要比 Eclipse 少很多。下面就介绍最常用的几个快捷键。



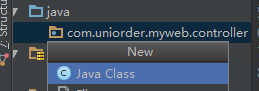
修改代码提示快捷键从CTRL+空格改成ALT+斜杠

## SHIFT+F6

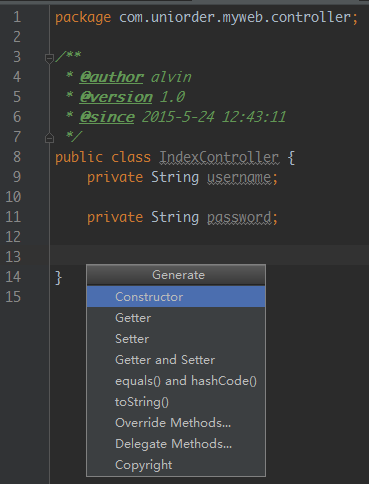
为文件、变量改名字。

## CTRL+INSERT 插入

这个快捷键是使用频率最高的快捷键，所有需要新建东西的地方都可以用这个快捷键完成操作，例如新建文件：



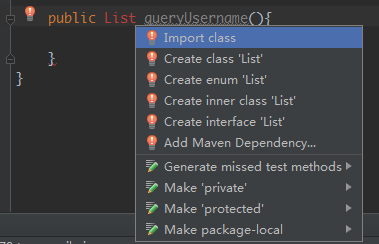
选中目录，按CTRL+INSERT创建新的文件



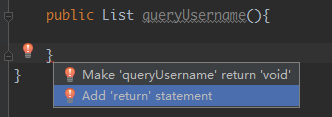
在Java 类中调出代码生成

## ALT+ENTER 改错

这个快捷键的作用是修补错误的代码，例如：没有导入包、错误的代码补全等的。



ALT+ENTER自动导入包

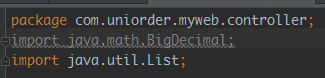


解决错误

总而言之，ALT+ENTER完成了Eclipse中CTRL+F1的功能，不过从操作上来讲，ALT+ENTER更加顺手一些。

## CTRL+ALT+O 清理包

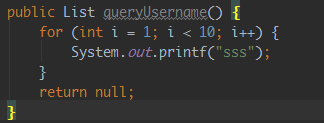
这个快捷键是清理没有使用到的包，并格式化导入的包。



CTRL+ALT+O可以清理掉灰色没有用到的包

## CTRL+ALT+L 格式化代码

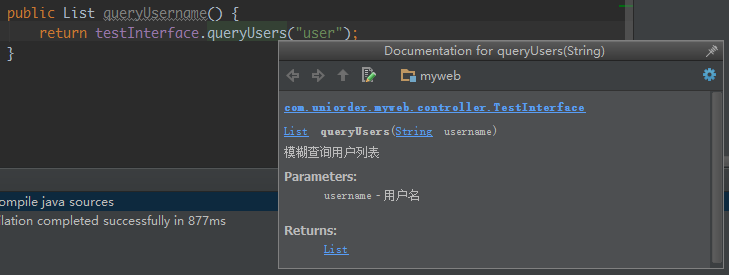
格式化代码，如果选中代码块，就格式化选中的代码块，否则就格式化整个类里的代码。



CTRL+ALT+T格式化代码

## CTRL+Q 显示注释

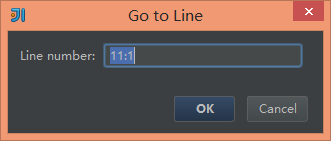
显示注释



CTRL+Q显示注释

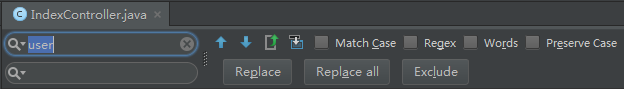
## CTRL+G 代码定位

代码定位，可以直接将光标跳转到你输入的代码行和列。



CTRL+G 代码定位

## CTRL+R 替换



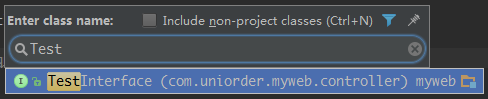
## CTRL+Z 恢复

Redo

## CTRL+SHIFT+Z 回滚

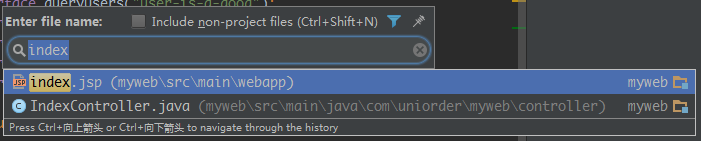
Undo，类似Eclipse中的Ctrl+Y

## CTRL +N 根据类名查找类



CTRL +N 查找类、借口、枚举……

## CTRL+SHIFT+N 根据文件名打开文件

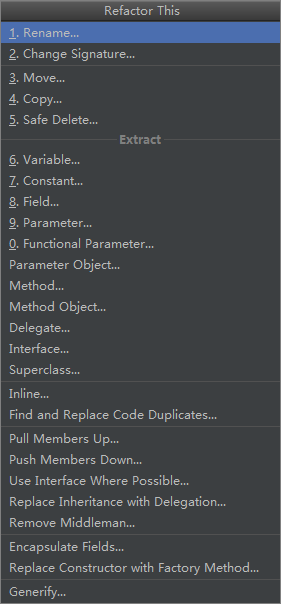


## SHIFT+F2 快速定位到出错的代码



SHIFT+F2 直接跳转到出错的代码

## CTRL+SHIFT+ALT+T

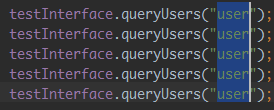


重构一切可以重构的东西……

## CTRL+W 单词选中



## 按住 ALT 列选模式



列选批量编辑

Java的编码style（format）

