

目錄

Introduction	1.1
1.IntelliJ IDEA 介绍（新用户必看）	1.2
2.本教程介绍（新用户必看）	1.3
3.Windows 下安装	1.4
4.Ubuntu 下安装	1.5
5.Mac 下安装	1.6
6.安装总结（新用户必看）	1.7
7.首次运行（新用户必看）	1.8
8.安装目录讲解、IDE 设置云同步（新用户必看）	1.9
9.界面讲解（新用户必看）	1.10
10.主题、字体、编辑区主题、文件编码修改、乱码问题（新用户必看）	1.11
11.各类文件类型图标讲解（新用户必看）	1.12
12.索引的讲解（新用户必看）	1.13
13.编译方式讲解（新用户必看）	1.14
14.项目相关概念讲解（新用户必看）	1.15
15.Hello World 的 Java 项目创建和项目配置文件讲解	1.16
16.版本控制讲解	1.17
17.实时代码模板讲解	1.18
18.文件代码模板讲解	1.19
19.Emmet 讲解	1.20
20.Postfix Completion 讲解	1.21
21.插件讲解	1.22
22.Eclipse 的 Java Web 项目环境搭建	1.23
23.Maven 项目介绍	1.24
24.Maven 的单模块 / 多模块之 Spring MVC + Spring + Mybatis 项目讲解（重点）	1.25
25.Maven 的单模块之 Spring MVC + Spring + Spring Data JPA 项目（基于 IntelliJ IDEA）	1.26
26.Debug 讲解	1.27
27.重构讲解	1.28
28.数据库管理工具	1.29

29.IntelliJ IDEA 常用设置-1	1.30
30.IntelliJ IDEA 常用设置-2	1.31
31.IntelliJ IDEA 常用设置-3	1.32
32.IntelliJ IDEA 常用设置-4	1.33
33.IntelliJ IDEA 常用快捷键讲解（Win+Linux）（新用户必看）	1.34
34.IntelliJ IDEA 常用快捷键讲解（Mac）（新用户必看）	1.35
35.从 Windows 过度到 Mac 必备快捷键对照表（新用户必看）	1.36
36.IntelliJ IDEA 的 Java 热部署插件 JRebel 安装及使用	1.37
37.IntelliJ IDEA 远程调试（Tomcat+Jetty）	1.38
38.最特殊的快捷键 Alt + Enter 介绍（新用户必看）	1.39
39.本教程总结	1.40

重要说明，看三遍

Fork 次库用于制作 **Gitbook** 方便阅读，仅为个人学习，请勿用于商业目的。

介绍(Introduce)

- 教程主要面向中文用户，如果你英文良好，建议直接阅读官网帮助文档
 - 官网帮助中心：<http://www.jetbrains.com/idea/webhelp/getting-help.html>
- 教程目前在不定时进行调整和补充，需要关注更新的请 `Watch`、`Star`、`Fork`。
- 特别需要友情提醒的是：请 `Fork` 之后，在我的基础上按你自己喜欢的方式整理一套属于你自己的快捷键列表，并导出为 PDF，以备不时查阅，对于提升开发效率是很有帮助的！文章的图片建议在需要的时候可以 右键 - 查看图像（在新标签页打开图片） 进行原图查看。或者建议你放大页面缩放比例（快捷键 `Ctrl` + 鼠标滚轮），可以更加清楚地看清图片细节。
- 同时邀请您一起参与完善该教程，帮助更多的人，欢迎反馈错误和意见！
- 本系列文章唯一授权的商业网站是：[极客学院](#)，其他商业网站一律禁止转载。公益站点、个人博客、公众号等载体请在转载写明出处链接。
- 如果你只是单纯要阅读的话，建议移步极客学院上观看，访问速度快很多：
 - 地址：<http://wiki.jikexueyuan.com/project/intelliJ-idea-tutorial/>
- 如果你想参与完善该教程，请移步到 Github 上进行 Fork：
 - 地址：<https://github.com/judasn/IntelliJ-IDEA-Tutorial/>
- 如果你需要一份电子版，请查看（制作电子版很费精力，不会常更新此文件）：
 - 百度云：<http://pan.baidu.com/s/1i3wFYPB>
 - Google Drive：<https://drive.google.com/file/d/0B5...UU/view?usp=sharing>
- 2016-10-25 更新，感谢 district10 童鞋做的一个在线阅读版本以及提供打包功能：
 - district10 主页：<https://github.com/district10>
 - 在线阅读：<http://whudoc.qiniudn.com/2016/IntelliJ-IDEA-Tutorial>
 - 下载打包：<http://whudoc.qiniudn.com/2016/publish-IntelliJ-IDEA-Tutorial-newMaster.zip> (90.4 MB)
 - 提供的转换脚本：把文件夹下的 Markdown 文件，转化成 GitHub 风格的 HTML。◦

目录(Contents)

- 1.IntelliJ IDEA 介绍（新用户必看）
- 2.本教程介绍（新用户必看）
- 3.Windows 下安装
- 4.Ubuntu 下安装
- 5.Mac 下安装
- 6.安装总结（新用户必看）
- 7.首次运行（新用户必看）
- 8.安装目录讲解、IDE 设置云同步（新用户必看）
- 9.界面讲解（新用户必看）
- 10.主题、字体、编辑区主题、文件编码修改、乱码问题（新用户必看）
- 11.各类文件类型图标讲解（新用户必看）
- 12.索引的讲解（新用户必看）
- 13.编译方式讲解（新用户必看）
- 14.项目相关概念讲解（新用户必看）
- 15.Hello World 的 Java 项目创建和项目配置文件讲解
- 16.版本控制讲解
- 17.实时代码模板讲解
- 18.文件代码模板讲解
- 19.Emmet 讲解
- 20.Postfix Completion 讲解
- 21.插件讲解
- 22.Eclipse 的 Java Web 项目环境搭建
- 23.Maven 项目介绍
- 24.Maven 的单模块 / 多模块之 Spring MVC + Spring + Mybatis 项目讲解（重点）
- 25.Maven 的单模块之 Spring MVC + Spring + Spring Data JPA 项目（基于 IntelliJ IDEA）
- 26.Debug 讲解
- 27.重构讲解
- 28.数据库管理工具
- 29.IntelliJ IDEA 常用设置-1
- 30.IntelliJ IDEA 常用设置-2
- 31.IntelliJ IDEA 常用设置-3
- 32.IntelliJ IDEA 常用设置-4
- 33.IntelliJ IDEA 常用快捷键讲解（Win+Linux）（新用户必看）
- 34.IntelliJ IDEA 常用快捷键讲解（Mac）（新用户必看）
- 35.从 Windows 过度到 Mac 必备快捷键对照表（新用户必看）
- 36.IntelliJ IDEA 的 Java 热部署插件 JRebel 安装及使用
- 37.IntelliJ IDEA 远程调试（Tomcat+Jetty）
- 38.最特殊的快捷键 Alt + Enter 介绍（新用户必看）
- 39.本教程总结

联系(Contact)

- Email : judas.n@qq.com (常用) or admin@youmeek.com (备用)
- Blog : <http://code.YouMeek.com>
- IntelliJ IDEA QQ 交流群：入群请看：<https://github.com/judasn/IntelliJ-IDEA-Java-Conversation>
- 欢迎捐赠 ^_^ : <http://www.youmeek.com/donate>

Github 协同视频教程(Participate)

- 如果您不会使用 Git 或是 Github 也没关系，请认真学习下面视频教程：
- Judas.n 录制
 - 视频格式：MP4
 - 分辨率：1920 X 1080
 - 片长：16 Min
 - 文件大小：62 M
- 下载
 - 百度云盘：<http://pan.baidu.com/s/1bogmTLd>
 - 360 网盘（2fb5）：<https://yunpan.cn/cYez7W9xnHs3c>

Github 常用按钮说明

- Watch : 关注该项目，作者有更新的时候，会在你的 Github 主页有通知消息。
- Star : 收藏该项目，在你的头像上有一个“Your stars”链接，可以看到你的收藏列表，以方便下次进来。
- Fork : 复制一份项目到自己的 Github 空间上，你可以自己开发自己的这个地址项目，然后 Pull Request 给项目原主人。

参与作者汇总(Author)

- 真心感谢这些志同道合的人，这个真的很重要，也希望你能一起参与（鞠躬）！
- 同时感谢那些通过私聊方式指出一些错误地方的朋友，使得该教程能得以更加完善，真心感谢（鞠躬）！

作者(按参与时间排序)	地址
Judas.n	http://code.YouMeek.com
温泉	https://github.com/wenquan0hf
zhenhappy	https://github.com/zhenhappy
two8g	https://github.com/two8g
Dectinc	https://github.com/Dectinc
Caliven	https://github.com/caliven
MinjieTao	https://github.com/MinjieTao
classloader	https://github.com/classloader
challengeof	https://github.com/challengeof
district10	https://github.com/district10

介绍

本系列教程介绍

本系列教程从 IntelliJ IDEA 的安装、卸载、软件设置、项目配置等各个方面进行讲解。通过本系列教程的学习，也希望你能爱上 IntelliJ IDEA，爱上它的体贴。同时学完本系列教程对于你学习 JetBrains 公司下的其他产品也有好处，其他产品包括：

- [PhpStorm](#) 主要用于开发 PHP
- [RubyMine](#) 主要用于开发 Ruby
- [PyCharm](#) 主要用于开发 Python
- [AppCode](#) 主要用于开发 Objective-C / Swift
- [CLion](#) 主要用于开发 C / C++
- [WebStorm](#) 主要用于开发 JavaScript、HTML5、CSS3 等前端技术
- [0xDBE](#) 主要用于开发 SQL
- [Android Studio](#) 主要用于开发 Android（Google 基本 IntelliJ IDEA 社区版进行迭代所以也姑且算上）

IntelliJ IDEA 介绍

- IntelliJ IDEA 官网：<https://www.jetbrains.com/idea/>

IntelliJ IDEA 在 2015 年 06 月官网主页是这样介绍自己的：

Excel at enterprise, mobile and web development with Java, Scala and Groovy, with all the latest modern technologies and frameworks available out of the box.

简明翻译：IntelliJ IDEA 主要用于支持 Java、Scala、Groovy 等语言的开发工具，同时具备支持目前主流的技术和框架，擅长于企业应用、移动应用和 Web 应用的开发。

IntelliJ IDEA 对自己的定义是很清晰的，对于新人来讲可能还不太理解，可能还会有误会，认为它博而不精，但是对于老用户来讲应该是非常认可上面这句话的。通过下面功能表格，新人对于 IntelliJ IDEA 所具备的功能会有一个新的认识。

如果用一句话来形容 IntelliJ IDEA，我会说：IntelliJ IDEA 是目前所有 **IDE** 中最具备沉浸式的 **IDE**，没有之一。

IntelliJ IDEA 主要功能介绍

- 语言支持上：

安装插件后支持	SQL类	基本JVM
PHP	PostgreSQL	Java
Python	MySQL	Groovy
Ruby	Oracle	
Scala	SQL Server	
Kotlin		
Clojure		

- 其他支持：

支持的框架	额外支持的语言代码提示	支持的容器
Spring MVC	HTML5	Tomcat
GWT	CSS3	TomEE
Vaadin	SASS	WebLogic
Play	LESS	JBoss
Grails	JavaScript	Jetty
Web Services	CoffeeScript	WebSphere
JSF	Node.js	
Struts	ActionScript	
Hibernate		
Flex		

上面特性只是 IntelliJ IDEA 的冰山一角，而且这个还不是 IntelliJ IDEA 最重要的地方，IntelliJ IDEA 最重要的特性就是人性化、智能，后面学习你会慢慢接触到。

更多官网信息

- IntelliJ IDEA 主要特性介绍 1 : <https://www.jetbrains.com/idea/features/>
- IntelliJ IDEA 主要特性介绍 2 : https://www.jetbrains.com/idea/features/editions_comparison_matrix.html
- 官网博客 : <http://blog.jetbrains.com/idea/>
- 官网 wiki : <http://wiki.jetbrains.net/intellij>
- 官网社区 : <http://devnet.jetbrains.com/community/idea>
- 官网快速入门 : <http://confluence.jetbrains.com/display/IntelliJIDEA/Quick+Start>
- 官网在线帮助文档 : <http://www.jetbrains.com/idea/webhelp/getting-help.html>

关于

学习前提

由于 IntelliJ IDEA 官网在亚洲没有设服务器，且官网用到一些类似 Twitter、Facebook 等站的脚本会使得你在国内出现访问巨慢或是不允许访问的特殊情况，所以建议你在访问官网、访问插件库、小版本本地迭代更新等操作的时候出现奇怪问题的时候，请自备VPN等网络加速工具。

很多用户都是先学习了 Eclipse、MyEclipse 再转到 IntelliJ IDEA 的，这里需要先说明的是，在学习 IntelliJ IDEA 过程中，你暂且要放下 Eclipse 下的开发思维方式，不能按 Eclipse 的软件思想或是结构去要求 IntelliJ IDEA，这样对你学习 IntelliJ IDEA 非常不利。

适用人群

用 IntelliJ IDEA 进行开发语言的学习者。

用 IntelliJ IDEA 进行开发语言的开发工作者。

其中对于语言开发者我是非常建议你使用 IntelliJ IDEA，因为一些代码格式、命名规范在 IntelliJ IDEA 下都是有良好的提示，对于我们所处的输入法下的中文全角符号也可以得到快速发现。特别是学习 Python 的学习者，当你在用 Pycharm 进行学习的时候，Pycharm 会时刻告诉你什么时候要注意空格、换行，提醒你有 PEP8 编码规范，你也可以通过快捷键快速格式化出适合 Python 要求的代码，这对于学习者来讲，真的很重要，它可以让你更专注于自己的代码。

教程演示的 IntelliJ IDEA 版本

IntelliJ IDEA 13 版本和 14 版本，在设置上差异很大，14 版本 IntelliJ IDEA 对整个 IDE 的设置进行了重新编排、归类，但是细节设置上所沿用的介绍是没有多大改变的。

目前（2015 年 06 月）IntelliJ IDEA 官网最新版本信息为：**Version:14.1.4 Build:141.1532.4 Released:June 19th, 2015**。

IntelliJ IDEA 有旗舰版和社区版本之分，本系列教程将以 14.1.4 的旗舰版进行演示和讲解。

其中旗舰版（Ultimate Edition）为收费版本，有 30 天试用期。如果你是学生、老师、开源项目参与者都可以向官网免费试用旗舰版，具体你可以查看下面链接。社区版（Community Edition）为免费版本，功能较旗舰版少了很多。

本教程使用的 IntelliJ IDEA 主题为较受欢迎的黑色：**Darcula**。

- 申请免费版本：<https://www.jetbrains.com/idea/buy/>
- 旗舰版和社区版差异细节：https://www.jetbrains.com/idea/features/editions_comparison_matrix.html

教程演示的系统环境

- 系统：Windows 8.1 64 位 简体中文版
- JDK 版本：1.8.0_05 64 位
- 建议使用 JDK 版本为：1.6 及 1.6 以上，更加详细的系统要求会在安装教程篇中进行讲解。

IntelliJ IDEA 版本迭代习惯

2015 年 IntelliJ IDEA 主版本是 14，目前（2015 年 06 月）最新版本是 14.1.4。与此同时，2015 年 06 月 17 日，官网开始提供 15 EAP 版本（Early Access Program 早期预览版）。如果你对 IntelliJ IDEA 下个大版本的新特性很感兴趣，你可以随时关注官网博客最新动态。

按正常情况来讲，IntelliJ IDEA 大版本是一年迭代一次。大版本下的小版本迭代时间没有固定，快的是一个月不到就迭代一次，慢的话基本在两到三个月迭代一次。相对其他 IDE 来讲迭代周期还是比较紧凑，但是作为用户你不用担心因为频繁迭代更新而引起的项目配置问题或是软件配置问题，后面有课程会专门对此进行说明。

- IntelliJ IDEA 官网博客：<http://eap.jetbrains.com/idea>

Windows 系统下安装 IntelliJ IDEA

系统要求

- 系统支持 : Microsoft Windows 8 / 7 / Vista / 2003 / XP (每个系统版本的 32 位和 64 位都可以)
- JDK 版本 : Oracle JDK 1.6 或以上
- 内存 : 最低要求 1 GB, 推荐 2 GB 以上
- 硬盘 : 最低要求 2 GB
- 显示器 : 最低要求 1024 X 768 分辨率
- 更多信息可以阅读

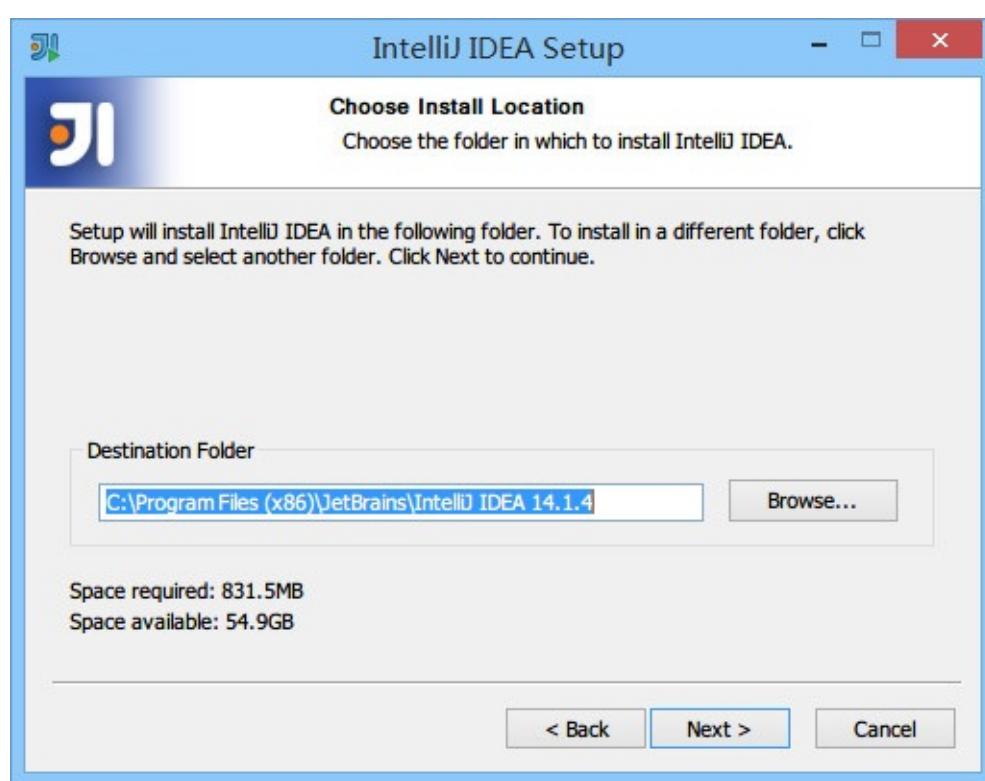
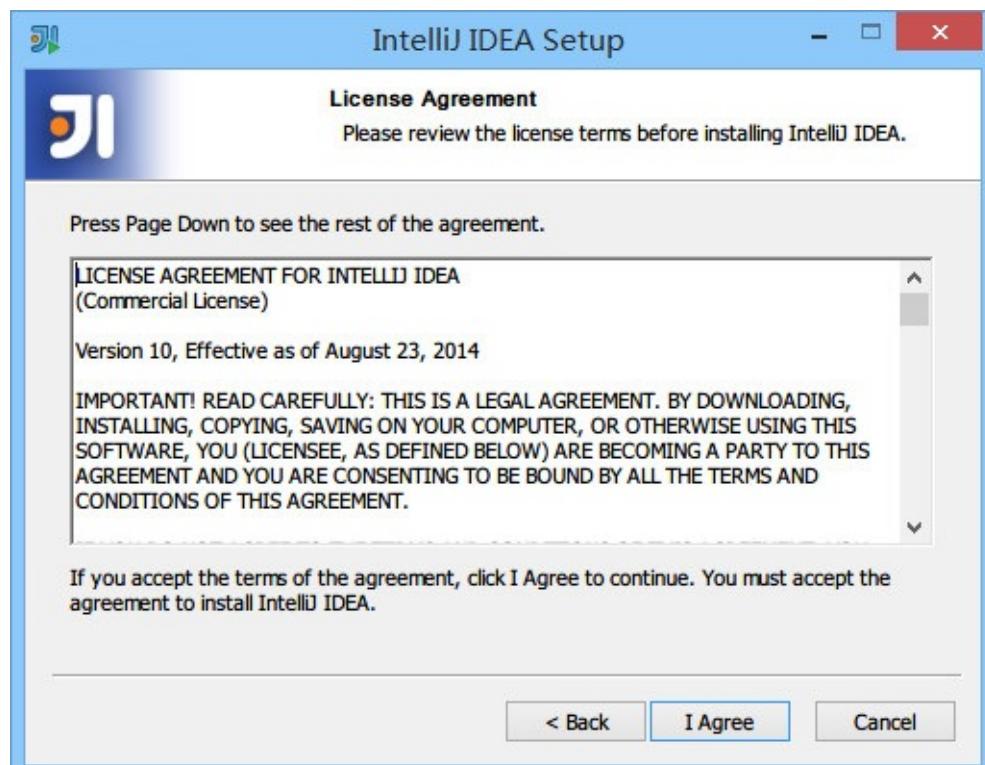
读 : https://www.jetbrains.com/idea/download/system_requirements.jsp?os=win

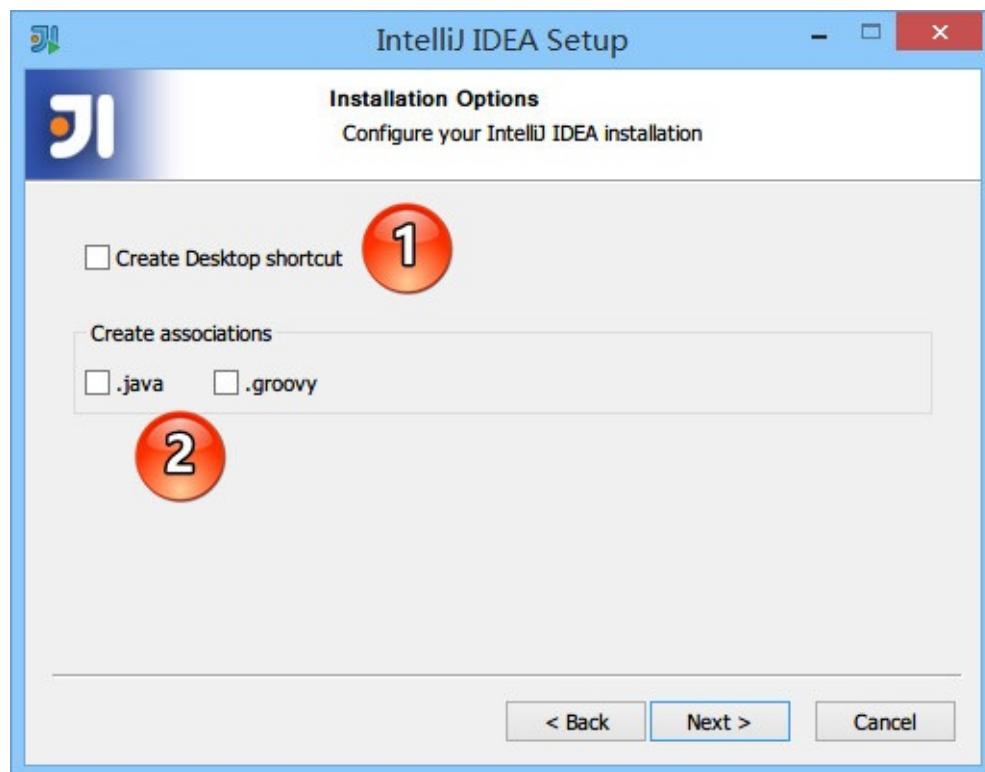
首次安装

- IntelliJ IDEA 的安装是非常简单的, 不需要做过多的选择, 可以说简单到都是 **Next** 即可。



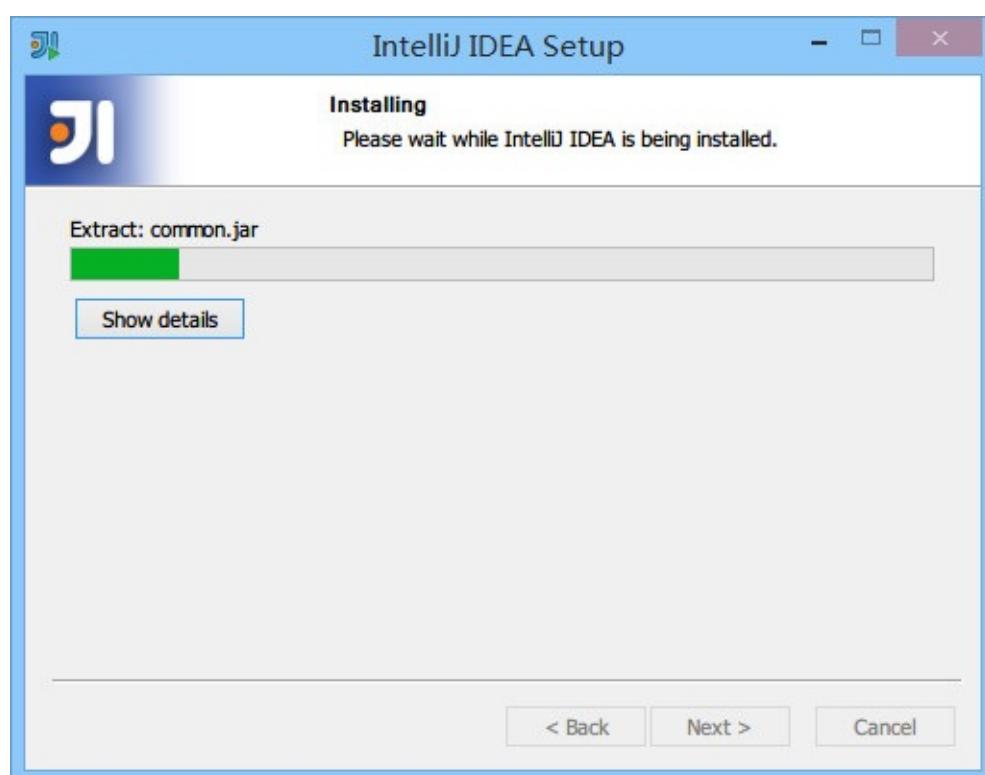
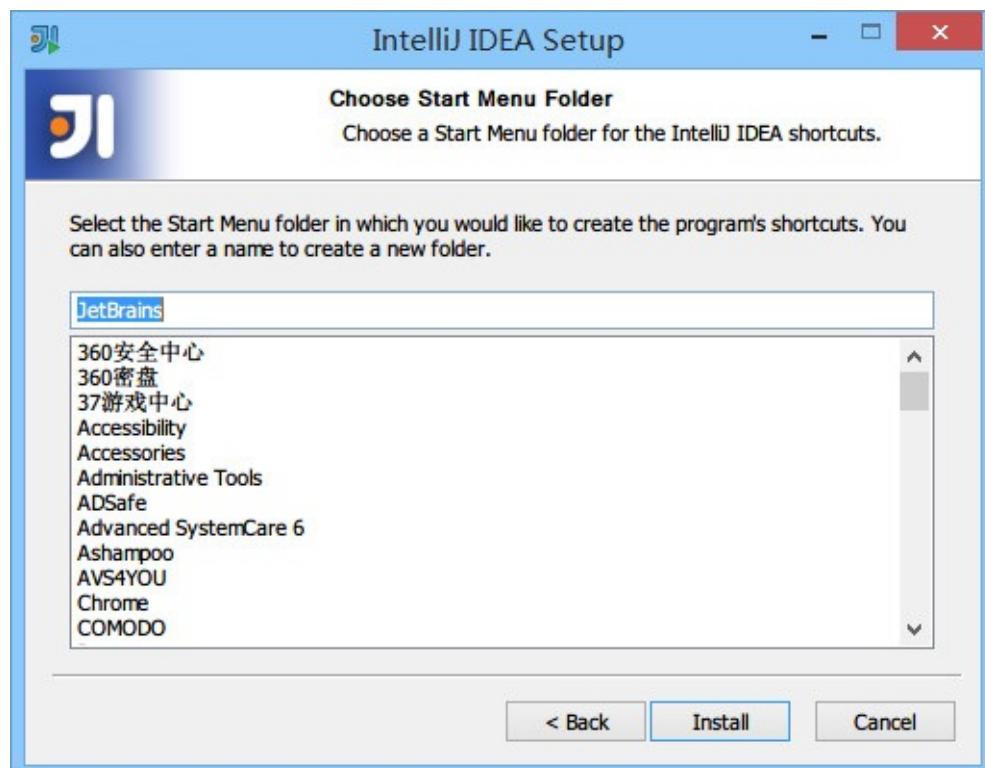
3. Windows 下安装





- 上图标记 1 表示在桌面上创建一个快捷图标，建议勾选上，方便我们在安装后定位 IntelliJ IDEA 安装目录。
- 上图标记 2 表示关联 Java 和 Groovy 文件，建议都不要勾选，正常我们会在 Windows 的文件系统上打开这类文件都是为了快速查阅文件里面的内容，如果用 IntelliJ IDEA 关联上之后，由于 IntelliJ IDEA 打开速度缓慢，这并不能方便我们查看。
- 建议在 Windows 系统上关联此类文件可以用 EmEditor、Notepad++ 这类轻便的编辑器。

3. Windows 下安装

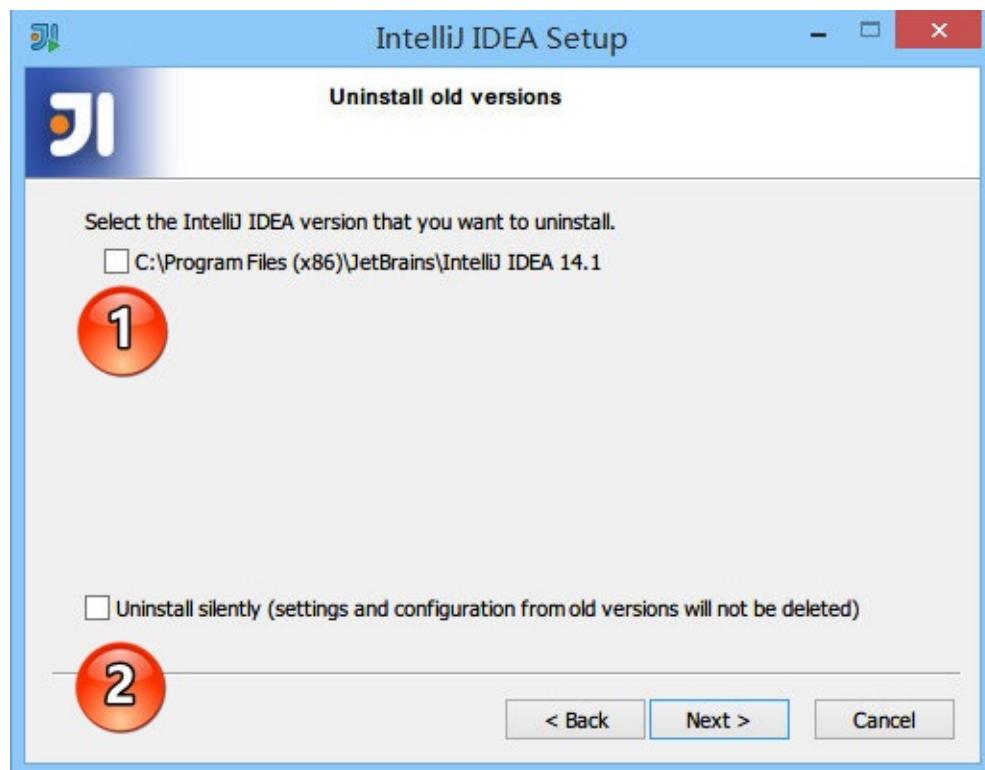




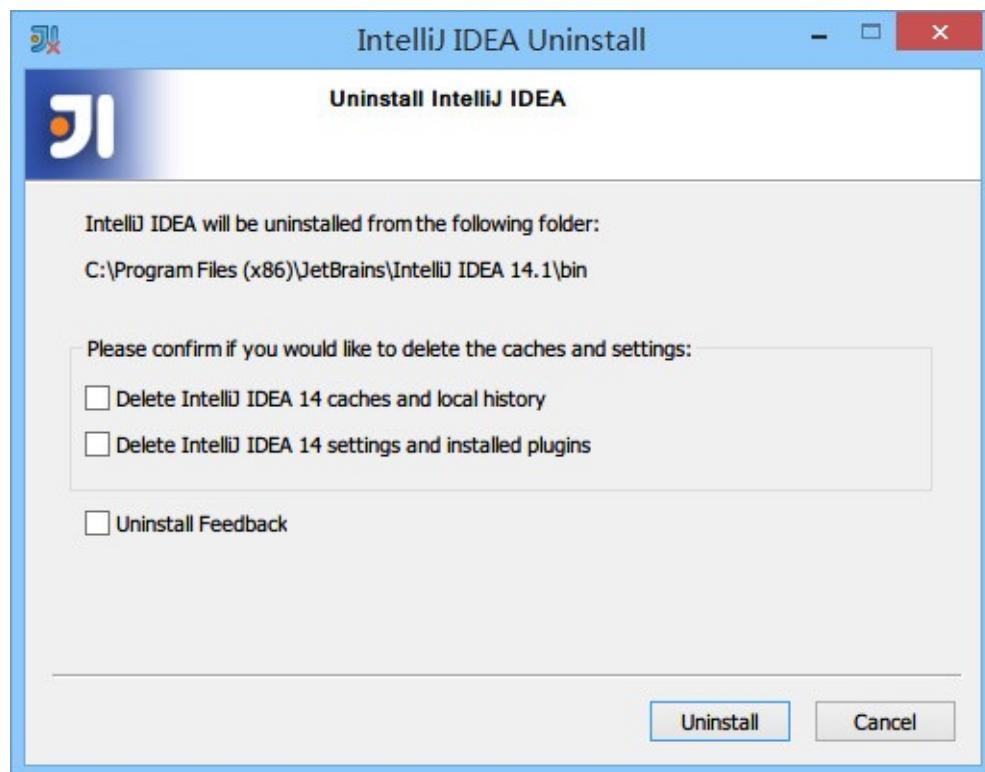
- 整个安装过程，一般的配置电脑安装所需的时间大约是 1~5 分钟。

已有旧版本安装新版本





- 上图，显示我目前电脑中已经有一个 IntelliJ IDEA 版本，如果我勾选了标记 1，则表示安装之前会先卸载掉电脑上的旧版本。
- 上图标记 2，如果勾选了，则 IntelliJ IDEA 会直接安静地卸载旧版本，而旧版本的个性化设置不会被删除。
- 在小版本迭代中建议是卸载掉旧版本的，然后再进行新版本安装，因为小版本迭代一般都是 Bug 的修复，保留旧版本没有多大意义。
- 在大版本迭代中建议是保留旧版本，也就是不勾选上图标注 1，IntelliJ IDEA 是支持一台电脑装多个版本的。
- 接下来的步骤我们假设勾选了标注 1 再进行安装。

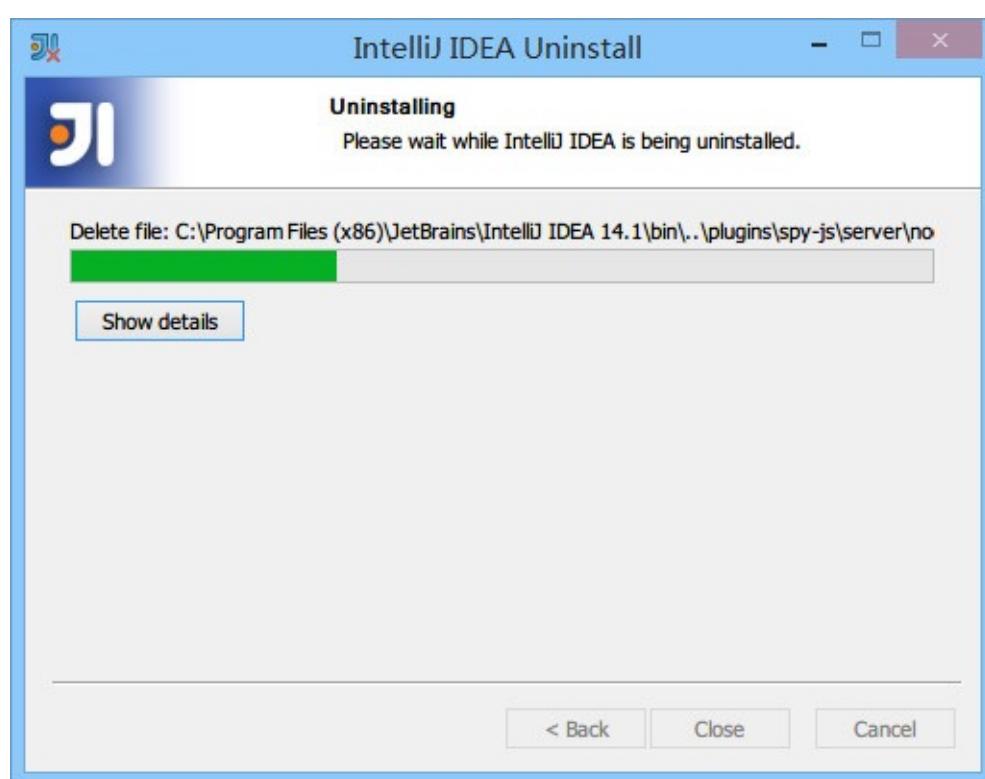
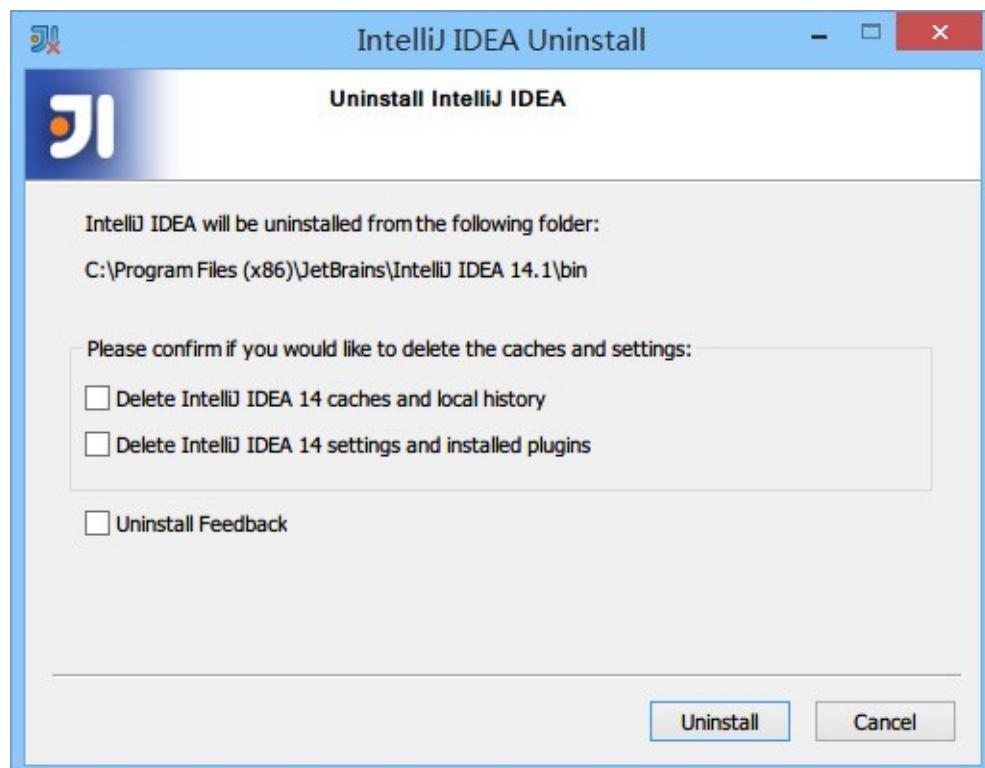


- 上图，由于上一步勾选了卸载旧版本选项，所以出现了选择删除旧版本的配置选项。
- 第一个选项：删除旧版本的缓存和本地历史记录。
- 第二个选项：删除旧版本的个人个性化设置。
- 建议两个都不要勾选。
- 点击 **uninstall**，进入全自动的卸载过程，卸载完成接下来的步骤跟上文“首次安装”一致，这里不再进行说明。

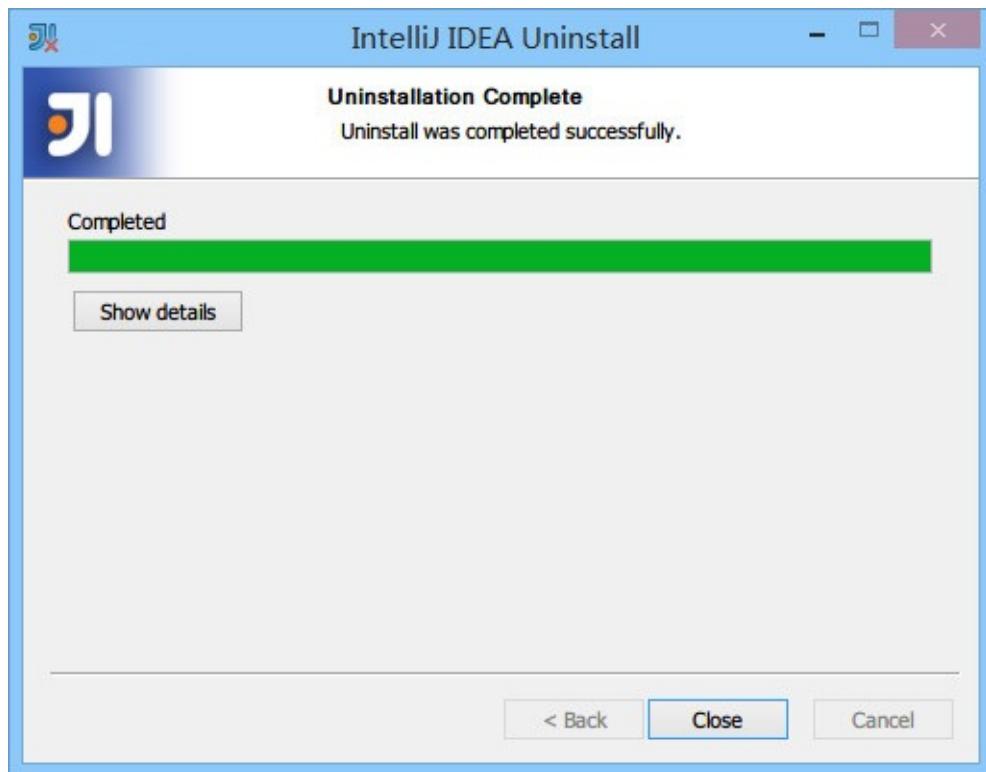
卸载

- 卸载过程在第 3 点已经有涉及到了，专门对 IntelliJ IDEA 进行卸载也是一样的流程。

3. Windows 下安装



3. Windows 下安装



Ubuntu 系统下安装 IntelliJ IDEA

系统要求

- 系统支持：只要是支持 GNOME 或 KDE 桌面系统，建议是 Ubuntu (32位和64位都可以)
- JDK 版本：Oracle JDK 1.6 或以上
- 内存：最低要求 1 GB，推荐 2 GB 以上
- 硬盘：最低要求 2 GB
- 显示器：最低要求 1024 X 768 分辨率
- 更多信息可以阅读

读：https://www.jetbrains.com/idea/download/system_requirements.jsp?os=linux

重要说明

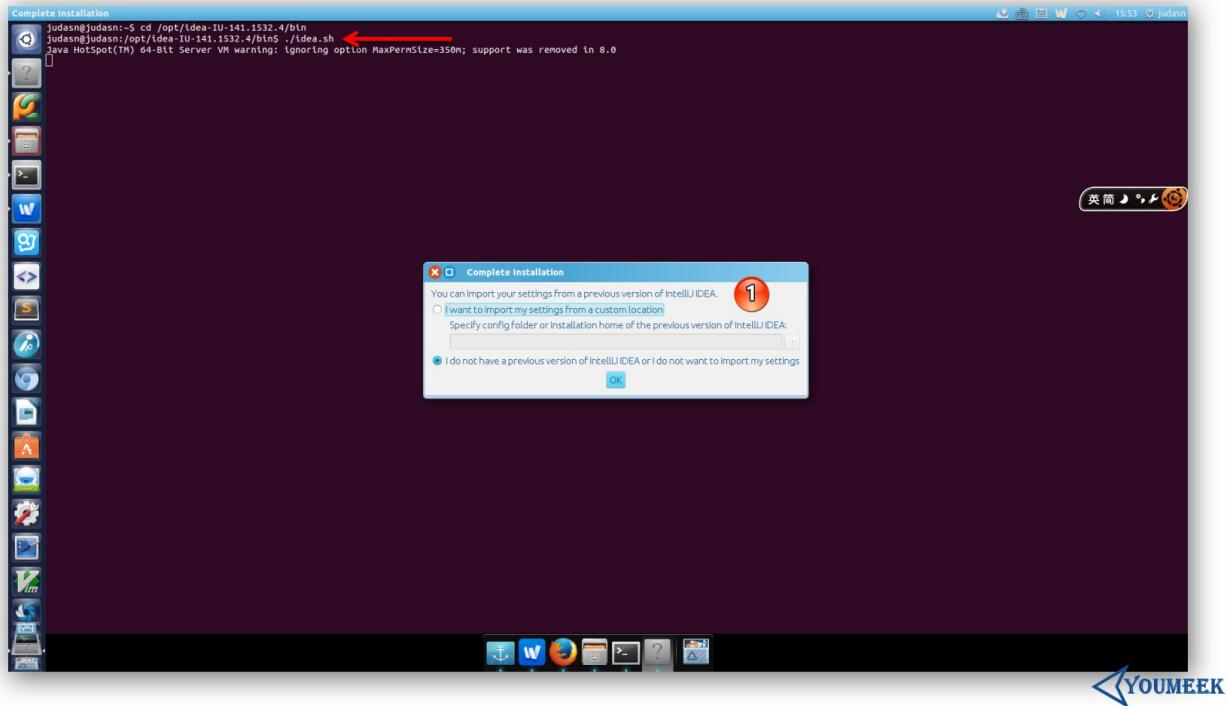
我这里以 Ubuntu 系统为例进行讲解。但是，在学习下面内容之前请先看下章节：[Windows 下安装](#)

因为它们配置流程是基本一样的，只是系统不同，开始的步骤不太一样而已，因此相同部分我这里是不会再讲的，我只讲 IntelliJ IDEA 在 Linux 安装特殊的地方。

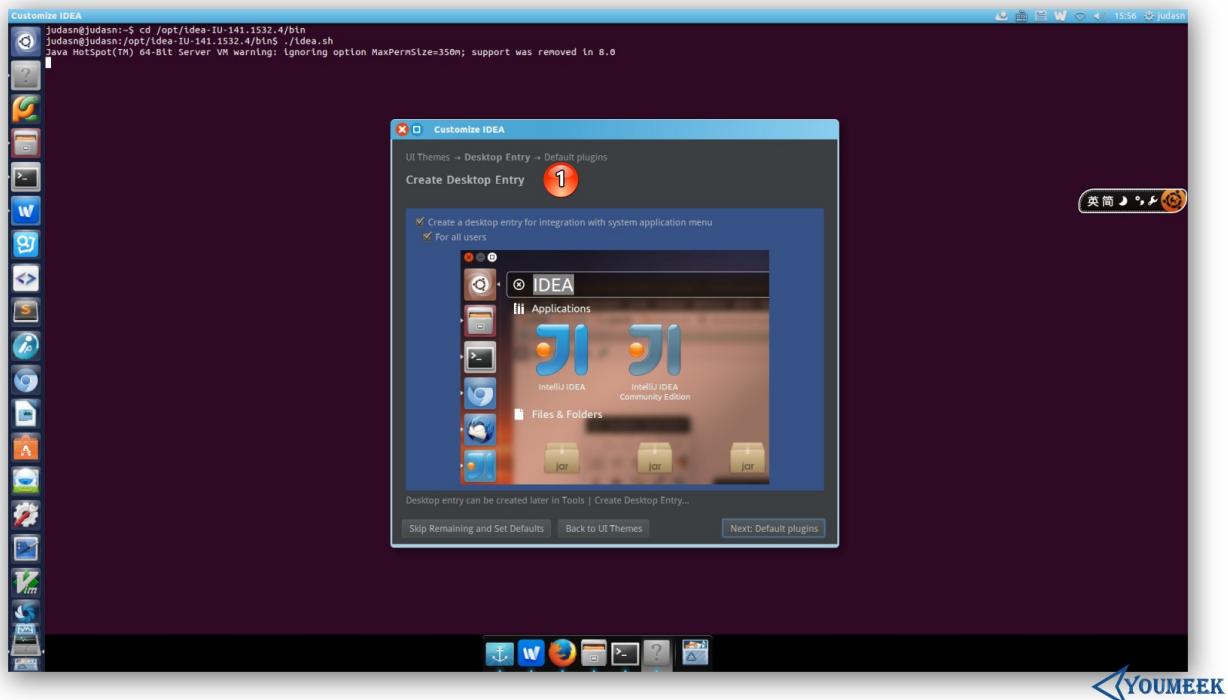
Ubuntu 下安装过程

先把你下载到的 `ideaIU-14.1.4.tar.gz` 移动到你平时存放软件的目录下，然后进行解压，我电脑是放在 `/opt` 下。

- 终端下解压命令：`tar xfz ideaIU-14.1.4.tar.gz`，解压出来的目录名称是：`idea-IU-141.1532.4`
- 可能在解压过程中你需要 `sudo` 命令权限，或者是切换到 `root` 账号下。如果你是切换到 `root` 用户下就一定要注意，解压完记得再切回来你常用的账户，不然等下生成的 IntelliJ IDEA 配置文件是放在 `/home/root` 下，这样就跟你常用的那个用户没啥关系了。



- 在假设你已经通过终端切换到了你常用的用户下之后，现在用终端进入解压目录下的 `bin` 子目录 下，然后在终端下运行启动命令：`./idea.sh`，运行的效果如上图箭头所示。剩下的配置步骤就跟 Windows 基本一样了，如标注 1 所示，所以这里不多讲。



- 其中，在整个首次启动的配置过程中，唯一跟 Windows 不太一样的就是上图标注 1 这个地方。原因是 Linux 下创建启动图标是非常非常非常的麻烦，所以 IntelliJ IDEA 帮我们考虑到了，所以只要勾选下即可解决这种麻烦事。



- 创建完启动图标之后，我们可以在如上图标注 1 所示的 Dash 这个地方找到 IntelliJ IDEA 图标。但是图标我们一般是放在启动栏上的，所以这里你可以按着箭头的方向拖动 IntelliJ IDEA 图标到启动栏上即可。



- 启动的最后效果如上图，是不是有点过于简单了？！

卸载

Linux 的卸载是不需要执行程序的，只需要：删除对应目录。

- 删除主程序目录，也就是我们本文上面讲的解压出来的 `idea-IU-141.1532.4` 。
- 如果不想保留你的配置文件，还可以删除配置目录，目录所在位置：`./home/你用登录名/.IntelliJldea14`

Mac 系统下安装 IntelliJ IDEA

系统要求

- 系统支持：Mac OS X 10.5 以上
- JDK 版本：Apple Java 6 或 Oracle Java 7 以上
- 内存：最低要求 1G，推荐 2G 以上
- 硬盘：最低要求 2G
- 显示器：最低要求 1024 X 768 分辨率
- 更多信息可以阅读

读：https://www.jetbrains.com/idea/download/system_requirements.jsp?os=mac

重要说明

我这里以 Mac 系统为例进行讲解。但是，在学习下面内容之前请先看下章节：[Windows 下安装](#)

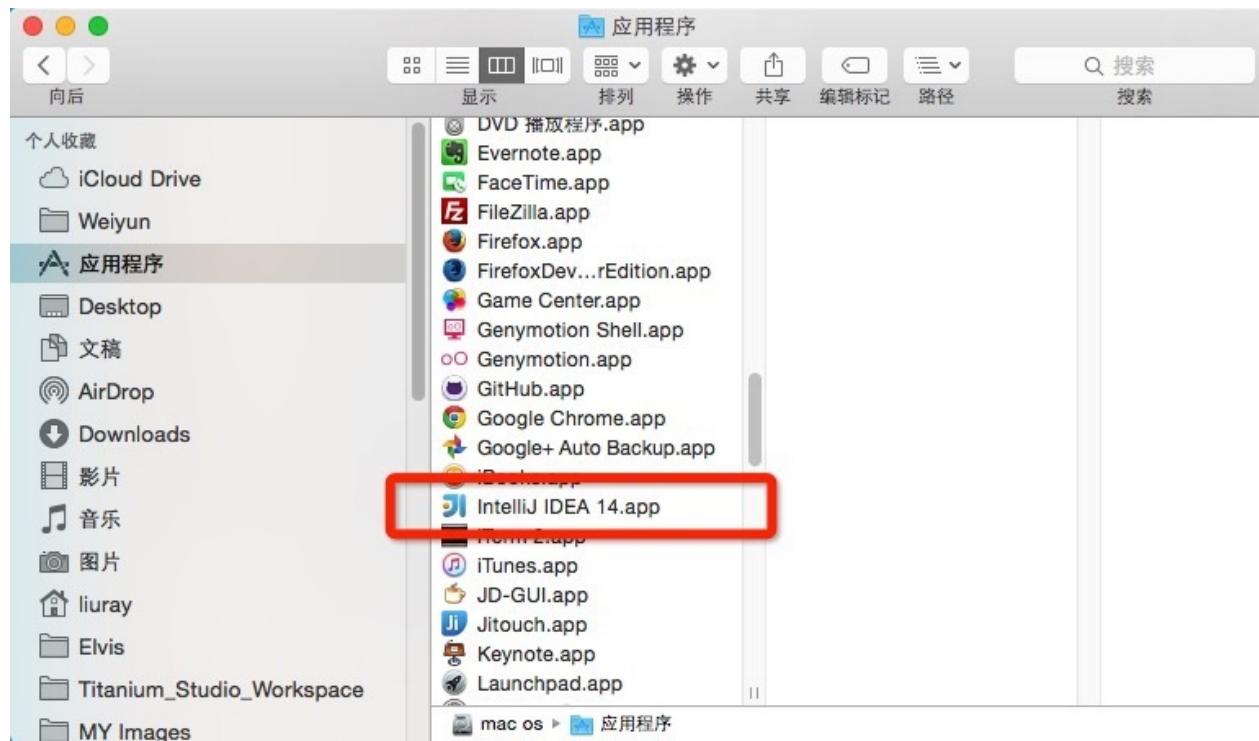
因为它们配置流程是基本一样的，只是系统不同，开始的步骤不太一样而已，因此相同部分我这里是不会再讲的，我只讲 IntelliJ IDEA 在 Mac 安装特殊的地方。

Mac 下安装过程



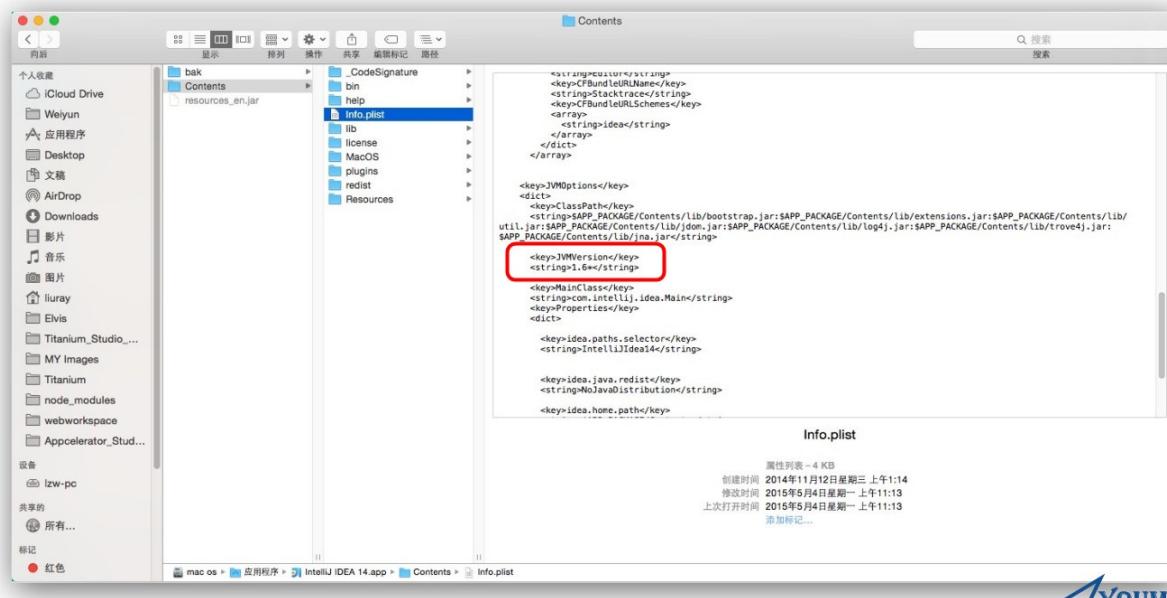
5. Mac 下安装

- 运行下载到 `ideaIU-14.1.4.dmg`，效果如上图所示。
- 根据提示把左侧的 IntelliJ IDEA 图标拖动到右侧目录图



- 拖动完成之后即可在 `应用程序` 中看到 IntelliJ IDEA 的启动图标，点击运行即可。

Mac 修改运行 JDK 版本



YOU MEEK

- 如果你的 Mac 安装有多个 JDK，你想使用高版本的 JDK 运行 IntelliJ IDEA 可以按如下方式进行修改：
- 在 应用程序 中找到 IntelliJ IDEA.app 然后对此进行 右键 > 显示包内容 > Contents > Info.plist ，效果如上图所示。
- 找到上图红圈标注的代码，修改 JVMVersion 的属性值，如果是 JDK 7，则改为 1.7* 。如果是 JDK 8，则改为 1.8* 。

安装总结

硬件建议

从上一讲的安装教程来看，IntelliJ IDEA 对硬件的要求看上去不是很高。可是实际在开发中其实并不是这样的，特别是开发 Java Web 的项目的计算机，2G 内存是基本不够用的。

我们现在来假设一种国内常见的开发环境：

有一个在开发的 Java Web 项目，它使用的框架为主流的：Struts + Spring + Hibernate，使用这三个框架的过程中，我们要引入大量的框架 jar 包，在我们的 Web 容器启动时，这些框架架包就要占用大量的内存，而且 IntelliJ IDEA 本身功能繁多，占用的内存也不算低，再加上我们这里还没计算计算机上的其他软件应用。所以基本上 2G 内存的计算机只适合写小程序、小项目或是开发静态页面。

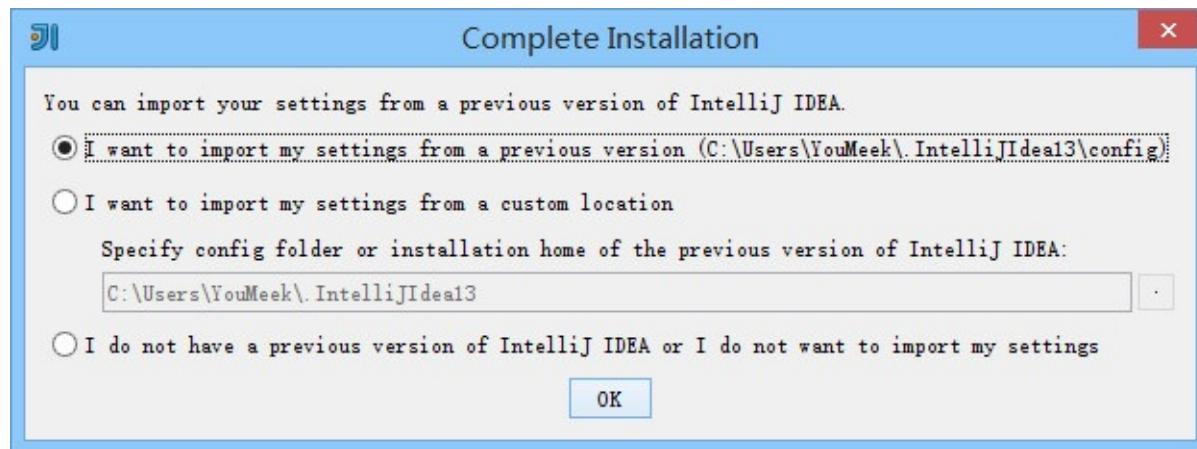
我个人建议，如果你是开发 Java Web 项目最好的方案是 8G 内存或是以上，硬盘能再用上固态是最好的，因为 IntelliJ IDEA 有大量的缓存、索引文件，把 IntelliJ IDEA 的缓存、索引文件放在固态上，IntelliJ IDEA 流畅度也会加快很多。

如果你正在使用 **Eclipse / MyEclipse**，想通过 **IntelliJ IDEA** 来解决计算机的卡、慢等问题，我这里可以直接明白地告诉你：这基本上是不可能的，本质上你应该对自己的硬件设备进行升级。

首次运行

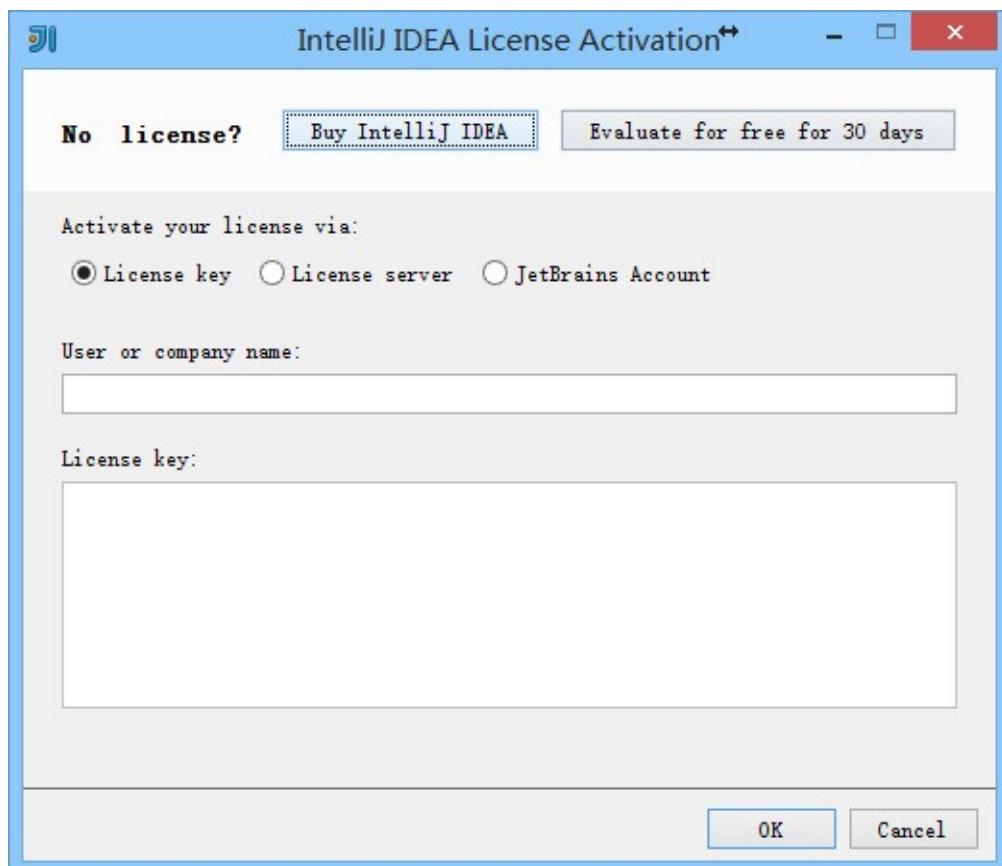
向导功能

- 假如你计算机上在过去已经有安装过 IntelliJ IDEA 14 的版本，且你在卸载 IntelliJ IDEA 的过程中，IntelliJ IDEA 的配置文件目录都没有删除，那安装新版本之后是不会启动向导的。
- 假如你计算机上没有安装过 IntelliJ IDEA，或是 卸载 IntelliJ IDEA 过程中你删除了 IntelliJ IDEA 的配置文件目录，则当你双击运行桌面上的 IntelliJ IDEA 快捷图标，将进入下面介绍的向导过程。

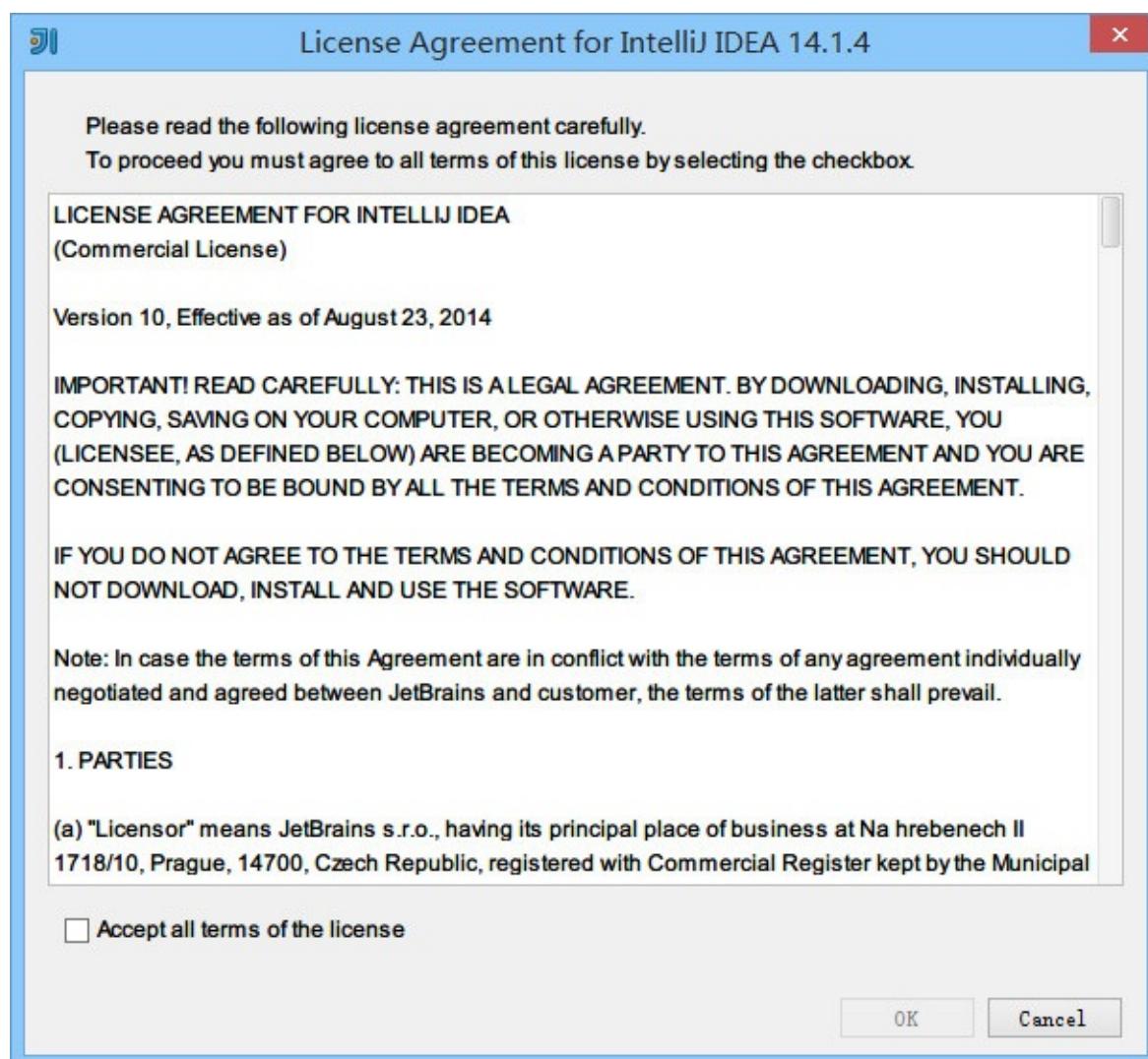


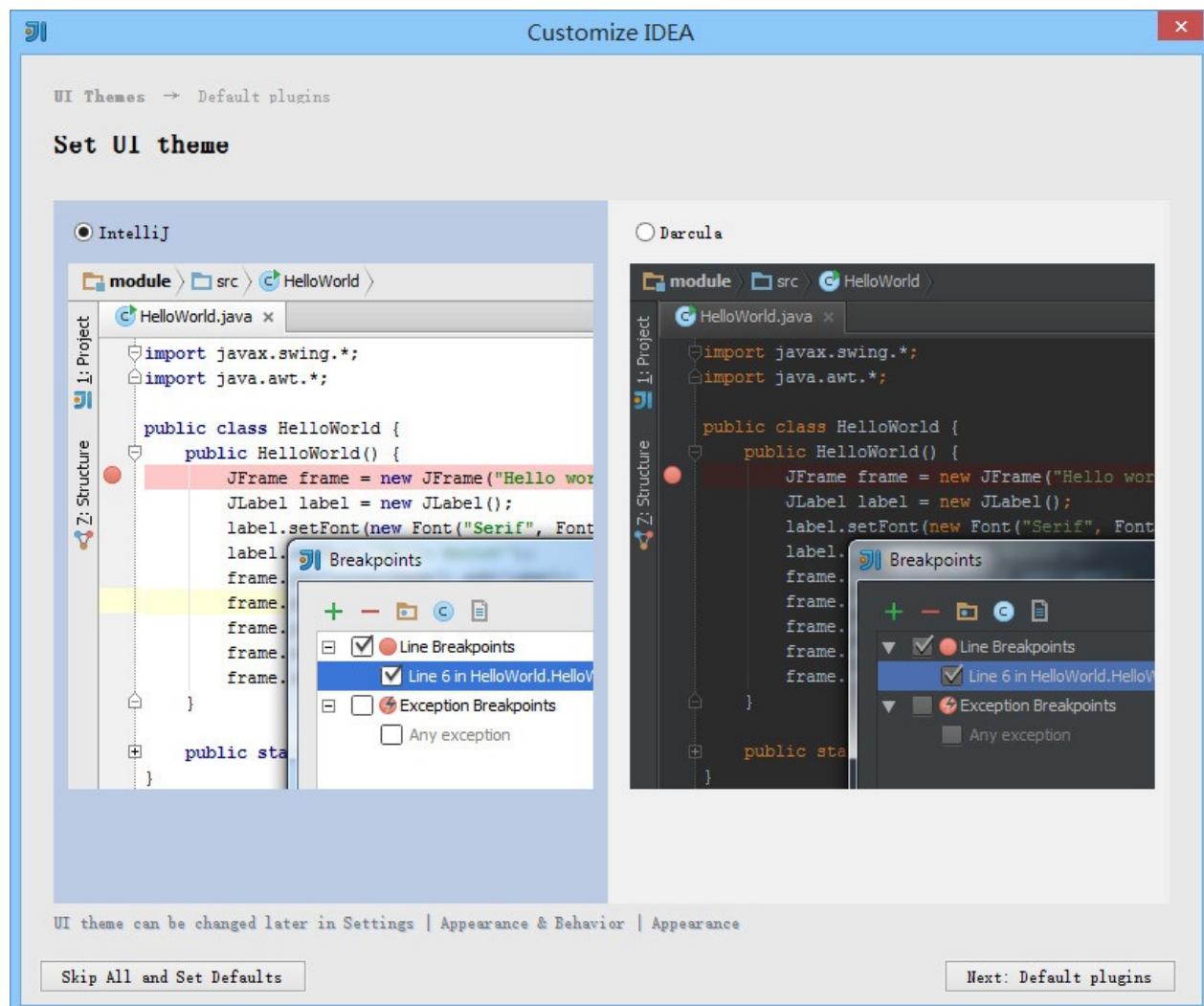
- 上图第一个单选按钮表示 IntelliJ IDEA 识别到我计算机上有 IntelliJ IDEA 13 版本的旧配置，如果我选择了该选项，则 IntelliJ IDEA 将自动把旧版本的配置文件转移到新版本的配置文件目录上。如果你计算机上首次安装一般是没有该选项的。
- 上图第二个单选按钮表示你可以指定 IntelliJ IDEA 导入你计算机上存在其他目录的 IntelliJ IDEA 配置文件目录，如果你有的话。
- 上图第三个单选按钮表示你没有任何早期版本的 IntelliJ IDEA 配置，你不导入任何配置，让 IntelliJ IDEA 生成一份新的配置。

7.首次运行（新用户必看）

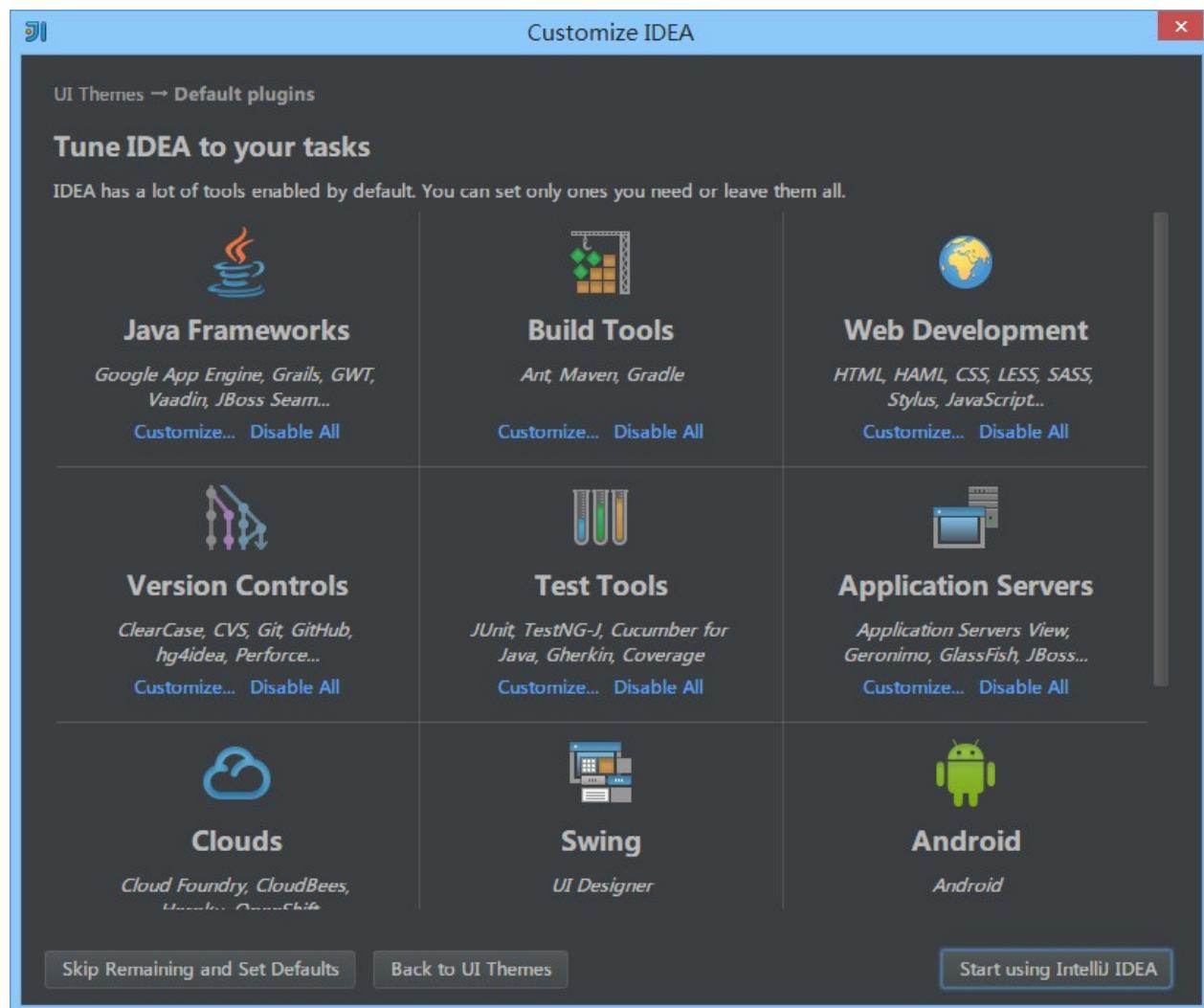


- 上图默认选择中的是 `Buy IntelliJ IDEA`，验证 IntelliJ IDEA 的许可有如图三种方式，我们这里使用的是 30 天试用版本进行演示，顾单击 `Evaluate for free for 30 days` 进行下一步。

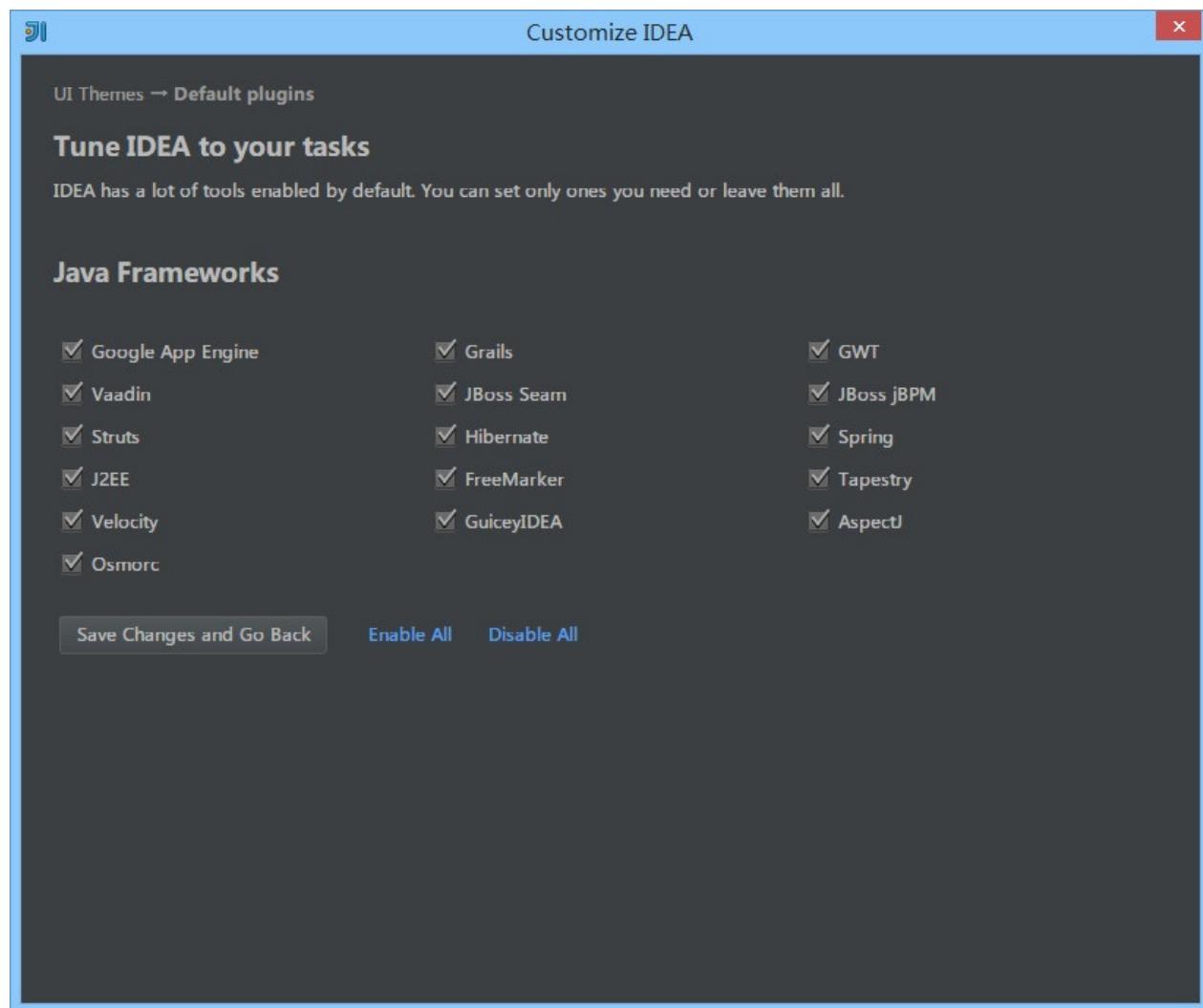




- 上图选择的时候 IntelliJ IDEA 主题 UI，在 Windows 系统版本中 IntelliJ IDEA 自带了 4 个主题，但是用的最多的就是上图这 2 种，其中大家基本偏爱黑色的 Darcula。这个没有好坏之分，根据你的喜好来进行选择，我们演示的版本就是用 Darcula。



- 上图显示了 IntelliJ IDEA 支持的主要的一些扩展功能或者说是工具、插件也可以。你可以根据自己开发的需求进行禁用一些扩展，这样可以稍微减轻 IntelliJ IDEA 运行时所占内存，加快运行速度，但是效果并不会很明显就是。
- 我们这里点击 Java Frameworks 的 Customize 进行下一步操作。



- 上图显示了 IntelliJ IDEA 所以支持的 Java Frameworks。我们可以根据自己的开发需求不启用指定框架的。去掉框架前面的勾选框就表示不启用该框架功能支持。
- 对于不启用的框架，我们也可以在后期进行重新勾选，这会在 IntelliJ IDEA 插件那一讲进行专门讲解。

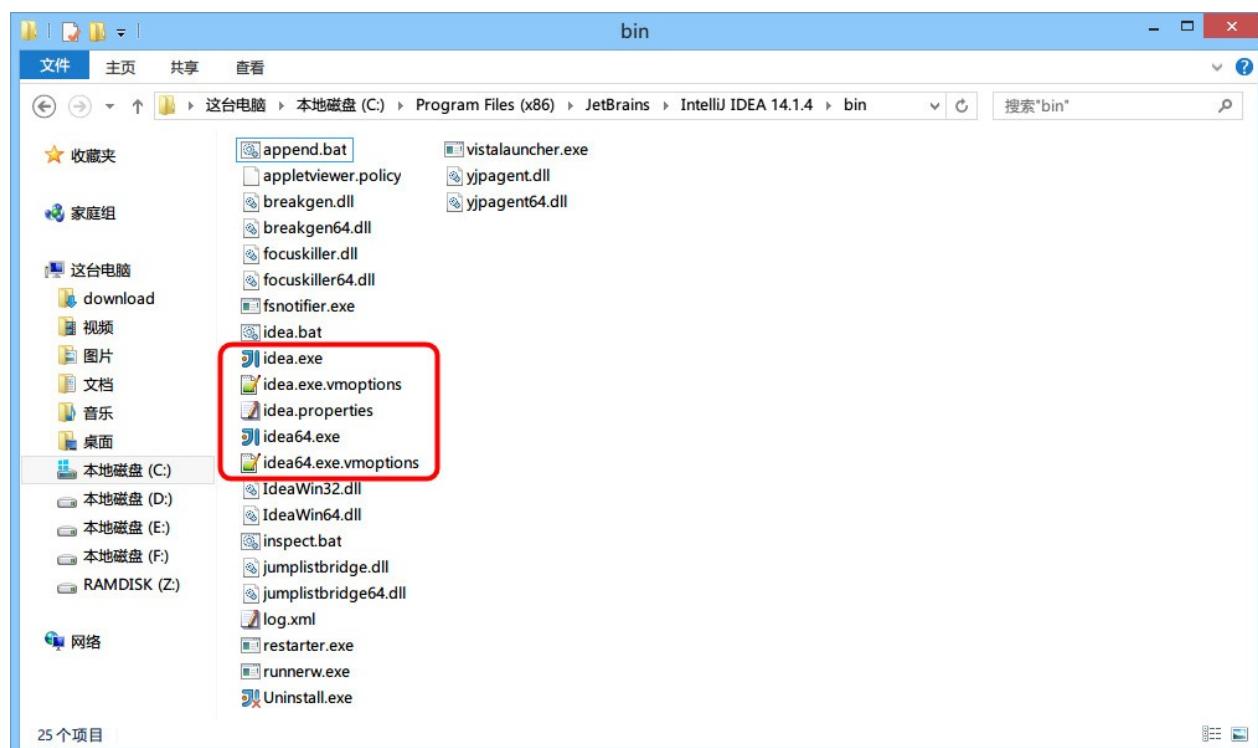


7.首次运行（新用户必看）

- 选择好自己所需的扩展功能后，按 `Start using IntelliJ IDEA` 显示上图启动界面，金黄色进度条走完之后，欢迎真正进入 IntelliJ IDEA 的编码世界！

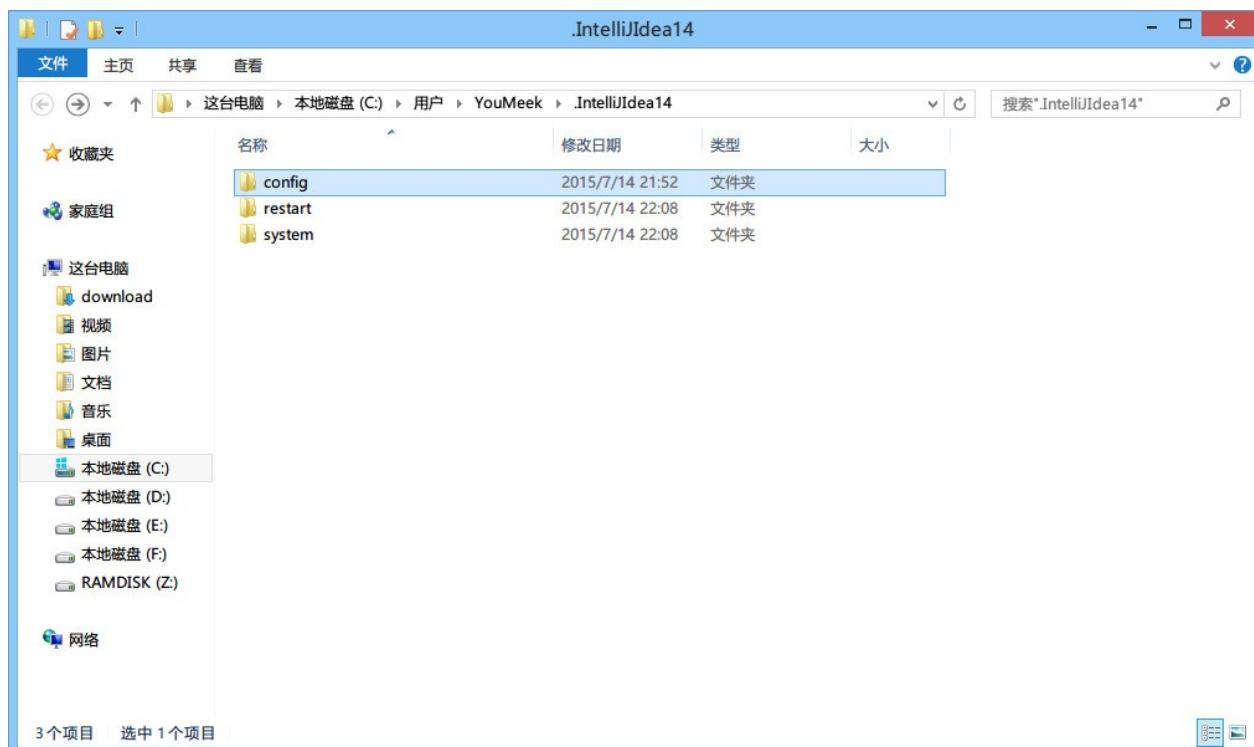
IntelliJ IDEA 相关核心文件和目录介绍

安装目录介绍



- IntelliJ IDEA 的安装目录并不复杂，上图为最常改动的 bin 目录，经常会改动的文件或是必须介绍的就是如图红色框中的几个。
- `idea.exe` 文件是 IntelliJ IDEA 32 位的可执行文件，IntelliJ IDEA 安装完默认发送到桌面的也就是这个执行文件的快捷方式。
- `idea.exe.vmoptions` 文件是 IntelliJ IDEA 32 位的可执行文件的 VM 配置文件，具体配置修改会下面进行专门讲解。
- `idea64.exe` 文件是 IntelliJ IDEA 64 位的可执行文件，要求必须电脑上装有 JDK 64 位版本。64 位的系统也是建议使用该文件。
- `idea64.exe.vmoptions` 文件是 IntelliJ IDEA 64 位的可执行文件的 VM 配置文件，具体配置修改会下面进行专门讲解。
- `idea.properties` 文件是 IntelliJ IDEA 的一些属性配置文件，具体配置修改会下面进行专门讲解。

设置目录介绍



- 不管你使用的是哪个操作系统，IntelliJ IDEA 的设置目录命名是统一的、有规律：`.IntelliJ IDEA14`。其中 14 表示大版本号，如果你电脑上还同时装有 13 的版本，那则还应该会有一个：`.IntelliJ IDEA13` 的设置目录，其他版本道理一样。
- 在三大主流的操作系统上，你只要对整个硬盘进行搜索：`.IntelliJ IDEA14`，即可找到，无需可以去记忆到底生成在哪个目录下。
- 对于这个设置目录有一个特性，就是你删除掉整个目录之后，重新启动 IntelliJ IDEA 会再自动帮你再生成一个全新的默认配置，所以很多时候如果你把 IntelliJ IDEA 配置改坏了，没关系，删掉该目录，一切都会还原到默认，我是很建议新人可以多自己摸索 IntelliJ IDEA 的配置，多几次还原，有助于加深对 IntelliJ IDEA 的了解。
- `config` 目录是 IntelliJ IDEA 个性化化配置目录，或者说是整个 IDE 设置目录。也是我个人认为最重要的目录，没有之一，如果你还记得安装篇的介绍的时候，安装新版本的 IntelliJ IDEA 会自动扫描硬盘上的旧配置目录，指的就是该目录。这个目录主要记录了：IDE 主要配置功能、自定义的代码模板、自定义的文件模板、自定义的快捷键、Project 的 tasks 记录等等个性化的设置。
- `system` 目录是 IntelliJ IDEA 系统文件目录，是 IntelliJ IDEA 与开发项目一个桥梁目录，里面主要有：缓存、索引、容器文件输出等等，虽然不是最重要目录，但是也是最不可或缺目录之一。

配置文件常见修改内容说明

```

1 -Xms128m
2 -Xmx750m
3 -XX:MaxPermSize=350m
4 -XX:ReservedCodeCacheSize=225m
5 -XX:+UseConcMarkSweepGC
6 -XX:SoftRefLRUPolicyMSPerMB=50
7 -ea
8 -Dsun.io.useCanonCaches=false
9 -Djava.net.preferIPv4Stack=true
10

```

N length : 200 lines : 10 Ln : 1 Col : 1 Sel : 0 | 0 DosWindows UTF-8 w/o BOM INS

- 上图是 64 位可执行文件的 JVM 配置文件内容，如果你是 32 位的系统你应该修改的是 `idea.exe.vmoptions` 文件里面的内容，但是由于 32 位系统内存一般都是 2G 左右的，所以也没有多大空间可以调整，所以一般无需调整的。
- 修改的原则主要是根据自己机器的内存情况来判断的，我个人是建议 8G 以下的机子或是静态页面开发者都是无需修改的。如果你是开发大型项目、Java 项目或是 Android 项目，并且内存大于 8G，建议进行修改，常修改的就是下面 4 个参数，我这里主要以我的机子会例进行建议，每个人机子情况不一，这里也只是做一个引子，最好的调整方式是你可以根据 `jconsole` 这类工具进行观察后个性化调整。

- `-Xms128m`，16 G 内存的机器可尝试设置为 `-Xms512m`
- `-Xmx750m`，16 G 内存的机器可尝试设置为 `-Xmx1500m`
- `-XX:MaxPermSize=350m`，16G 内存的机器可尝试设置为 `-XX:MaxPermSize=500m`
- `-XX:ReservedCodeCacheSize=225m`，16G 内存的机器可尝试设置为 `-XX:ReservedCodeCacheSize=500m`

```

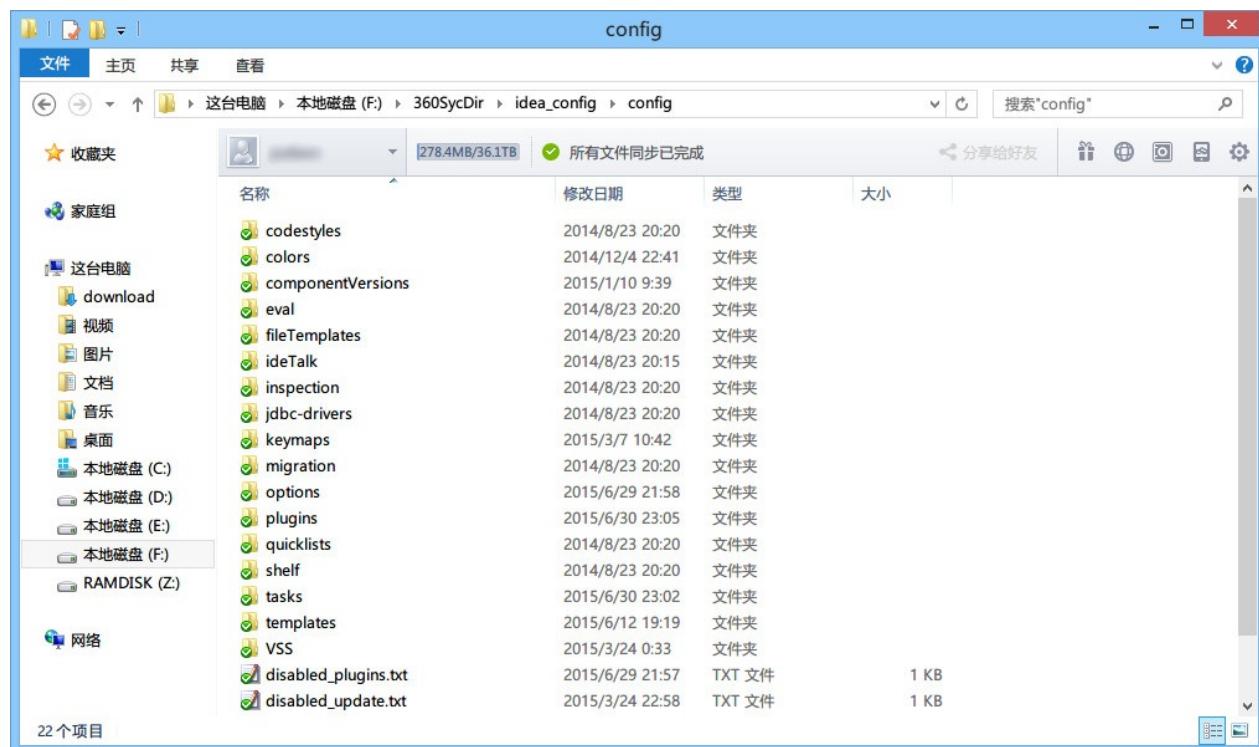
C:\Program Files (x86)\JetBrains\IntelliJ IDEA 14.1.4\bin\idea.properties - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 格式(M) 语言(L) 设置(T) 宏(O) 运行(R) 插件(P) 窗口(W) ?
idea.properties
1 # Use ${idea.home.path} macro to specify location relative to IDE installation home.
2 # Use ${xxx} where XXX is any Java property (including defined in previous lines of this file) to refer to its value.
3 # Note for Windows users: please make sure you're using forward slashes (e.g. c:/idea/system).
4
5 # -----
6 # Uncomment this option if you want to customize path to IDE config folder. Make sure you're using forward slashes.
7 # -----
8 # idea.config.path=${user.home}/.IntelliJIdea/config
9
10 # -----
11 # Uncomment this option if you want to customize path to IDE system folder. Make sure you're using forward slashes.
12 # -----
13 # idea.system.path=${user.home}/.IntelliJIdea/system
14
15 # -----
16 # Uncomment this option if you want to customize path to user installed plugins folder. Make sure you're using forward slashes.
17 # -----
18 # idea.plugins.path=${idea.config.path}/plugins
19
20 # -----
21 # Uncomment this option if you want to customize path to IDE logs folder. Make sure you're using forward slashes.
22 # -----
23 # idea.log.path=${idea.system.path}/log
24
25 # -----
26 # Maximum file size (kilobytes) IDE should provide code assistance for.
27 # The larger file is the slower its editor works and higher overall system memory requirements are
28 # if code assistance is enabled. Remove this property or set to very large number if you need
29 # code assistance for any files available regardless their size.
30 # -----
31 idea.max.intellisense.filesize=2500
32
33 #
34 # This option controls console cyclic buffer: keeps the console output size not higher than the specified buffer size (Kb
35 # Older lines are deleted. In order to disable cycle buffer use idea.cycle.buffer.size=disabled

```

Properties file length: 7745 lines : 136 Ln: 103 Col: 1 Sel: 0 | 0 Dos\Windows UTF-8 w/o BOM INS

- 上图是 IntelliJ IDEA 一些属性配置，没有 32 位和 64 位之分，修改原则主要根据个人对 IntelliJ IDEA 的个性化配置情况来分析。常修改的就是下面 4 个参数：
 - `idea.config.path=${user.home}/.IntelliJIdea/config`，该属性主要用于指向 IntelliJ IDEA 的个性化配置目录，默认是被注释，打开注释之后才算启用该属性，这里需要特别注意的是斜杠方向，这里用的是正斜杠。
 - `idea.system.path=${user.home}/.IntelliJIdea/system`，该属性主要用于指向 IntelliJ IDEA 的系统文件目录，默认是被注释，打开注释之后才算启用该属性，这里需要特别注意的是斜杠方向，这里用的是正斜杠。如果你的项目很多，则该目录会很大，如果你的 C 盘空间不够的时候，还是建议把该目录转移到其他盘符下。
 - `idea.max.intellisense.filesize=2500`，该属性主要用于提高在编辑大文件时候的代码帮助。IntelliJ IDEA 在编辑大文件的时候还是很容易卡顿的。
 - `idea.cycle.buffer.size=1024`，该属性主要用于控制控制台输出缓存。有遇到一些项目开启很多输出，控制台很快就被刷满了没办法再自动输出后面内容，这种项目建议增大该值或是直接禁用掉，禁用语句
`idea.cycle.buffer.size=disabled`。

设置目录进行多台设置同步化处理



- 上图是我的个性化配置目录，我是存放在 F 盘，同时该目录也是在 360 同步盘中。这样做主要是为了让我的多台设置可以同时使用一个个性化配置，保证个人开发习惯，额外作用就是在服务器上一个备份作用。
- 设置方式很简单，修改 `idea.properties` 属性文件中的 `idea.config.path` 值，我的机器为：`idea.config.path=F:/360SycDir/idea_config/config`

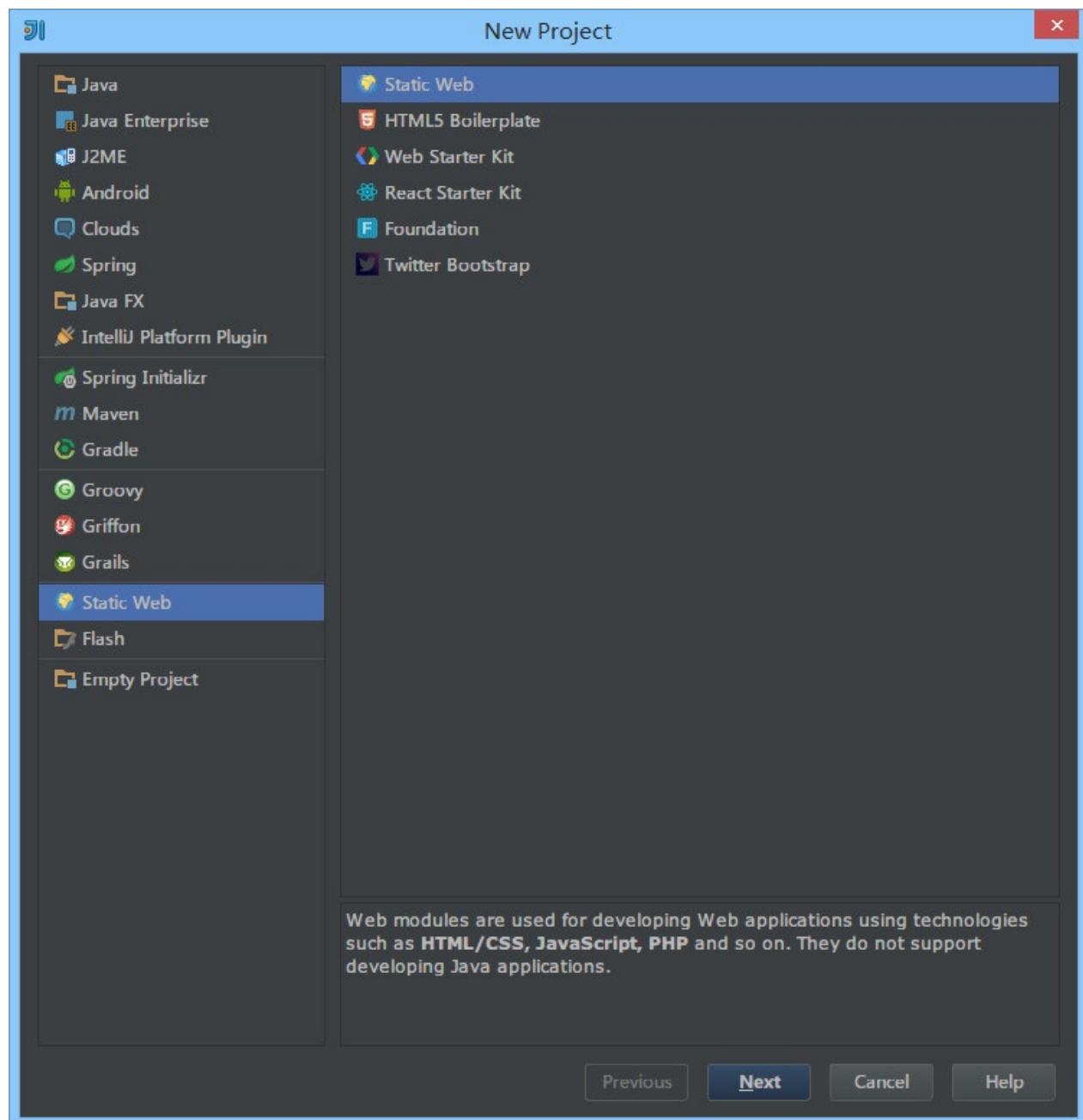
IntelliJ IDEA 界面介绍

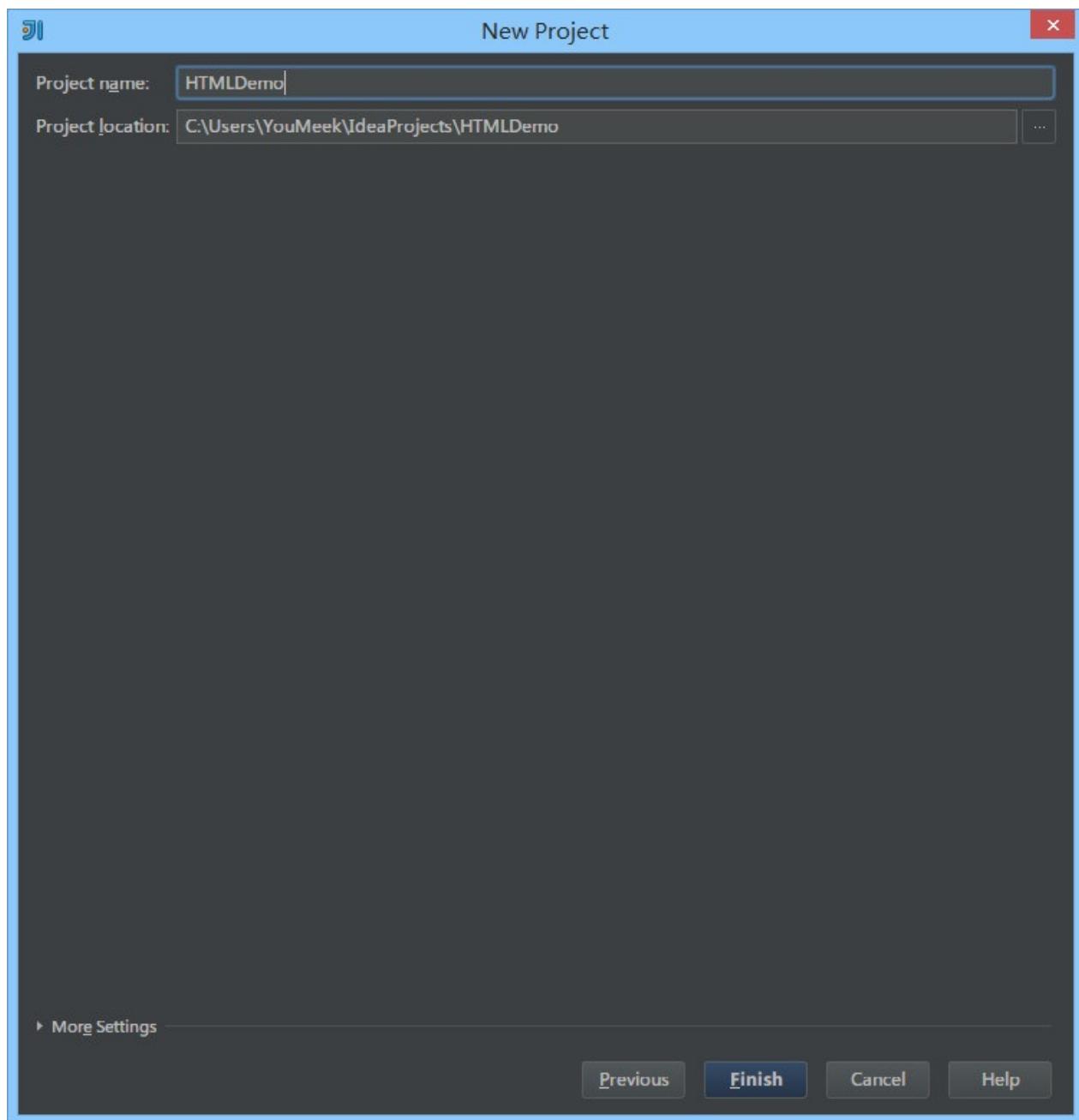
首次打开



- 重点说明：IntelliJ IDEA 是没有类似 Eclipse 的工作空间的概念（Workspaces），最大单元就是 Project。如果你同时观察多个项目的情况，IntelliJ IDEA 提供的解决方案是打开多个项目实例，你可以理解为开多个项目窗口。
- 命令 Create New Project 创建一个新项目。
- 命令 Import Project 导入一个已有项目。
- 命令 Open 打开一个已有项目，可以直接打开 Eclipse 项目，但是由于两者 IDE 下的项目配置不一样，所以项目还是需要配置的。
- 命令 Check out from Version Control 可以通过服务器上的项目地址 Checkout Github 上面项目或是其他 Git 托管服务器上的项目。
- 为了介绍 IntelliJ IDEA 界面，我们这里创建一个新 HTML 项目。

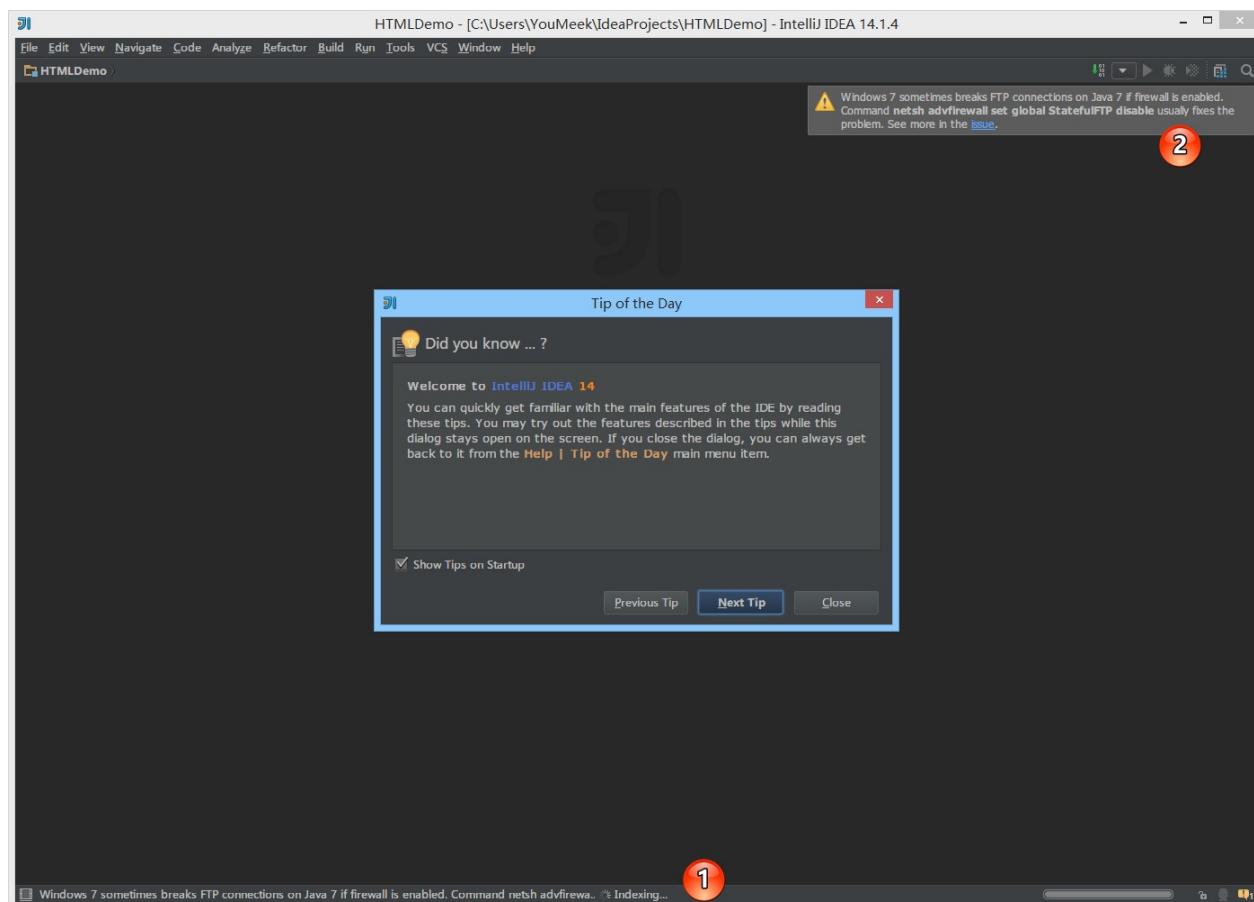
新建 HTML 项目





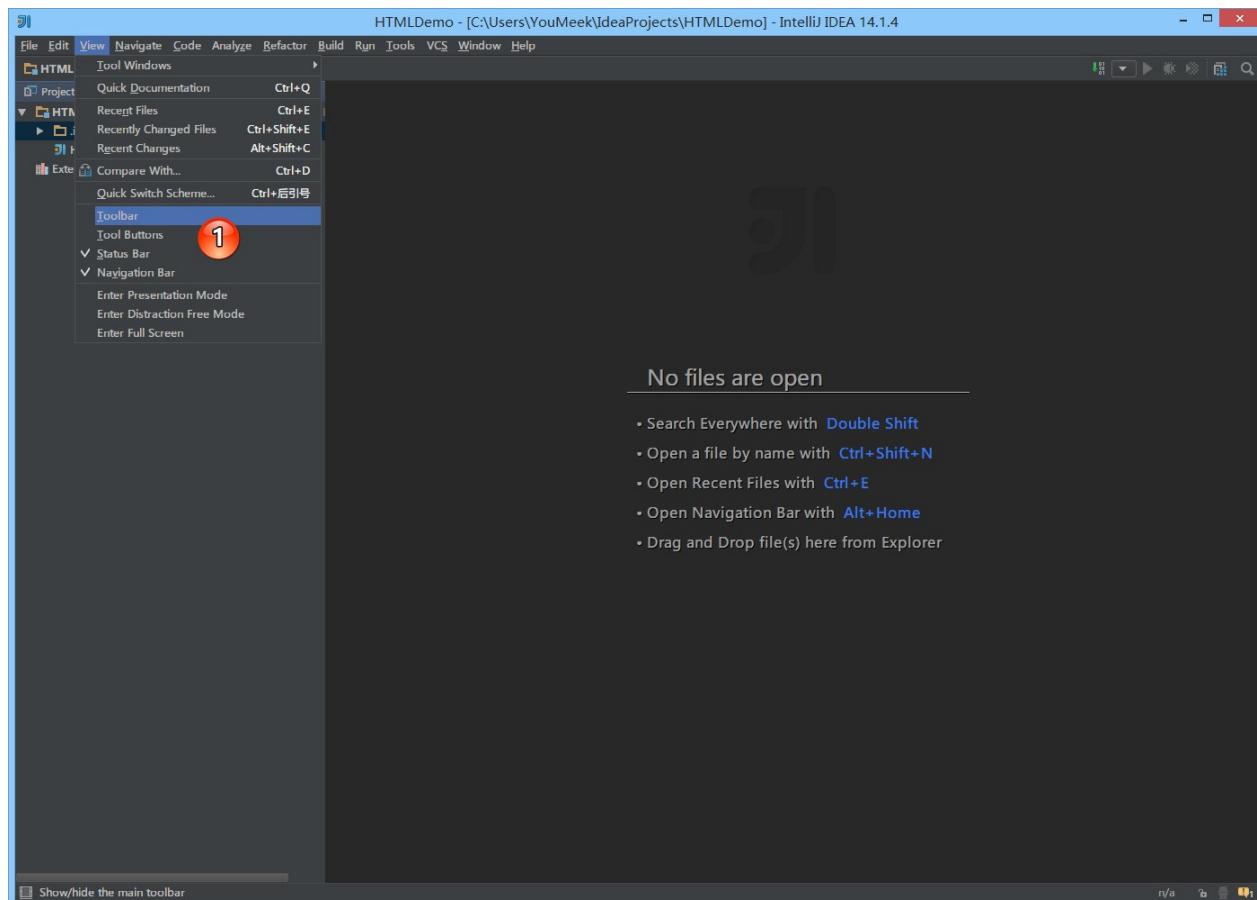
界面介绍

9.界面讲解（新用户必看）

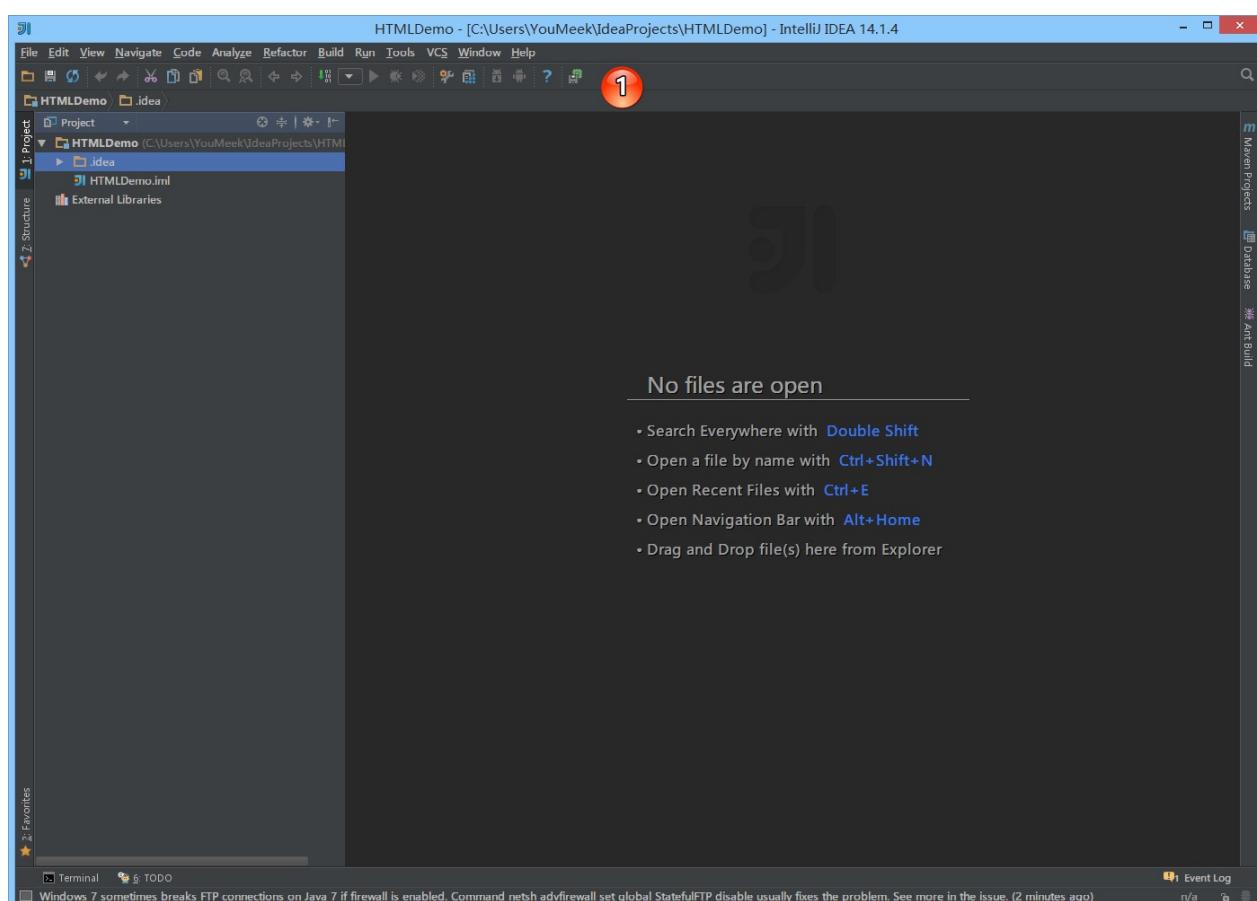


- 对于首次创建或打开的新项目，IntelliJ IDEA 都会创建项目索引，如上图标注 1 所示。大型项目在创建索引过程中可能必须会卡顿，所以强烈建议 创建索引过程最好不要动项目。
- 每次进入项目中，IntelliJ IDEA 会根据当前项目情况进行问题反馈，或是升级更新提示等。使用方式如上图标注 2 所示。

9. 界面讲解（新用户必看）



- 如上图，IntelliJ IDEA 默认界面是隐藏掉 `Toolbar` 和 `Tool Buttons`，我个人习惯看到这两个，所以一般都会进行开启。

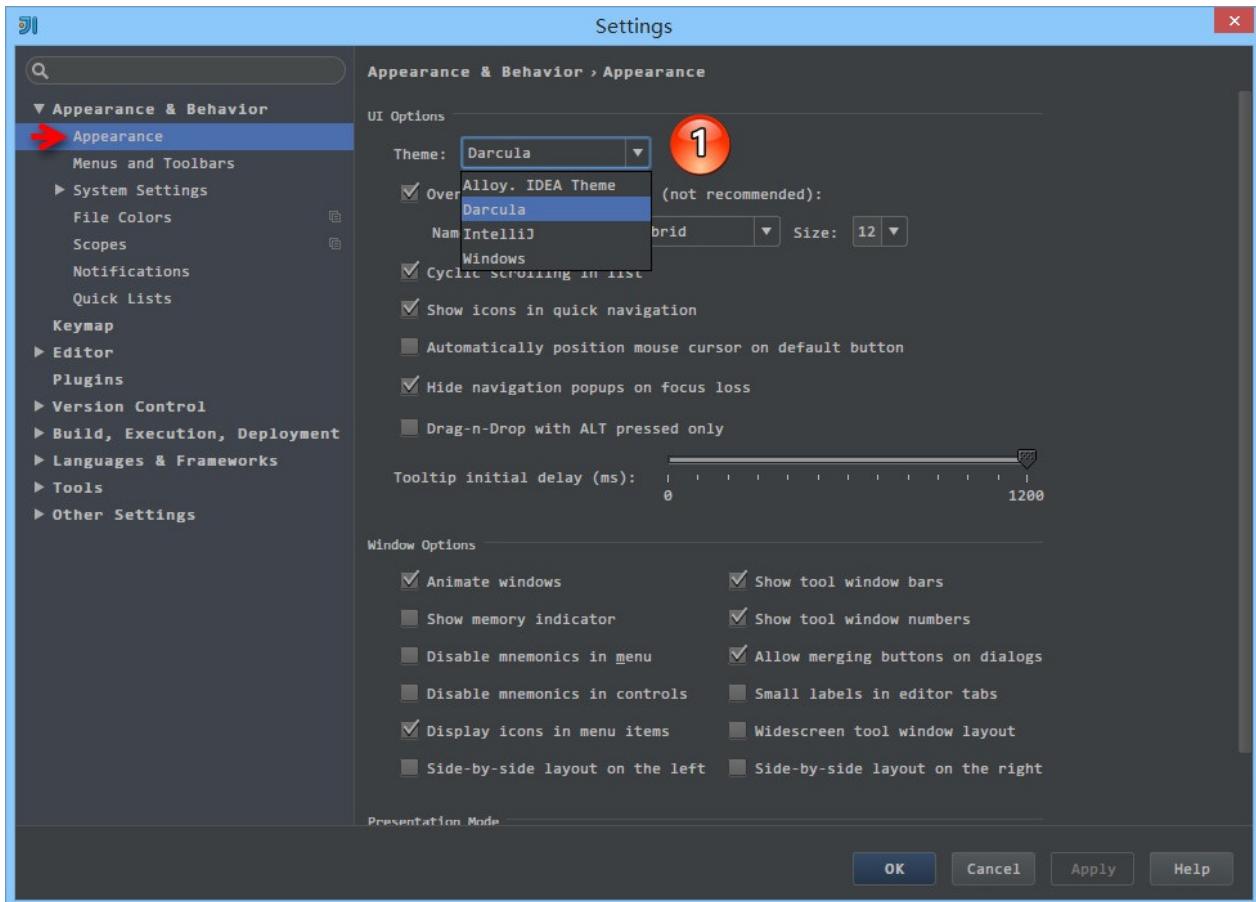


9. 界面讲解（新用户必看）

- 如上图标注 1 所示为开启 Toolbar 和 Tool Buttons 效果。

IntelliJ IDEA 主题、字体、编辑区主题、文件编码修改、乱码问题

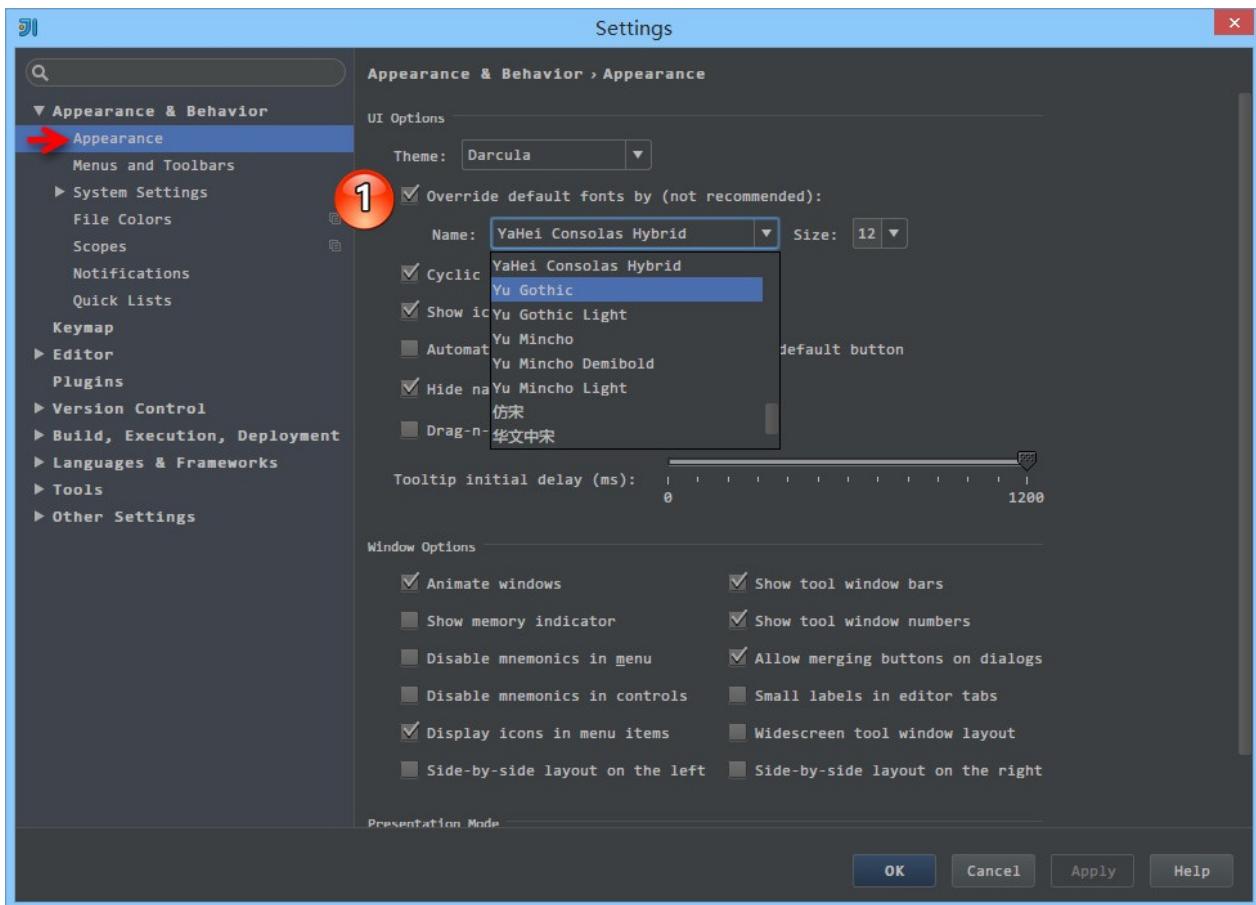
主题修改



- 上图标注 1 所示为 IntelliJ IDEA 修改主题的地方，在 Windows 系统上 IntelliJ IDEA 默认提供的主题有四套：Darcula、IntelliJ、Windows、Alloy. IDEA Theme。除了 Darcula 是黑色主题，其他三套都是以白色为背景的。
- 其他操作系统上不一定会也有四套主题的，主题的选择上大家根据自己喜好即可。改变主题需要重启 IntelliJ IDEA 方可看到效果。

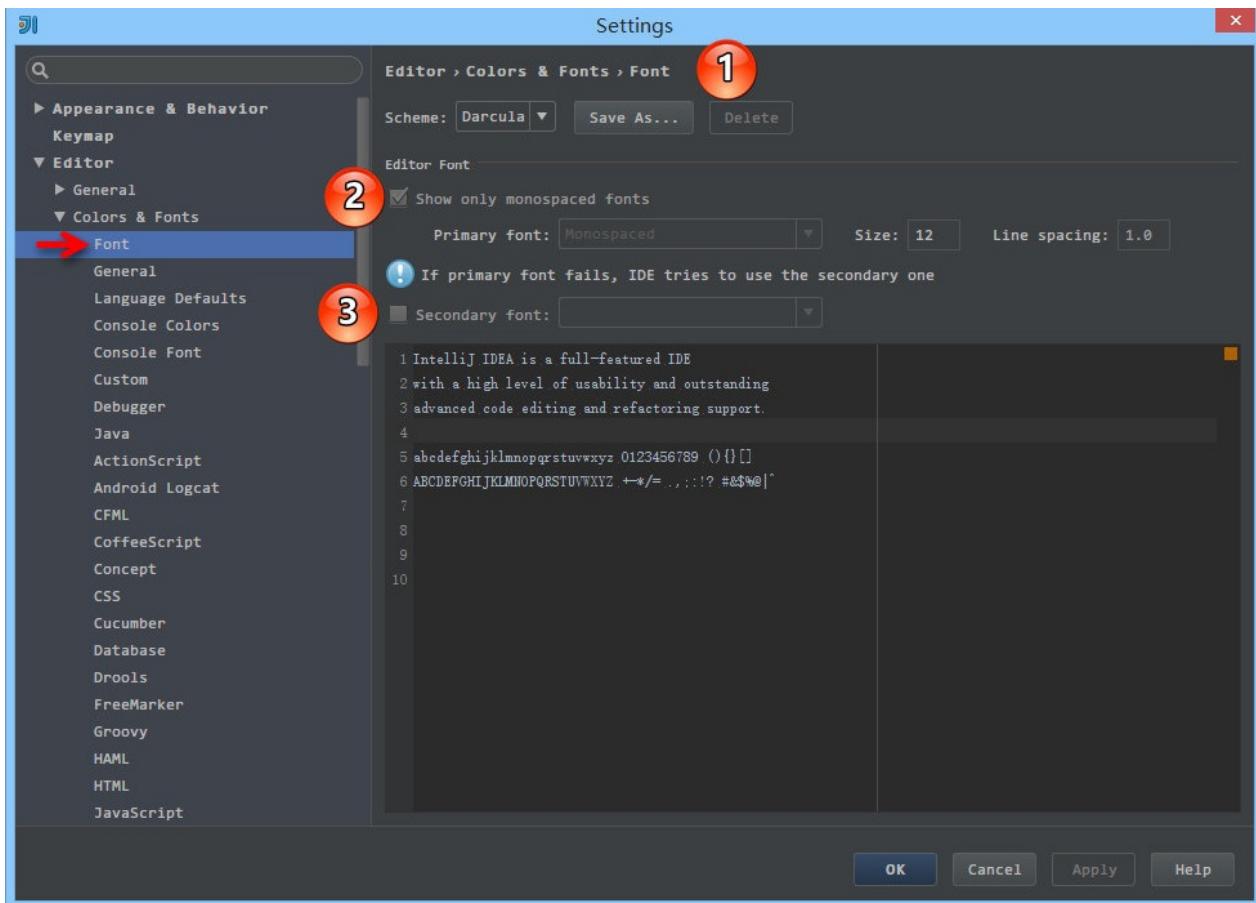
字体修改

主题字体修改



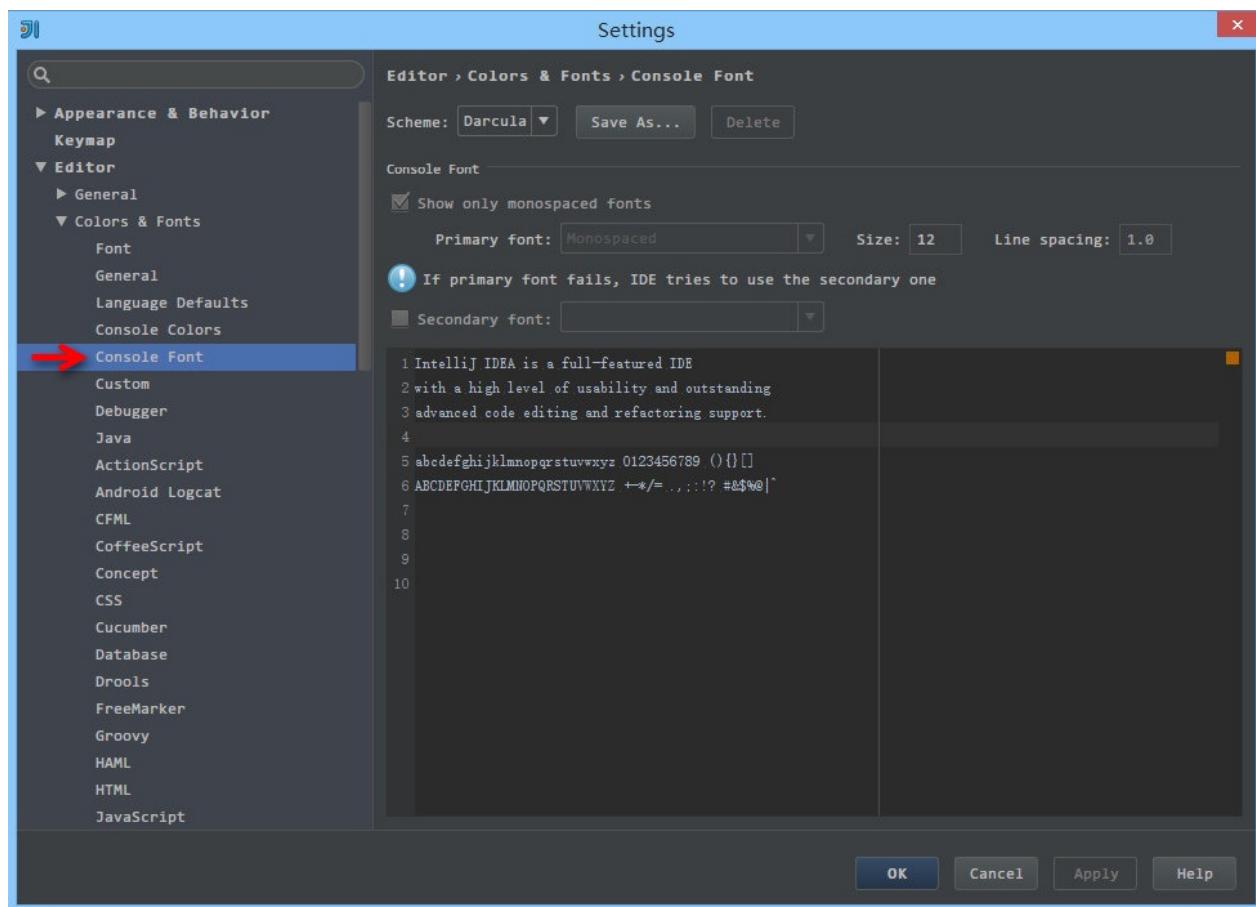
- 如上图标注 1 所示，IntelliJ IDEA 主题字体的修改要先勾选 `Override default fonts by`。默认 IntelliJ IDEA 是不推荐修改的，但是由于字体是有分包含中文和不包含中文之分的，一般使用英文的国家是不需要额外担心乱码问题的，而我们需要。
- 字体的审美上每个人不一样，但是如上一段说的，这里的字体修改是需要知道一个前提的，那就是你选择的那个字体必须含有中文，比如微软雅黑和宋体这类是包含中文的，而 `Courier New` 和 `Monaco` 这类只是单纯的英文字体。
- 如果你选择的字体不包含中文，那可能会在很多位置上出现类似 口口口口 这样的乱码问题，比如文件名含有中文、字体是中文名字的都会变成 口口口口。
- 在修改 IntelliJ IDEA 的主题字体的时候，不建议把字体调成很大，因为很多人遇到这样一种情况：显示器分辨率低，主题字体又大，在 IntelliJ IDEA 的某些操作的工具菜单、右键菜单选项中部分选项超出了分辨率显示范围，没办法被选中。当然了，如果你一定要把字体改大，又不用大分辨率显示器，那可以通过 IntelliJ IDEA 的 `Menus and Toolbars` 删除部分你认为用不到的菜单，但是一般不建议这样做。
- 还需要特别注意的时候，如果你是开着 IntelliJ IDEA 的时候，新装了一个字体的话，那必须重启 IntelliJ IDEA 之后才能在下拉列表找到新装的字体。

代码编辑字体修改



- 如上图标注 1 所示，默认 IntelliJ IDEA 是不能直接在默认的代码模板上修改字体的，需要先 `Save As` 一份出来，然后才可以修改。这种设计在 IntelliJ IDEA 其他很多设置也是如此的，所以如果你还看到类似有 `Copy`、`Save As` 这类选项的按钮就要想到是此设计思想。
- 如上图标注 2 所示，勾选的 `Show only monospaced fonts` 表示筛选显示系统上的等宽字体。由于 Windows 系统上等宽字体并不多，勾选此选项出现的下拉字体可选择就很少。取消勾选之后，就可以显示系统上所有已安装的字体。
- 如上图标注 3 所示，其中编码字体有第一字体（`Primary font`）和第二字体（`Secondary font`）之分。当有些字符在第一字体支持不了的时候，会去使用第二字体进行支持。
- 我个人习惯上：英文字体使用 `Monaco`，由于此字体不支持中文，所以我把这个设置为第一字体，第二字体使用 `Yahei Consolas Hybrid` 进行支持，该字体含有中文。这两个字体都不是系统自带的，需要自行下载安装。
- 如果你的第一字体不包含中文的话，第二字体包含中文，那在有些地方也还是会出 现 `□□□□□` 这类问题，比如 `Ctrl + Shift + N` 进行查找文件的时候，如果你输入中文也会变成 `□□□□□`，我个人文件名为中文的不多，所以就容忍了这种情况。如果你不愿意容忍这种情况，那还是回到最开始的要求：第一字体包含中文。

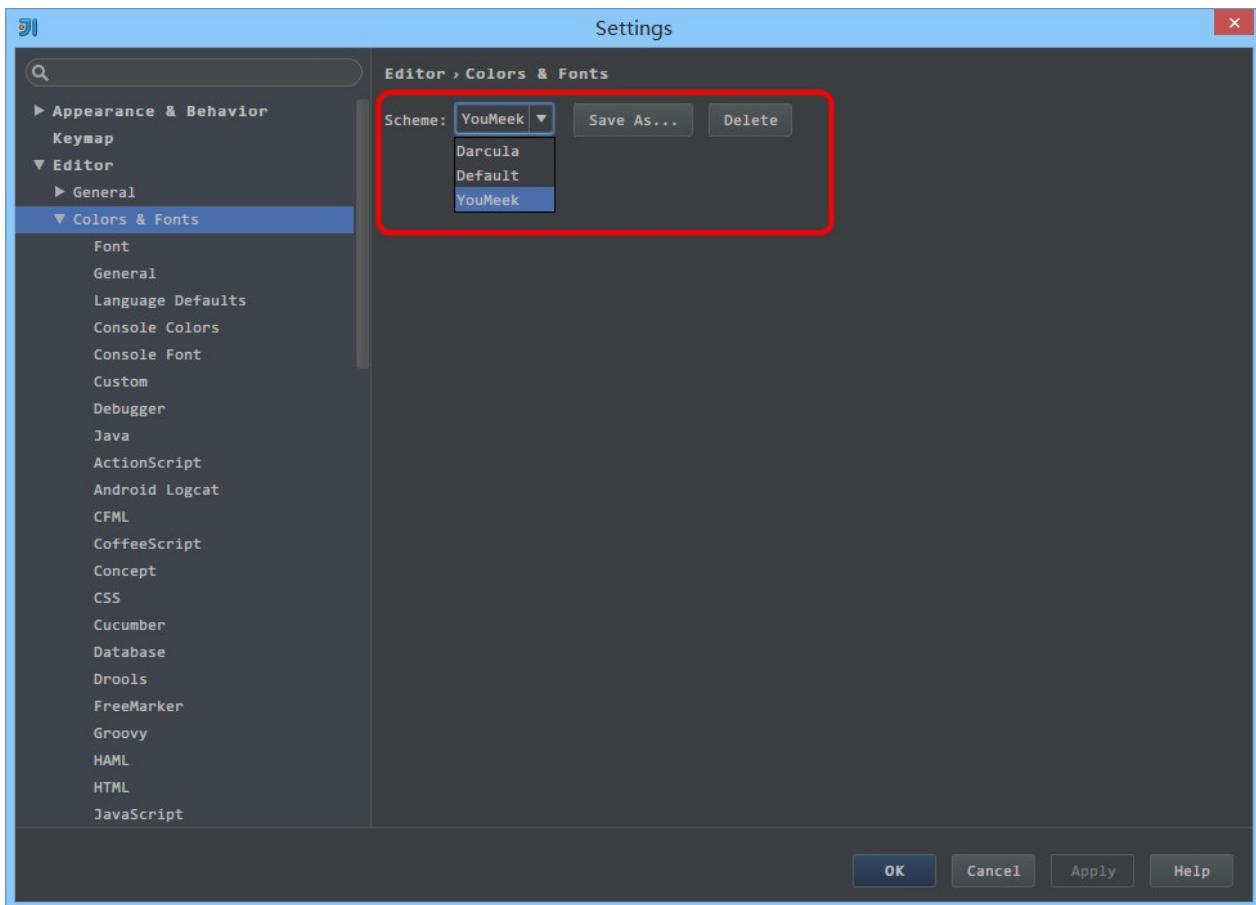
控制台输出字体修改



- 如上图为控制台输出内容字体修改，有很多 IntelliJ IDEA 新人在做输出的时候出现乱码原因就是因为没有在这里进行设置。
- 控制台输出字体修改的原理跟代码编辑字体修改是一样的，所以这里不进行讲解。

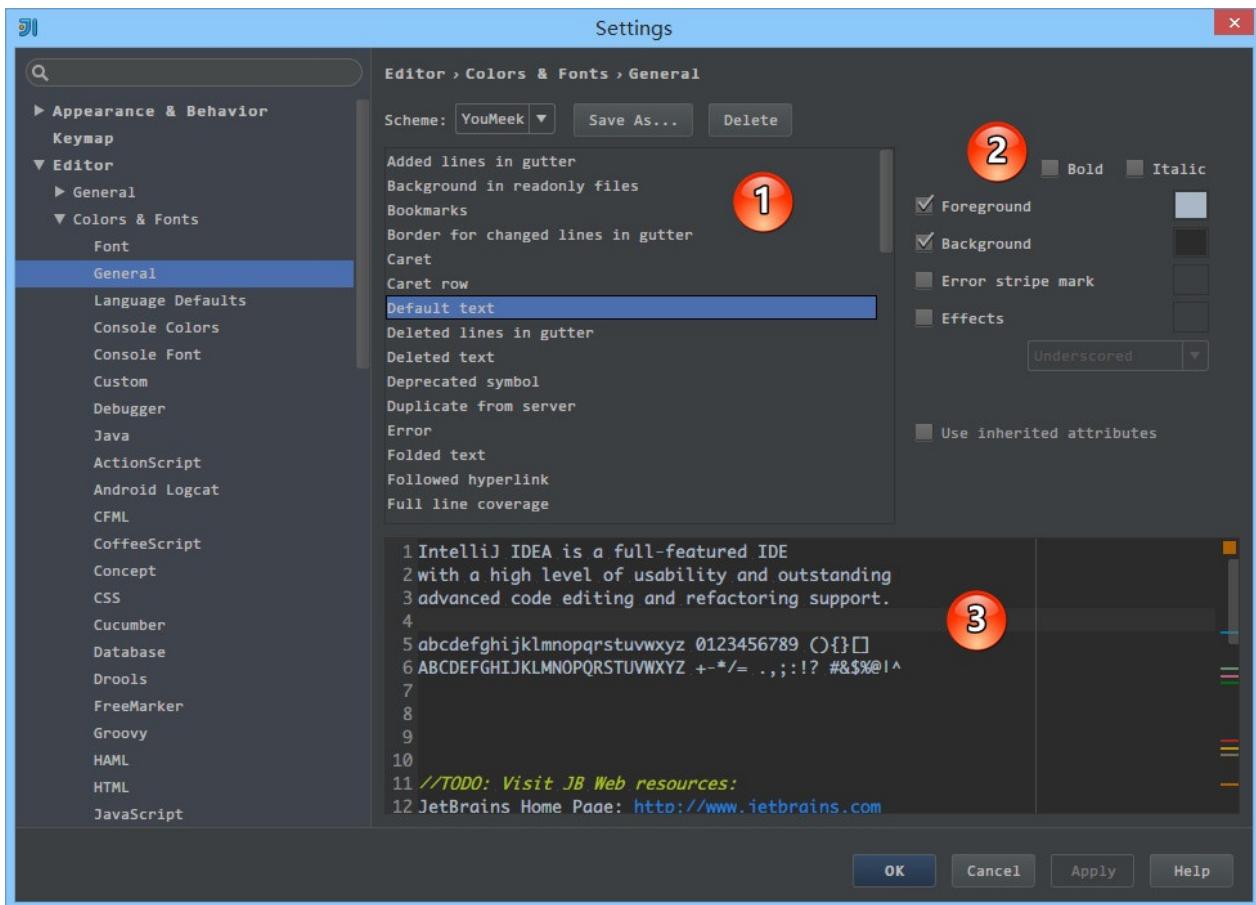
编辑区主题修改

编辑区主题介绍



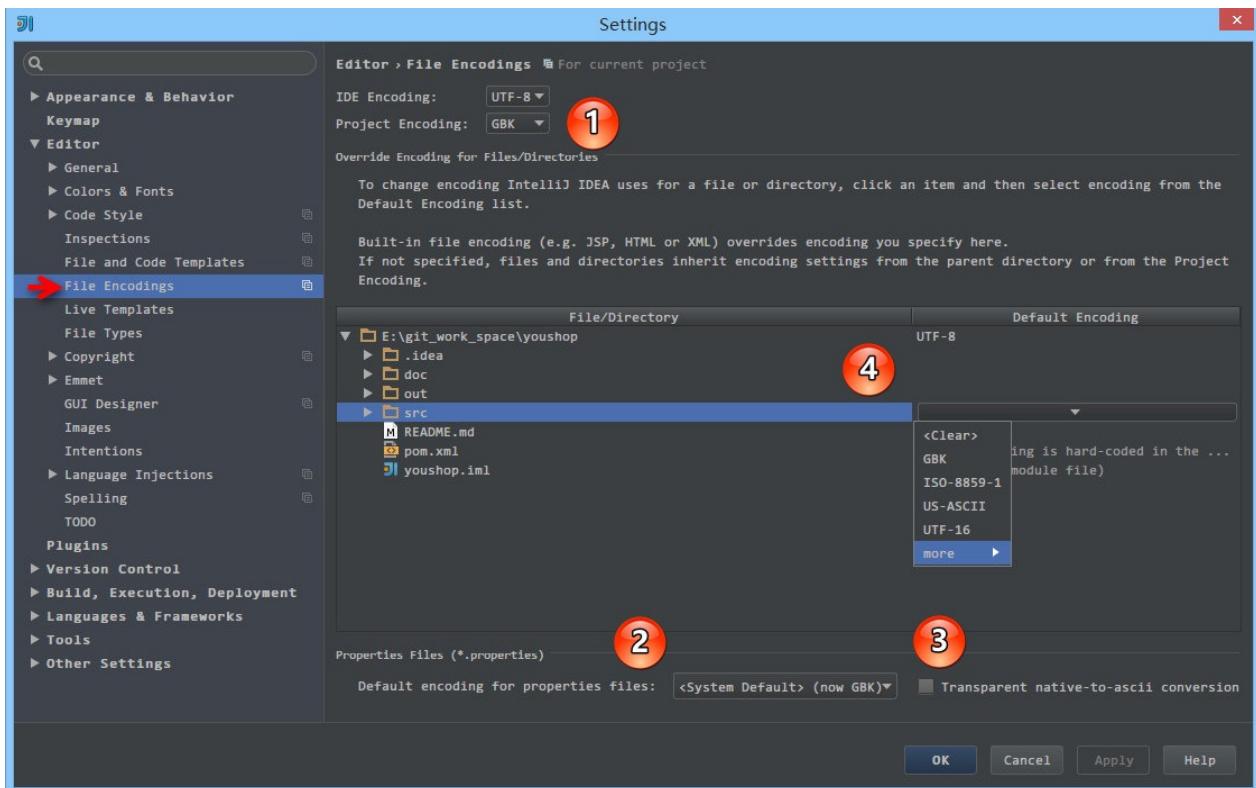
- 编辑区主题，也就是代码书写区的主题修改。基本上大家在 General 上都有对此进行小修小改，我下面也主要介绍下我个人在 General 上常修改的一些地方，其他特性的颜色修改我一般默认，但是修改方法原理一样。
- 如上图红圈下拉所示，展示的是我当前电脑可以选择的编辑区主题。
- 对于编辑区的主题，也有人制作成模板在网络上提供下载。这里主要介绍两个站点：
 - <http://www.ideacolorthemes.org/themes/>，主要提供 jar 文件下载。
 - <http://www.phpstorm-themes.com/>，主要提供 xml 和 icl 文件下载。
 - 对应文件如何安装请查看网站对应的 Help 页面，都有详细说明的。

编辑区主题细节修改



- 上图标注 1 为可修改的通用细节内容
- 上图标注 2 为可修改属性，其中并不是每个细节都可以修改所有属性的。比如细节：`Default text` 是可以勾选 `Bold`，而 `Caret row` 则是无法勾选 `Bold`，因为只有文本才有加粗的属性需求。
- 上图标注 3 为修改后的预览区，预览区是实时动态展示的。
- 在 `General` 区，我常修改的有：
 - `Default text`，指的是默认代码文本，我一般会修改其 `Background` 属性。
 - `Caret row`，指的是光标所在行，我一般会修改其 `Background` 属性。
 - `Vertical indent guide`，指的是垂直缩进线，我一般会修改其 `Foreground` 属性。
 - `Identifier under caret`，指的是光标所在位置的相同标识符呈现什么效果，我一般会修改其 `Background` 属性。
 - `Text search result`，指的是在查找模式下，匹配字符的样式，我一般会修改其 `Background` 属性。

文件编码修改



- 上图标注 1 所示，IDE 的编码默认是 `UTF-8`，Project Encoding 虽然默认是 `GBK`，但是一般我都建议修改为 `UTF-8`。
- 上图标注 2 所示，IntelliJ IDEA 可以对 Properties 文件进行专门的编码设置，一般也建议改为 `UTF-8`，其中有一个重点就是属性 `Transparent native-to-ascii conversion`，
- 上图标注 3 所示，对于 Properties 文件，重要属性 `Transparent native-to-ascii conversion` 主要用于转换 `ascii`，一般都要勾选，不然 Properties 文件中的注释显示的都不会是中文。
- 上图标注 4 所示，IntelliJ IDEA 除了支持对整个 Project 设置编码之外，还支持对目录、文件进行编码设置。如果你要对目录进行编码设置的话，可能会出现需要 Convert 编码的弹出操作选择，强烈建议在转换之前做好文件备份，不然可能出现转换过程变成乱码，无法还原。

```
youshop - [E:\git_work_space\youshop] - [youshop] - ...\\src\\test\\java\\com\\youmeek\\youshop\\testmybatis\\TestMyBatis.java - I... - □ ×
```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

youshop src test java com youmeek youshop testmybatis TestMyBatis

TestMyBatis.java

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath:spring-and-mybatis.xml"})
```

```
public class TestMyBatis {
    private static Logger logger = Logger.getLogger(TestMyBatis.class);
    @Resource
    private SysUserServiceI userServiceI = null;

    @Resource
    private SysAreaServiceI sysAreaServiceI = null;

    @Test
    public void test1() {
        SysUser sysUser = userServiceI.getById("1");
        logger.info(JSON.toJSONString(sysUser));
        System.out.println("-----");
        SysArea sysArea = sysAreaServiceI.getById("中文");
        logger.info(JSON.toJSONString(sysArea));
    }
}
```

Maven Projects Database Bean Validation CDI JSF Ant

Frameworks detected: Spring framework is detected in the project Configure (20 minutes ago)

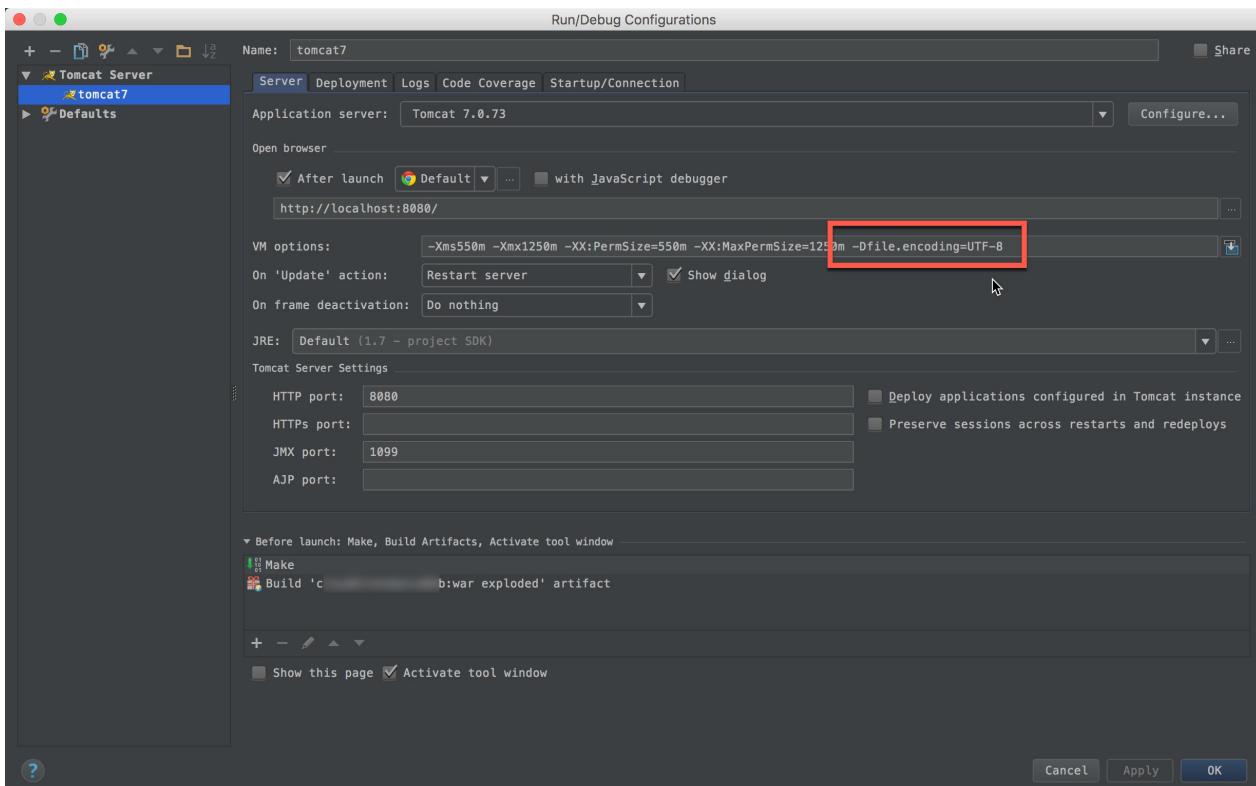
- 如上图演示，对单独文件的编码修改还可以点击右下角的编码设置区。如果代码内容中包含中文，则会弹出演示中的操作选择。

- `Reload` 表示使用新编码重新加载，新编码不会保存到文件中，重新打开此文件，旧编码是什么依旧还是什么。
 - `Convert` 表示使用新编码进行转换，新编码会保存到文件中，重新打开此文件，新编码是什么则是什么。
 - 含有中文的代码文件，`Convert` 之后可能会使中文变成乱码，所以在转换成请做好备份，不然可能出现转换过程变成乱码，无法还原。

由于编码问题引起的编译错误

- 编译报错：找不到符号、未结束的字符串文字 等的解决办法：
- 由于 UTF-8 编码文件有分 有BOM 和 无BOM 之分，默认情况下 IntelliJ IDEA 使用的编译器是 javac ，而此编译只能编译 无BOM 的文件，有很多 Eclipse 用户在使用 IntelliJ IDEA 开发 Eclipse 项目的时候常常会遇到此问题。主要是因为 Eclipse 的编译器是 Eclipse ，此编译器支持 有BOM 的文件编译。顾，解决办法是对于此文件进行 BOM 去除。
- 批量去除 BOM ，你可以 Google：批量去除 BOM 、批量转换无 BOM 等关键字，网络上已有提供各种方案。
- 除了通过去除 BOM 还有设置 IntelliJ IDEA 的编译器为 Eclipse ，但是一般不建议这样做。
- 如果上述问题都无法解决，而且你也确认 IntelliJ IDEA 各个配置编码的地方都是 UTF-8 ，报错文件编码也是是 UTF-8 无 BOM 的话，那还有一种可能也会出现这种情况：项目配置文件有问题。项目编码的配置文件在： / 项目目录 /.idea/encodings.xml 。如果你会修改此文件可以进行修改，如果不会，那就删除掉 .idea 整个目录，重启 IntelliJ IDEA 重新配置这个项目即可。

Tomcat 控制台输出乱码



- 如果你的 Tomcat 控制台输出乱码，并且你已经保证了本文上面的控制台字体设置你设置的字体包含中文，那你还可以尝试下在 Tomcat 的 VM 参数上加上： -Dfile.encoding=UTF-8
- 如果你是 Mac 系统，很有可能是需要的。

编程字体推荐

- 下载地址：<http://pan.baidu.com/s/1kVoF32R>
 - Microsoft YaHei Consolas
 - Microsoft YaHei Mono

常见文件类型的图标介绍

Java 类相关图标介绍

Symbols

In this section:

- Common
- Data Sources

Common

Icon	Description
Class	
Abstract Java class	
Annotation	
Enumeration	
Exception	
Final Java class	
Interface	
Java class that contains declaration of the main() method.	
Test case	
Java class located out of the source root. Refer to the section Configuring Content Roots for details.	
Java class excluded from compilation.	
Method	
Field	
Variable	
Property	
Parameter	
Element	
Directory	
Package	
Source root	
Test root	
Excluded root	

Visibility modifiers

private
protected
package protected
public

Data sources

Icon	Description
DB	DB data source. Also, DBMS-specific icons are used: DB2, Derby, H2, HSQLDB, MySQL, ODBC, Oracle, PostgreSQL, SQL Server, SQLite, Sybase.
RO	DB data source with the read-only status, e.g. RO for DB2.
DDL	DDL data source
Schema	
Table	
Column	A NOT NULL column
PK	Column with a primary key
FK	Column with a foreign key
Index	Column with an index
PK	Primary key
FK	Foreign key
Index	Index

View
StoredProcedure or function

See Also

Procedures:

- Auto-Completing Code

Reference:

- File Types
- Structure Tool Window, File Structure Popup

Web Resources:

- Developer Community..

- 官网地址：<http://www.jetbrains.com/idea/webhelp/symbols.html>
- 对于各个图标，上图的 `Description` 写得非常详细，但是有几个还是需要进行特别的说明下。
 - `Source root`，你可以理解为源目录，源码的作用就是用来专门放 Java 类文件，相对于编译出来的 `class` 文件而言，它就是源。我们一般默认名字叫 `src` 的目录就是源目录，但是其实并不是这样的，在 IntelliJ IDEA 中，即使叫 `srcs` 也是可以设置为 `Source root`，所以源目录跟目录命名是没有关系的，而是在于 IntelliJ IDEA 支持对任意目录进行设置为 `source root`，具体设置在会后面章节进行详解。`Source root` 的作用是标记该目录下的文件是可编译的。
 - `Java class located out of the source root`，由于上一条我们知道 `Source root` 目录是用来告诉 IntelliJ IDEA 这是编译目录，而假如你 Java 类文件没有放在该目录或是该目录的子包下，那该 Java 类则无法编译，就会被表示成这个图标。

各类文件类型图标介绍

File Types Recognized by IntelliJ IDEA

IntelliJ IDEA recognizes numerous file types. Each file type is denoted with a special icon. Custom file types are also allowed. Each file type is associated with one or more extensions that match a certain pattern.

The file types and their extensions are configurable in the [File Types dialog](#).

"Recognized" does not mean "supplied with extensive support". For example, psd files are recognized in the Community Edition and marked with the corresponding icon, although the edition does not provide Photoshop development support.

The default types include:

File Type	Icon	Recognized in
ActionScript files		Ultimate Edition
Active Server Pages files		Ultimate Edition
Android files		Ultimate Edition: all Android-related file types; Community Edition: Android IDL files, Android rendescript files
Apache Config files		Ultimate Edition
Archive files		Ultimate Edition, Community Edition
AspectJ files		Ultimate Edition, Community Edition
C# files		Ultimate Edition, Community Edition
C/C++ files		Ultimate Edition, Community Edition
Command Shell files		Ultimate Edition
CSS files		Ultimate Edition, Community Edition
CoffeeScript files		Ultimate Edition
Cucumber feature files		Ultimate Edition
ColdFusion files		Ultimate Edition
Eclipse project files		Ultimate Edition, Community Edition
Diagram files		Ultimate Edition
Drools files		Ultimate Edition
Erlang files		Ultimate Edition, Community Edition
EJB QL files		Ultimate Edition, Community Edition

Files marked as plain text		Ultimate Edition, Community Edition
Files opened in associated applications		Ultimate Edition, Community Edition
FreeMarker template files		Ultimate Edition
Gant scripts		Ultimate Edition
Gradle scripts		Ultimate Edition
Groovy files		Ultimate Edition, Community Edition
Groovy Server Pages		Ultimate Edition
GUI Form		Ultimate Edition, Community Edition
HAML files		Ultimate Edition
HTML files		Ultimate Edition, Community Edition
IntelliJ IDEA project, module or workspace files.		Ultimate Edition, Community Edition
IDL files		Ultimate Edition, Community Edition
Image files		Ultimate Edition, Community Edition
Java class files		Ultimate Edition, Community Edition
Java source files		Ultimate Edition, Community Edition
JavaFX files		Ultimate Edition, Community Edition
JavaScript files		Ultimate Edition
JSF files		Ultimate Edition, Community Edition
JSHint configuration files		Ultimate Edition
JSON files		Ultimate Edition
JSTestDriver Config files		Ultimate Edition
Java Server Pages files		Ultimate Edition, Community Edition
JSP files		Ultimate Edition, Community Edition
LESS files		Ultimate Edition
Patch files		Ultimate Edition, Community Edition
Perl files		Ultimate Edition, Community Edition
PHP files		Ultimate Edition, Community Edition
Properties files		Ultimate Edition, Community Edition
Puppet files		Ultimate Edition

Regular expressions		Ultimate Edition, Community Edition
RELAX NG Compact Syntax		Ultimate Edition, Community Edition
SASS files		Ultimate Edition
SCSS files		Ultimate Edition
Scala files		Ultimate Edition
Smarty, Smarty config files		Ultimate Edition
SQL files		Ultimate Edition, Community Edition
Drools Expert files		Ultimate Edition
Text files		Ultimate Edition, Community Edition
TypeScript files		Ultimate Edition
Velocity template files		Ultimate Edition
XHTML files		Ultimate Edition, Community Edition
XML DTD files		Ultimate Edition, Community Edition
XML files		Ultimate Edition, Community Edition
YAML files		Ultimate Edition

See Also

Procedures:

- » [Creating and Registering File Types](#)

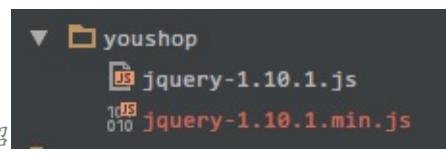
Reference:

- » [File Types](#)
- » [Register New File Type Association Dialog](#)

Web Resources:

- » [Developer Community](#)

- 官网地址：<http://www.jetbrains.com/idea/webhelp/file-types-recognized-by-intellij-idea.html>
- 对于各个图标，上图的介绍得非常清楚，我这边只做一个类型文件的补充。



- JavaScript 两种图标介绍
- 如上图所示，对于压缩过的 JavaScript 文件，图标会有 010 图案。

IntelliJ IDEA 缓存和索引介绍和清理方法

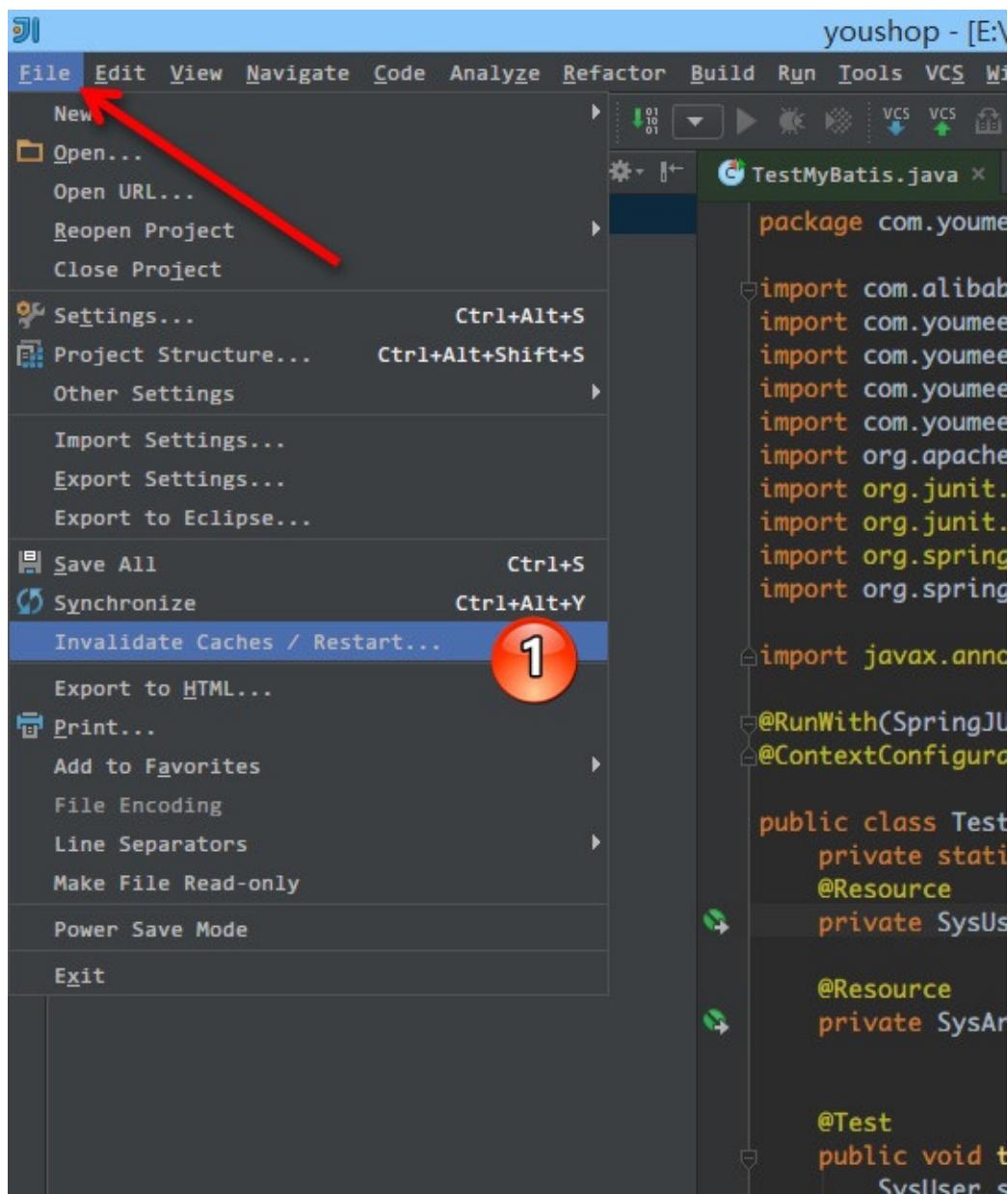
缓存和索引介绍

在《[IntelliJ IDEA 界面介绍](#)》章节里已经点到了 IntelliJ IDEA 首次加载项目的时候，都会创建索引，而创建索引的时间跟项目的文件多少成正比，我也简单强调了 IntelliJ IDEA 索引的重要性。这里我们再对此进行详细说明索引、缓存对 IntelliJ IDEA 的重要性。

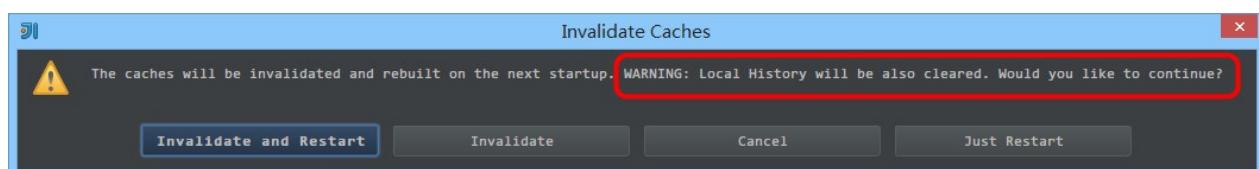
通过《[常见文件类型的图标介绍](#)》章节，你已经认识到 IntelliJ IDEA 下各个文件类型的图标是什么样子的。其中有一个图标我是专门进行了讲解： Java class located out of the source root。我们也都知道该图标是表示 Java 类文件没有在 Source root 目录下的文件夹下会显示此图标，但是其实还有一种情况也是会显示此图标的。那就是：在 IntelliJ IDEA 创建索引过程中，所有的 Java 类都是这个图标，如果你项目大的话很容易观察到的，几个文件的小项目倒是不一定能看到。所以在 IntelliJ IDEA 创建索引过程即使你编辑了代码也是编译不了、运行不起来的，所以还是安安静静等 IntelliJ IDEA 创建索引完成。

IntelliJ IDEA 的缓存和索引主要是用来加快文件查询，从而加快各种查找、代码提示等操作的速度，所以 IntelliJ IDEA 的索引的重要性我再唠叨一万遍都不为过。但是，IntelliJ IDEA 的索引和缓存并不是一直会良好地支持 IntelliJ IDEA 的，这某些特殊条件下，IntelliJ IDEA 的缓存和索引文件也是会损坏的，比如断电、蓝屏引起的强制关机，当你重新打开 IntelliJ IDEA，基本上百分八十的可能 IntelliJ IDEA 都会报各种莫名其妙错误，甚至项目打不开，IntelliJ IDEA 主题还原成默认状态。也有一些即使没有断电、蓝屏，也会有莫名其妙的问题的时候，也很有可能是 IntelliJ IDEA 缓存和索引出问题，这种情况还不少。遇到此类问题也不用过多担心，下面就来讲解如何解决。

清除缓存和索引



- IntelliJ IDEA 已经自带提供清除缓存、索引的路口，如上图标注 1 所示。

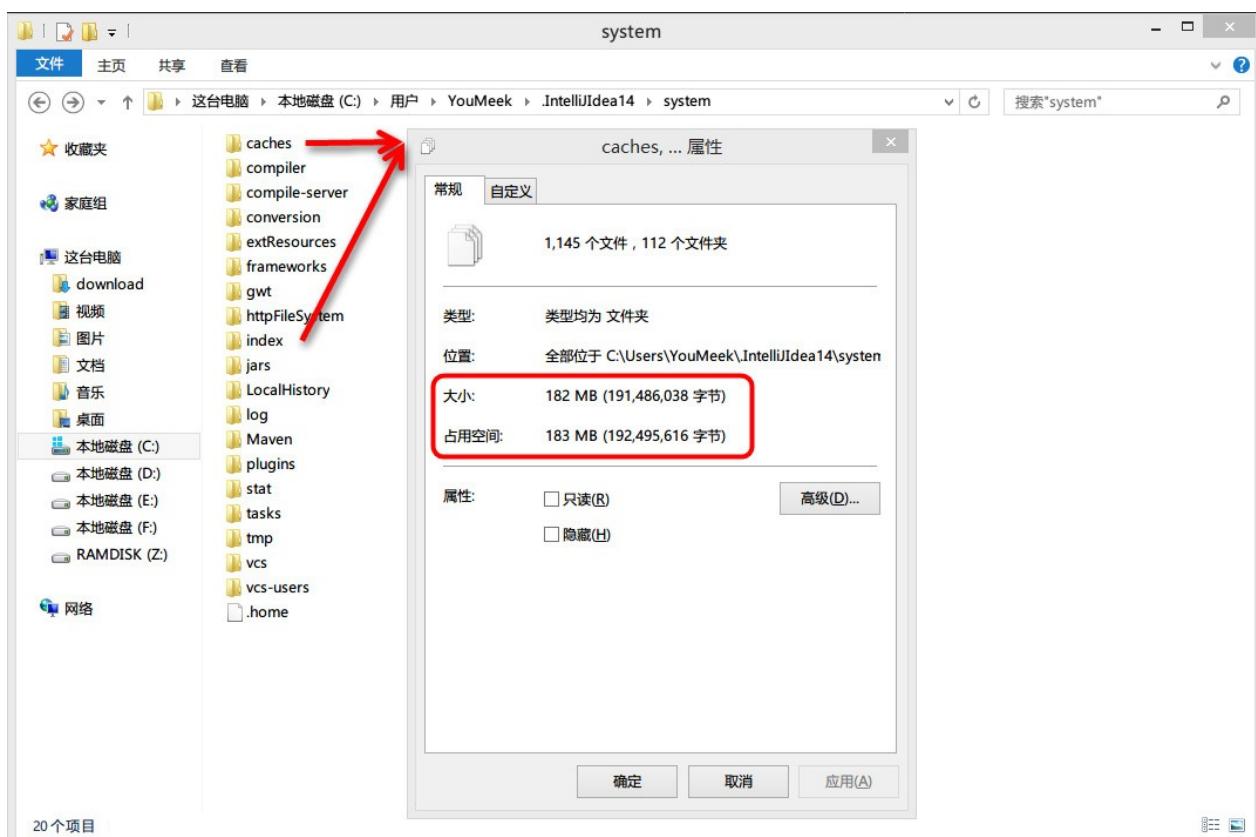


- 一般建议点击 `Invalidate and Restart`，这样会比较干净。
- 但是有一个需要提醒的是，如上图红圈标注的地方：清除索引和缓存会使得 IntelliJ IDEA 的 `Local History` 丢失，所以如果你项目没有加入到版本控制，而你又需要你项目文件的历史更改记录，那你最好备份下你的 `LocalHistory` 目录。目录地址在：`C:\Users\当前登录的系统用户名\.IntelliJIdea14\system\LocalHistory` 建议使用硬盘的全文搜索，这样效率更高。

通过上面方式清除缓存、索引本质也就是去删除 C 盘下的 system 目录下的对应的文件而已，所以如果你不用上述方法也可以删除整个 system。当 IntelliJ IDEA 再次启动项目的时候会重新创建新的 system 目录以及对应项目缓存和索引。

如果你遇到了因为索引、缓存坏了以至于项目打不开，那也建议你可以直接删除 system 目录，一般这样都可以很好地解决你的问题。

其他



- 目前我电脑的 IntelliJ IDEA 是新装的，也就打开了几个小项目，所有打开的项目大小加起来不到 5M，但是他们创建的索引大家就已经上百兆了，如上图所示。所以如果你 C 盘空间不足的情况下，最好转移下 system 目录，方法可以根据《IntelliJ IDEA 相关核心文件和目录介绍》讲解的方法进行。

IntelliJ IDEA 编译方式介绍

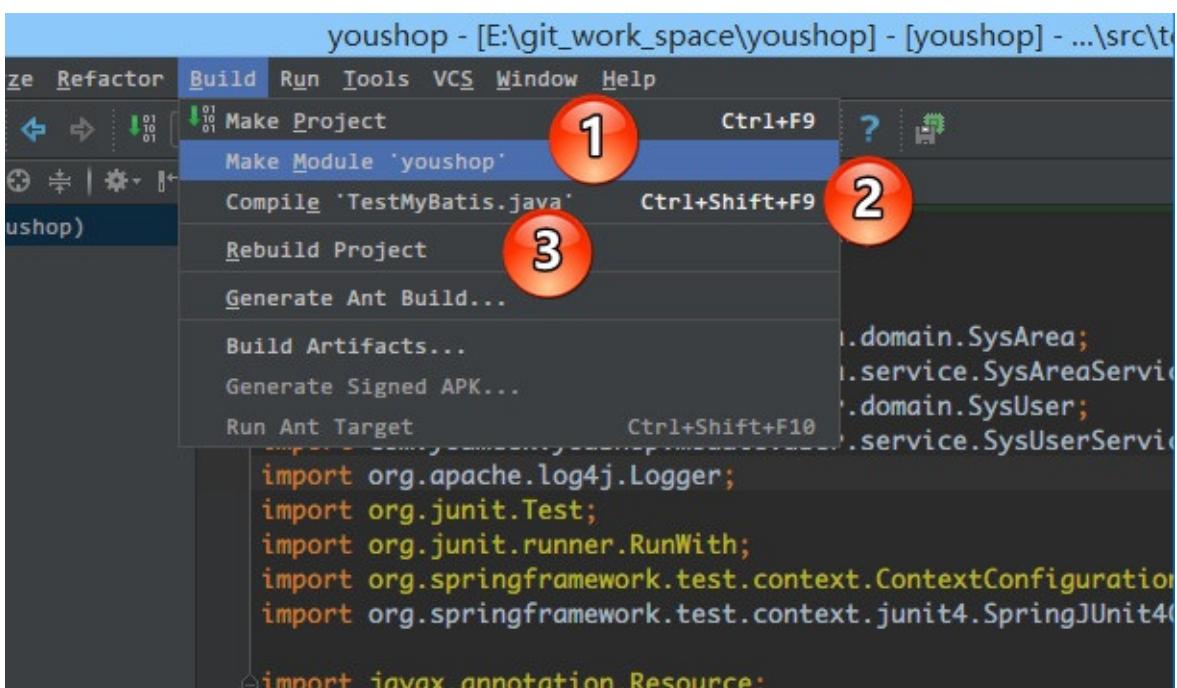
编译方式介绍

相比较于 Eclipse 的实时自动编译，IntelliJ IDEA 的编译更加手动化，虽然 IntelliJ IDEA 也支持通过设置开启实时编译，但是不建议，因为太占资源了。IntelliJ IDEA 编译方式除了手工点击编译按钮进行编译之外，还有就是在容器运行之前配置上一个编译事件，先编译后运行。默认下 IntelliJ IDEA 也都是这样的设置，所以实际开发中你也不用太注意编译这件事。虽然 IntelliJ IDEA 没有实时编译，但是对于代码检查完全是没有影响。但是多个类之间的关联关系还是要等 Make 或 Rebuild 触发的时候才会做相关检查的。

在 IntelliJ IDEA 里，编译方式一共有三种：

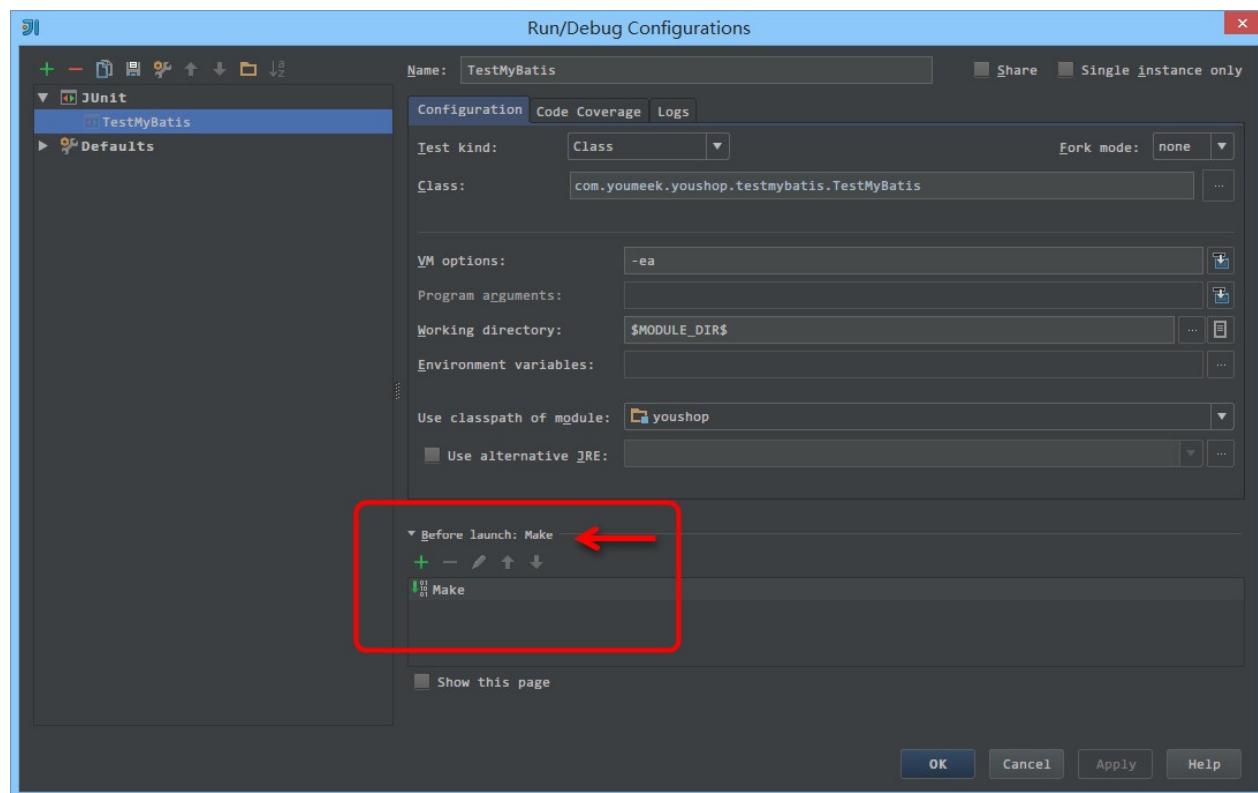
- Compile：对选定的目标（Java 类文件），进行强制性编译，不管目标是否是被修改过。
- Rebuild：对选定的目标（Project），进行强制性编译，不管目标是否是被修改过，由于 Rebuild 的目标只有 Project，所以 Rebuild 每次花的时间会比较长。
- Make：使用最多的编译操作。对选定的目标（Project 或 Module）进行编译，但只编译有修改过的文件，没有修改过的文件不会编译，这样平时开发大型项目才不会浪费时间在编译过程中。

编译触发按钮

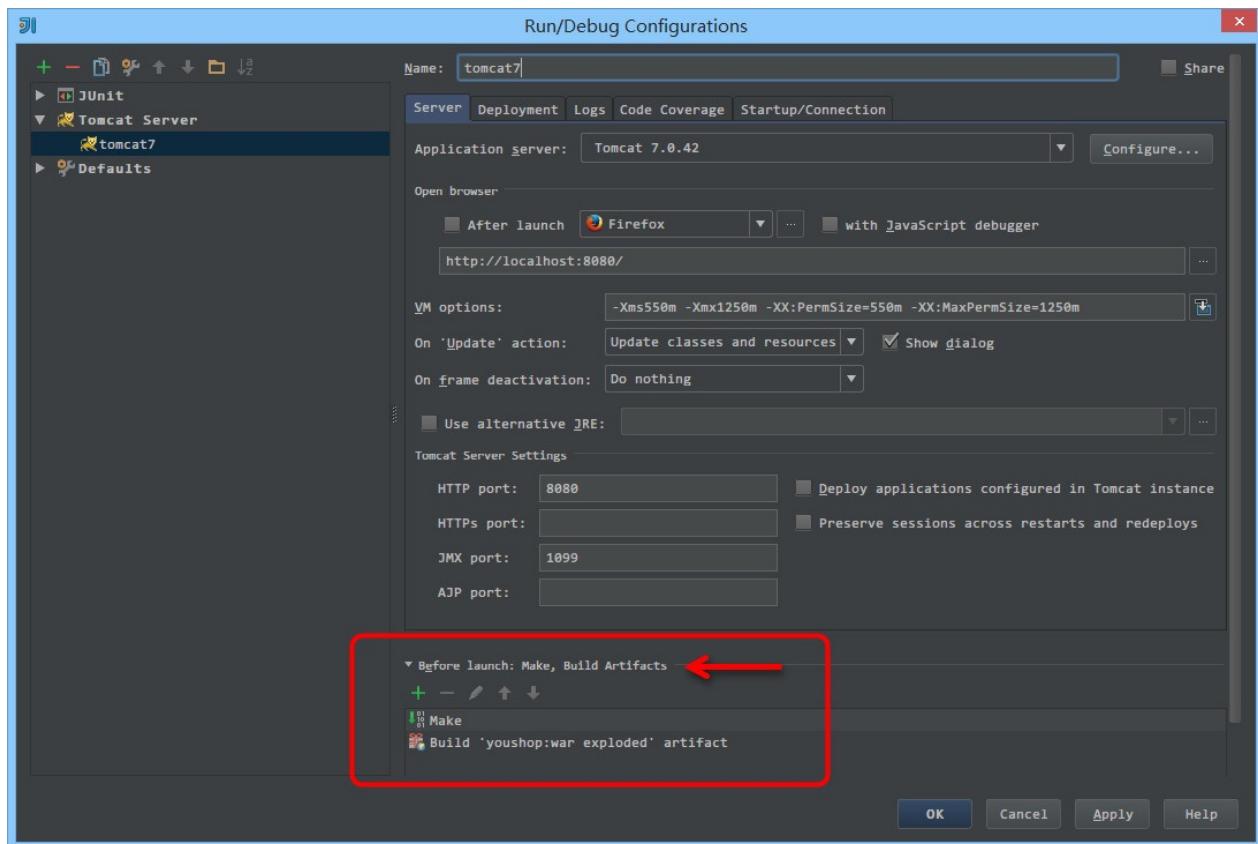


- 如上图标注 1 所示，Compile 的操作有： Compile 指定类
- 如上图标注 1 所示，Rebuild 的操作有： Rebuild Project
- 如上图标注 1 所示，Make 的操作有： Make Project 、 Make Module

运行之前的编译

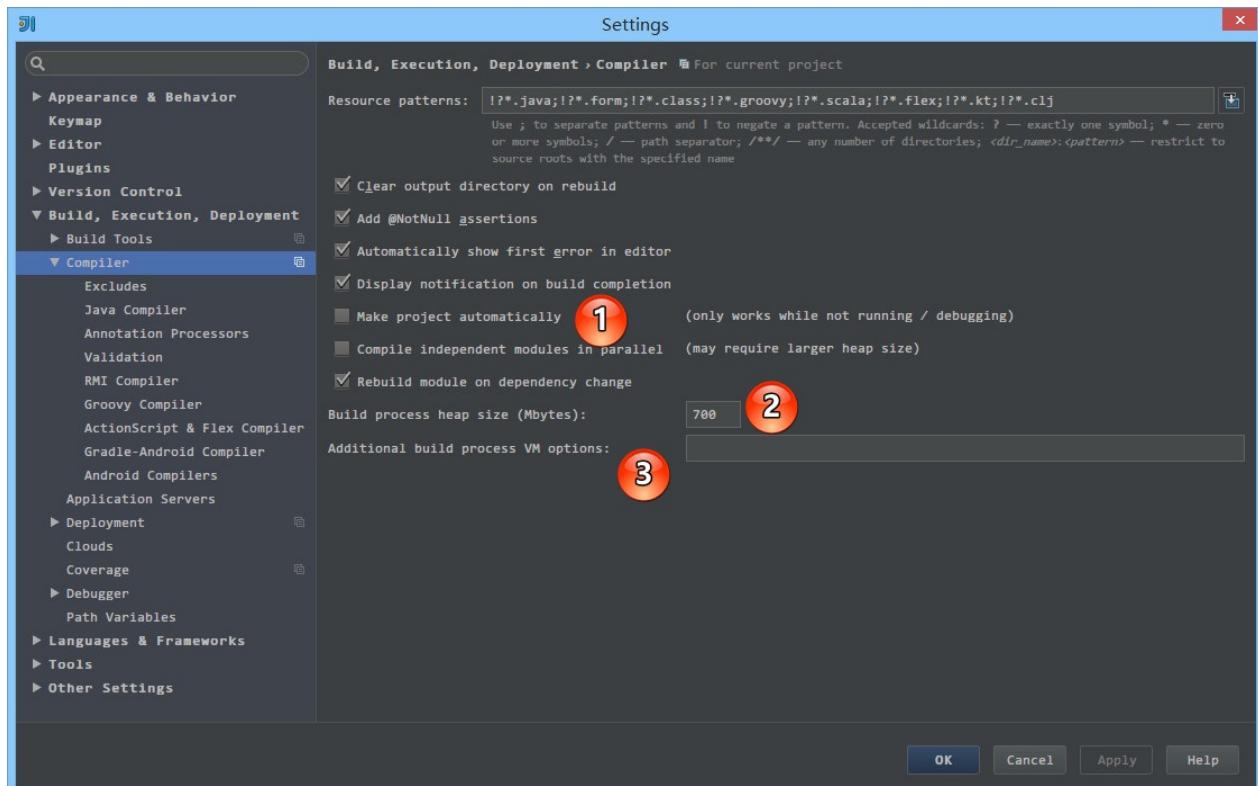


- 如上图所示，IntelliJ IDEA 默认在运行 JUnit 之前会先进行 Make 操作。

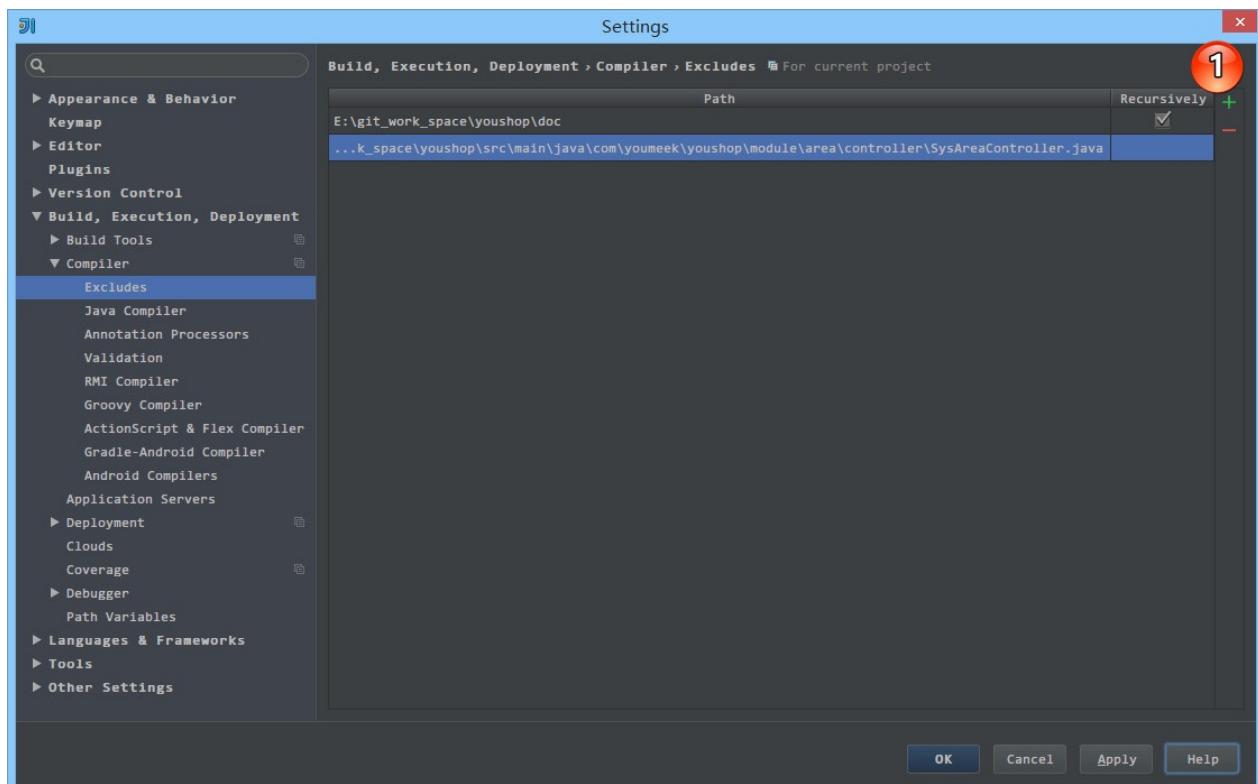


- 如上图所示，IntelliJ IDEA 默认在运行 tomcat 之前会先进行 Make 操作。

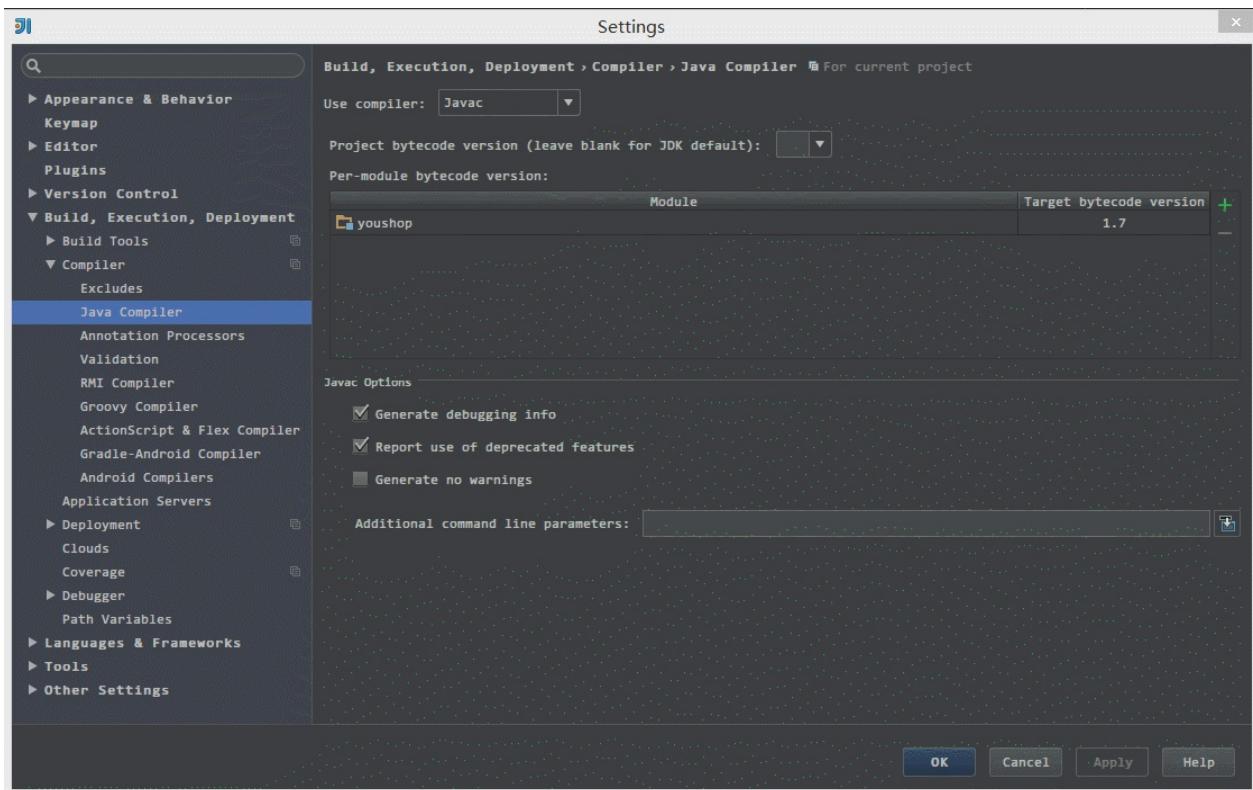
编译器的设置和选择



- 上图标注 1 所示，也是我们本文前面讲的，IntelliJ IDEA 是支持自动编译的，默认是不开启的，也建议不用开启，原因前面已经说了。
- 上图标注 2 所示，设置编译 heap 大小，默认是 700，建议使用 64 位的用户，在内存足够的情况下，建议改为 1500 或以上。如果你在编译的时候出错，报：OutOfMemoryError，一般也是要来改这个地方。
- 上图标注 3 所示，还可以设置编译时的 VM 参数，这个你可以根据需求进行设置，一般人是用不上的。



- 如上图标注 1 所示，可以添加目录或文件进行编译排除。
- 在项目中，如果有任何一个可编译的文件无法编译通过，则 IntelliJ IDEA 是无法运行起来的，必须等你全部问题解决，编译通过之后才可运行。但是可能开发过程中，某一个包目录的文件编译无法通过，但是我们又不急着改，那我们就可以考虑把该包加入到排除编译列表中，则项目就可以运行起来。



- 如上图动态 Gif 所示，IntelliJ IDEA 支持常见的几种编译器：Javac、Eclipse、Ajc 等。默认是 Javac，也推荐使用 Javac。
- Project bytecode version 针对项目字节码编译版本，一般选择的是当前项目主 JDK 的版本。
- Per-module bytecode version 可以针对 Project 下各个 Module 的特殊需求单独设置不同的 bytecode version，前提是电脑上必须有安装对应的 JDK 版本。

IntelliJ IDEA 项目相关的几个重要概念介绍

必备材料介绍

- IntelliJ IDEA 对其他 IDE 转过来的用户有特别优待，对其专门整理了非常棒的资料，还请其他 IDE 过来的用户抽时间查看，会有很大帮助：
 - Eclipse 用户可以看：<https://www.jetbrains.com/idea/help/eclipse.html>
 - NetBeans 用户可以看：<https://www.jetbrains.com/idea/help/netbeans.html>

Project 和 Module 介绍

这两个概念是 IntelliJ IDEA 的必懂知识点之一，请务必要学会。

如果你是 Eclipse 用户，并且已经看了上面给的链接，那 IntelliJ IDEA 首先告诉你一个非常重要的事情：IntelliJ IDEA 没有类似 Eclipse 工作空间（workspace）的概念的。很多从 Eclipse 转过来的人总是下意识地要再同一个窗口管理 n 个项目，这在 IntelliJ IDEA 是无法得到。

IntelliJ IDEA 提供的体验是：一个 Project 打开一个 Window 窗口。

对于 Project，IntelliJ IDEA 是这样解释的：

- Whatever you do in IntelliJ IDEA, you do that in the context of a project. A project is an organizational unit that represents a complete software solution. It serves as a basis for coding assistance, bulk refactoring, coding style consistency, etc.
- Your finished product may be decomposed into a series of discrete, isolated modules, but it's a project definition that brings them together and ties them into a greater whole.
- Projects don't themselves contain development artifacts such as source code, build scripts, or documentation. They are the highest level of organization in the IDE, and they define project-wide settings as well as collections of what IntelliJ IDEA refers to as modules and libraries.
 - 链接地址：<https://www.jetbrains.com/idea/help/project.html>

对于 Module，IntelliJ IDEA 是这样解释的：

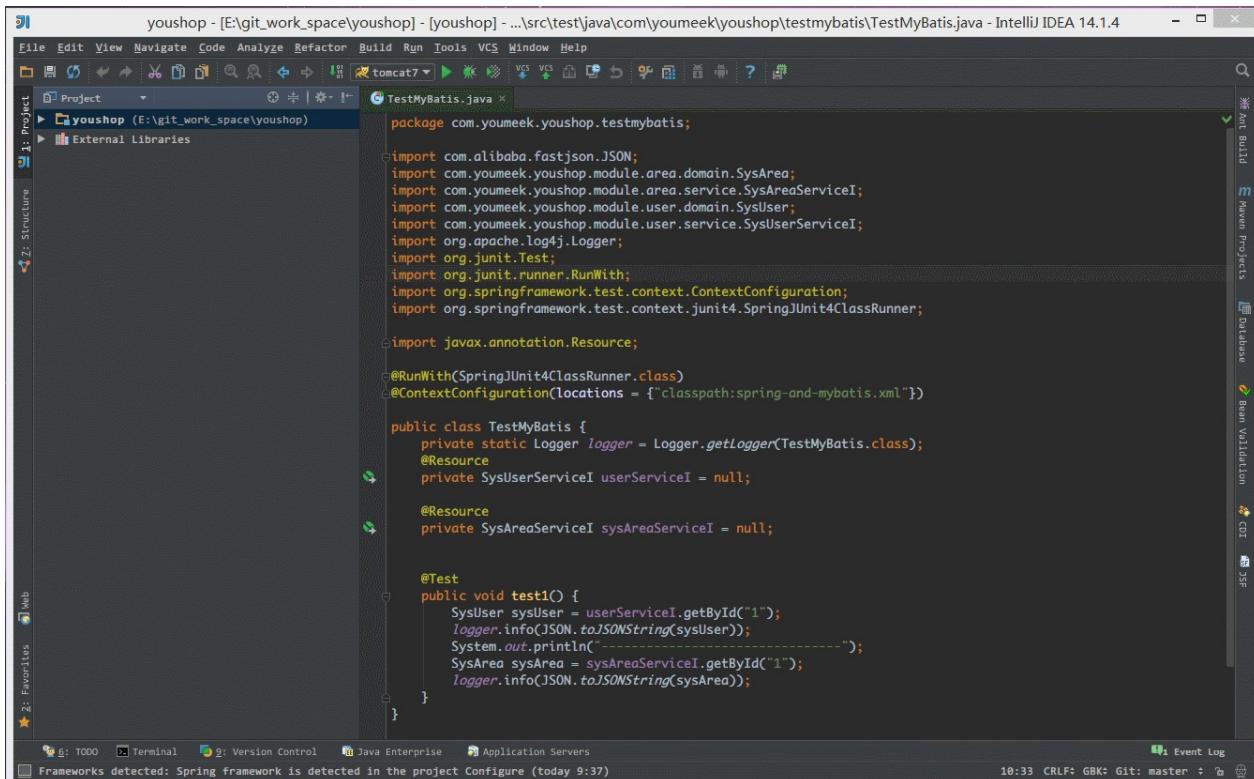
- A module is a discrete unit of functionality which you can compile, run, test and debug independently.
 - Modules contain everything that is required for their specific tasks: source code, build scripts, unit tests, deployment descriptors, and documentation. However, modules exist and are functional only in the context of a project.
 - Configuration information for a module is stored in a .iml module file. By default, such a file is located in the module's content root folder.
 - Development teams, normally, share the .iml module files through version control.
- 链接地址：<https://www.jetbrains.com/idea/help/module.html>

通过上面的介绍我们知道，在 IntelliJ IDEA 中 Project 是最顶级的级别，次级别是 Module。一个 Project 可以有多个 Module。目前主流的大型项目结构都是类似这种多 Module 结构，这类项目一般是这样划分的，比如：core Module、web Module、plugin Module、solr Module 等等，模块之间彼此可以相互依赖。通过这些 Module 的命名也可以看出，他们之间应该都是处于同一个项目业务情况下的模块，彼此之间是有不可分割的业务关系的。

所以我们现在总结：一个 Project 是由一个或多个 Module 组成，模块之间尽量是处在同一个项目业务的情况下，彼此之间互相依赖关联。这里用的是 尽量，因为 IntelliJ IDEA 的 Project 是一个没有具备任何编码设置、构建等开发功能的，主要起到一个项目定义、范围约束、规范等类型的效果，也许我们可以简单地理解为就是一个单纯的目录，只是这个目录命名上必须有其代表性的意义。

下面我们以著名的 spring-framework 项目为例介绍多 Module 的结构的：

- 项目主页：<https://github.com/spring-projects/spring-framework>：
- 该项目的 Project 命名是：spring-framework。该目录主要作用为各个 Module 的顶层目录进行约束，告诉协同者，这个目录下都是 spring-framework 相关的，我绝不会放 Android 相关源码、文档、文件在上面的。该目录并不是以一个实际性的目录来体现的，所以你访问主页是看不到的，但是当你 checkout 的时候，你必须为这个项目命名，至于命名默认就是 spring-framework。
- 该 Project 下有二十来个 Module，各个 Module 的命名也是有含义的，比如：spring-core、spring-jdbc、spring-jms、spring-orm、spring-web、spring-webmvc 等等，我们通过这些命名也能清楚地知道他们要表达的含义，这些 Module 下也都各自有 src 编码目录，可以自行编码和构建。



- 相比较于多 Module 项目，小项目就无需搞得这么复杂。只有一个 Module 的结构 IntelliJ IDEA 也是支持的，并且 IntelliJ IDEA 创建项目的时候，默认就是单 Module 的结构的。
- 如上图 Gif 图演示，在输入 `Project name` 的时候，`Module name` 和 `Module file Location` 自动进行改变，同时 `Project location` 和 `Module file Location` 完全一样，这也就表示，Project 目录和 Module 目录是同一个，所以此时 Project 目录下就会有 `src` 目录，但是我们应该明白其本质还是 Module 的目录。

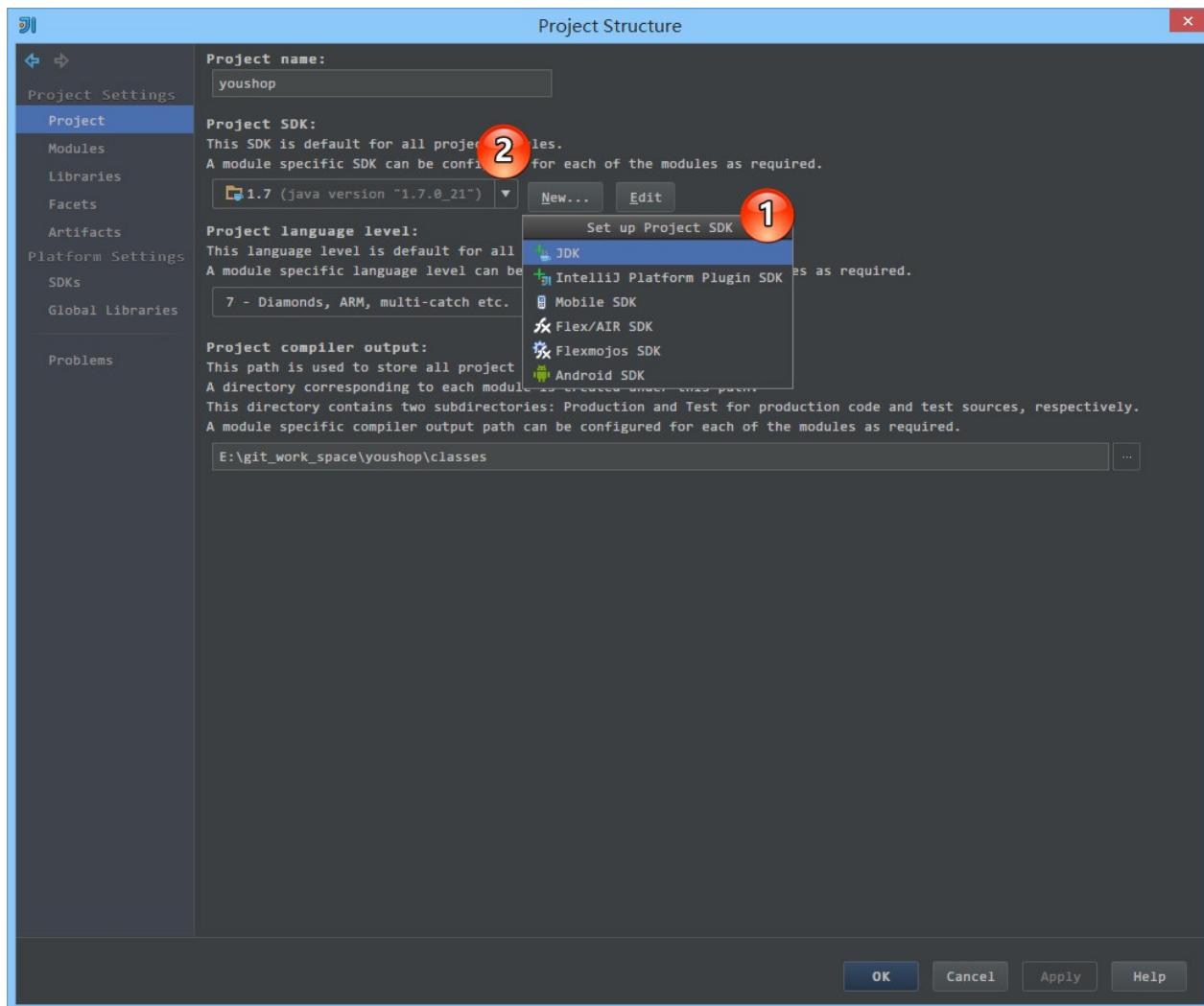
关于 IntelliJ IDEA 的 Project 和 Module 终于解释清楚了，但是由于 IntelliJ IDEA 官网上又有一段话对此解释得不够好，特别是对 Eclipse 用户来讲：<https://www.jetbrains.com/idea/help/eclipse-faq.html>，其中有这样两句话：

- An Eclipse workspace is similar to a project in IntelliJ IDEA
- An Eclipse project maps to a module in IntelliJ IDEA

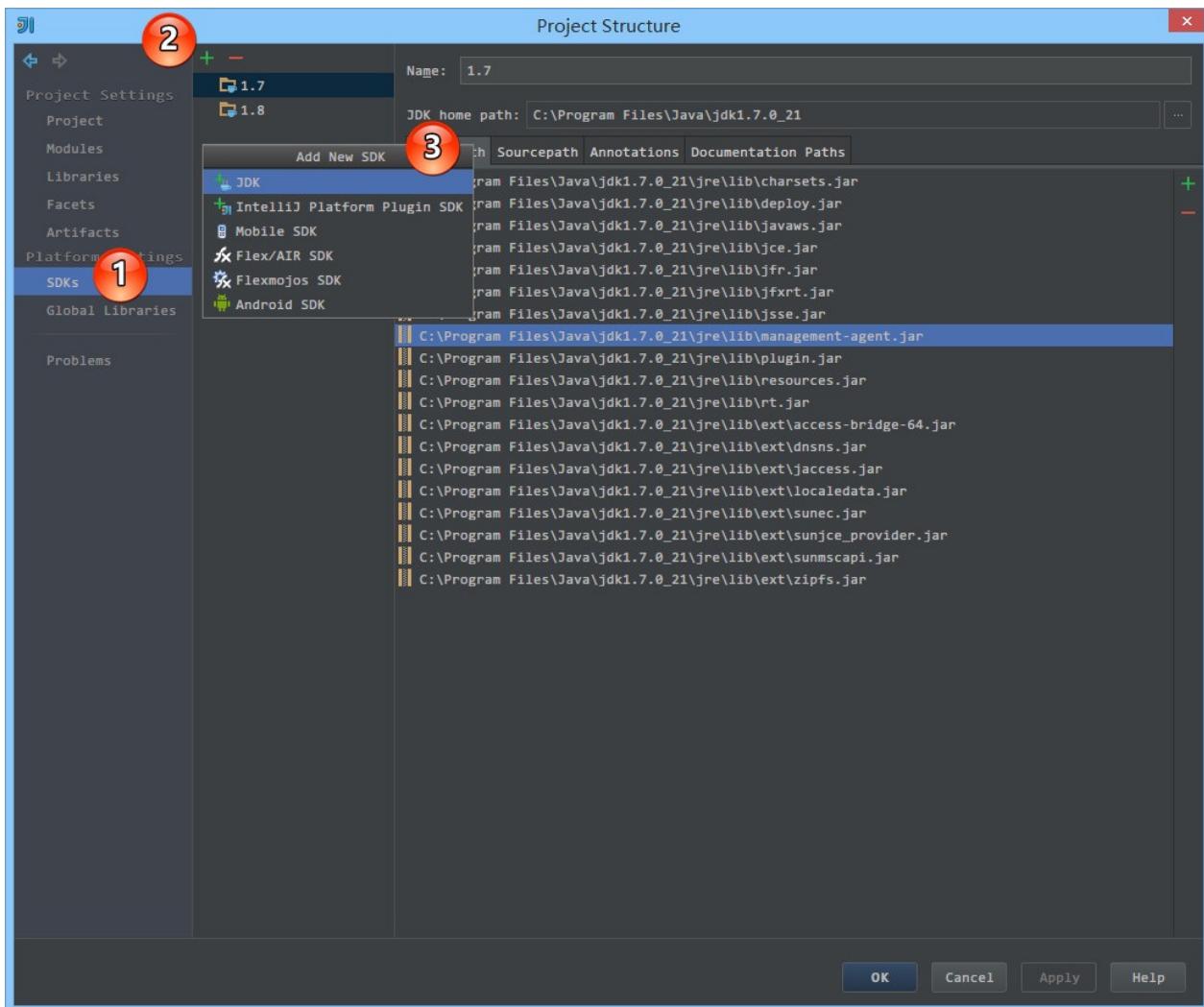
你可以把 IntelliJ IDEA 的 `Project` 当做 `workspace` 使用，IntelliJ IDEA 也是支持的，但是就像我们前面解释的那么那么多，这样是非常不符合其初衷的，所以请别把这段话当做教义去学习。对此 zeroturnaround 的大牛也有针对此进行了说明：

[http://zeroturnaround.com/rebellabs/getting-started-with-intellij-idea-as-an-eclipse-user/3/](http://zeroturnaround.com/rebellabs/getting-started-with-intellij-idea-as-an-eclipse-user/)

SDK (Software Development Kit) 介绍



- 按 `Ctrl + Shift + Alt + S` 弹出项目结构设置区，如上图所示。
- 如上图标注 1 所示，IntelliJ IDEA 支持 6 种 SDK。最常用的就是 `JDK` 和 `Android SDK`，其中在创建 `Android SDK` 的时候如果你没有先配置一个 `JDK` 的话，IntelliJ IDEA 则会提示你要先配置一个 `JDK`，然后才能配置 `Android SDK`。
- 如上图标注 2 所示，下拉会展示已经创建的所有 SDK，可以很方便地不同 SDK 中切换。在开发 Java 项目过程中，由于 IntelliJ IDEA 支持管理多个 `JDK`，所以你完全不用担心你系统上不同项目需要不同 `JDK`。

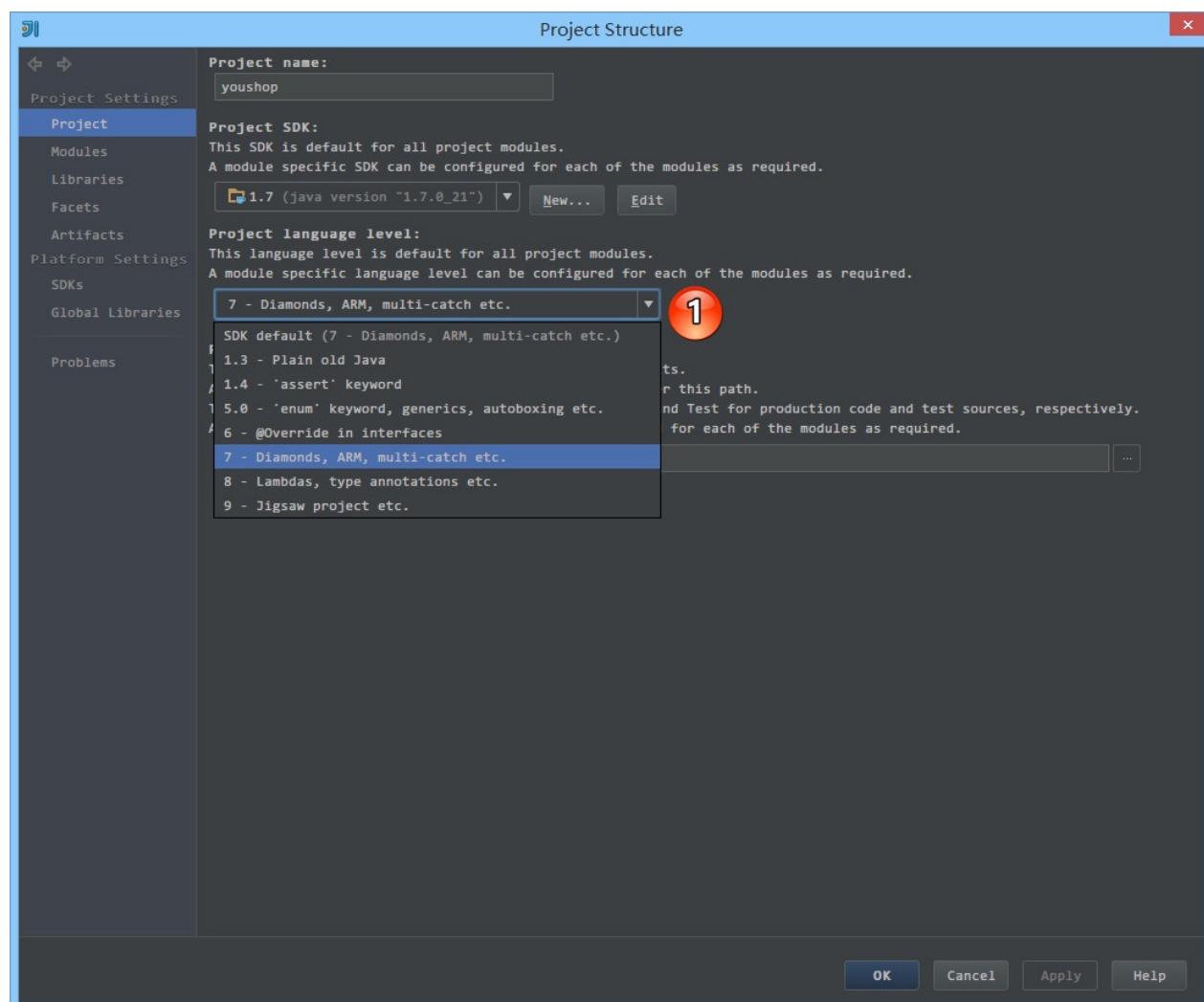


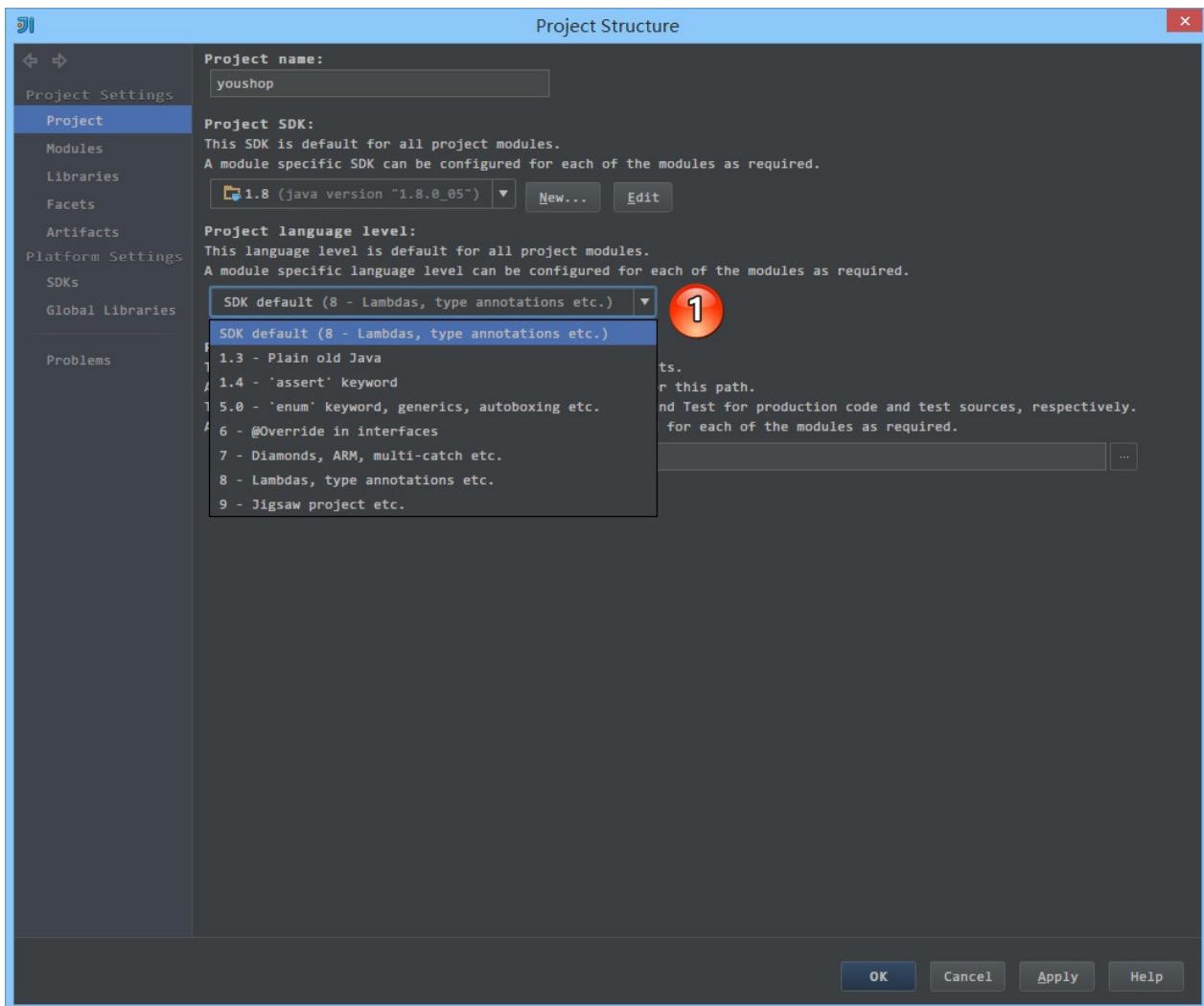
- 如上图标注 1 所示，`SDKs` 为 SDK 的统一管理处。
- 如上图标注 2 所示，加号可以添加新 SDK，支持的类型如标注 3 所示；减号可以删除光标所选的 SDK。
- 官网介绍：<https://www.jetbrains.com/idea/help/sdk.html>

language level 介绍

其他 IDE 没有看到类似 `language level` 的设置，所以这个功能应该算是 IntelliJ IDEA 特有的，可是 IntelliJ IDEA 官网也没有专门介绍 `language level` 的地方，也许 IntelliJ IDEA 认为这个知识点属于 JDK 范畴所以没加以介绍吧。所以这里主要我对此的一些理解。我们应该知道 Java JDK 在每个新版本都会有其新特性，而新版本一般也会向下兼容旧版本的特性，IntelliJ IDEA 是对这些 JDK 的新特性是这样介绍的：

- JDK 6 的新特性：@Override in interfaces
 - JDK 7 的新特性：Diamonds，ARM，multi-catch etc.
 - JDK 8 的新特性：Lambdas，type annotation etc.
 - JDK 9 的新特性：Jigsaw project etc.
- etc. == et cetera == and so on == 等等





- 如上第一张图标注 1，使用的是 JDK 7，显示的 `SDK default` 为 `7 - Diamonds, ARM, multi-catch etc.`
- 如上第二张图标注 1，使用的是 JDK 8，显示的 `SDK default` 为 `8 - Lambdas, type annotation etc.`

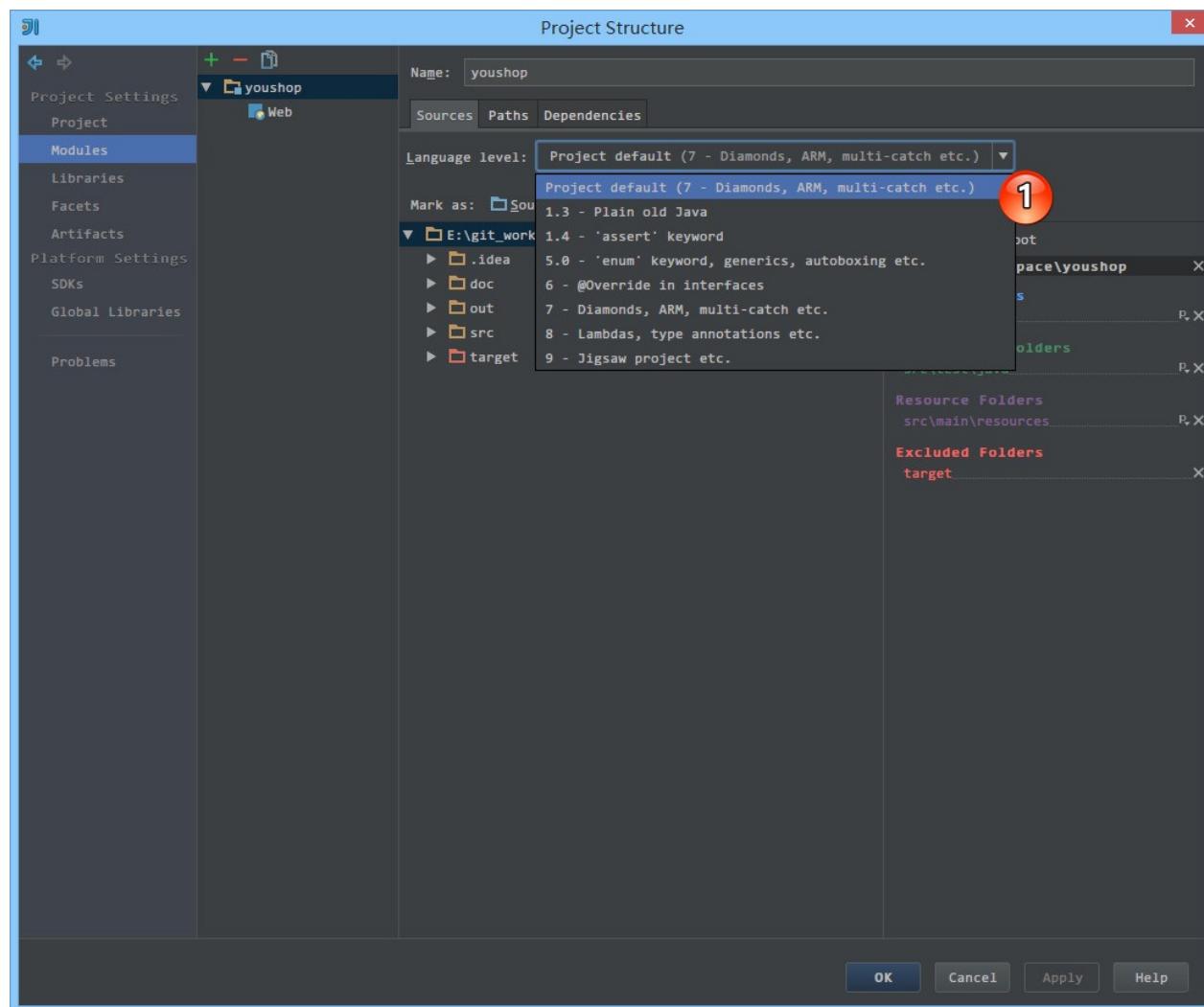
当我们使用 JDK 8 的时候，我们只能向下兼容 JDK 8 及其以下的特性，所以只能选择 8 及其以下的 `language level`。所以当我们项目使用的是 JDK 8，但是代码却没有使用 JDK 8 的新特性，最多使用了 JDK 7 的特性的时侯我们可以选择 `7 - Diamonds, ARM, multi-catch etc.`。

对此我们总结 `language level`：限定项目编译检查时最低要求的 JDK 特性。

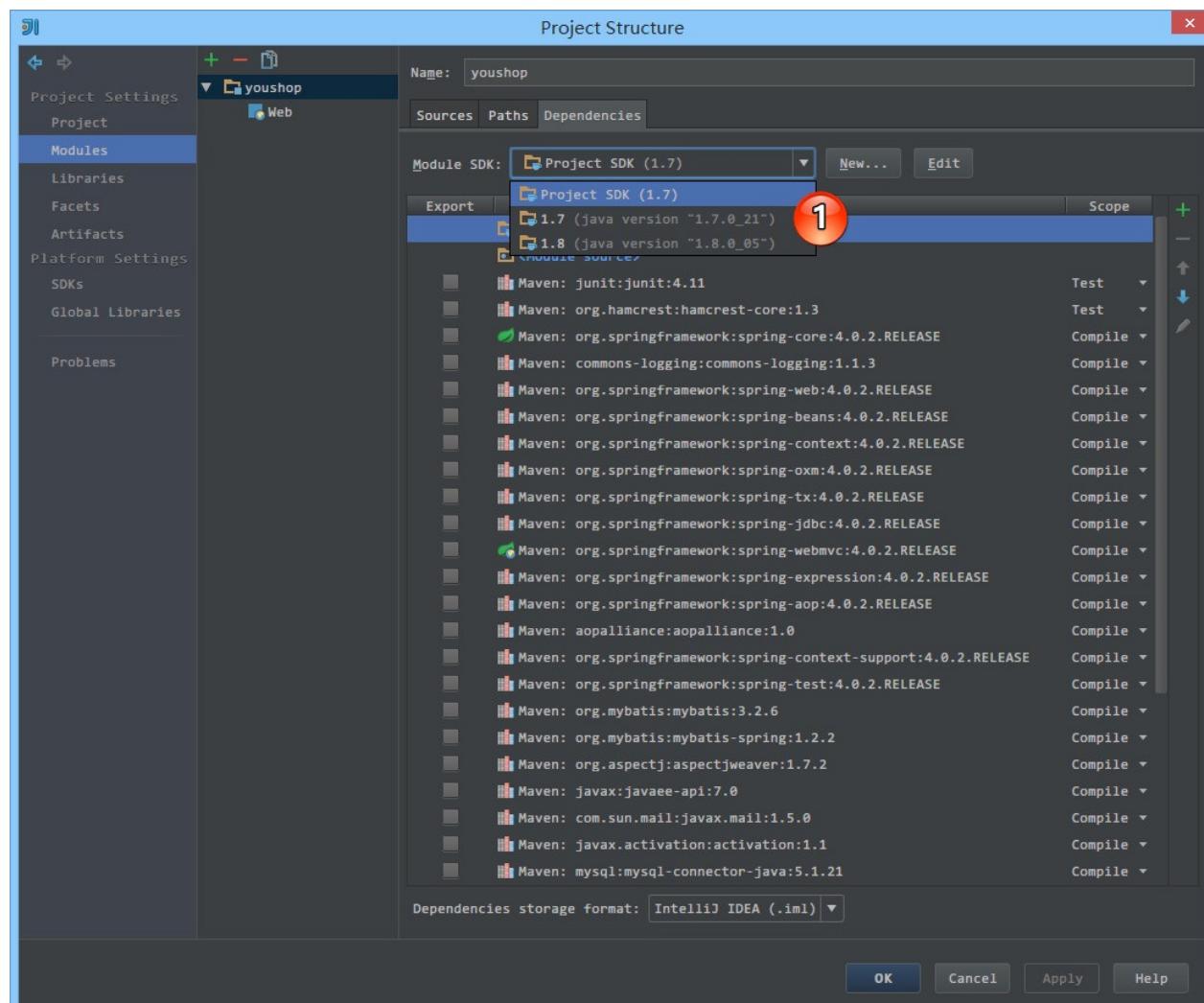
现在假设我们有一个项目代码使用的 JDK 8 新特性：`lambda` 语法，但是 JDK 选择的却是 JDK 7，即使 `language level` 选择了 `8 - Lambdas, type annotation etc.`，也是没有多大意义的，一样会编译报错。

Module 下的 SDK 和 language level

对于大型项目，各个 Module 用到的 `SDK` 和 `language level` 很有可能是各不一样的，IntelliJ IDEA 对此也进行了支持。



- 如上图标注 1 所示，可以针对 Module 选择其他 SDK，默认选择的是 Project SDK



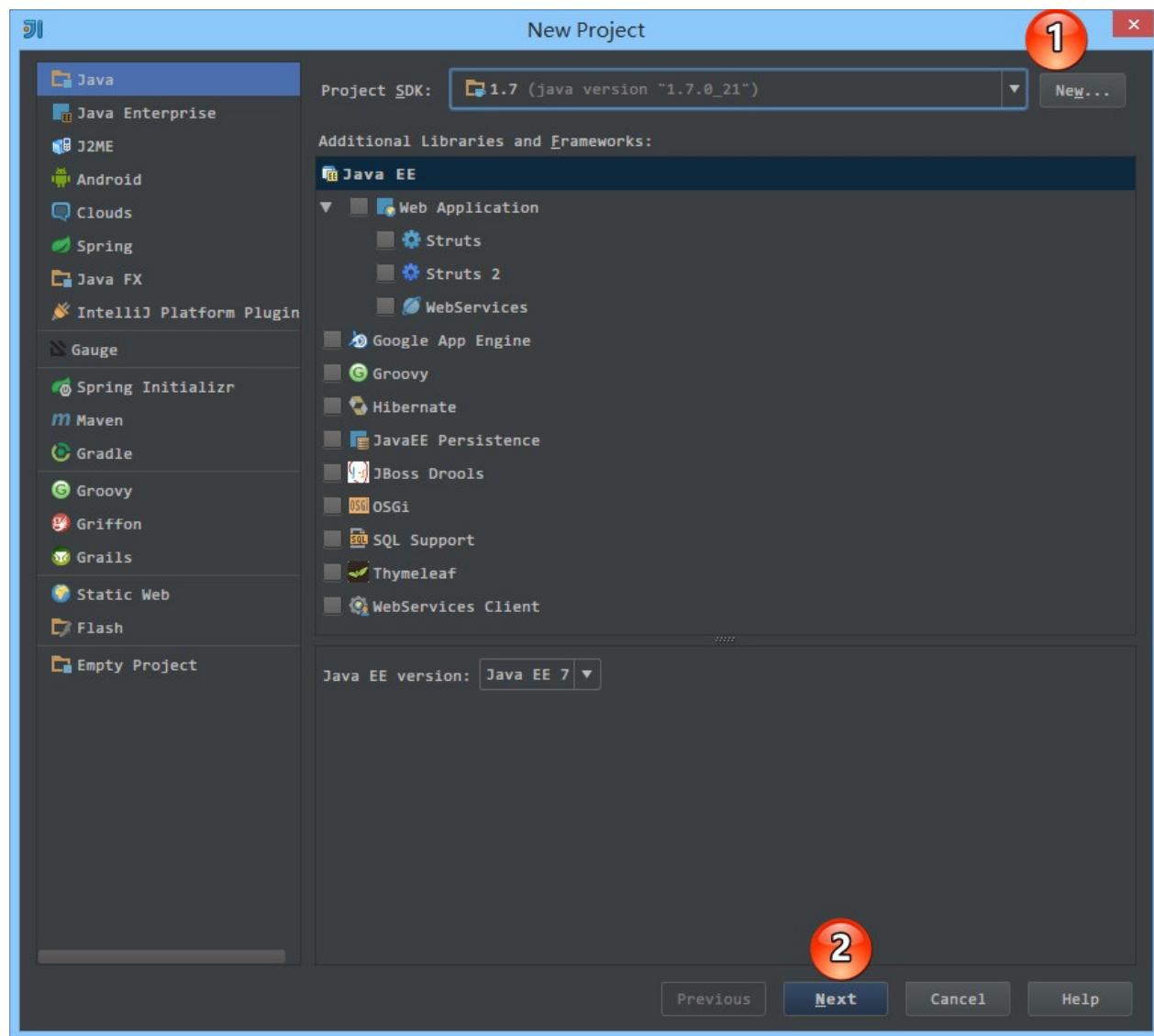
- 如上图标注 1 所示，可以针对 Module 选择其他 language level，默认选择的是 Project language level

Hello World 项目创建与项目配置文件介绍

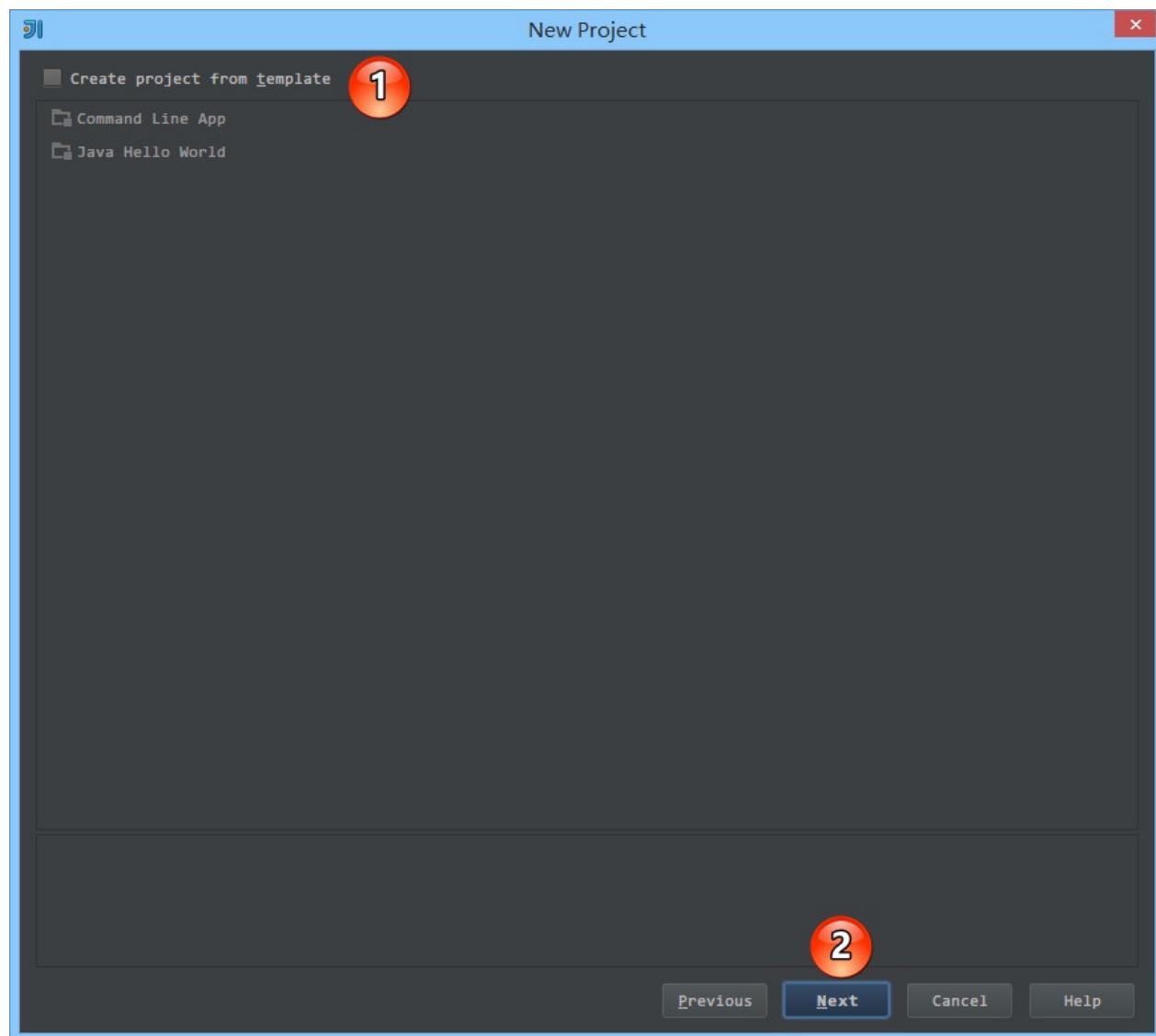
Hello World 项目创建



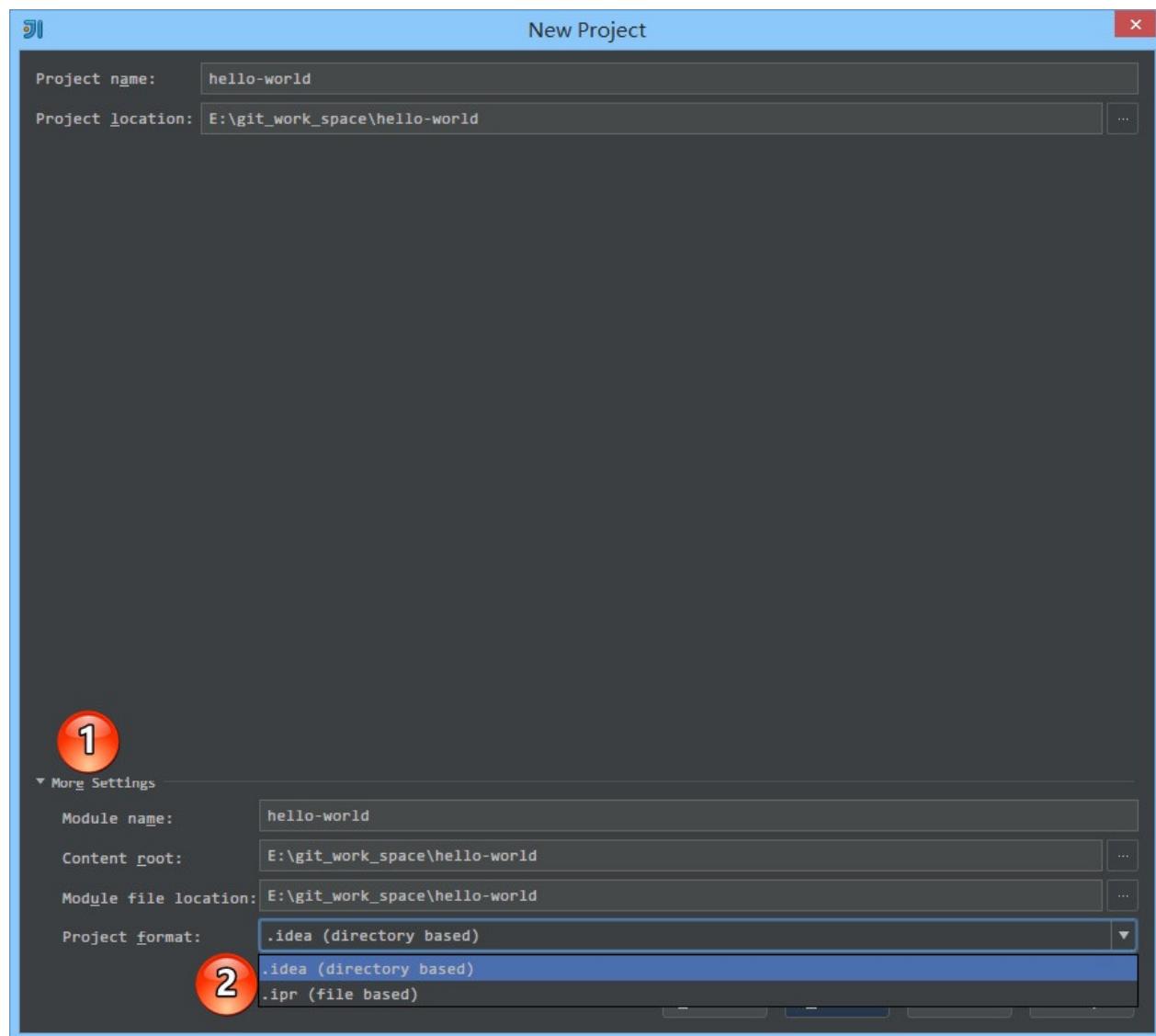
- 如上图标注 1 所示，点击 Create New Project 进入向导式创建项目



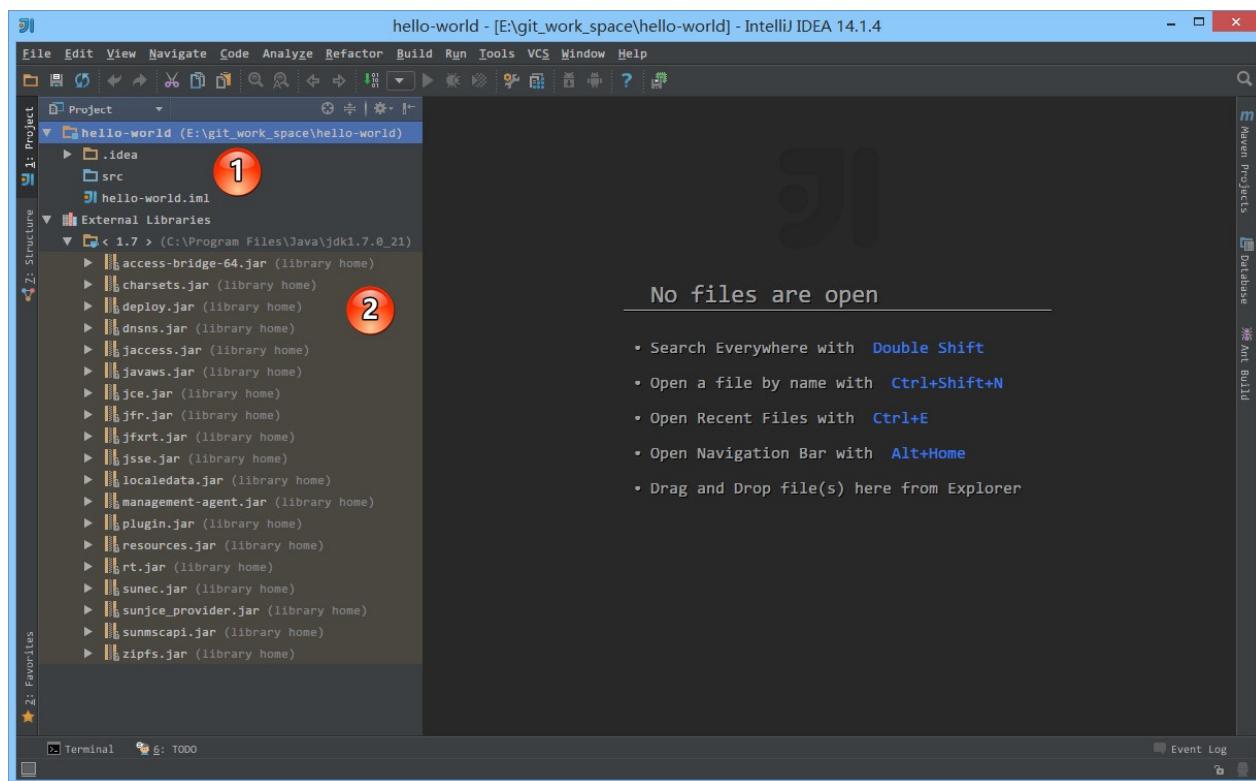
- 如上图标注 1 所示，如果此时 IntelliJ IDEA 还没有配置任何一个 SDK 的话，可以点击 `New...` 先进行 SDK 的配置。
- 如上图标注 2 所示，配置好 SDK 或选择好 SDK 之后，点击 `Next` 进入下一步。



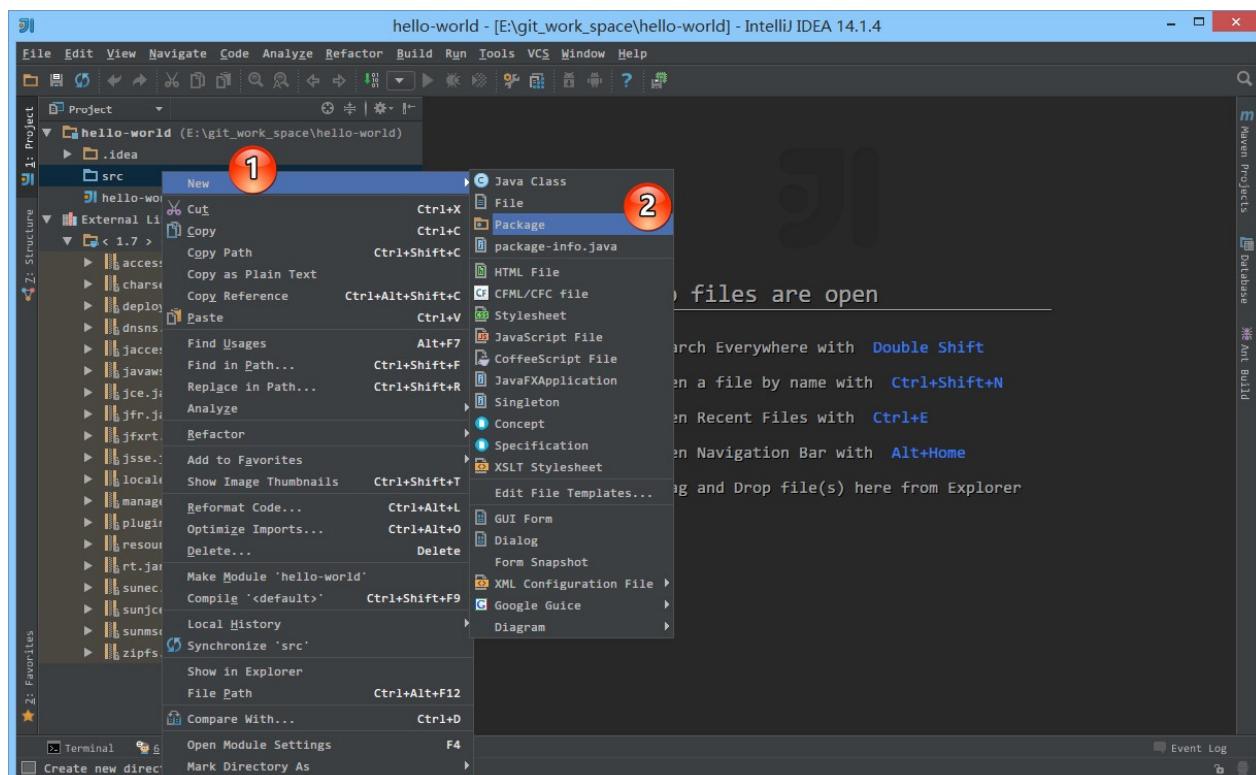
- 如上图标注 1 所示，可以选择模板快速创建项目。
 - `Command Line App` 会自动创建一个带有 `main` 方法的类。
 - `Java Hello World` 会自动创建一个带有 `main` 方法的并且会打印输出 `Hello World` 的类。
- 我们这里不勾选使用模板，而是手工创建，所以我们点击上图标注 2，进入下一步。



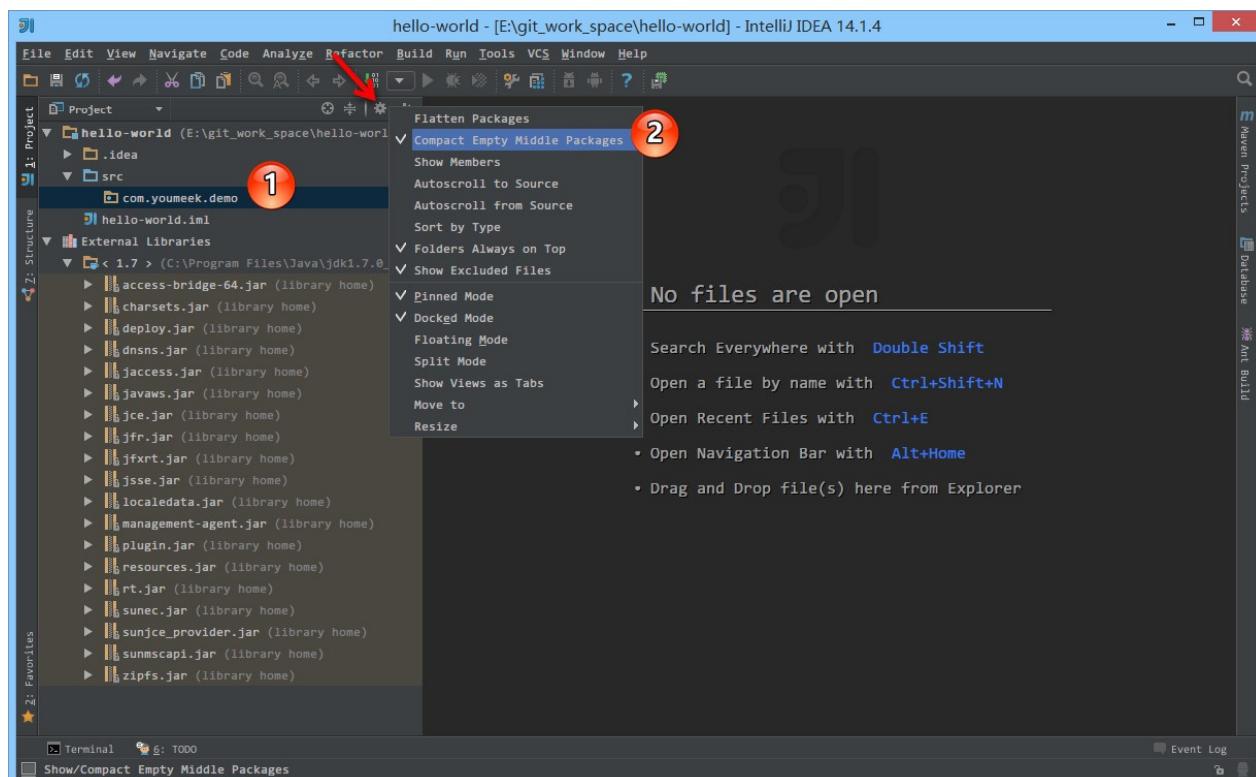
- 如上图标注 1 所示，默认 More Settings 是没有展开的，点击此处可以展开更多细节的信息。
- 如上图标注 2 所示，IntelliJ IDEA 的项目格式文件主要提供两种方式
 - .idea (directory based) 创建项目的时候自动创建一个 .idea 的项目配置目录来保存项目的配置信息。这是默认选项。
 - .ipr (file based) 创建项目的时候自动创建一个 .ipr 的项目配置文件来保存项目的配置信息。
 - 需要特别注意的是，我这边默认创建的项目编码是 GBK，而如果你需要 UTF-8 的话，修改编码的方式请看第 10 讲。



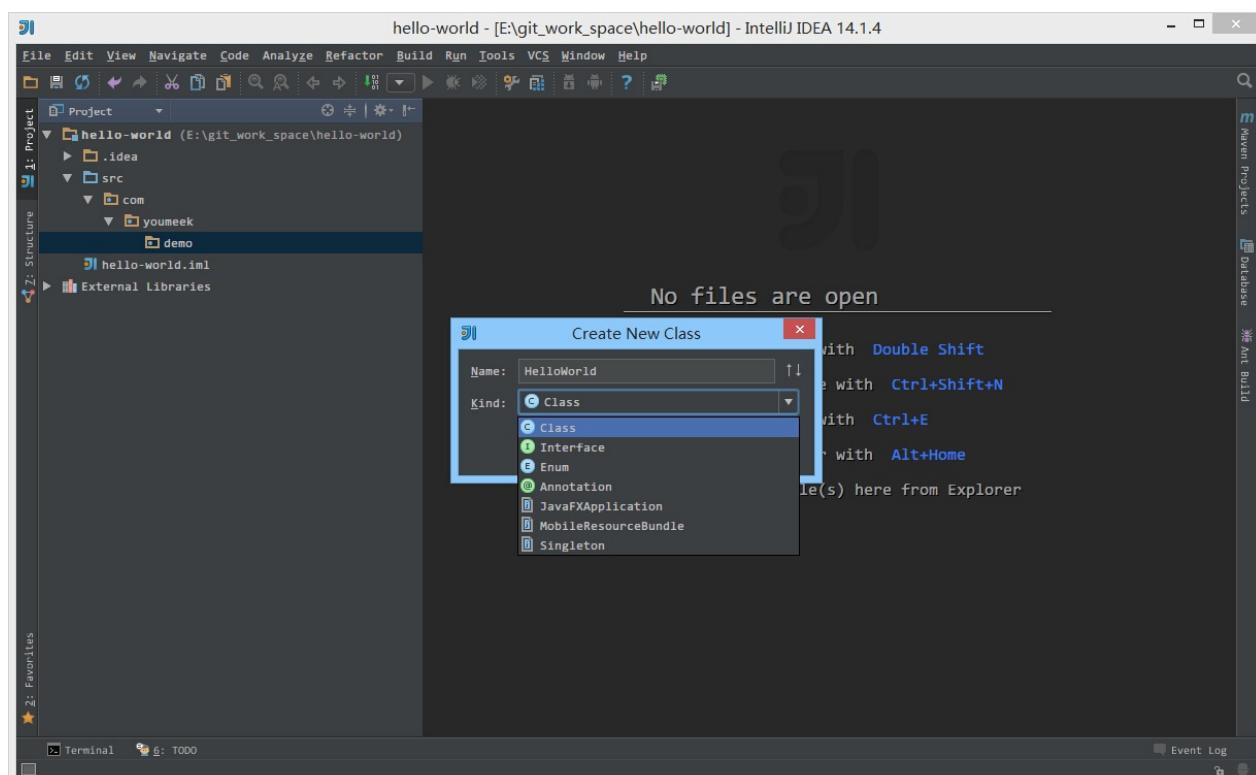
- 如上图标注 1 所示，根据《常见文件类型的图标介绍》章节我们知道，`src` 目录为蓝色表示 `Source root`，我们可以在此目录下创建包和类。
- 如上图标注 2 所示，由于该项目使用的是 `JDK 7`，所以项目是基于 `JDK 7`，我们可以调用 `JDK 7` 中的类。



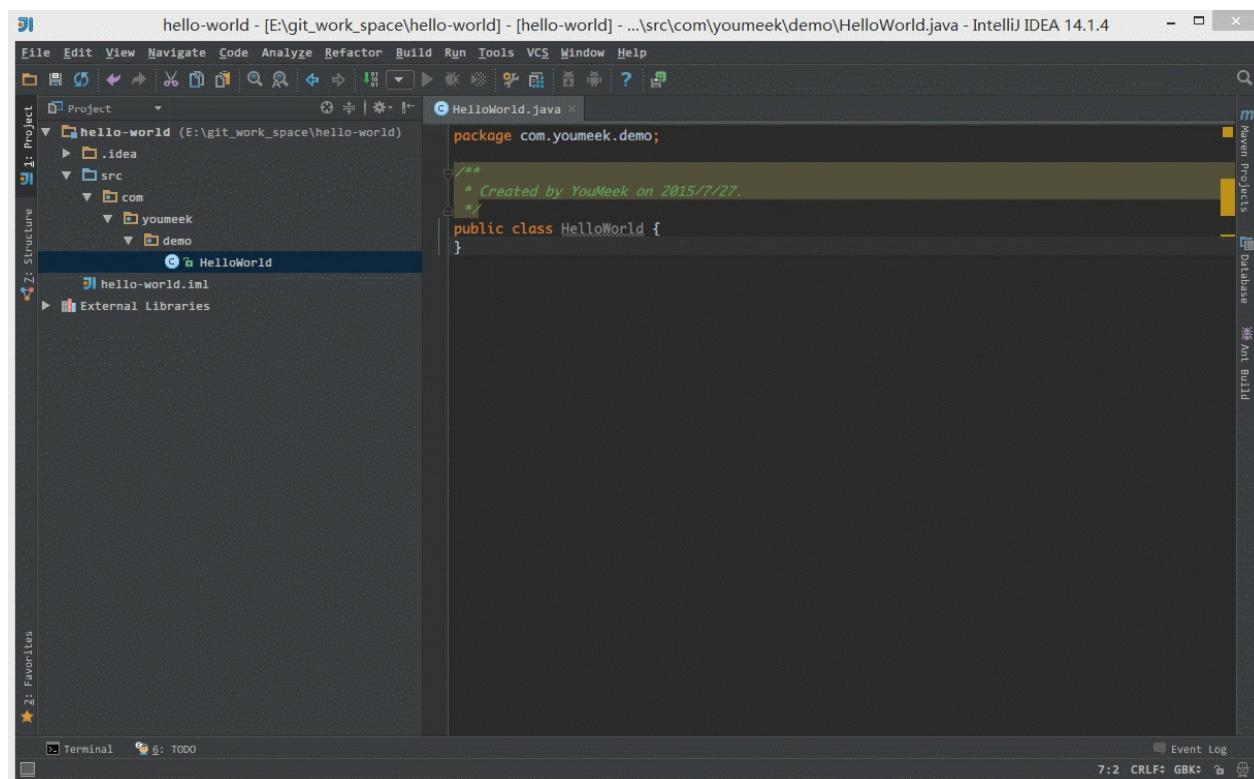
- 如上图标注 1, 2 所示，在 `src` 目录右键，选择 `New` 创建包目录。



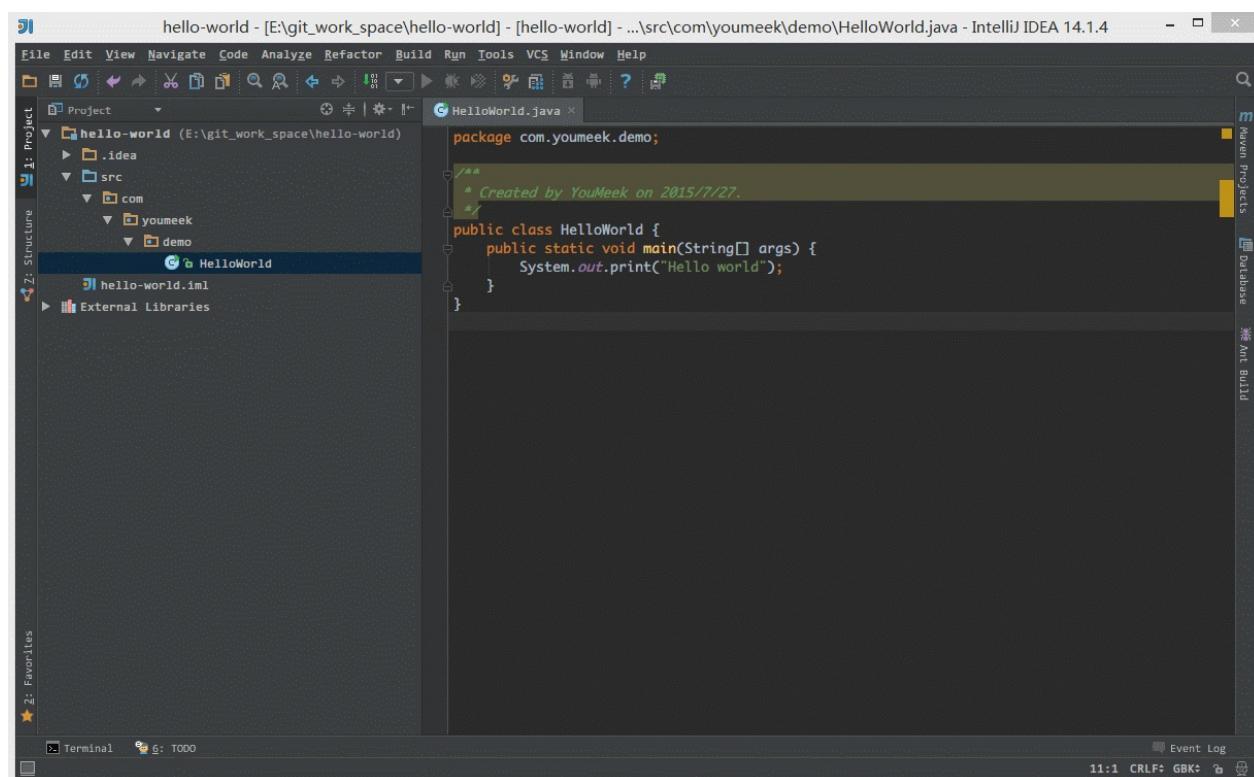
- 如上图标注 1 所示，在没有文件的情况下包目录默认是连在一起的，这不方便看目录层级关系。
- 如上图标注 箭头 所示，点击此齿轮，在弹出的菜单中去掉选择标注 2 选项：Compact Empty Middle Packages。



- 如上图所示，在包下可以直接创建 Class、Interface、Enum、Annotation 等常见类型文件。

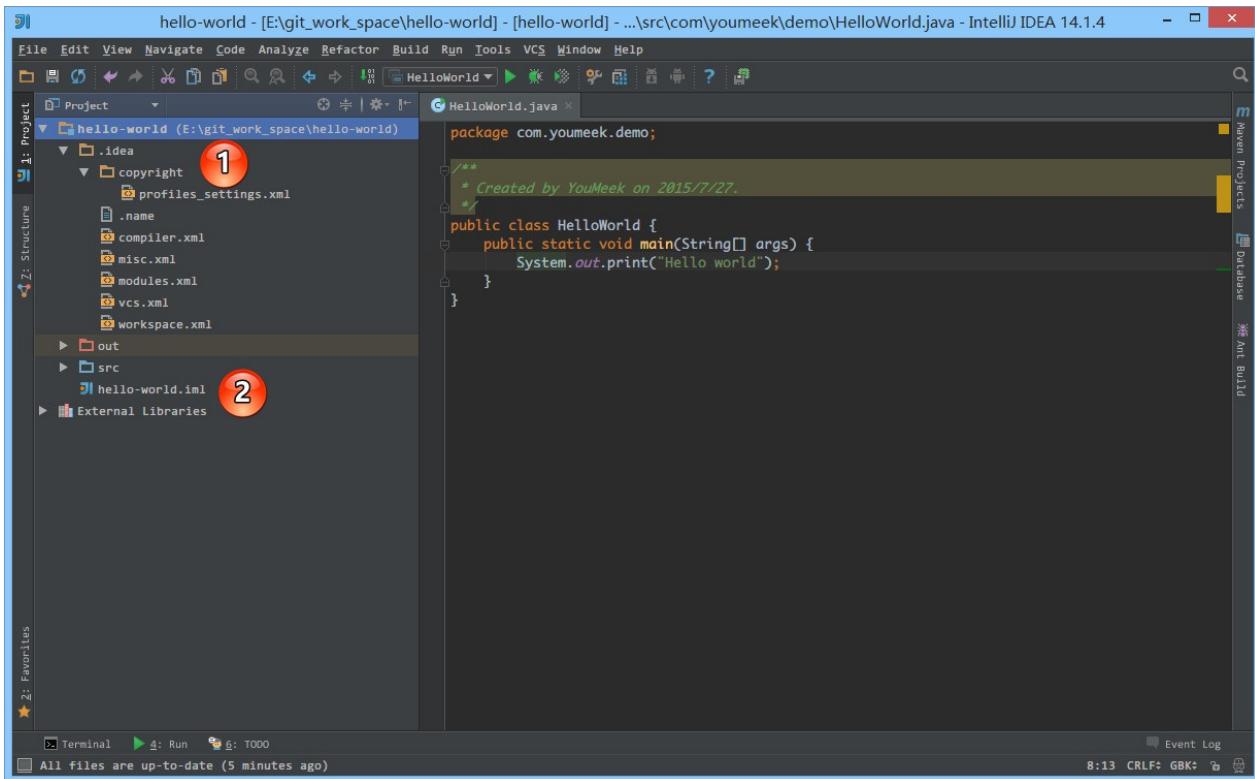


- 如上图 Gif 演示，写一个 main 方法，打印输出 Hello world。



- 如上图 Gif 演示，运行 main 方法，打印输出 Hello world。

项目配置文件介绍



- 如上图标注 1 所示，.idea 即为 Project 的配置文件目录。
- 如上图标注 2 所示，.iml 即为 Module 的配置文件。
- 通过上面的了解我们也知道 IntelliJ IDEA 项目的配置变动都是以这些 XML 文件的方式来表现的，所以我们也可以通过了解这些 XML 文件来了解 IntelliJ IDEA 的一些配置。也因为此特性，所以如果在项目协同中，我们要保证所有的项目配置一致，就可以考虑把这些配置文件上传到版本控制中（包括.idea 目录和 .iml 文件）。如果把这些文件加入到版本控制之后，那又有一点是需要考虑的，那就是协同者 Checkout 项目下来之后，按自己的需求进行项目配置的之后，项目的 XML 文件也会跟着变化。此时协同者的这些变化的文件就不应该再上传到版本控制中。至于如何更好地控制这些不想随时提交的文件，在接下来的版本控制专讲中会进行详细讲解。

特别介绍

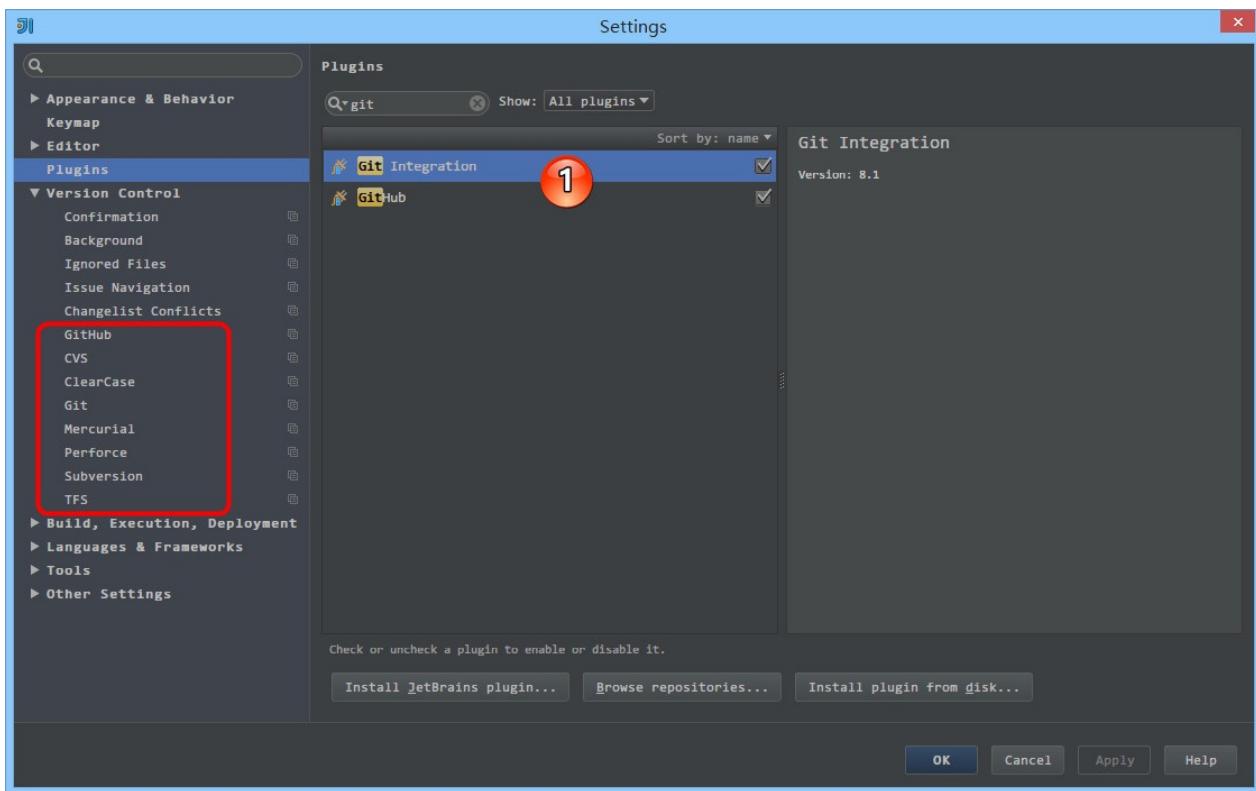
- IntelliJ IDEA 是一个没有 **Ctrl + S** 的 IDE，所以每次修改完代码你只要管着运行或者调试即可，无需担心保存或者丢失代码。
- 现在，放心、大胆地开始你的 Hello World。

版本控制的使用

IntelliJ IDEA 下的版本控制介绍

这一章节放在这么靠前位置来讲是因为版本控制在我心目中的地位比后面的实战知识点都来得重要。不管是个人开发或是团队开发，版本控制都是可以很好地被使用的，目前我找不到任何开发者不使用版本控制的理由。而且对于 IDE 来讲，集成版本控制的本身就是它最大的亮点之一，很多开发者也是为此而使用它。

在本章节中也会对 IntelliJ IDEA 的相关版本控制进行了介绍，会开始涉及到一些 IntelliJ IDEA 人性化设置，也希望你能从这一讲开始认识到 IntelliJ IDEA 的优雅。



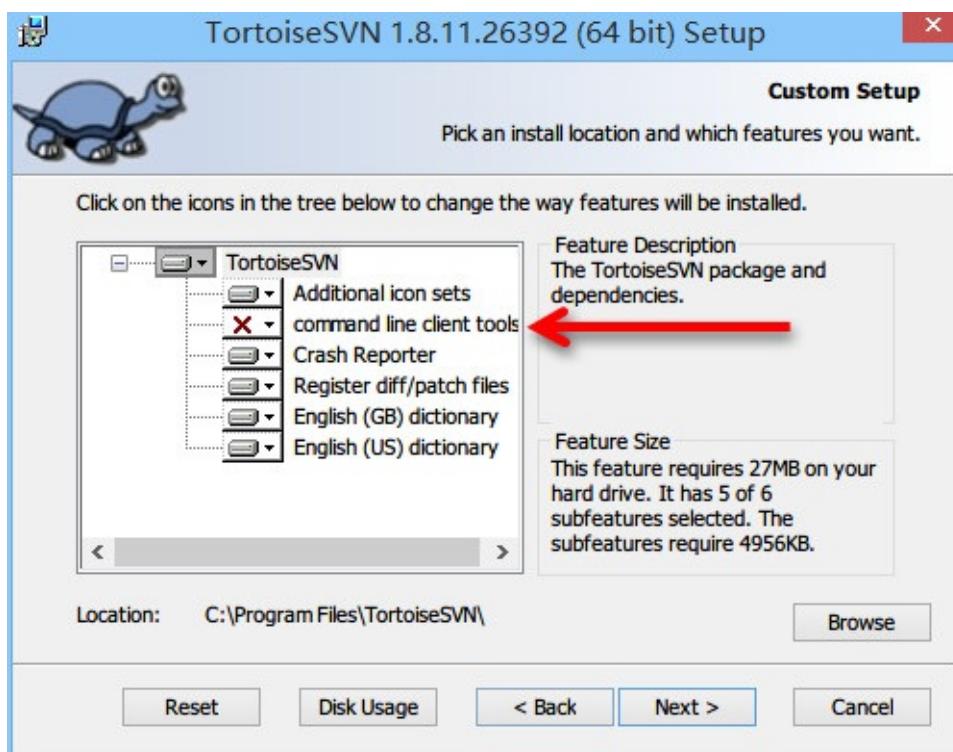
- 很多人认为 IntelliJ IDEA 自带了 SVN 或是 Git 等版本控制工具，认为只要安装了 IntelliJ IDEA 就可以完全使用版本控制应有的功能。这完全是一种错误的解读，IntelliJ IDEA 是自带对这些版本控制工具的支持插件，但是该装什么版本控制客户端还是要照样装的。
- 如上图标注 1 所示，IntelliJ IDEA 对版本控制的支持是以插件化的方式来实现的。旗舰版默认支持目前主流的版本控制软件：CVS、Subversion（SVN）、Git、ClearCase、Mercurial、Perforce、TFS。又因为目前太多人使用 Github 进行协同或是项目版本管理，所以 IntelliJ IDEA 同时自带了 Github 插件，方便 Checkout 和管理你的 Github 项目。

SVN 的配置

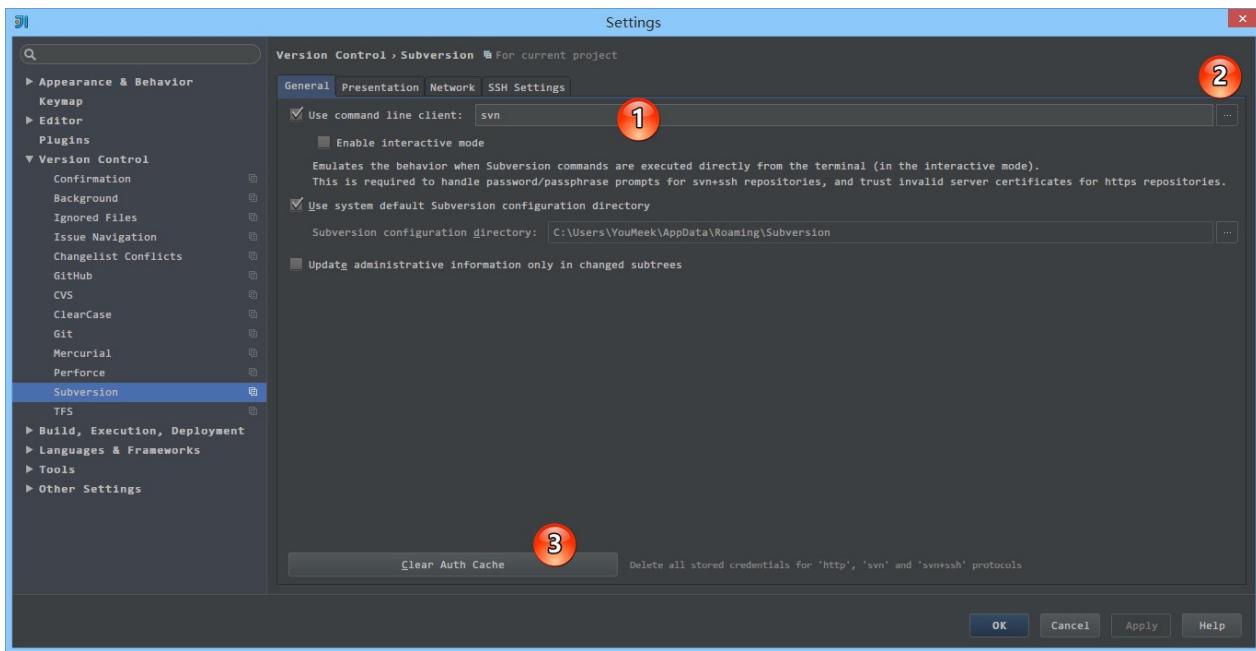
要在 IntelliJ IDEA 中使用 SVN，需要先安装 SVN 客户端或是 TortoiseSVN 这类图形化工具，Windows 系统这里推荐安装 TortoiseSVN，即使在不使用 IntelliJ IDEA 也可以方便管理我们的项目。

SVN 主要使用的版本有 1.6、1.7、1.8，最新的 1.9。推荐大家使用 1.8 的。如果你的项目使用的是 1.6 的版本，在安装 1.8 之后是可以直接对项目文件进行升级的，所以无需担心，也因此更加推荐大家使用 1.8。

- Subversion 官网下载：<https://subversion.apache.org/download/#recommended-release>
- TortoiseSVN 官网下载：<http://tortoisessvn.net/downloads.zh.html>



- 如上图箭头所示，在安装 TortoiseSVN 的时候，默认 command line client tools 是不安装的，这里建议勾选上。



- 如上图标注 1 所示，勾选 Use command line client
- 如上图标注 2 所示，建议 svn 的路径自己根据安装后的路径进行选择，不然有时候 IntelliJ IDEA 无法识别到会报： Cannot run program "svn" 这类错误。
- 如上图标注 3 所示，当使用一段时间 SVN 以后，发现各种 SVN 相关问题无法解决，可以考虑点击此按钮进行清除一下缓存。

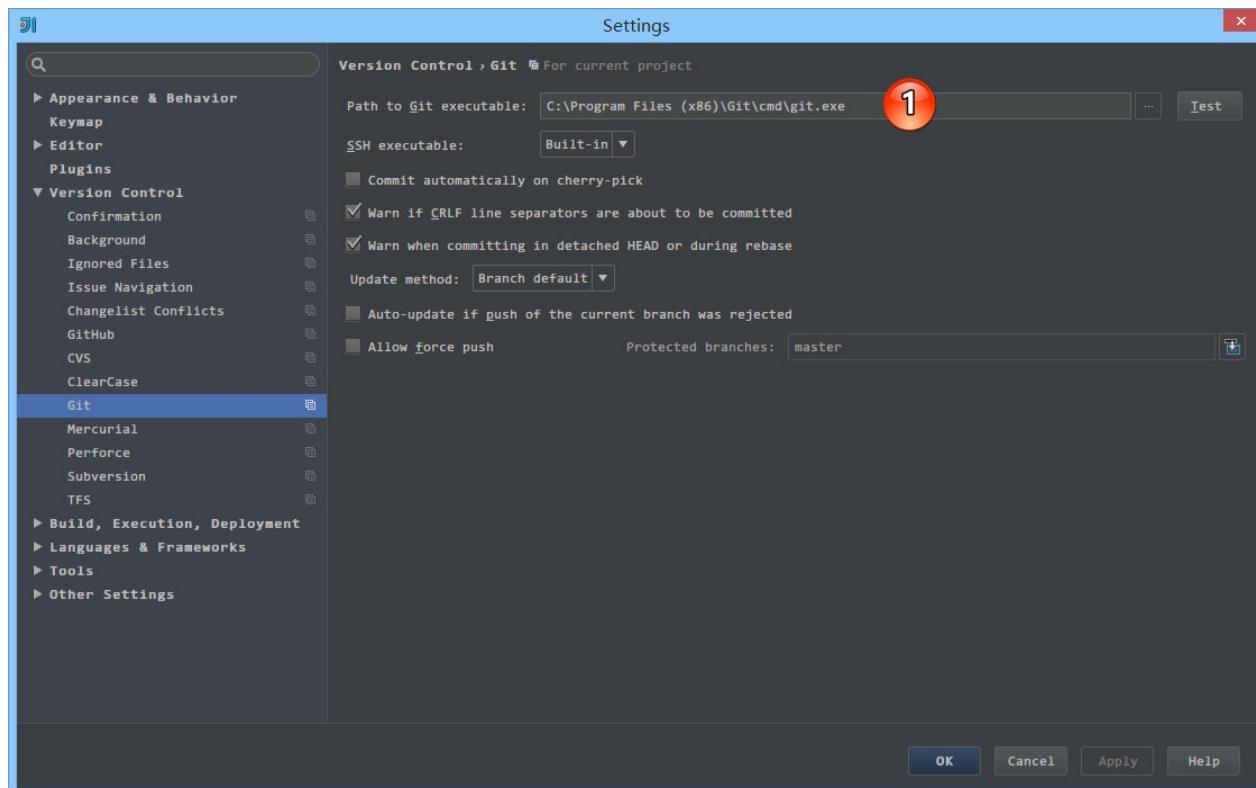
根据目前的使用经验来看，IntelliJ IDEA 下 SVN 的使用经历并不算愉快，至少比 Git 不好用很多，经常遇到很多问题，所以这里也算是先给大家提个醒。如果紧急情况下 IntelliJ IDEA 无法更新、提交的时候，要记得使用 TortoiseSVN 来操作。

Git 的配置

要在 IntelliJ IDEA 中使用 Git，需要先安装 Git 客户端，这里推荐安装官网版本。

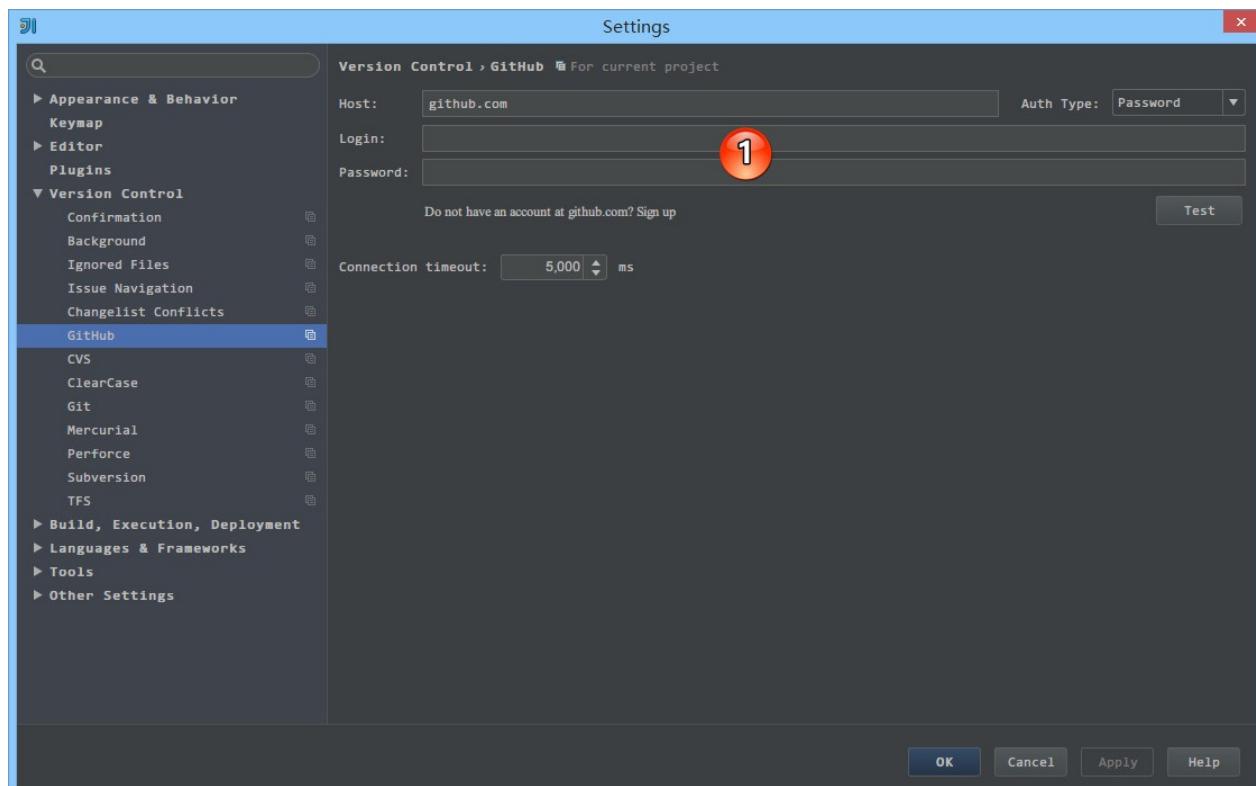
Git 主要的版本有 1.X、2.X，最新的是 2.X，使用版本随意，但是不要太新了，不然可能 IntelliJ IDEA 小旧版本会无法支持可能。

- Git 官网下载：<http://git-scm.com/>
- TortoiseGit 官网下载：<http://download.tortoisegit.org/tgit/>



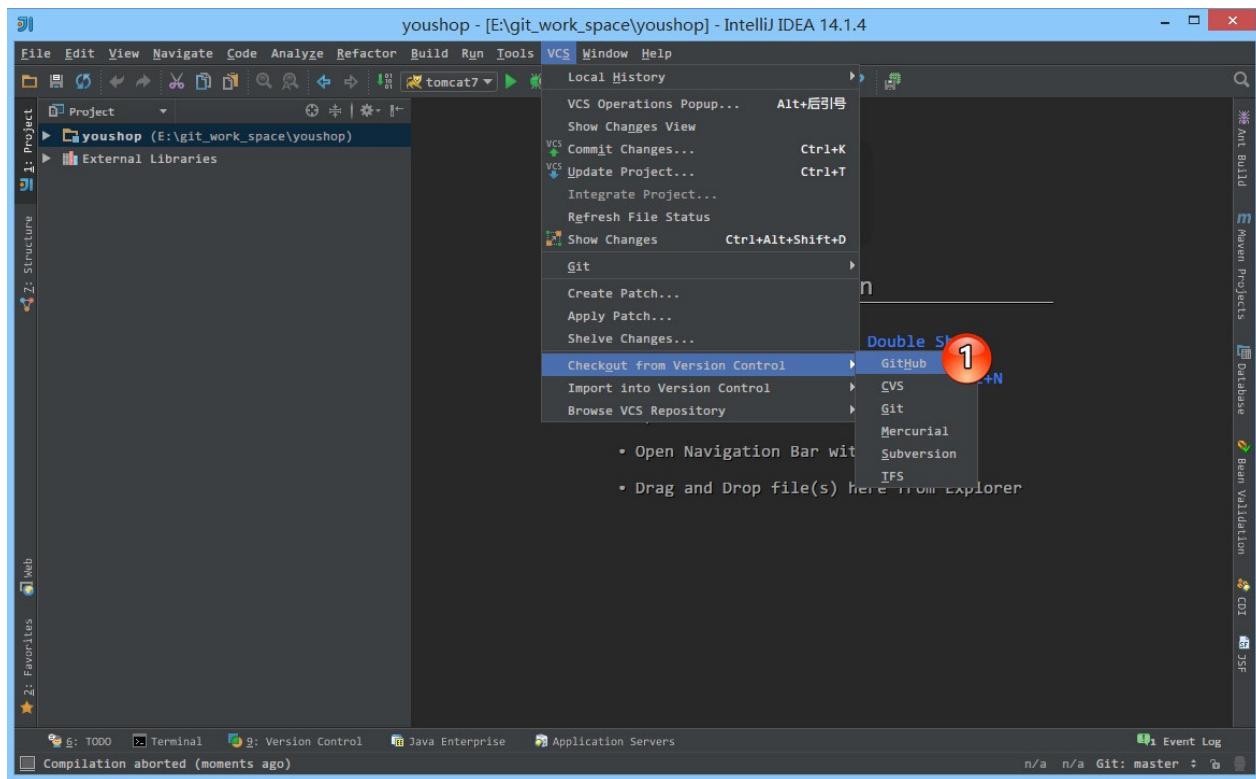
如上图标注 1 所示，确定好该路径下是否有对应的可执行文件。

Github 的配置和使用

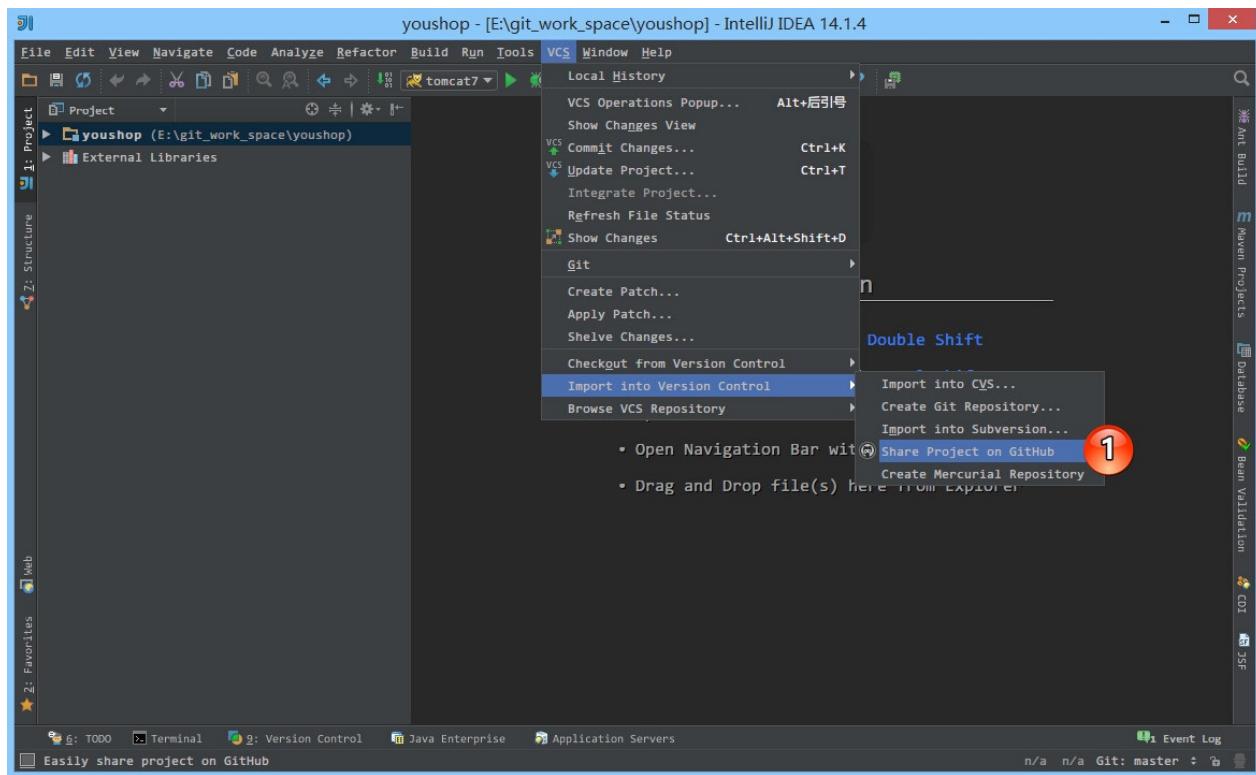


- 如上图标注 1 所示，填写你的 Github 登录账号和密码，点击 **Test** 可以进行测试是否可以正确连上。

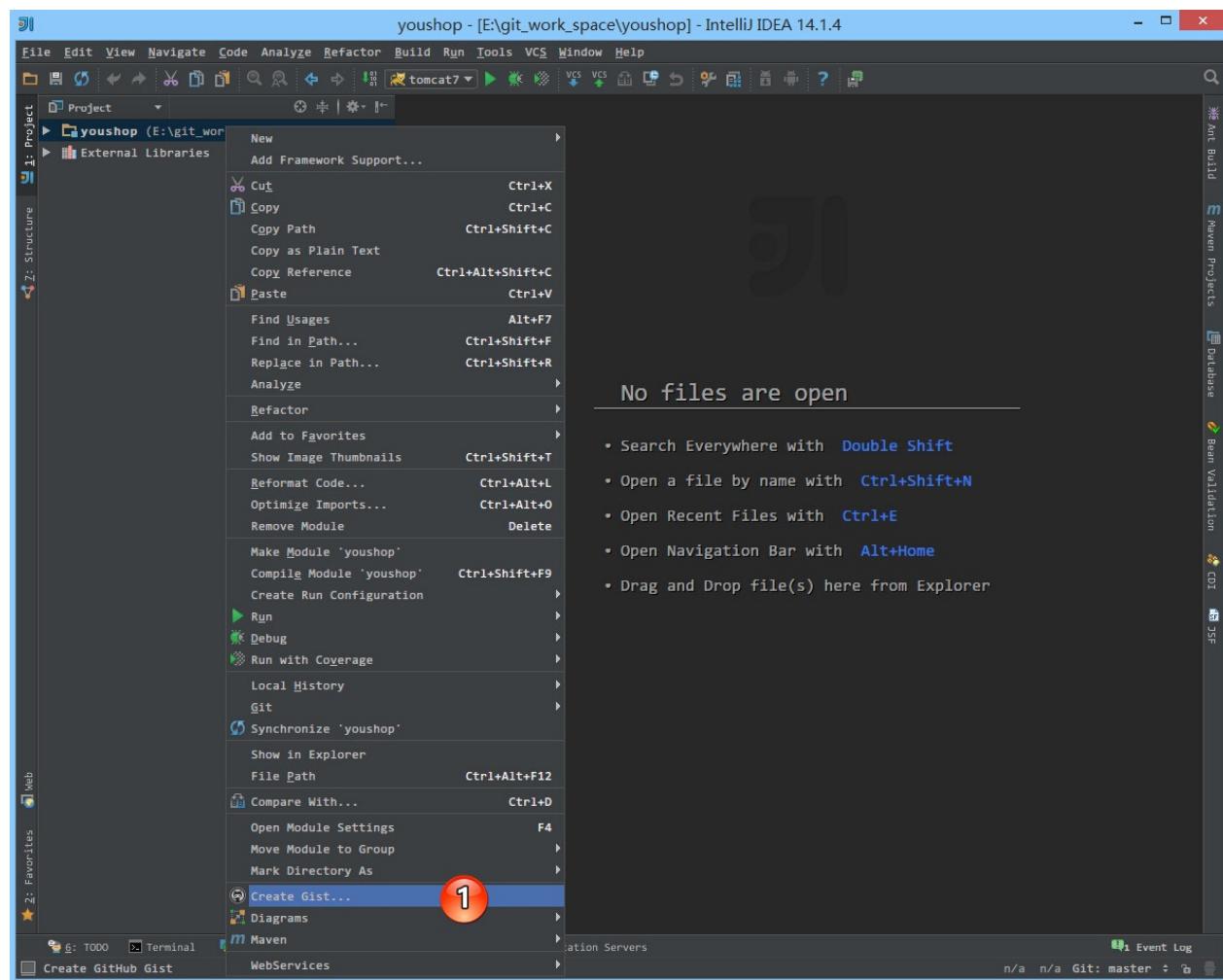
16. 版本控制讲解



- 如上图标注 1 所示，支持直接从你当前登录的 Github 账号上 Checkout 项目。

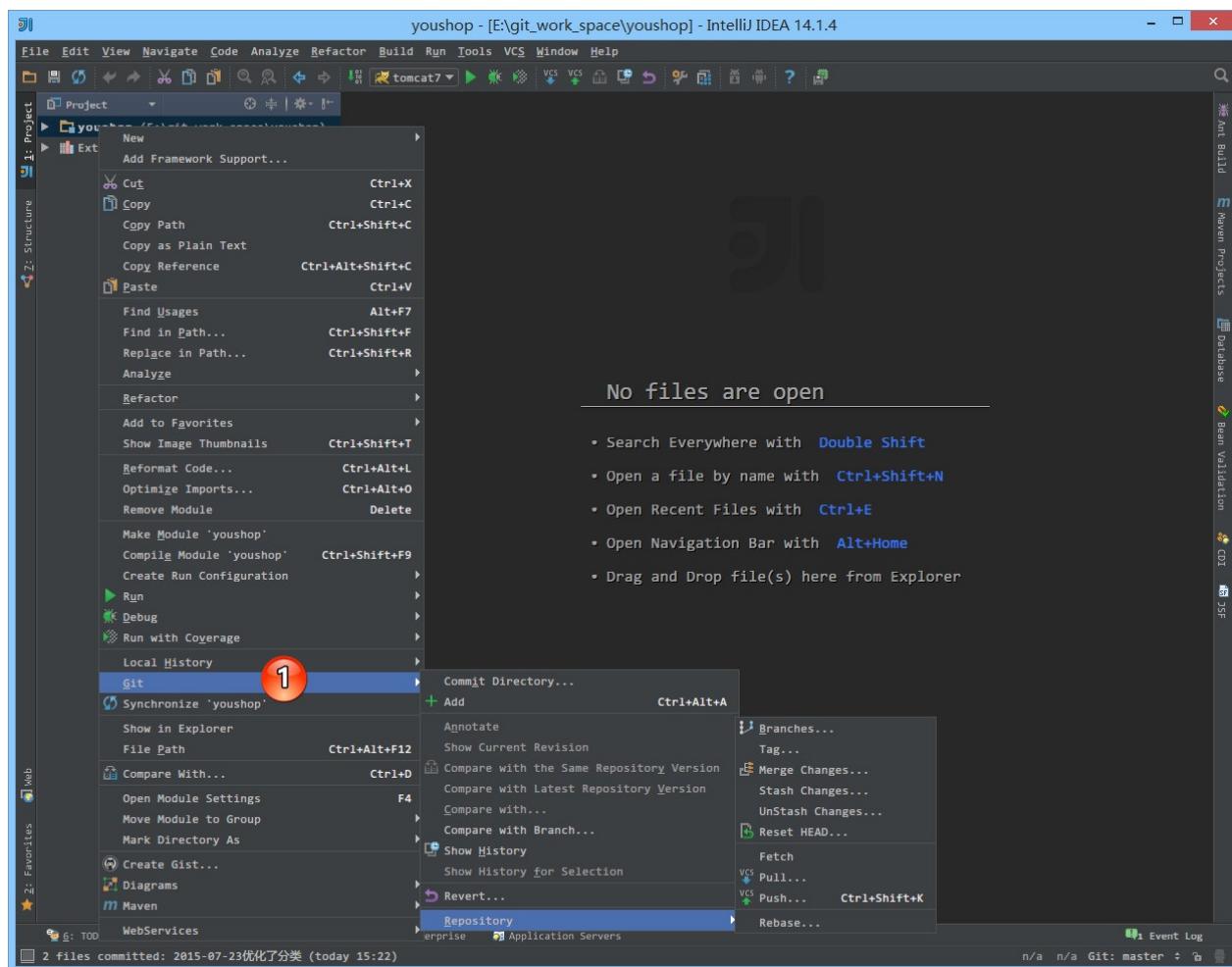


- 如上图标注 1 所示，支持把当前本地项目分享到你的 Github 账号上。



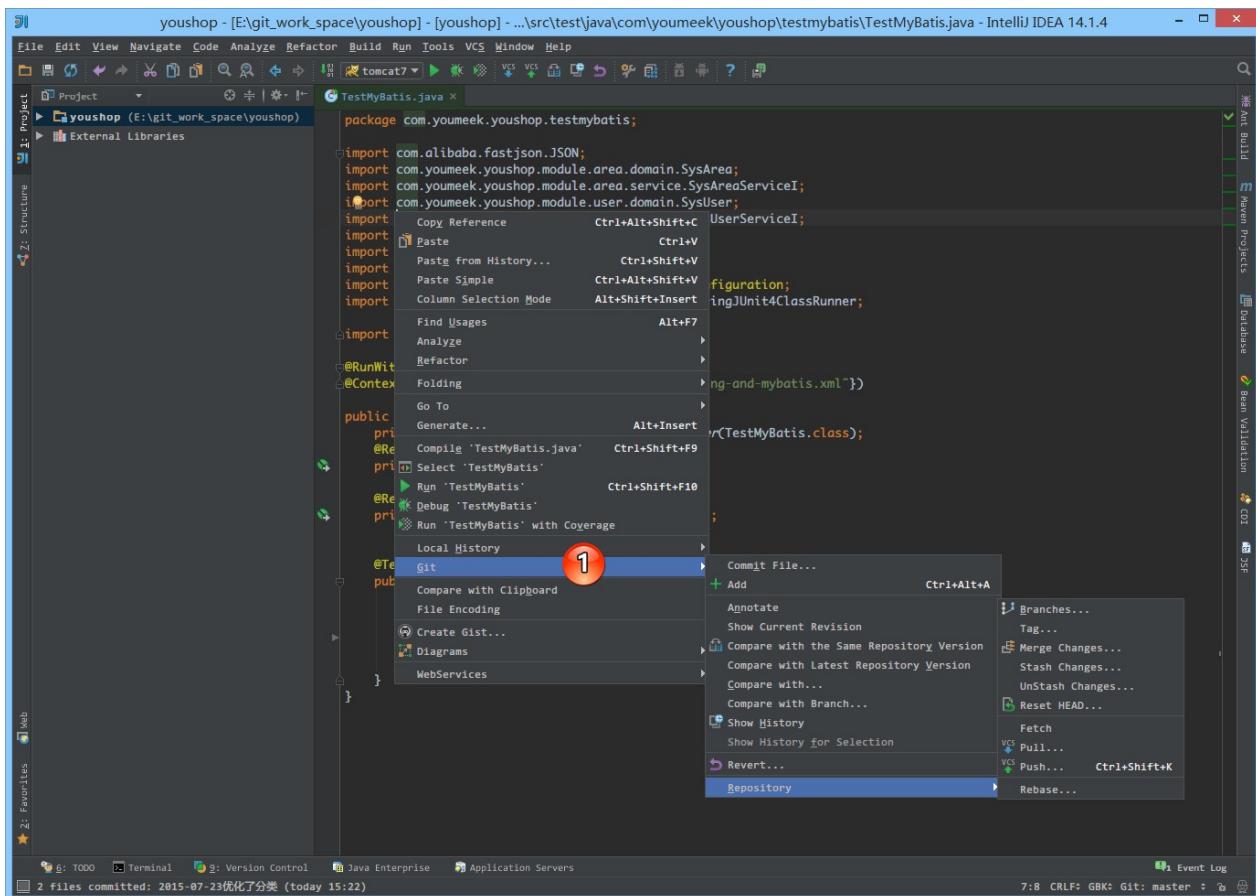
- 如上图标注 1 所示，支持创建 Gist。Github 的 Gist 官网地址：<https://gist.github.com/>

版本控制主要操作按钮

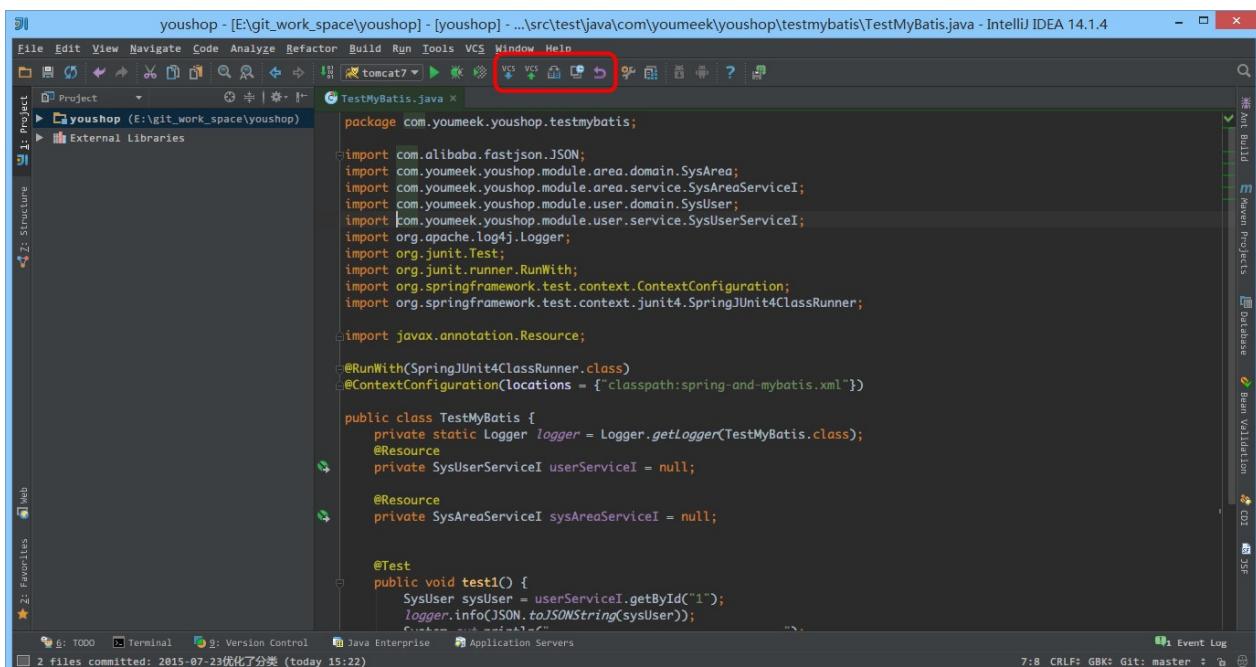


- 如上图标注 1 所示，对目录进行右键弹出的菜单选项。

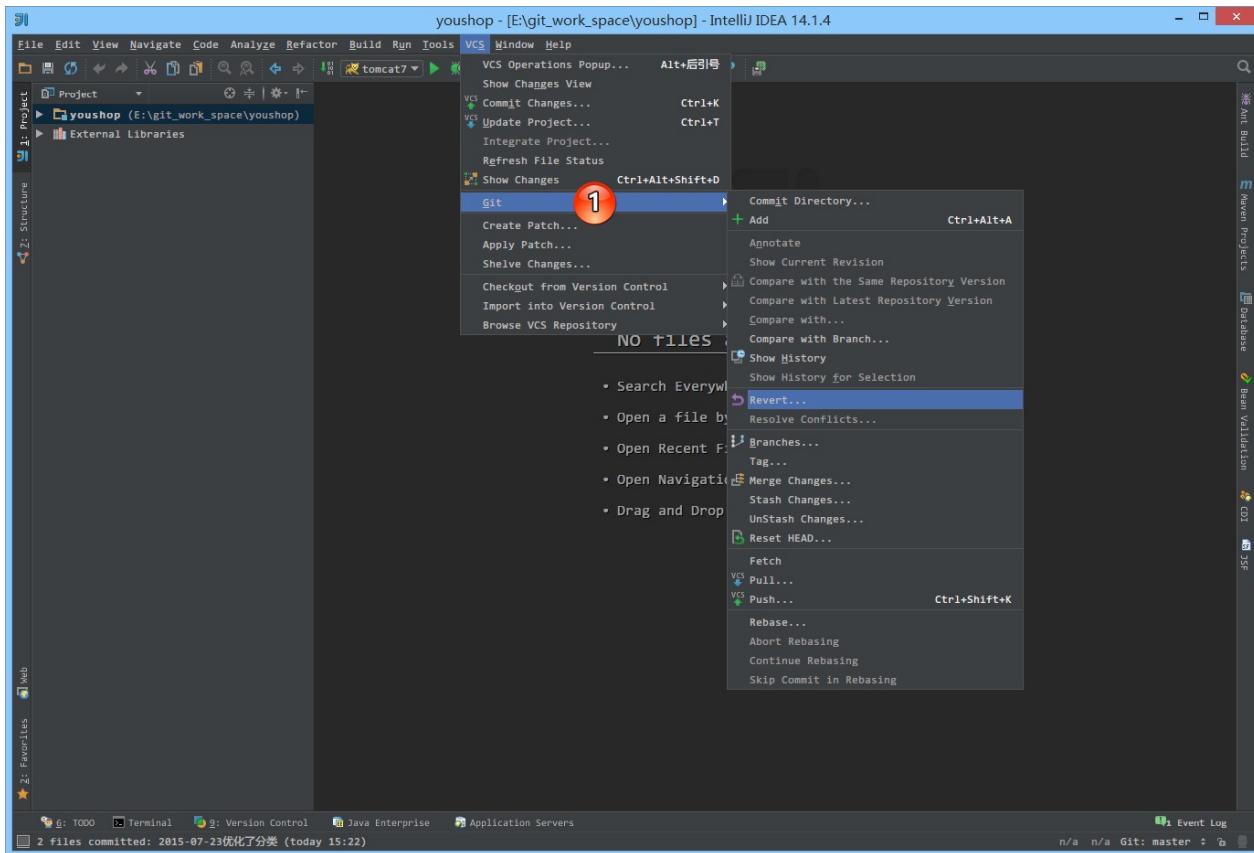
16. 版本控制讲解



- 如上图标注 1 所示，对文件进行右键弹出的菜单选项。

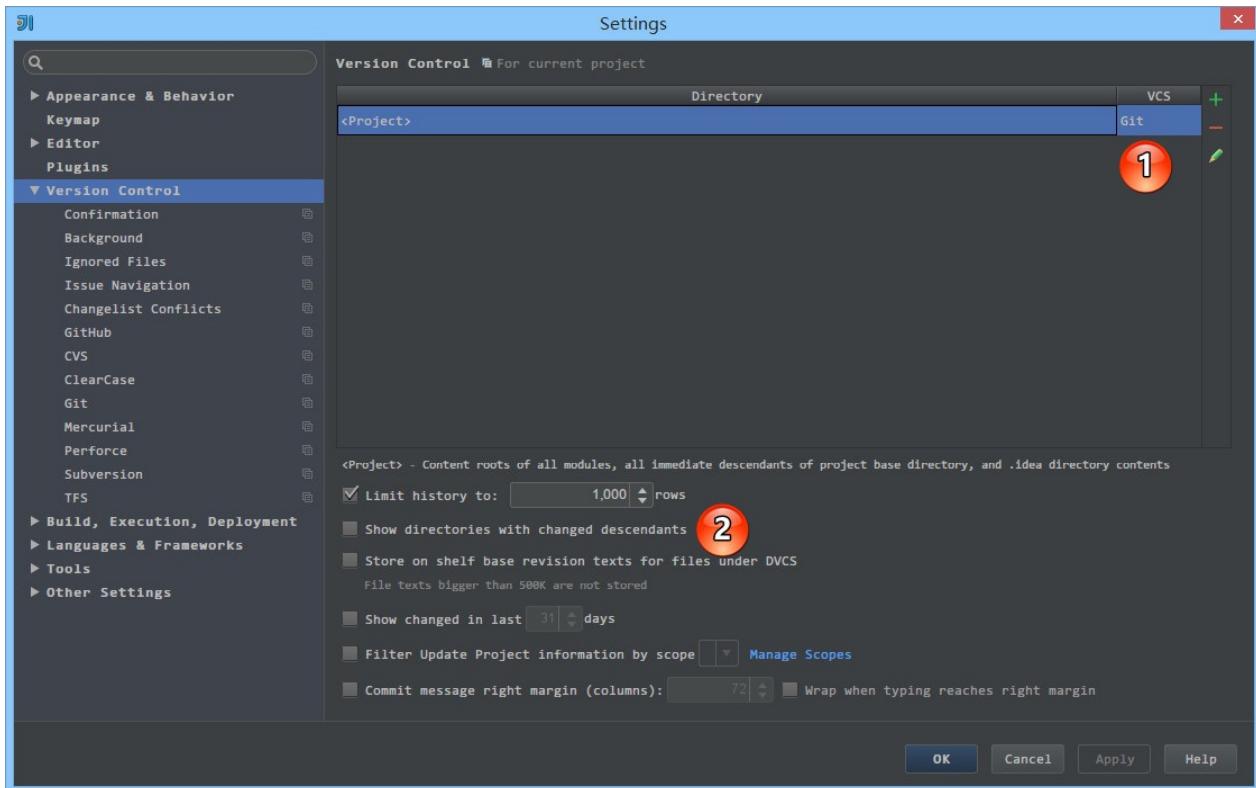


- 如上图标注红圈所示，为工具栏上版本控制操作按钮，基本上大家也都是使用这里进行操作。
- 第一个按钮：Update Project 更新项目。
- 第二个按钮：Commit changes 提交项目上所有变化文件。点击这个按钮不会立马提交所有文件，而是先弹出一个被修改文件的一个汇总框，具体操作下面会有图片进行专门介绍。
- 第三个按钮：Compare with the Same Repository Version 当前文件与服务器上该文件通版本的内容进行比较。如果当前编辑的文件没有修改，则是灰色不可点击。
- 第四个按钮：Show history 显示当前文件的历史记录。
- 第五个按钮：Revert 还原当前被修改的文件到违背修改的版本状态下。如果当前编辑的文件没有修改，则是灰色不可点击。

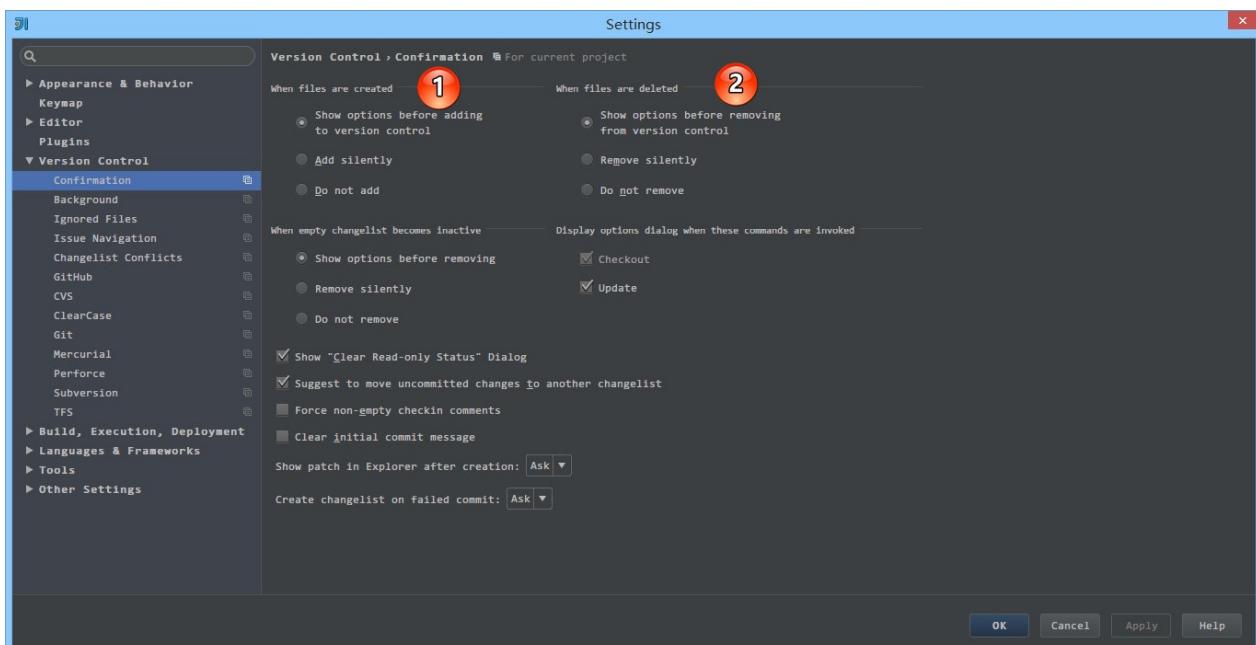


- 如上图标注1所示，菜单栏上的版本控制操作区。

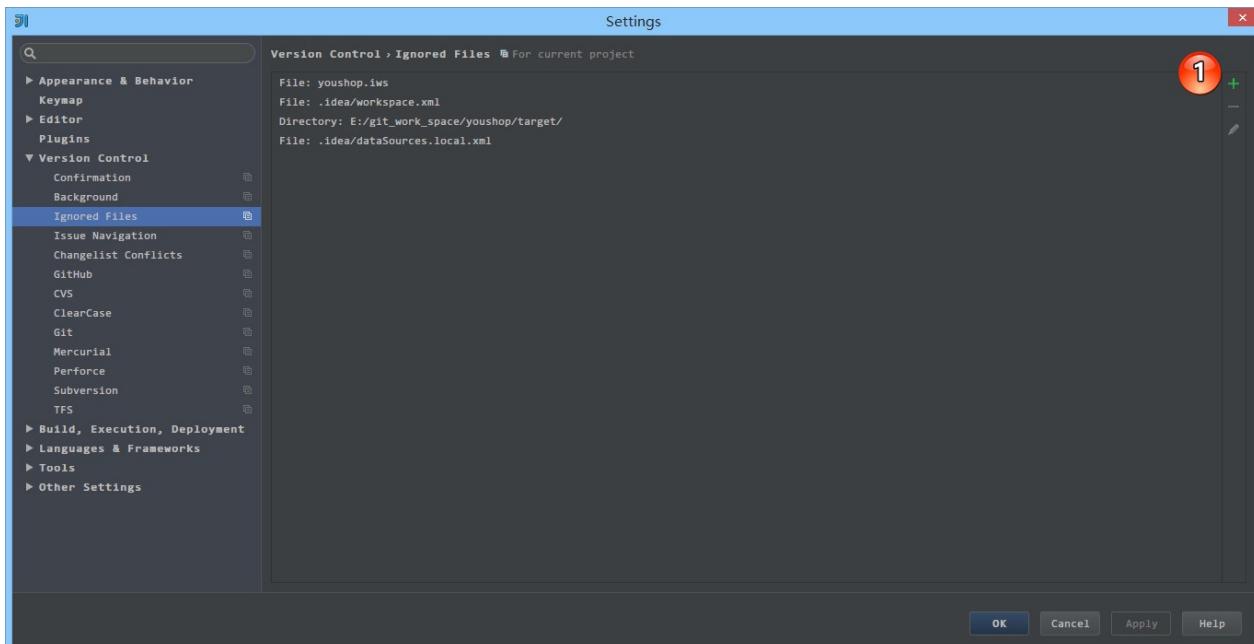
版本控制相关的常用设置说明



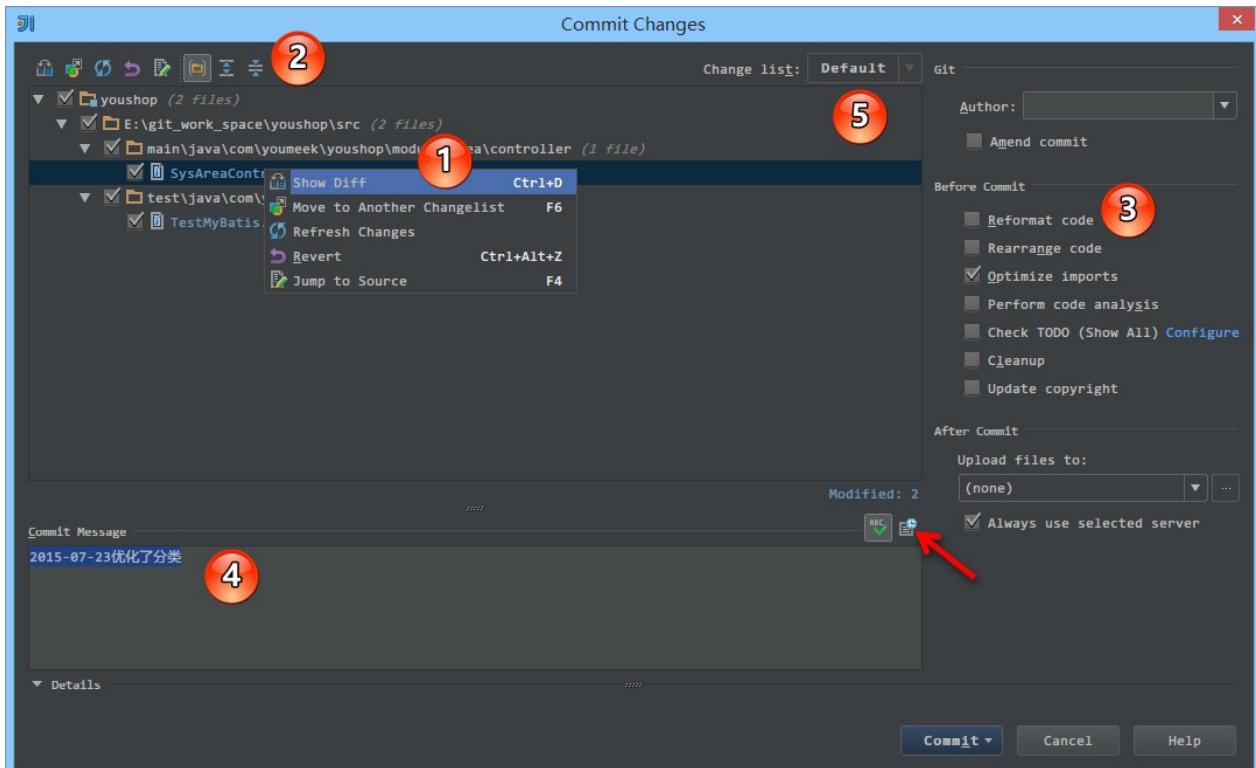
- 如上图标注 1 所示，当前项目使用的版本控制是 Git。如果你不愿意这个项目继续使用版本控制可以点击旁边的减号按钮，如果你要切换版本控制，可以点击 Git，会出现 IntelliJ IDEA 支持的各种版本控制选择列表，但是我们一般情况下一个项目不会有多个版本控制的。
- 如上图标注 2 所示，Show directories with changed descendants 表示子目录有文件被修改了，则该文件的所有上层目录都显示版本控制被概念的颜色。默认是不勾选的，我一般建议勾选此功能。



- 如上图标注 1 所示，When files are created 表示当有新文件放进项目中的时候 IntelliJ IDEA 做如何处理，默认是 Show options before adding to version control 表示弹出提示选项，让开发者决定这些新文件是加入到版本控制中还是不加入。如果不弹出提示，则选择下面两个选项进行默认操作。
- 如上图标注 2 所示，When files are deleted 表示当有新文件在项目中被删除的时候 IntelliJ IDEA 做如何处理，默认是 Show options before removing from version control 表示弹出提示选项，让开发者决定这些被删除的是否从版本控制中删除。如果不弹出提示，则选择下面两个选项进行默认操作。

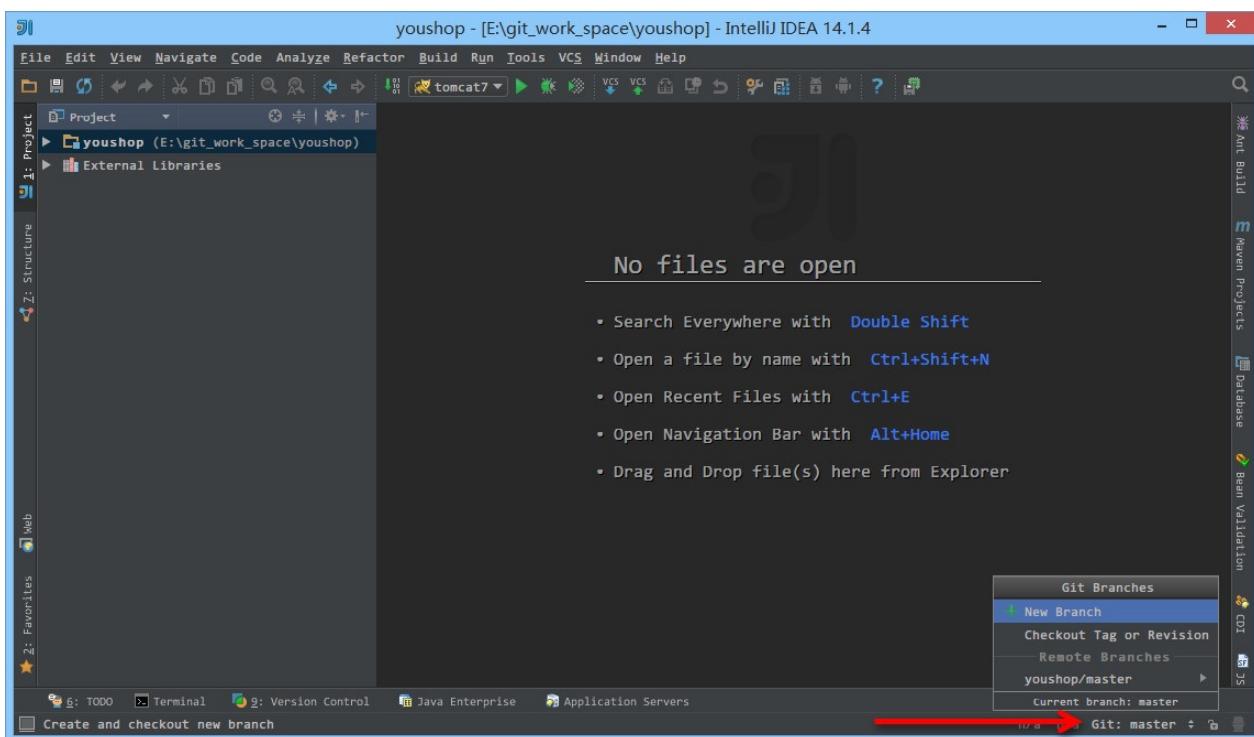


- 如上图标注 1 所示，对于不想加入到版本控制的文件，可以添加到忽略列表中。但是如果已经加入到版本控制的文件使用此功能，则表示该文件或目录无法再使用版本控制相关的操作，比如提交、更新等。我个人使用过程中发现在 SVN 上此功能不太好用，Git 上是可以用的。



- 上图所示的弹出层就是本文上面说的 `Commit Changes` 点击后弹出的变动文件汇总弹出层。
- 如上图标注 1 所示，可以在文件上右键进行操作。
 - `Show Diff` 当前文件与服务器上该文件通版本的内容进行比较。
 - `Move to Another Changelist` 将选中的文件转移到其他的 `Change list` 中。`Change list` 是一个重要的概念，这里需要进行重点说明。很多时候，我们开发一个项目同时并发的任务可能有很多，每个任务涉及到的文件可能都是基于业务来讲的。所以就会存在一个这样的情况：我改了 30 个文件，其中 15 个文件是属于订单问题，剩下 15 个是会员问题，那我希望提交代码的时候是根据业务区分这些文件的，这样我填写 `Commit Message` 是好描述的，同时在文件多的情况下，我也好区分这些要提交的文件业务模块。所以我一般会把属于订单的 15 个文件转移到其他的 `Change list` 中，先把专注点集中在 15 个会员问题的文件，先提交会员问题的 `Change list`，然后在提交订单会员的 `Change list`。我个人还有一种用法是把一些文件暂时不提交的文件转移到一个我指定的 `Change list`，等后面我觉得有必要提交了，再做提交操作，这样这些文件就不会干扰我当前修改的文件提交。总结下 `Change list` 的功能就是为了让你更好地管理你的版本控制文件，让你的专注点得到更好的集中，从而提供效率。
 - `Jump to Source` 打开并跳转到被选中。
 - 如上图标注 2 所示，可以根据工具栏按钮进行操作，操作的对象会鼠标选中的文件，多选可以按 `Ctrl` 后不放，需要注意的是这个更前面的复选框是没有多大关系的。

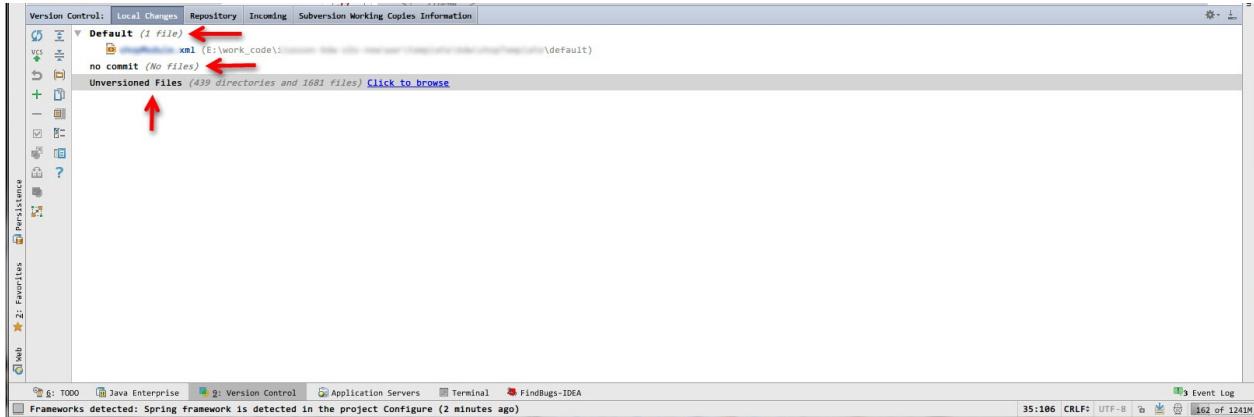
- 如上图标注 3 所示，可以在提交前自动对被提交的文件进行一些操作事件（该项目使用的 Git，使用其他版本控制可能有些按钮有差异。）：
- `Reformat code` 格式化代码，如果是 Web 开发建议不要勾选，因为格式化 JSP 类文件，格式化效果不好。如果都是 Java 类则可以安心格式化。
- `Rearrange code` 重新编排代码，IntelliJ IDEA 支持各种复杂的编排设置选项，这个会在后面说。设置好了编码功能之后，这里就可以尝试勾选这个进行自动编排。
- `Optimize imports` 优化导入包，会在自动去掉没有使用的包。这个建议都勾选，这个只对 Java 类有作用，所以不用担心有副作用。
- `Perform code analysis` 进行代码分析，这个建议不用在提交的时候处理，而是在开发完之后，要专门养成对代码进行分析的习惯。IntelliJ IDEA 集成了代码分析功能。
- `Check TODO` 检查代码中的 `TODO`。`TODO` 功能后面也会有章节进行讲解，这里简单介绍：这是一个记录待办事项的功能。
- `Cleanup` 清除下版本控制系统，去掉一些版本控制系统的错误信息，建议勾选。
- 如上图标注 4 所示，填写提交的信息。
- 如上图标注 5 所示，`Change list` 改变列表，这是一个下拉选项，说明我们可以切换不同的 `Change list`，提交不同的 `change list` 文件。
- 如上图标注箭头所示，我们可以查看我们提交历史中使用的 `Commit Message`，有些时候，我们做的是同一个任务，但是需要提交多次，为了更好管理项目，建议是提交的 `Message` 是保持一致的。



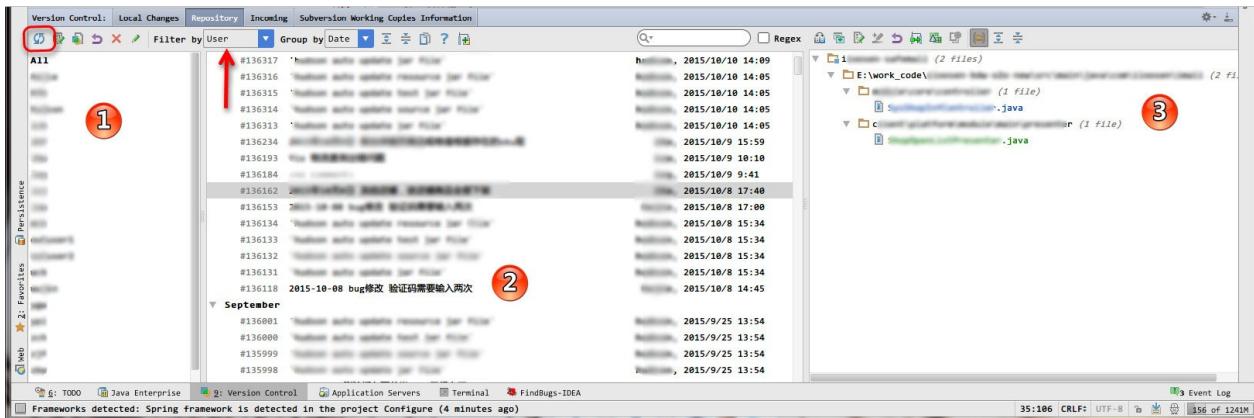
- 如上图标注箭头所示，如果你使用的 Git，点击此位置可以切换分支和创建分支，以及合并、删除分支等操作。

SVN 的使用

SVN 的这个窗口有的 IntelliJ IDEA 上叫 `changes`，有的叫 `version control`，具体是什么原因引起这样的差异，我暂时还不清楚。但是不管叫法如何里面的结构是一样的，所以对使用者来讲没多大影响，但是你需要知道他们其实是一样的功能即可。



上图 `Local Changes` 这个 Tab 表示当前项目的 SVN 中各个文件的总的情况预览。这里的 `Default` 是 IntelliJ IDEA 的默认 `change list` 名称，`no commit` 是我自己创建的一个 `change list`，我个人有一个习惯是把一些暂时不需要提交的先放这个 `list` 里面。`change list` 很常用而且重要，本文前面也有强调过了，所以一定好认真对待。`unversioned Files` 表示项目中未加入版本控制系统中的文件，你可以点击 `Click to browse`，会弹出一个弹出框列表显示这些未被加入的文件。



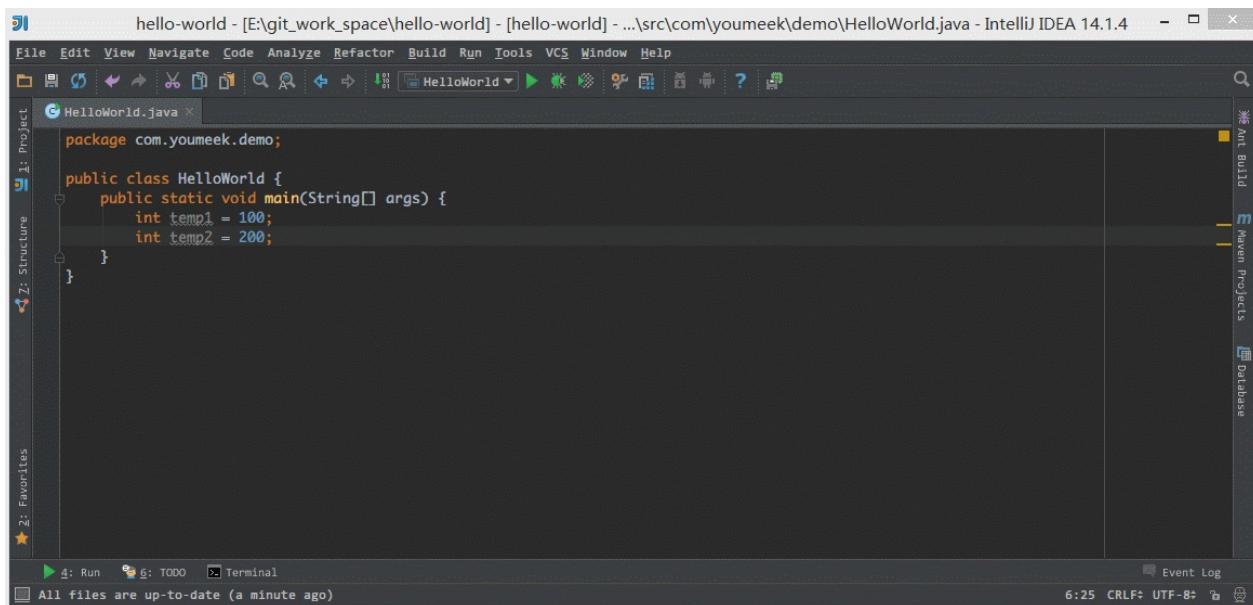
上图 `Repository` 这个 Tab 表示项目的 SVN 信息汇总，内容非常的详细，也是我平时用最多的地方。如果你点击这个 Tab 没看到数据，是因为你需要点击上图红圈这个刷新按钮。初次使用下默认的过滤条件不是我上图这样的，我习惯根据 `User` 进行过滤筛选，所以上图箭头中的 `Filter` 我是选择 `User`。选择之后，如上图标注 1 所示，显示了这个项目中参与提交的各个

用户名，选择一个用户之后，上图标注 2 所以会显示出该用户提交了哪些记录。选择标注 2 区域中的某个提交记录后，标注 3 显示对应的具体提交细节，我们可以对这些文件进行右键操作，具体操作内容跟本文上面提到的那些提交时的操作按钮差不多，这里不多讲。

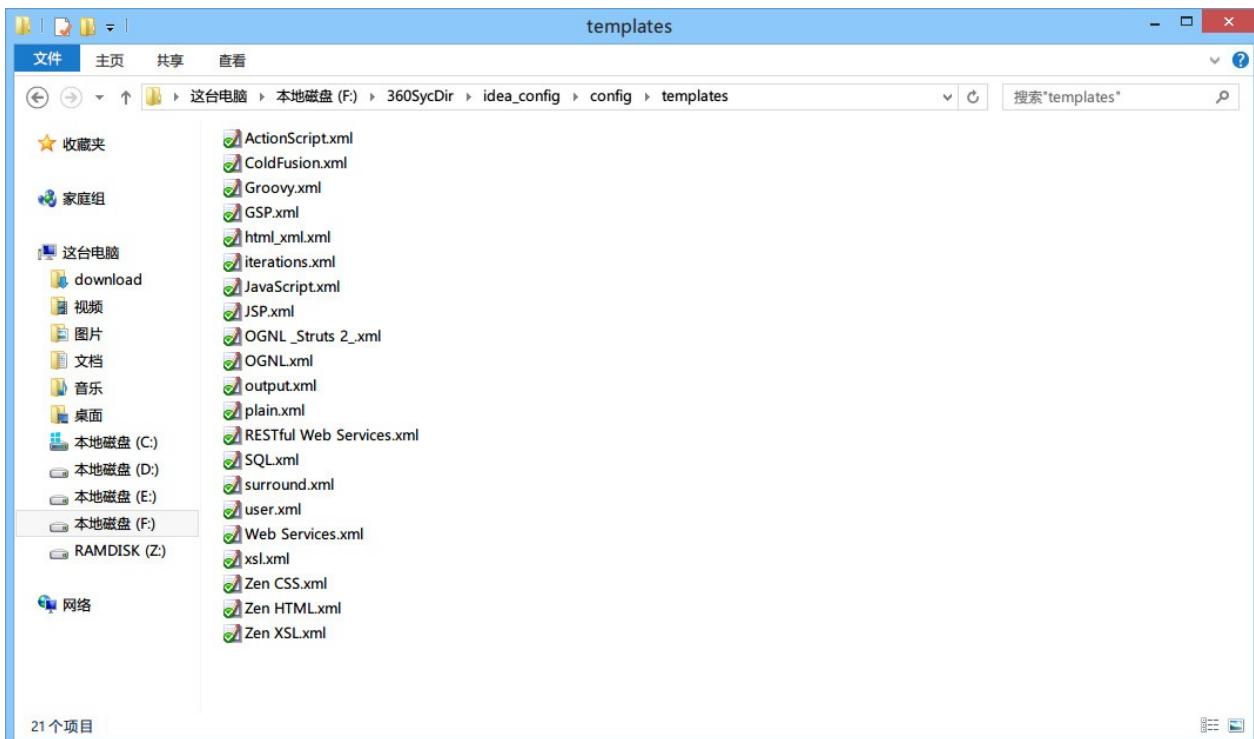
总的来说，SVN 这个功能用来管理和审查开发团队中人员的代码是非常好用的，所以非常非常建议你一定要学会该功能。

实时代码模板的使用

实时代码模板的介绍



- 上图 Gif 演示为最好的介绍 Live Templates。
- 实时代码模板需要字符串前缀，如 Gif 演示中，在输入 sys 后生成一段输出语句，其中 sys 前缀是我自己设置的。
- 实时代码模板支持变量参数设置，如 Gif 演示中，在输入 temp1 的时候，后面自动也生成了一个 temp1，这是因为两者的变量名是一致的，所以我设置了一个变量值内容之后，相同变量值的内容也会跟着出现。
- 实时代码模板支持最后位置的定位，如 Gif 演示中，在输入完 temp1 之后，按 Enter 后自动跳转到语句末。虽然默认没有设置的时候也是在这个位置，但是我是通过设置来定位到此位置的。
- 实时代码模板支持获取当前类名和当前方法名（本质是内置函数），如 Gif 演示中，在输入完 sys 生成输出语句中自动带有当前类名和方法名。
- 综上效果：实时代码模板只是为了让我们更加高效的写一些固定模式的代码，以提高编码效率，同时也可以增加个性化。比如 Gif 演示中，我在开发中如果需要写输出的话，我个人习惯输出语句中前缀是自己的标记：横线和域名，以区分其他人输出，方便做全文搜索。
- 官网介绍 Live Templates : <https://www.jetbrains.com/idea/help/live-templates.html>



- 如上图标注所示，实时代码模板本质是用 XML 文件来保存的，所以传播自己的实时代码模板只要传播对应的文件即可。
- IntelliJ IDEA 的实时代码模板保存在 `/templates` 目录下，其他系统目录位置如下：
(因为目录名在各个系统上是一致的，建议用硬盘搜索工具搜索即可)
 - Windows: `.\config\templates`
 - Linux: `~/.config/templates`
 - OS X: `~/Library/Preferences//templates`

调用常规的实时代码模板主要是通过两个快捷键：`Tab` 和 `Ctrl + J`。虽然 IntelliJ IDEA 支持修改此对应的快捷键，但是默认大家都是这样使用的，所以没有特别愿意就不要去改。

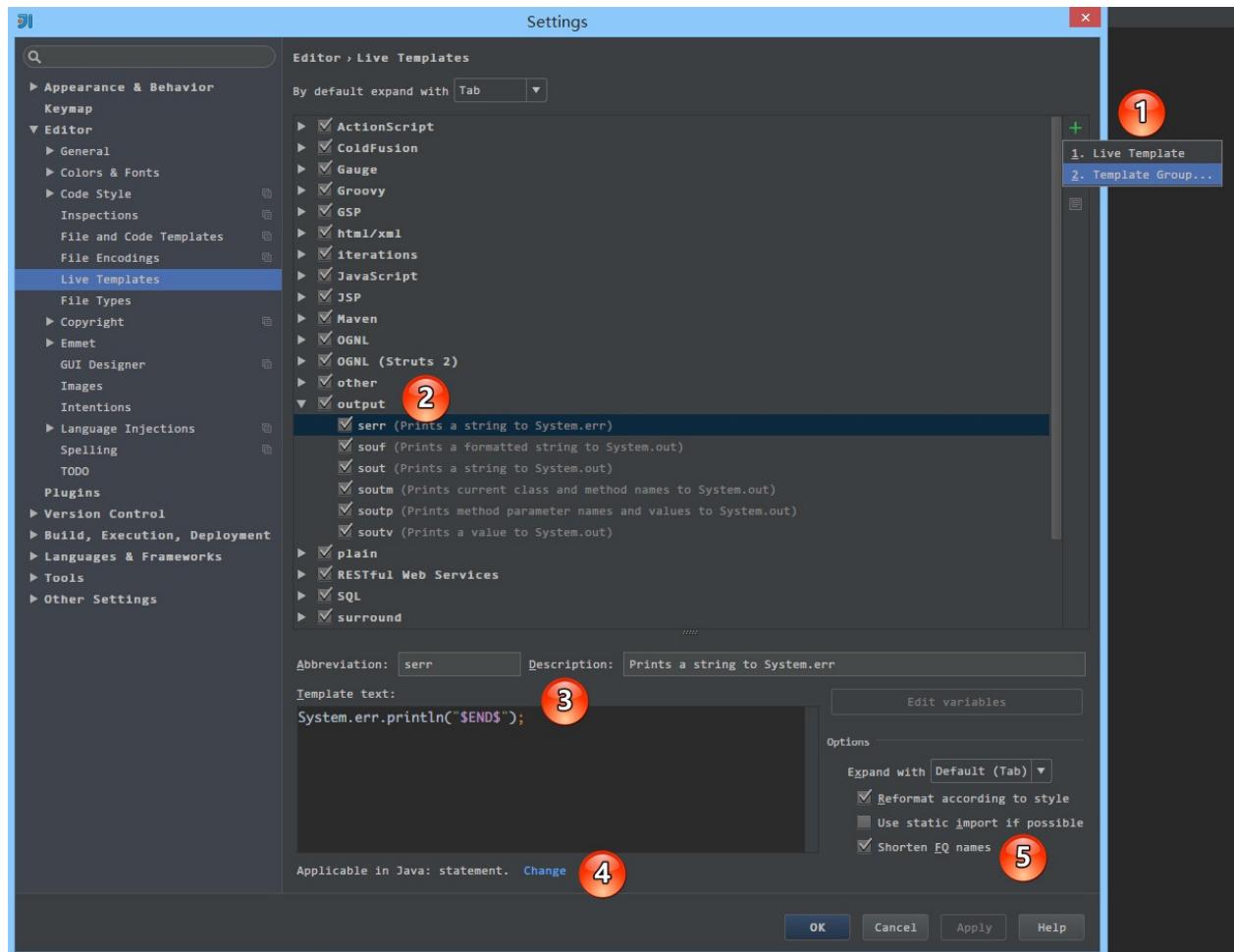
该两个快捷键的使用方法：如第一张 Gif 演示图所示，在输入 `sys` 后按 `Tab` 键，即立即生成预设语句。如果按 `Ctrl + J` 则会先提示与之匹配的实时代码模板介绍，然后还需按 `Enter` 才可完成预设语句的生成。

自带变量参数介绍

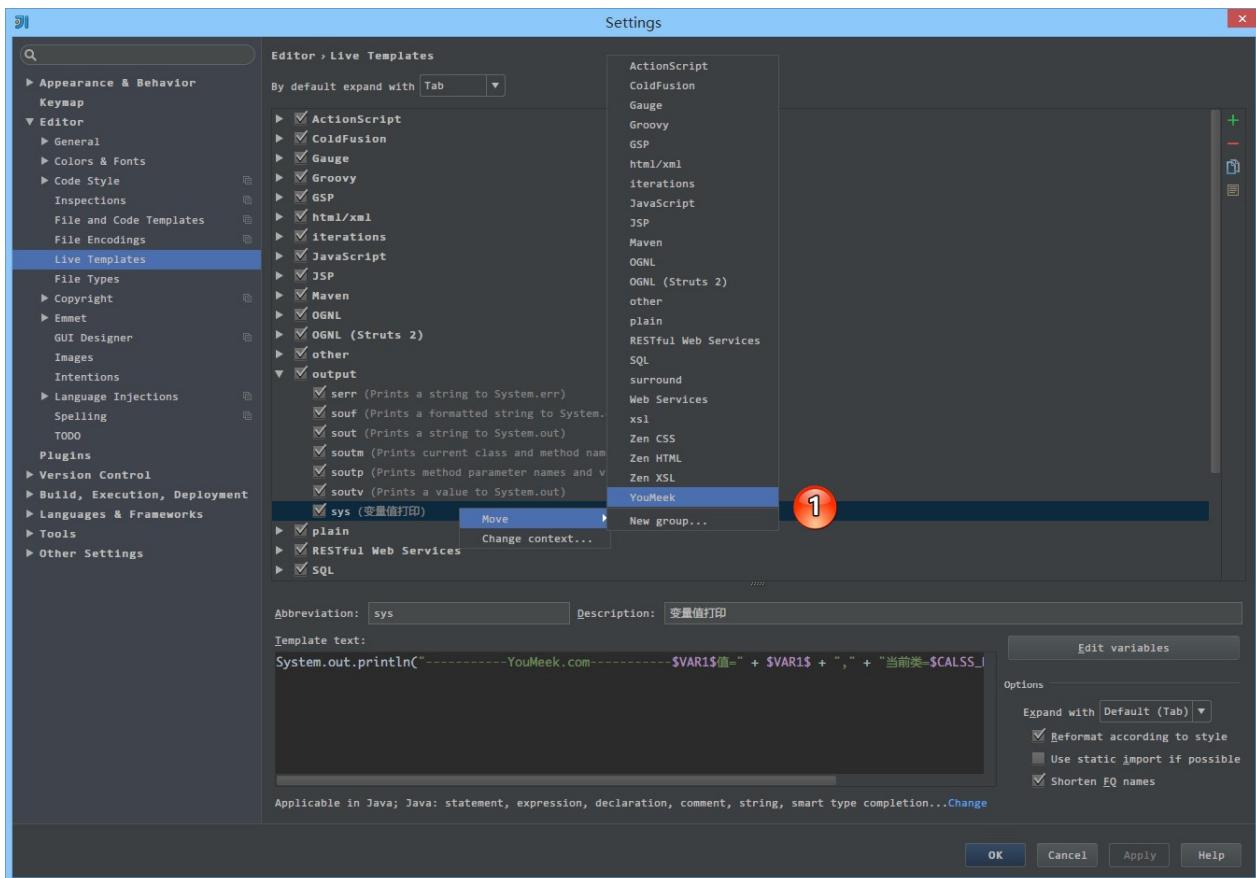
在文章开头我们已经有提到 IntelliJ IDEA 实时代码模板是有变量参数设置的，其中 IntelliJ IDEA 自带了两个变量参数：

- `END`，表示最后都编辑完后光标所处的位置
 - `$SELECTION$`，表示设置环绕实时代码模板，环绕功能下面会模板专门进行介绍。
 - 除了上面两个变量参数外，其他一律都会自定义变量。
 - 官网介绍：<https://www.jetbrains.com/idea/help/live-template-variables.html>
- `END` indicates the position of the cursor after the template is expanded.
 - `$SELECTION$` is used in surround templates and stands for the code fragment to be wrapped.

实时代码模板的设置



- 如上图标注 1 所示，除了 IntelliJ IDEA 预设的模板之外，我们还可以创建新组和新实时代码模板，其中组是用来包含实时代码模板的。
- 如上图标注 2 所示，`output` 是组名，告诉开发者，这组里面实时代码模板都是用来做输出的。`serr` 表示实时代码模板输出 `System.err` 的简称，所以这个不是一个完整的单词，不需要读懂，只需要记忆。
- 如上图标注 3 所示，实时代码模板的内容，其中用到了预设的变量 `END`。
- 如上图标注 4 所示，可设置该实时代码模板的试用范围，比如图上的 `serr` 这是 Java 代码，所以试用范围我们就应该只是 Java 文件上或是 JSP 这类文件上，设置在 HTML 或是 CSS 文件上就完全没有多大意义了。
- 如上图标注 5 所示，勾选了辅助的功能：
 - `Reformat according to style` 对生成的代码进行格式化。
 - `Shorten FQ names` 关于此设置的说明我没有找到，所以暂时无法给个很少的解释，只是看到材料说明一般需要勾上此设置，如果您有好的答案，还请联系我。

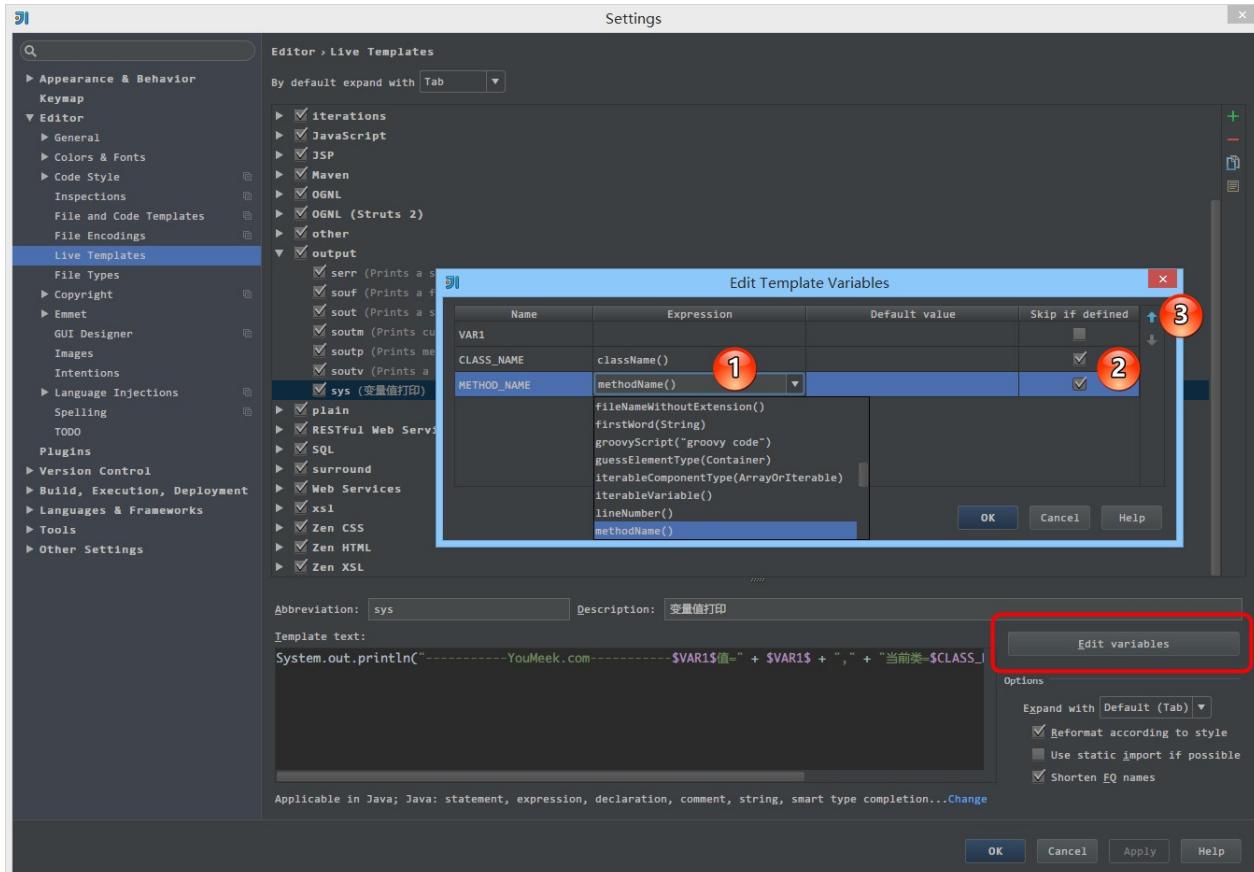


- 如上图标注 1 所示，我们可以对实时代码模板进行组的转移。

变量参数和函数的介绍

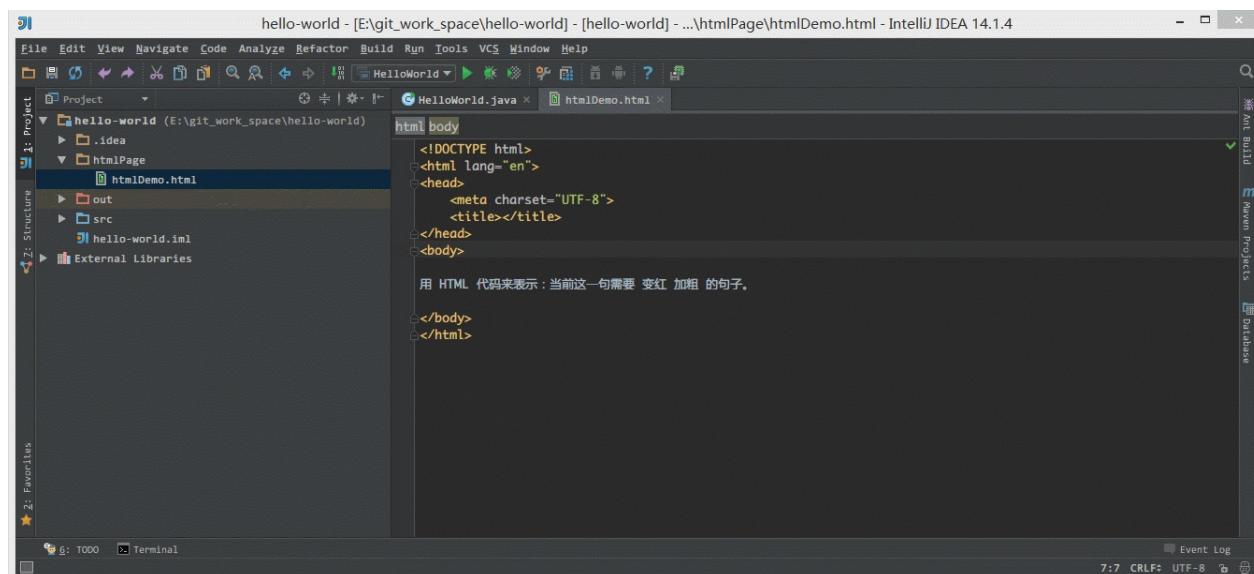
如文章开头演示的 Gif 实时代码模板，其实实时代码模板内容为： `System.out.println("-----
---YouMeek.com-----$VAR1$值=\"$ + $VAR1$ + \" + \"当前类
=$CLASS_NAME$.METHOD_NAME$()");$END$`

- `$VAR1$`、`$CLASS_NAME$`、`$METHOD_NAME$` 都为自己定义的变量名。设置变量名只要用两个 `$` 包住即可。
- 每个变量在代码输出的时候都是一次光标位置，光标跳动顺序从左到右，每次跳动按 `Enter`。



- 如上图标注红圈所示，可以对编辑代码内容的变量。
- 如上图标注 1 所示，演示中变量名 `$CLASS_NAME$`、`$METHOD_NAME$` 会自动输出当前类名和方法名是因为使用对应的函数 `className()` 和 `methodName()`。
- 如上图标注 2 所示，如上面介绍的，每个变量都是一次光标位置，但是演示中变量名 `$CLASS_NAME$`、`$METHOD_NAME$` 没有进入，是因为勾选了 `Skip if defined`。
- 如上图标注 3 所示，可以改变变量光标跳转时的顺序。
- 内置函数介绍：<https://www.jetbrains.com/idea/help/live-template-variables.html>
 - 对于官网这些函数这里就不在累赘，官网有详细的介绍，如果你还看不懂，可以看 IntelliJ IDEA 已经预设各个实时代码模板，基本上常用的函数都有被引用过，你可以通过学习这些预设的实时代码模板来揣测其函数的用法。

环绕功能介绍

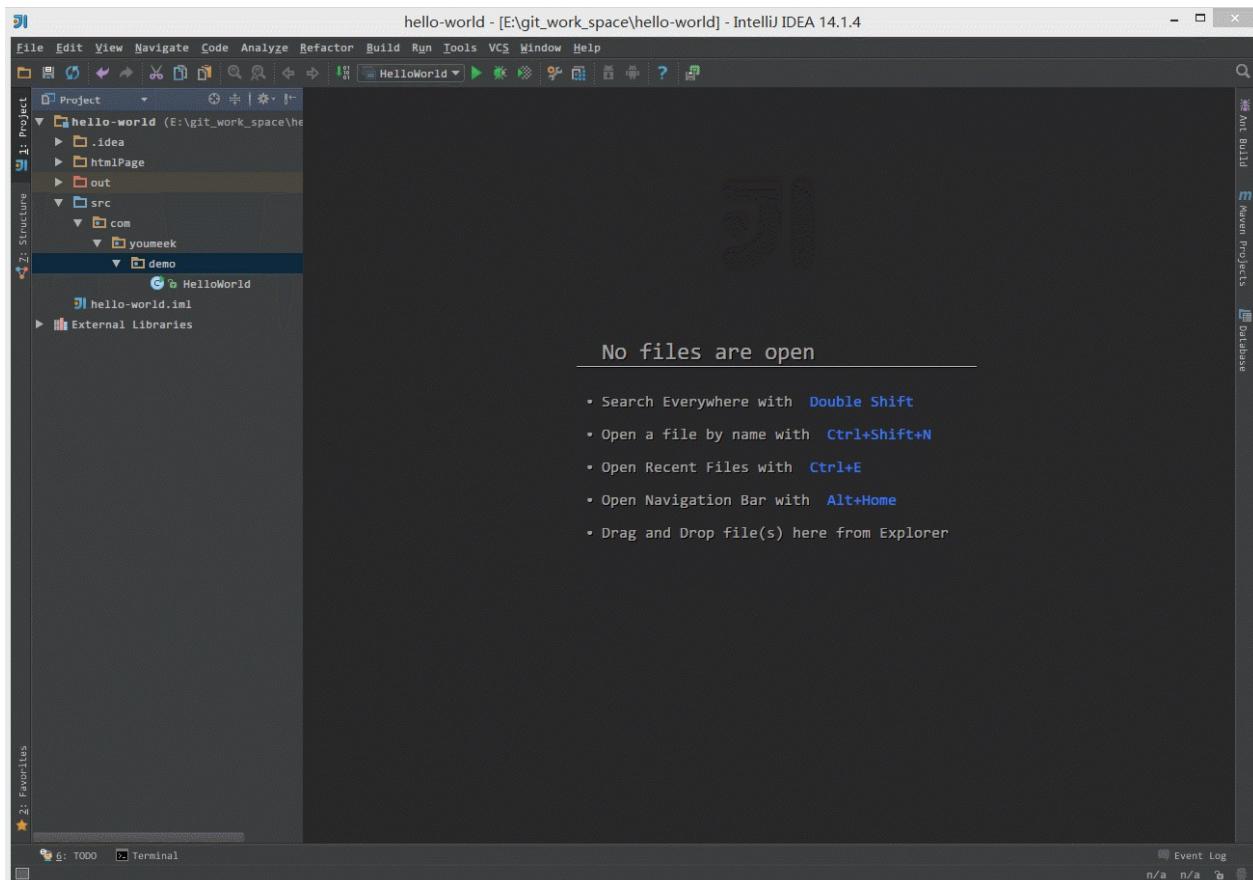


- 上图 Gif 演示为最好的介绍 surround templates。
- 如上图 Gif 演示，环绕模板的前提是：选中要被环绕的内容。
- 在设置实时代码模板的时候，如果含有预设的变量 \$SELECTION\$ 则该实时代码模板是环绕模板。
- 该功能对于前端开发者来讲是非常方便的，默认 IntelliJ IDEA 已经自带了 HTML 标签的环绕功能。

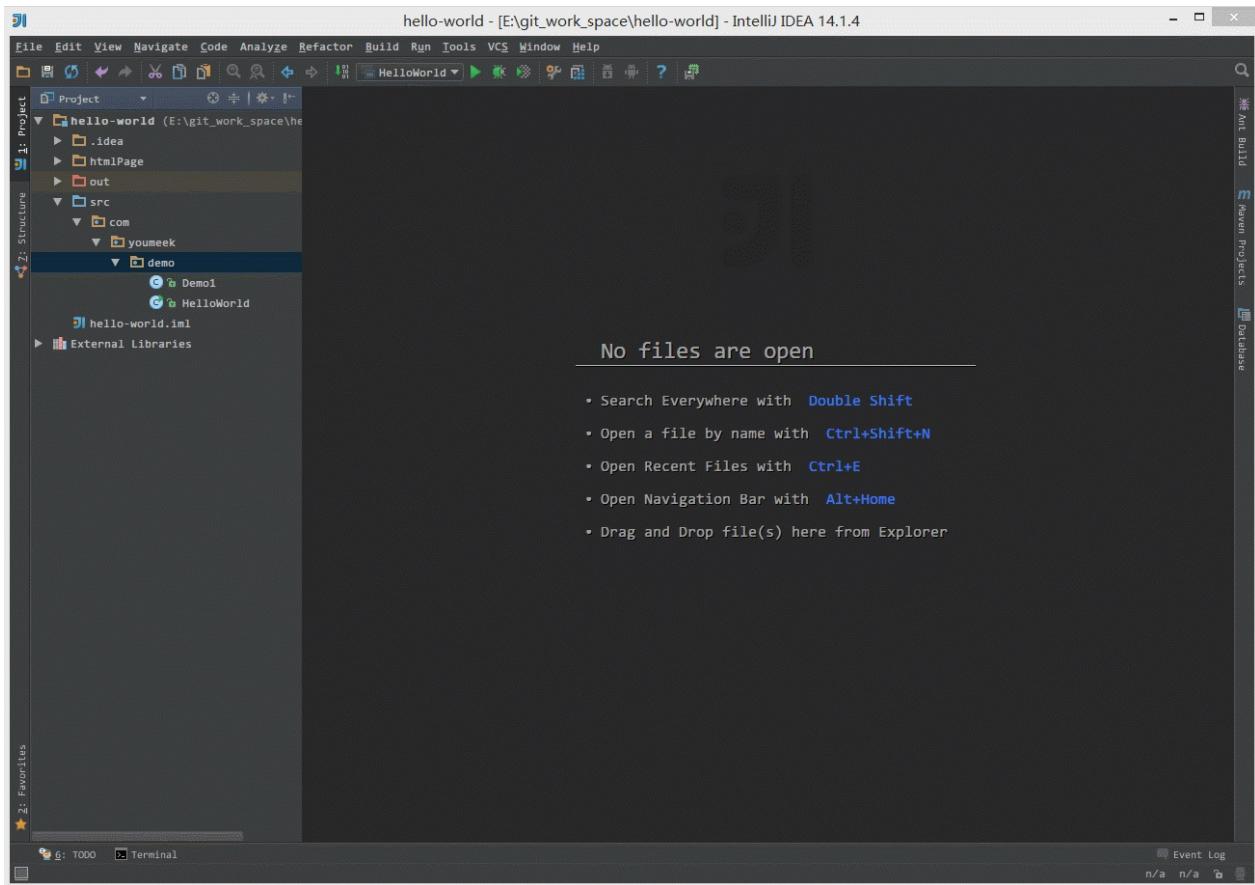
文件代码模板的使用

文件代码模板的介绍

文件代码模板可以简单理解为：我们在项目中创建某些类型文件时，就已经在对应这些新文件中预设了代码内容。因为文字表达都带有点无力，所以下面用 Gif 动态图来演示。



- 如上图 Gif 所示，IntelliJ IDEA 默认新建类自带的类注释格式一般不够友好或是规范，所以我们一般需要自己根据公司编码规范进行设置。

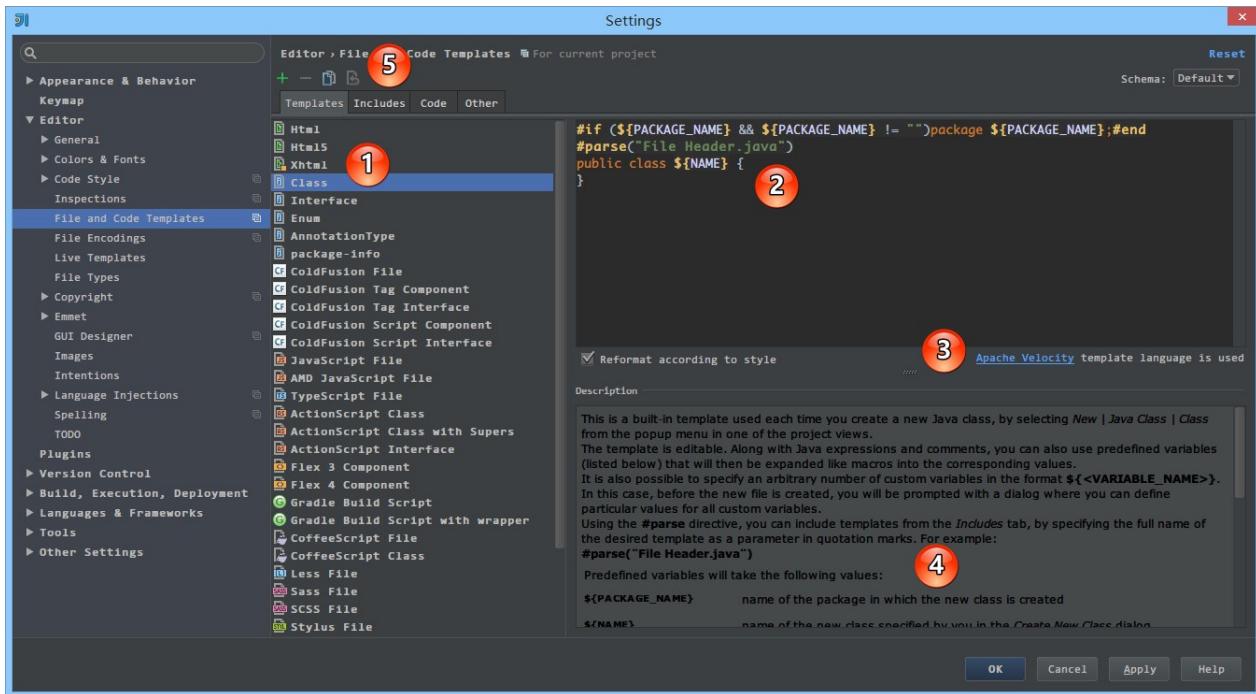


- 如上图 Gif 所示，这是根据我自己的需求进行设置的类注释，这种注释方式会更好。

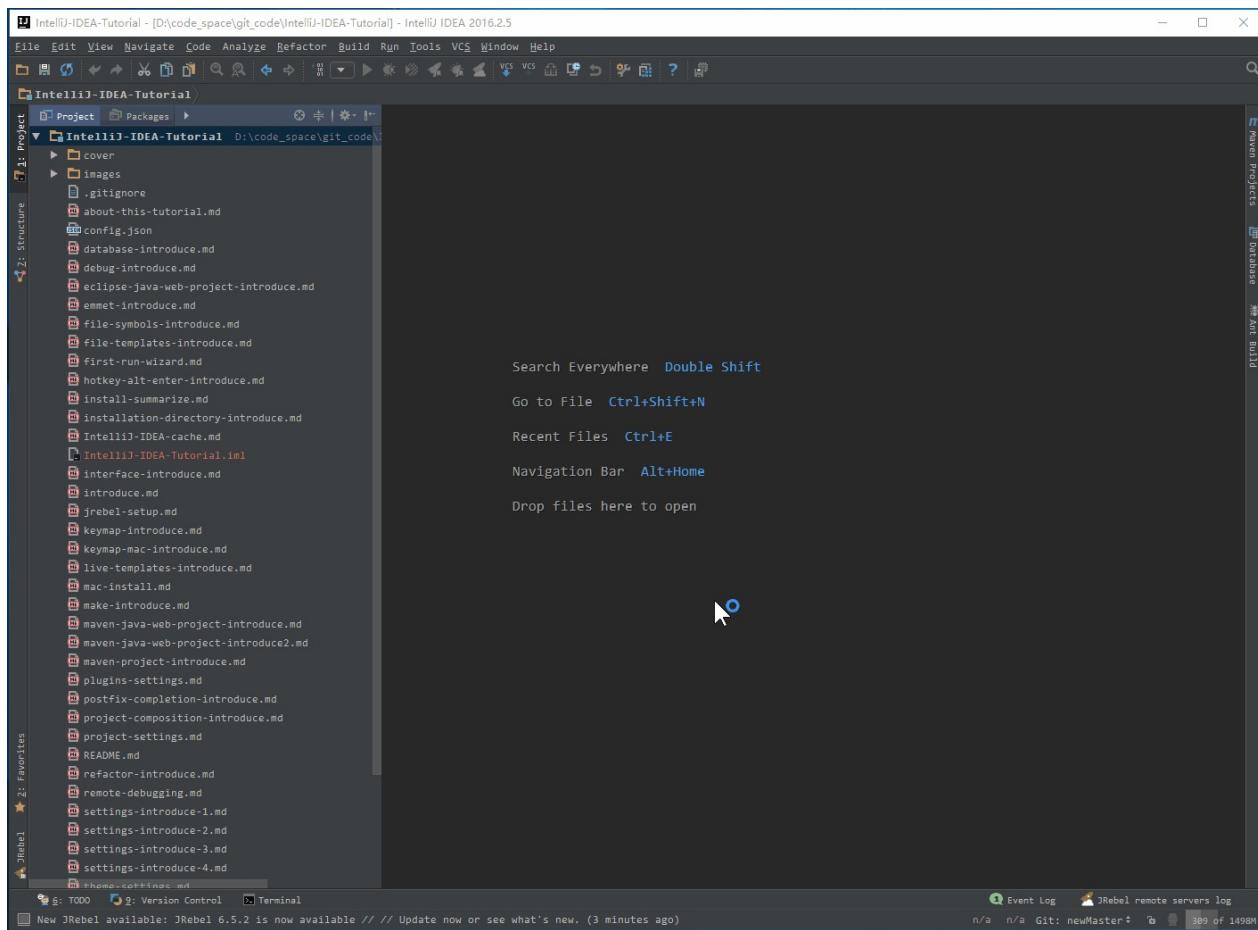
根据演示我们应该已经很好地理解了文件代码模板是什么东西了。对此我们可以衍生出很多玩法，比如：我们的项目 Controller、Service、Dao 等常用新对象都是要各自继承某个类、实现某些接口或预设某些方法，也都可以通过这样的文件代码模板来实现。

- 官网介绍：<https://www.jetbrains.com/idea/help/file-and-code-templates.html>

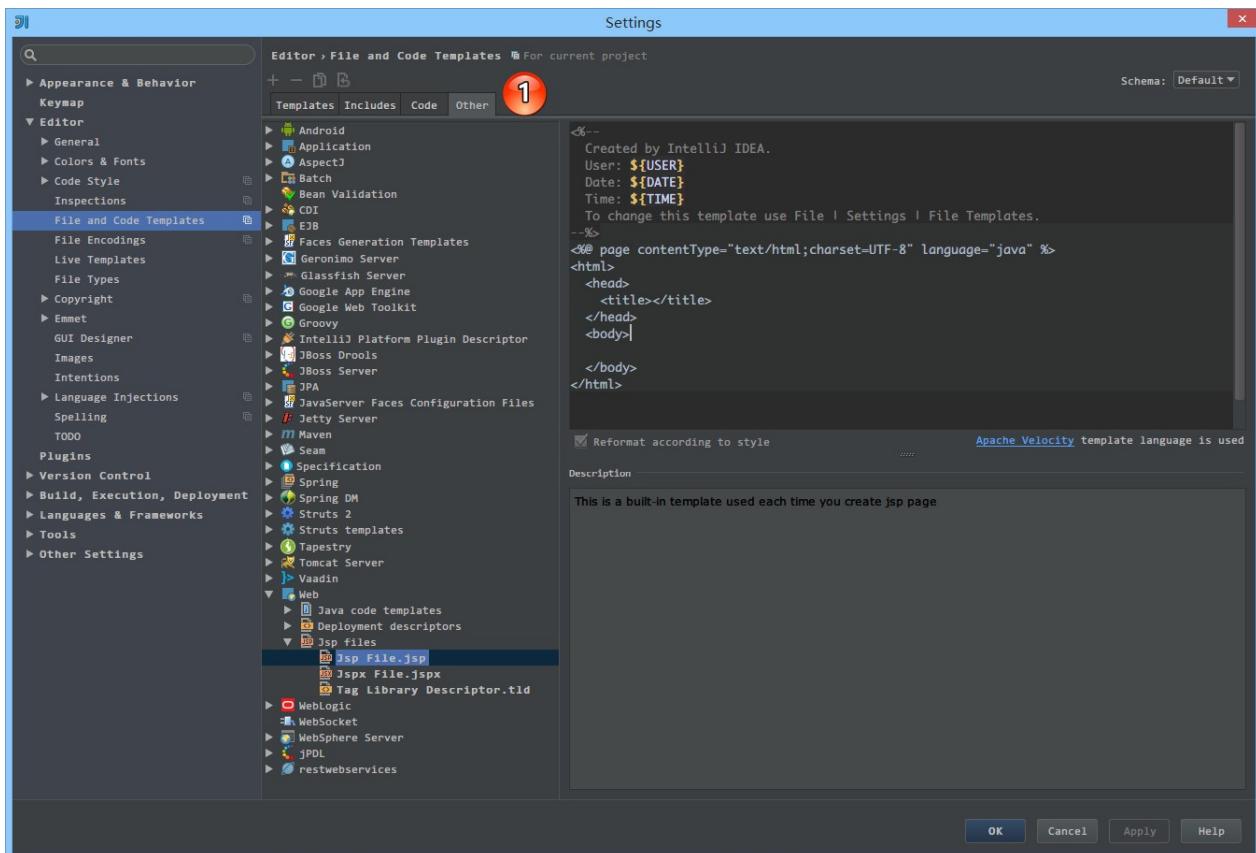
文件代码模板的设置



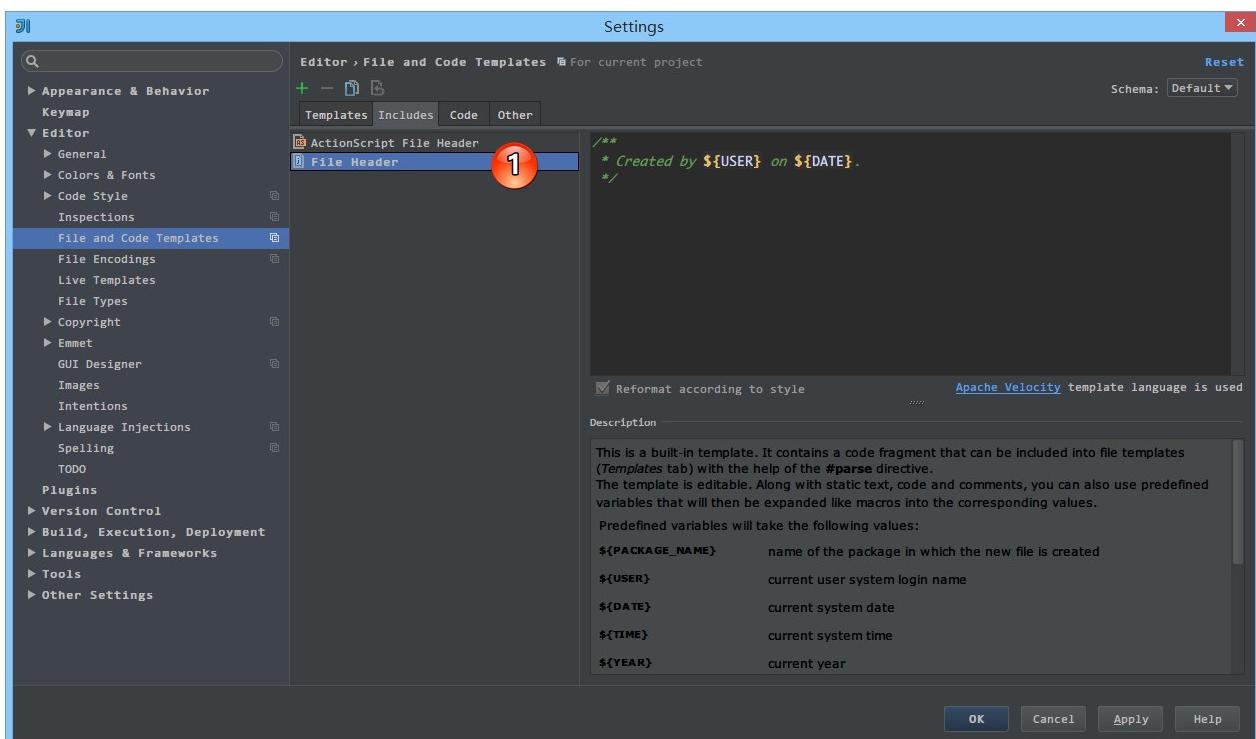
- 如上图标注 1 所示，文件代码模板支持的类型基本常见的文件类型都涵盖了。
- 如上图标注 2 所示，这是 Java 文件模板新建的代码模板，其中 `${PACKAGE_NAME}`、 `${NAME}` 是 IntelliJ IDEA 预设的变量。
- 如上图标注 3 所示，IntelliJ IDEA 的文件代码模板是可以使用 Velocity Template Language (VTL) 进行书写的。如图 2 上的 `#if ... #end` 和 `#parse` 都是 VTL 的语法。
- 如上图标注 4 所示，介绍当前文件模板的一些信息，以及一些预设变量的作用介绍。
- 如上图标注 5 所示，这四个按钮是文件代码模板的主要操作按钮，四个操作分别是：
 - `Create Template` 创建一个文件代码模板。
 - `Remove Template` 删除一个文件代码模板，标注 1 所示的这些预设模板是不允许删除的，只能能删除预设之外的新增的。
 - `Copy Template` 复制一个文件代码模板。
 - `Reset To Default` 对被修改的预设文件代码模板，还原到默认状态。



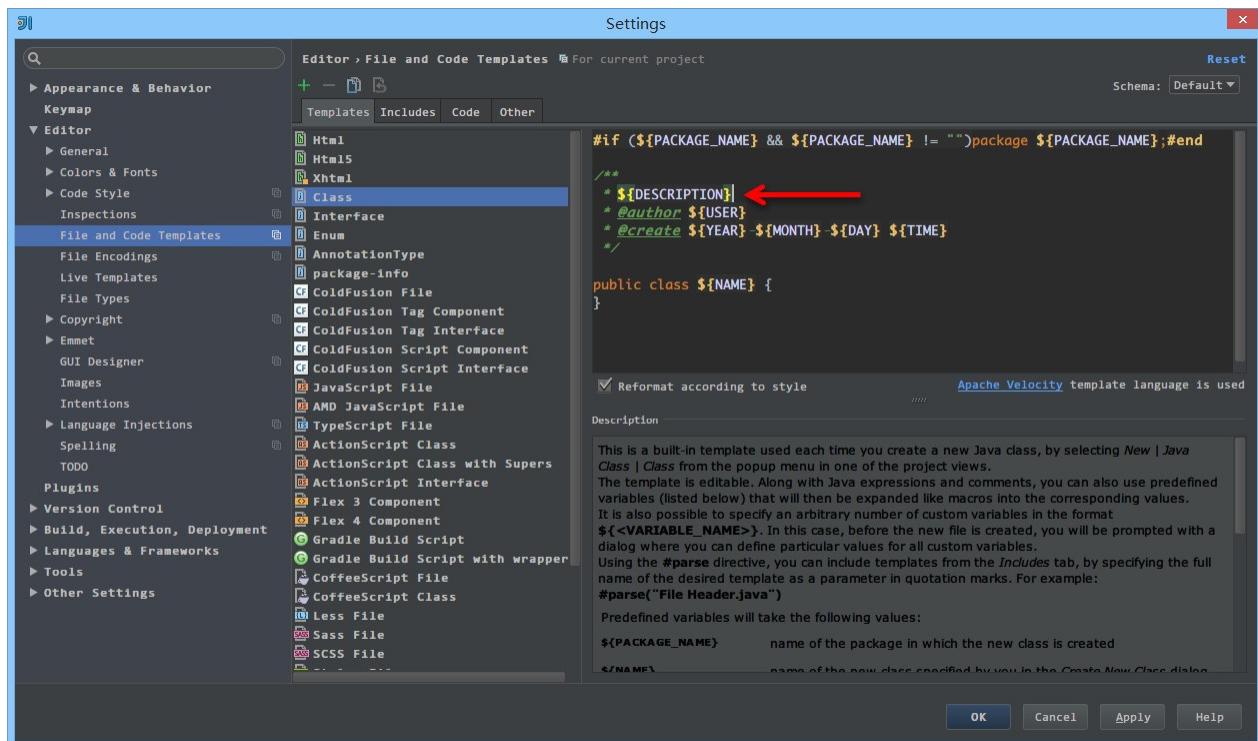
- 如上图 Gif 演示，IntelliJ IDEA 默认是没有提供 XML 文件的创建的，所以我自己创建了一个名为： YouMeek XML 的文件模板，该模板后缀为： xml ，里面的初始化内容为：`<?xml version="1.0" encoding="UTF-8"?>`。初始化的内容你可以根据自己的需求进行补充。



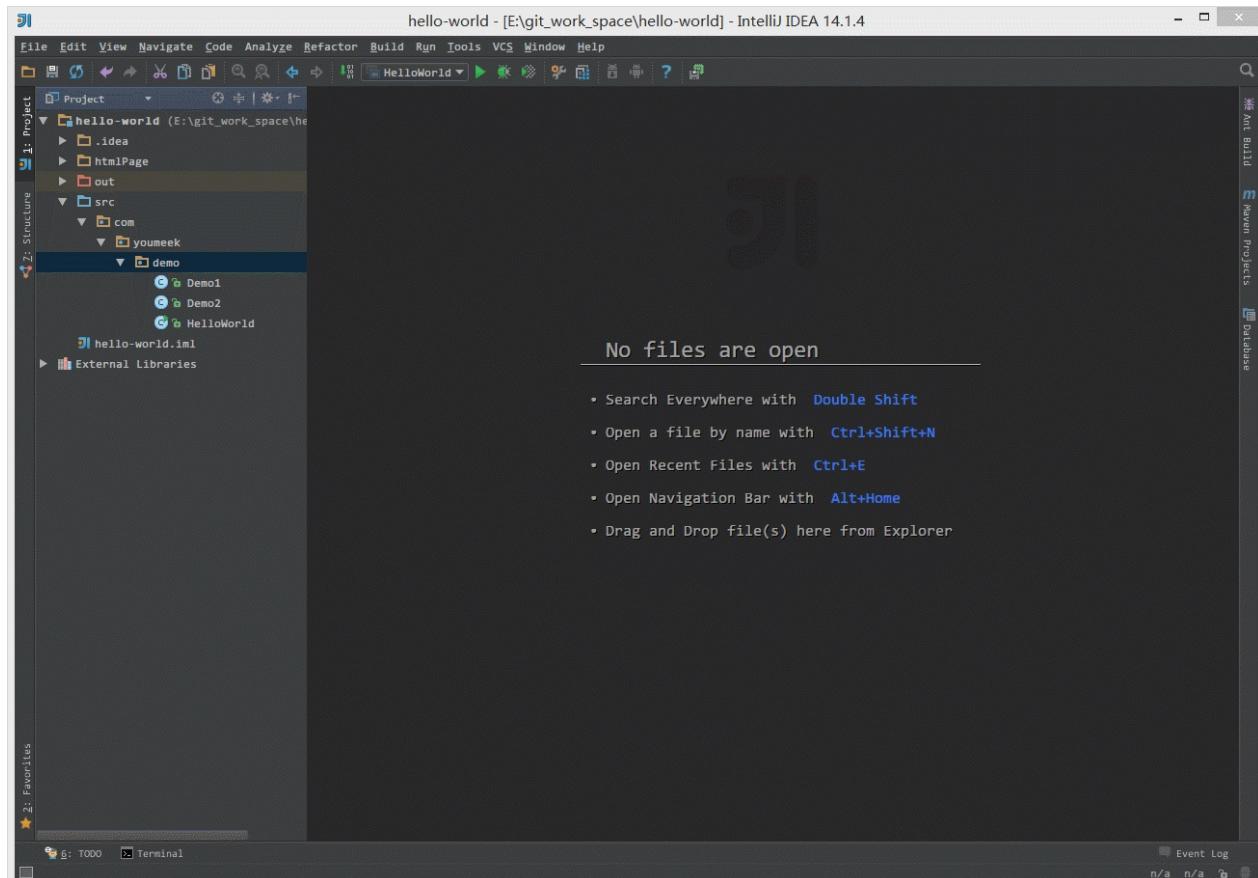
- 如上图标注 1 所示，IntelliJ IDEA 还支持其他常见会新建的文件类型，基本上我们根本不用担心有不支持的文件类型，常用的基本都被涵盖了。



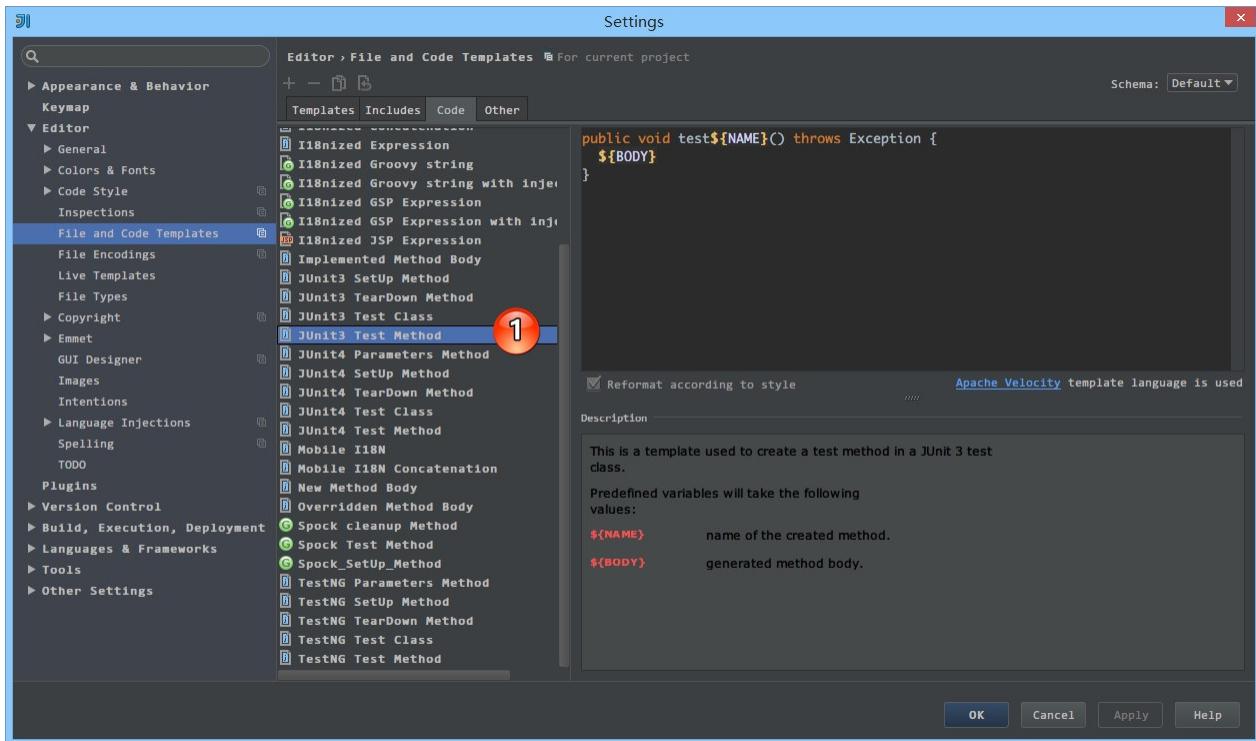
- 如上图标注 1 所示，该 `File Header` 文件就是前面 Java 文件代码模板中引入语句中 `#parse("File Header.java")` 文件。



- 如上图标注箭头所示， `${DESCRIPTION}` 是我们自己定义的变量，自定义变量格式有两种方式： `$变量名` 或 `${变量名}`。
- 自定义的变量在创建对应的文件时候就必须输入内容，这个在本文最上面的 Gif 演示中已经体现了。



- 如上图 Gif 所示，当我们需要用到一个固定值的自定义变量的时候并且该变量多个地方被引用，我们可以通过 VTL 语法的 `#set($变量名 = "变量值内容");` 来设置。



- 因为 `Code` 模块改动的人不多，所以放在最后。我们是无法新建、删除 `Code` 里面的代码模板。
- 如上图标注 1 所示，在我们通过快捷方式生成某个类的测试方法时候会自动调用此 `Code` 模板，所以我们可以解释该 `Code` 功能主要用在 IntelliJ IDEA 自动帮我们生成某些代码的时候引用的模板。

文件代码模板预设的变量

因为变量的命名太明了不过了，所以这里不多讲解，直接贴出官网的英文解释：

- \${PACKAGE_NAME} - the name of the target package where the new class or interface will be created.
- \${PROJECT_NAME} - the name of the current project.
- \${FILE_NAME} - the name of the PHP file that will be created.
- \${NAME} - the name of the new file which you specify in the New File dialog box during the file creation.
- \${USER} - the login name of the current user.
- \${DATE} - the current system date.
- \${TIME} - the current system time.
- \${YEAR} - the current year.
- \${MONTH} - the current month.
- \${DAY} - the current day of the month.
- \${HOUR} - the current hour.
- \${MINUTE} - the current minute.
- \${PRODUCT_NAME} - the name of the IDE in which the file will be created.
- \${MONTH_NAME_SHORT} - the first 3 letters of the month name. Example: Jan, Feb, etc.
- \${MONTH_NAME_FULL} - full name of a month. Example: January, February, etc.

PHP 的文件类型预设的变量比上面的还多一点，具体可以查阅官网：

- 官网变量介绍：<https://www.jetbrains.com/idea/help/file-template-variables.html>

Emmet 的使用

Emmet 的介绍

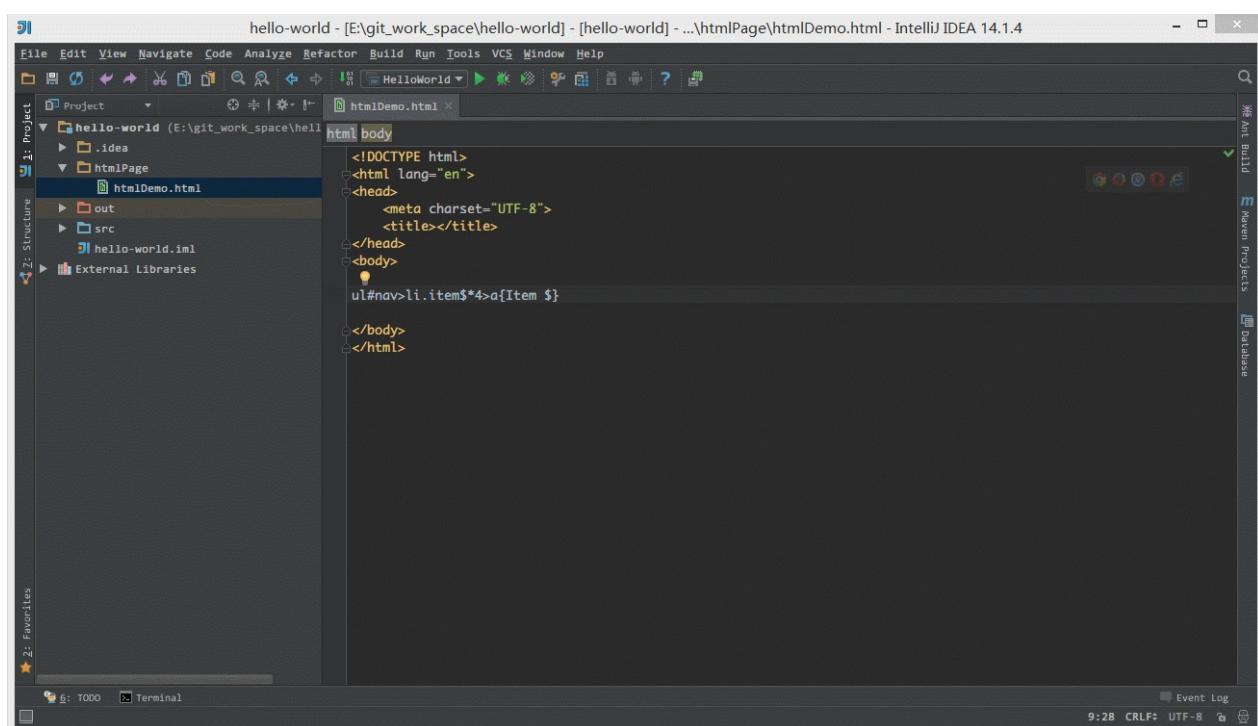
Emmet 的前身叫做：Zen Coding，也许熟知旧名的人不在少数。Emmet 一般前端工程师用得比较多，具体它是做什么的，我们通过下面两张 Gif 演示图来说明：

Emmet — the essential toolkit for web-developers

Emmet is a plugin for many popular text editors which greatly improves HTML & CSS workflow:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Demo</title>
</head>
<body>
</body>
</html>
```

Watch demo

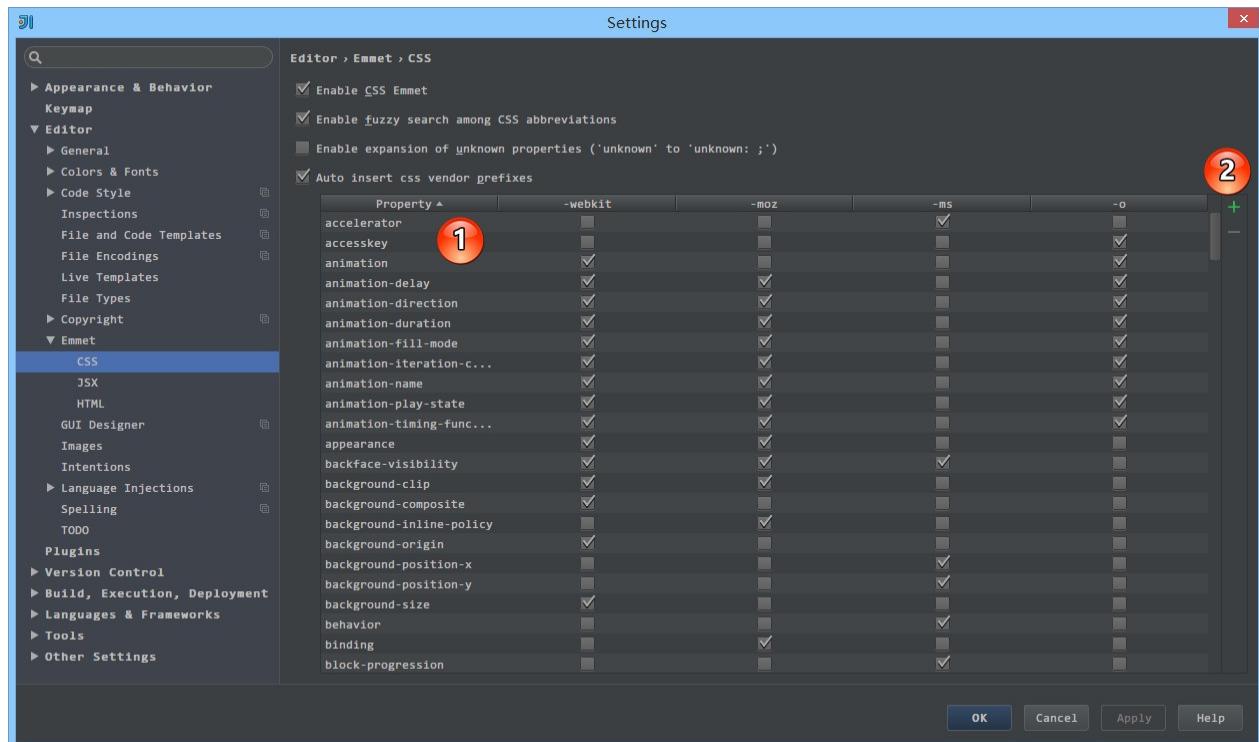


- IntelliJ IDEA 自带 Emmet 功能，使用的快捷键是 Tab。

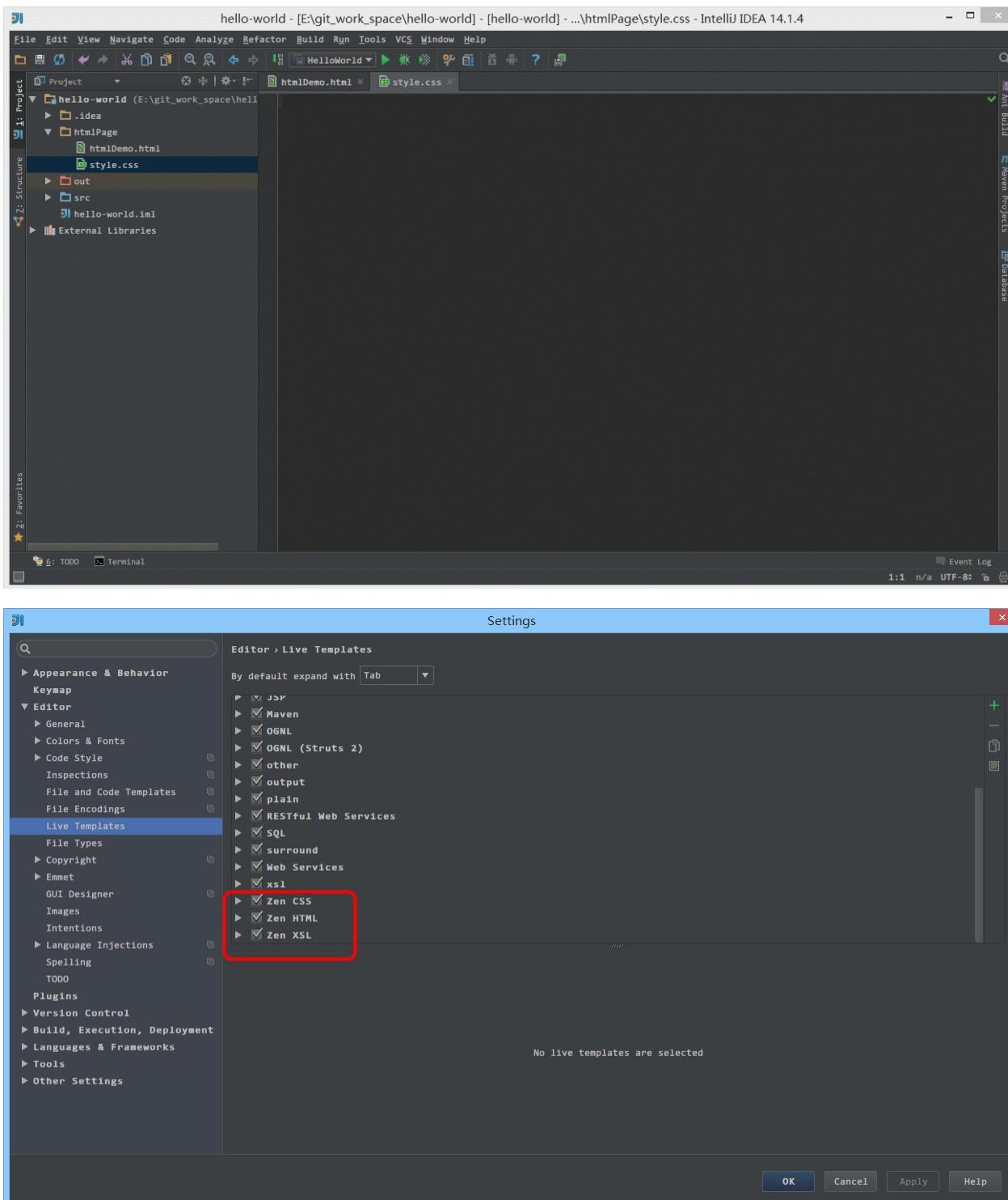
Emmet 资料介绍：

- Emmet 官网：<http://emmet.io>
- Emmet 官网文档：<http://docs.emmet.io/>
- Emmet 速查表：<http://docs.emmet.io/cheat-sheet/>
- Emmet 项目主页：<https://github.com/emmetio/emmet>

Emmet 的设置



- 如上图标注 1 所示，IntelliJ IDEA 支持主流四个浏览器内核的一些特别 CSS 书写。
- 如上图标注 2 所示，可以增加或是删除某些属性。
- 具体使用，如下图 Gif 演示。

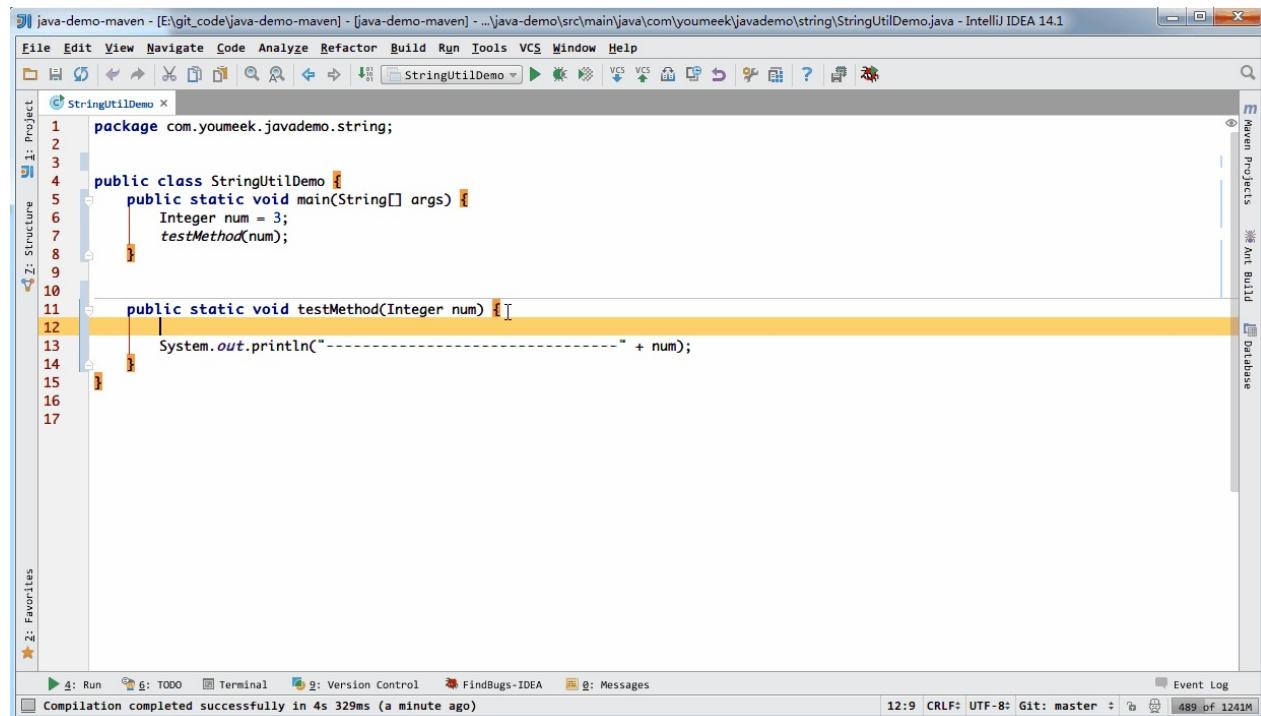


- 如上图标注红圈所示，在 `Live Templates` 中也有预设三套代码模板。

Postfix Completion 的使用

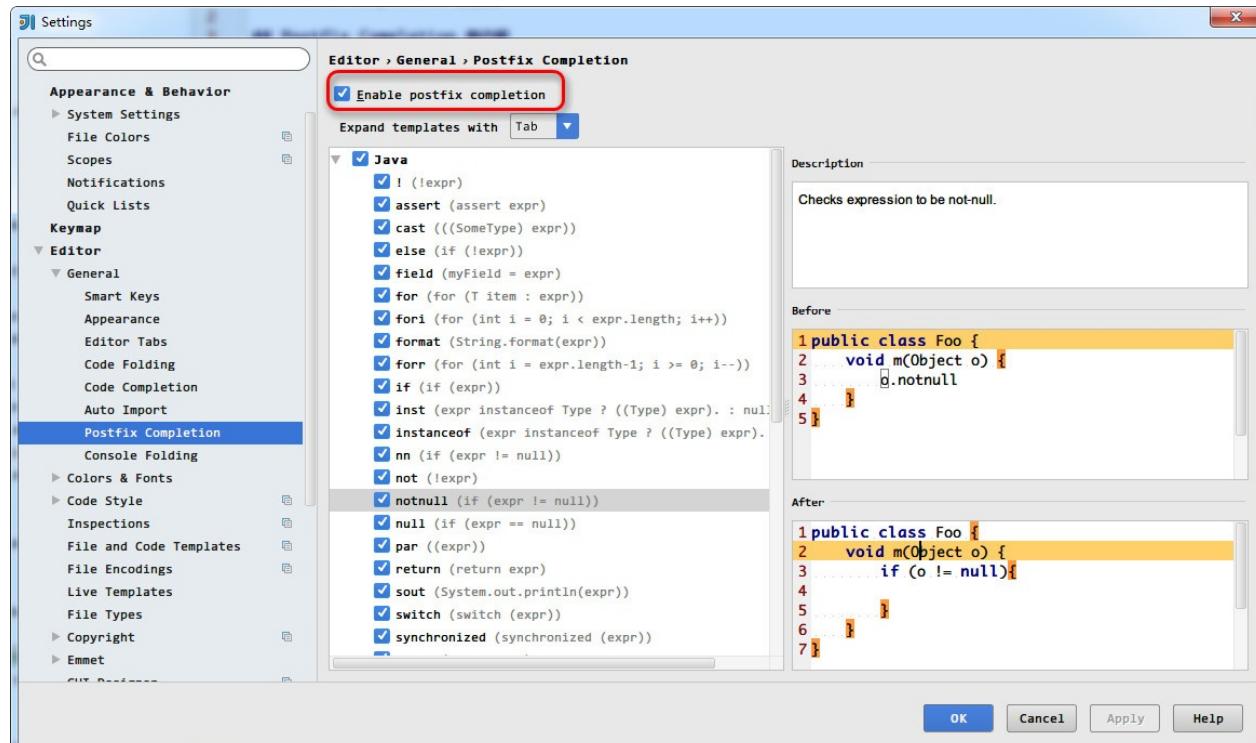
Postfix Completion 的介绍

Postfix Completion 功能本质上也是代码模板，只是它比 Live Templates 来得更加便捷一点点而已。具体它是做什么的，我们通过下面一张 Gif 演示图来说明：



- 如上图标注 1 所示，非空的判断在 Java 代码中应该是非常常见的一句话代码，如果用 Live Templates 当然也是可以快速生成，但是没有上图 Gif 这种 Postfix Completion 效果快。也许只是快了那么 0.01 秒，但是有如此好用的功能不用也是一种浪费。

Postfix Completion 的设置



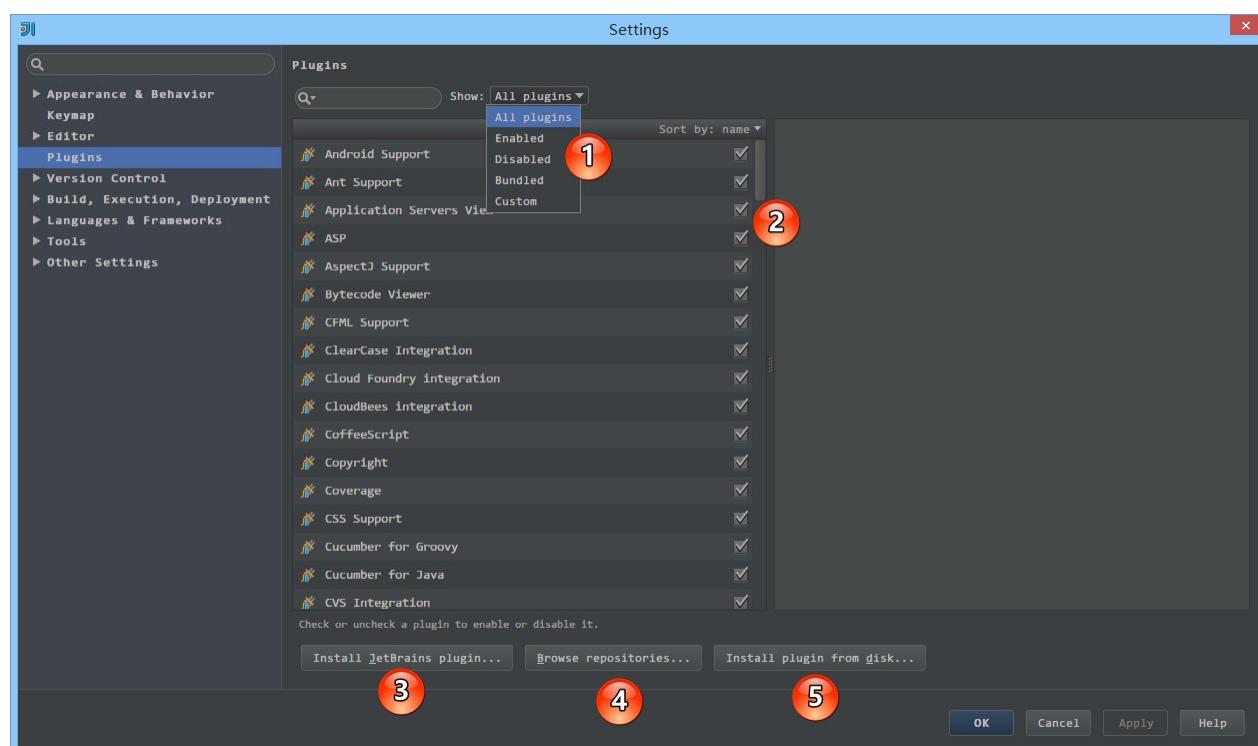
- 如上图红圈所示，IntelliJ IDEA 只提供了是否启用的开关，其他的设置就没了。所以如果目前的版本中你还无法想对该功能进行自定义。IntelliJ IDEA 也对常用到的一些固定格式的代码进行了归纳，基本目前也够用了。

插件的使用

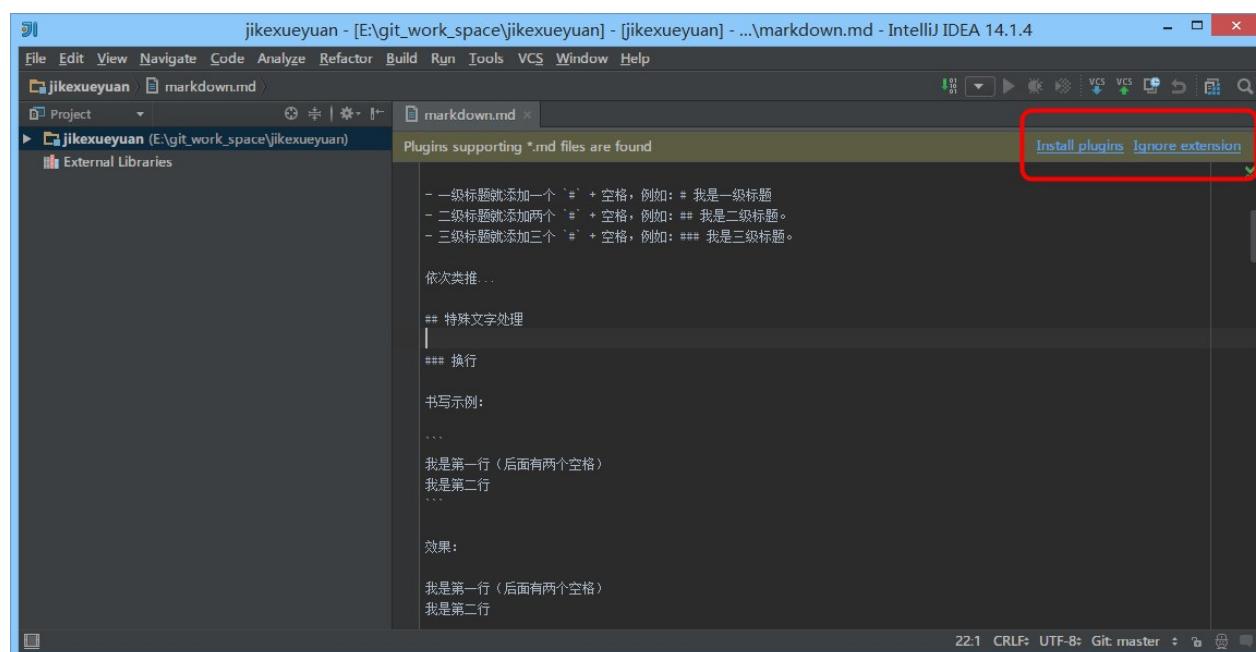
插件的设置

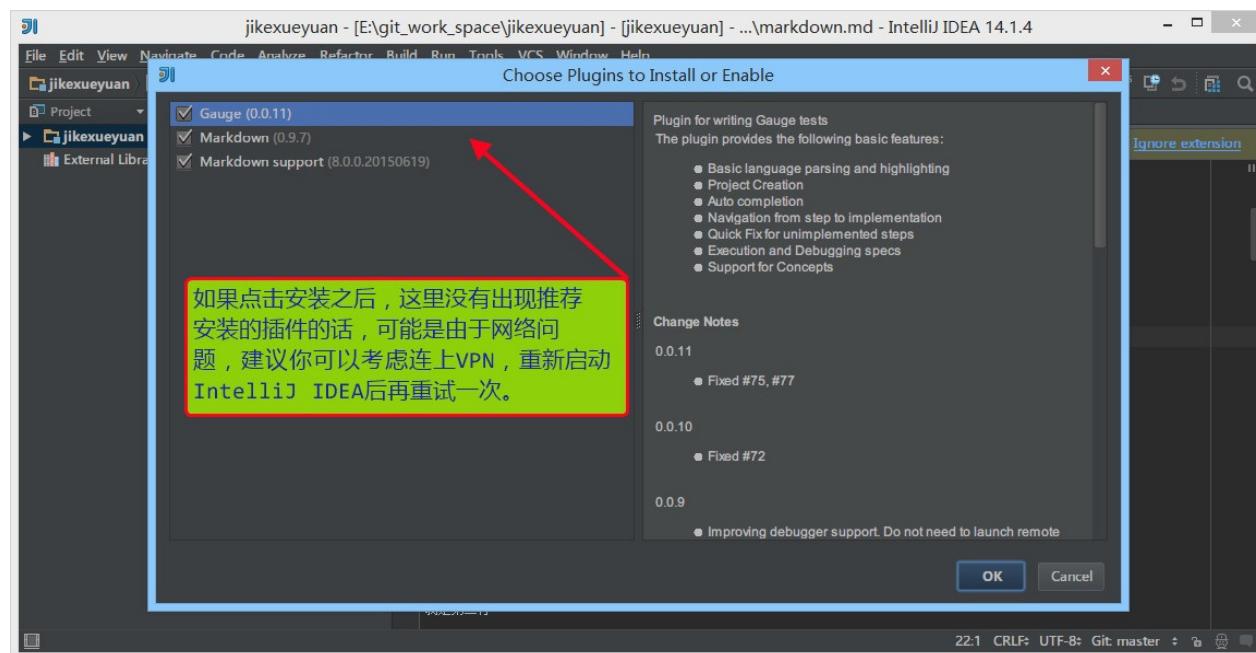
在 IntelliJ IDEA 的安装讲解中我们其实已经知道，IntelliJ IDEA 本身很多功能也都是通过插件的方式来实现的，只是 IntelliJ IDEA 本身就是它自己的插件平台最大的开发者而已，开发了很多优秀的插件。

- 官网插件库：<https://plugins.jetbrains.com/>



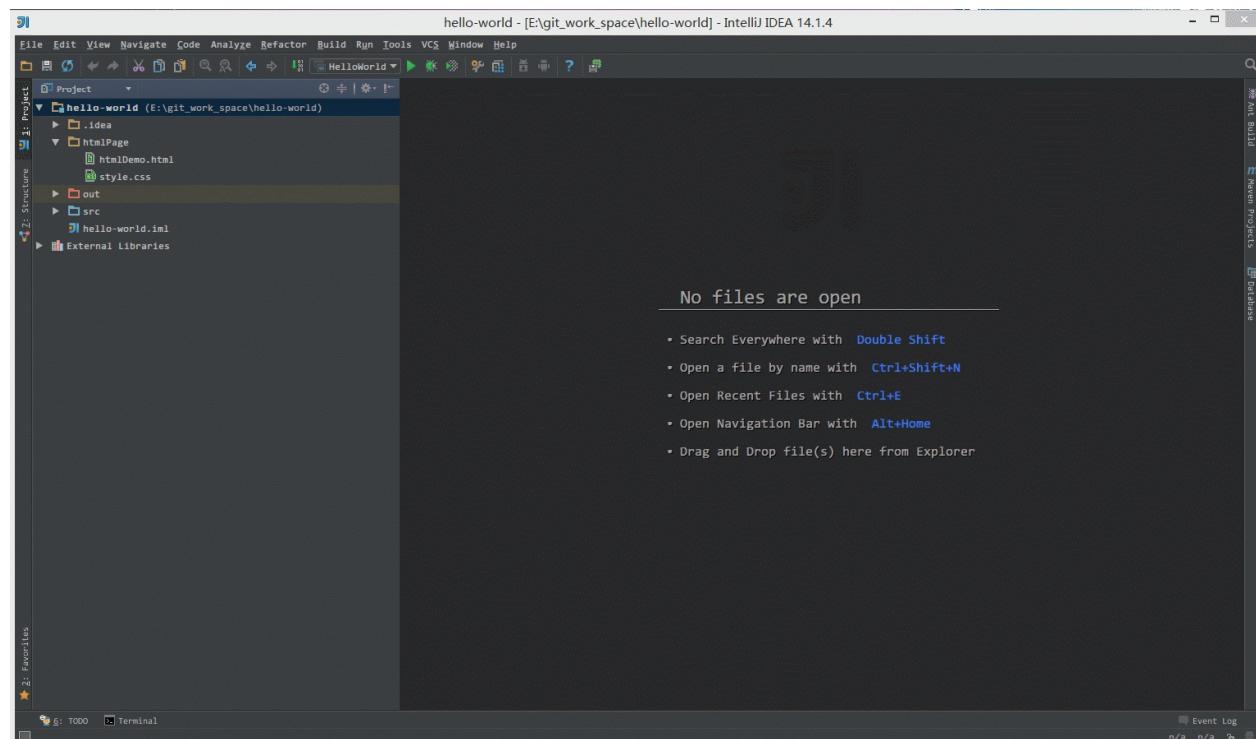
- 如上图标注 1 所示，IntelliJ IDEA 对插件进行了很好的分类：
- All plugins 显示所有插件。
- Enabled 显示当前所有已经启用的插件。
- Disabled 显示当期那所有已经禁用的插件。
- Bundled 显示所有 IntelliJ IDEA 自带的插件。
- Custom 显示所有我们自行安装的插件，如果你自己装了很多次插件的话，这个选项会用得比较多。
- 如上图标注 2 所示，启用的插件是被勾选的，如果要禁用一个插件，去掉勾选即可。
- 如上图标注 3 所示，弹出 IntelliJ IDEA 公司自行开发的插件仓库列表，供下载安装。
- 如上图标注 4 所示，弹出插件仓库中所有插件列表供下载安装。
- 如上图标注 5 所示，浏览本地的插件文件进行安装，而不是从服务器上下载并安装。
- 需要严重注意的是：在国内的网络下，很经常出现显示不了插件列表，或是显示了插件列表，无法下载完成安装。这时候请自行开 VPN，一般都可以得到解决。



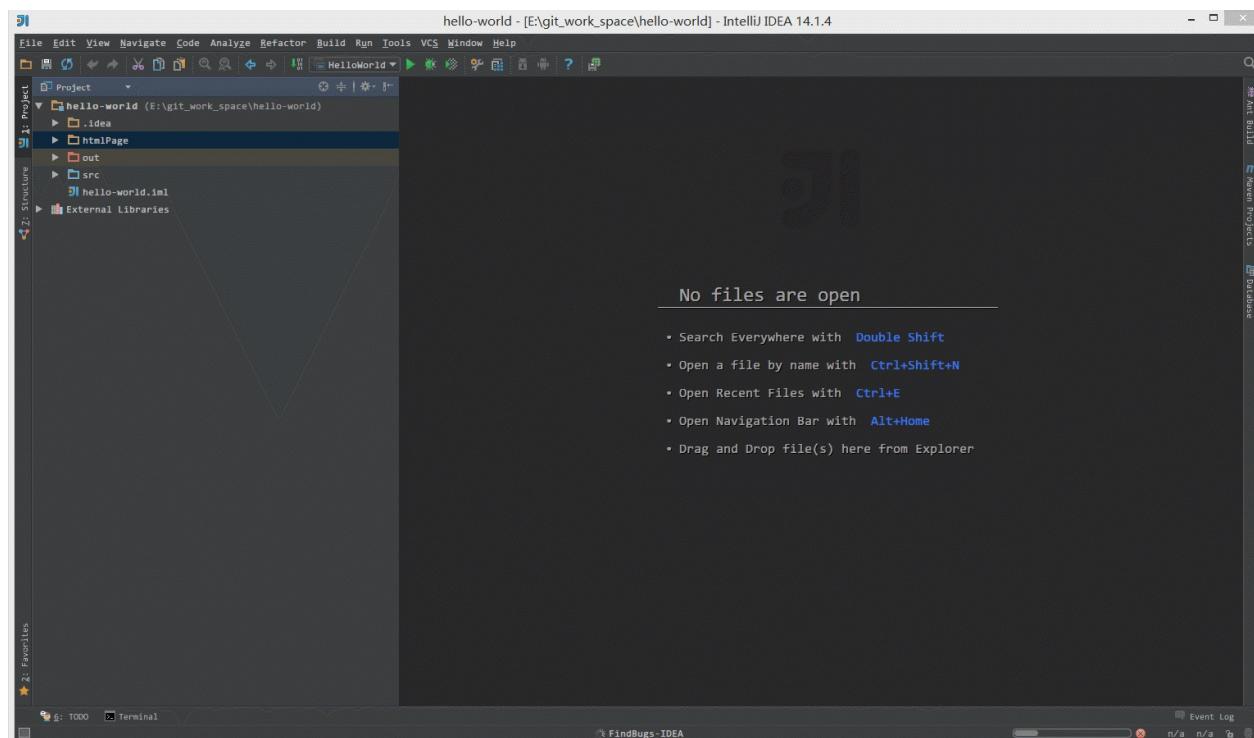


- 如上图红圈所示，如果 IntelliJ IDEA 打开一个它可以识别并且支持的文件类型，它会提示你它有对应的插件进行支持，你可以安装对应的插件来加以兼容此类文件。

插件的安装



- 如上图 Gif 演示，在线安装 IntelliJ IDEA 插件库中的插件。



- 如上图 Gif 演示，离线安装本地插件文件。

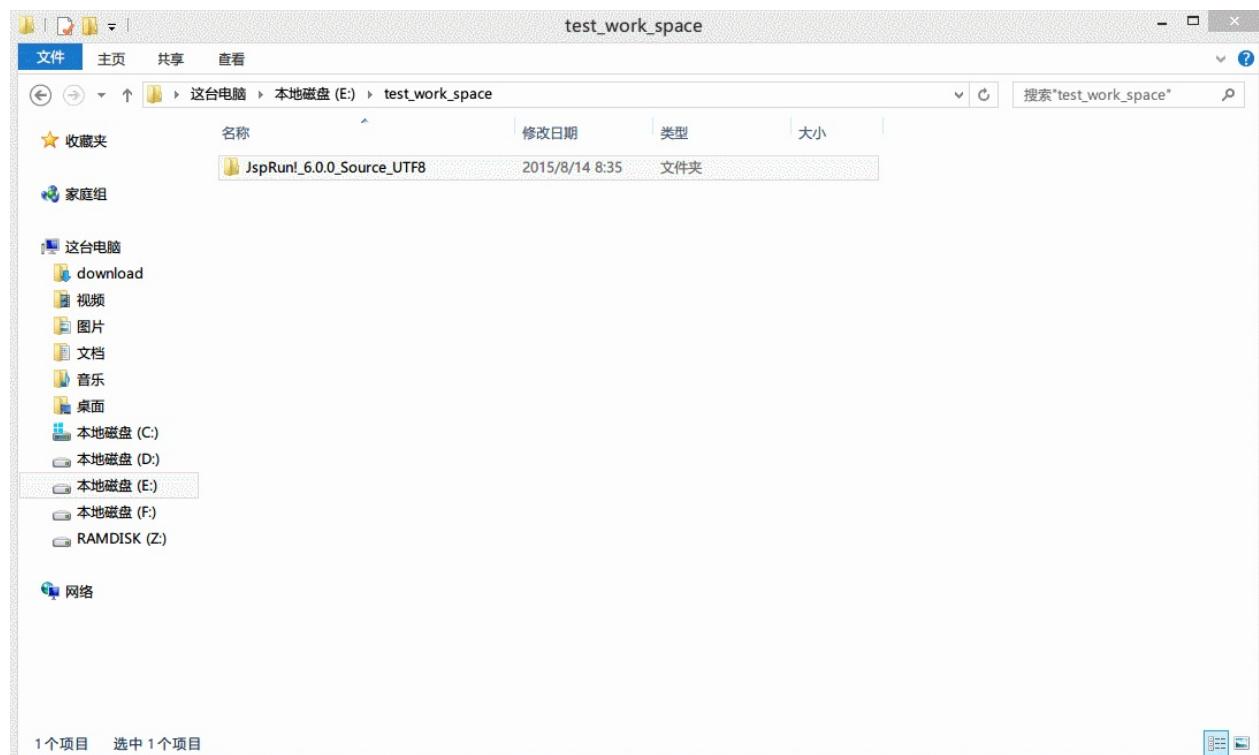
插件推荐

插件名称	插件介绍	官网地址
Key promoter	快捷键提示	https://plugins.jetbrains.com/plugin/4455?pr=idea
String Manipulation	驼峰式命名和下划线命名交替变化	https://plugins.jetbrains.com/plugin/2162?pr=idea
CheckStyle-IDEA	代码规范检查	https://plugins.jetbrains.com/plugin/1065?pr=idea
FindBugs-IDEA	潜在 Bug 检查	https://plugins.jetbrains.com/plugin/3847?pr=idea
MetricsReloaded	代码复杂度检查	https://plugins.jetbrains.com/plugin/93?pr=idea
Statistic	代码统计	https://plugins.jetbrains.com/plugin/4509?pr=idea
JRebel Plugin	热部署	https://plugins.jetbrains.com/plugin/?id=4441
CodeGlance	在编辑代码最右侧，显示一块代码小地图	https://plugins.jetbrains.com/plugin/7275?pr=idea

GsonFormat	把 JSON 字符串直接实例化成类	https://plugins.jetbrains.com/plugin/7654?pr=idea
MultiMarkdown	书写 Markdown 文章	https://plugins.jetbrains.com/plugin/7896?pr=idea
Eclipse Code Formatter	使用 Eclipse 的代码格式化风格，在一个团队中如果公司有规定格式化风格，这个可以使用。	https://plugins.jetbrains.com/plugin/6546?pr=idea
Jindent-Source Code Formatter	自定义类、方法、doc、变量注释模板	http://plugins.jetbrains.com/plugin/2170?pr=idea
ECTranslation	翻译插件	https://github.com/Skykai521/ECTranslation/releases

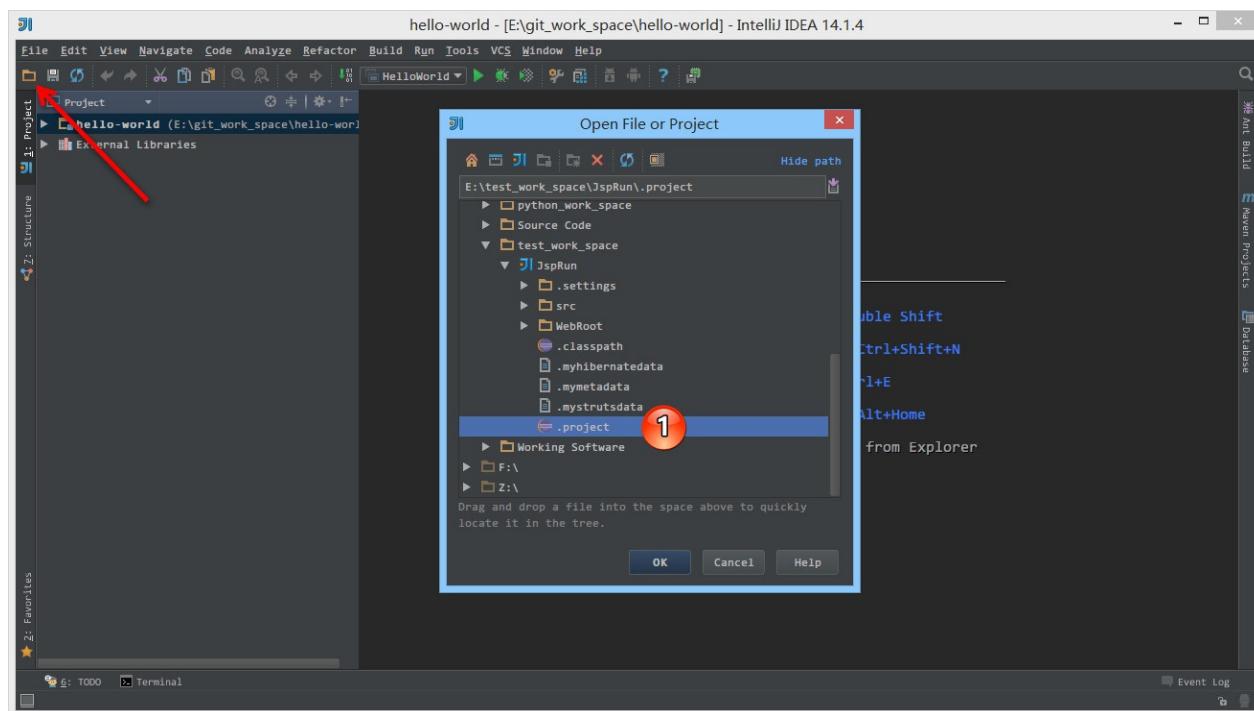
Eclipse 的 Java Web 项目环境搭建

Eclipse 项目结构

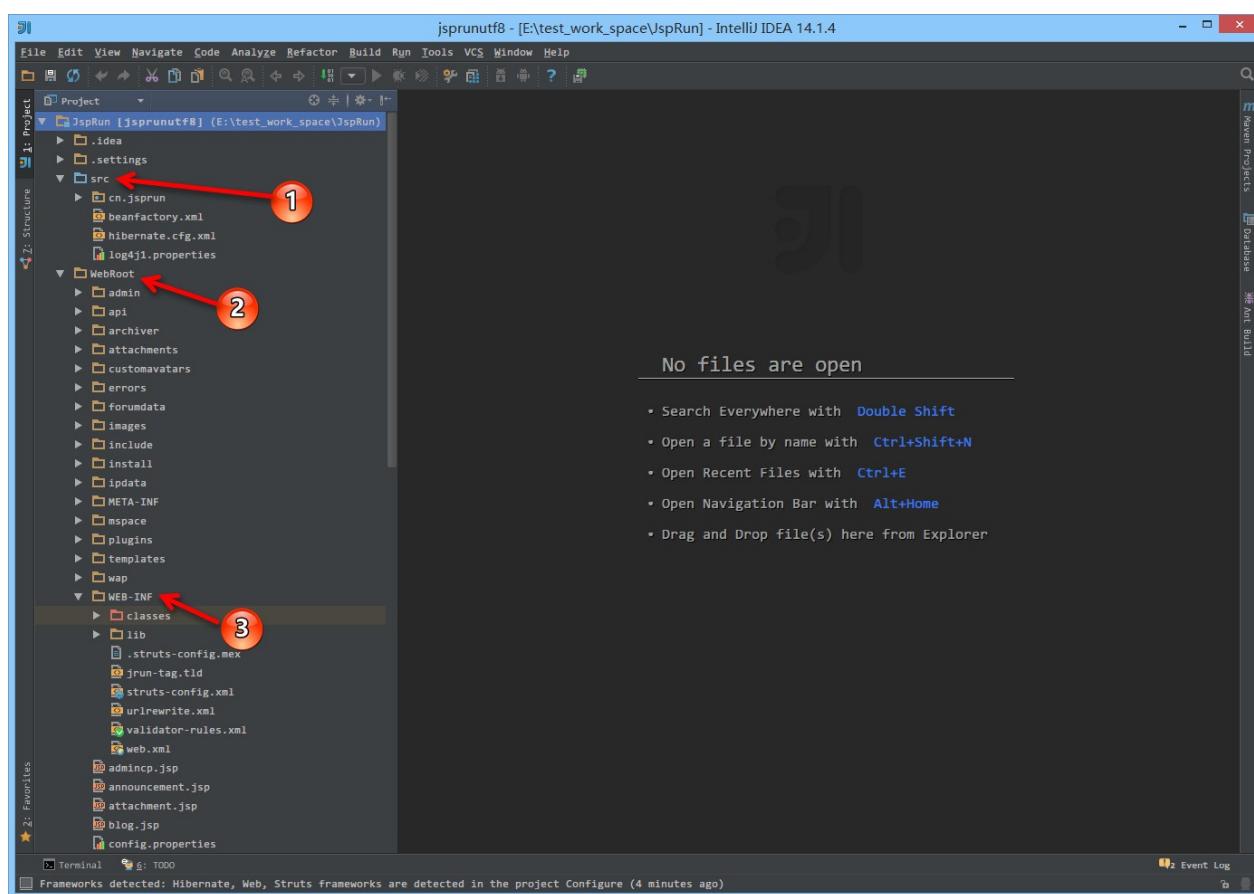


- 如上图 Gif 演示，在 Eclipse 下，一般中小项目的项目结构基本都是这种模式的，所以我们这里也用一个开源项目 JspRun 进行讲解。
- 下载地址：<http://pan.baidu.com/s/1i3zrSf7>

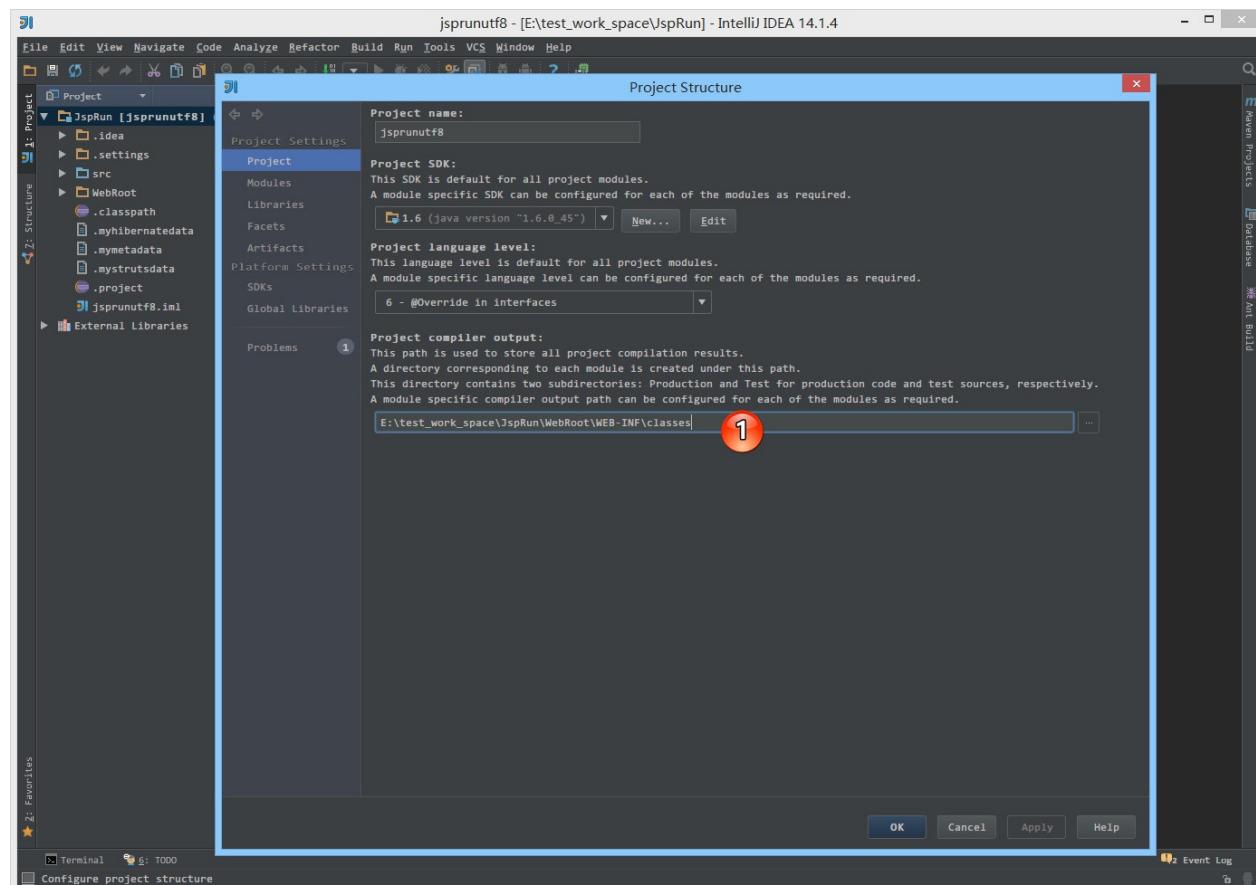
Eclipse 项目配置



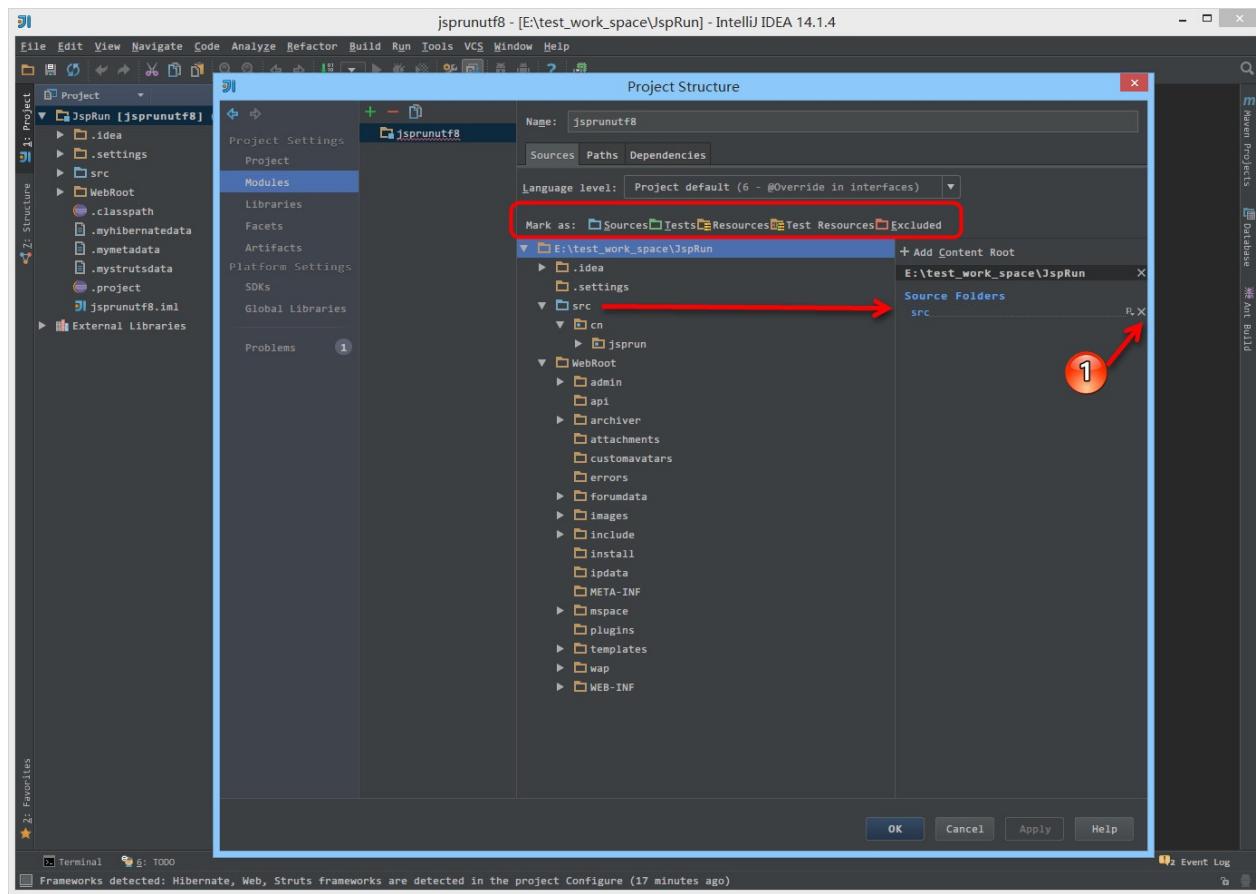
- 如上图箭头所示，在 IntelliJ IDEA 下是支持直接打开 Eclipse 项目的，无需导入。
- 如上图标注 1 所示，选择项目的 .project 文件打开即可。



- 如上图标注 1 所示，IntelliJ IDEA 能自动识别到可编译的 `src` 类目录，所以这里文件夹是蓝色的。
- 如上图标注 2 所示，Java Web 项目 `WebRoot` 是整个项目输出的根目录，所以这个区域也是非常重要的，只是无需配置。
- 如上图标注 3 所示，`WEB-INF` 下的一些配置文件，以及两个目录 `classes` 和 `lib` 都是至关重要的，其中 `classes` 是红色目录，也就是被排除的，因为编译的文件对开发来讲是没有多大意义的，所以进行了排除。但是这并不会影响容器去使用它。

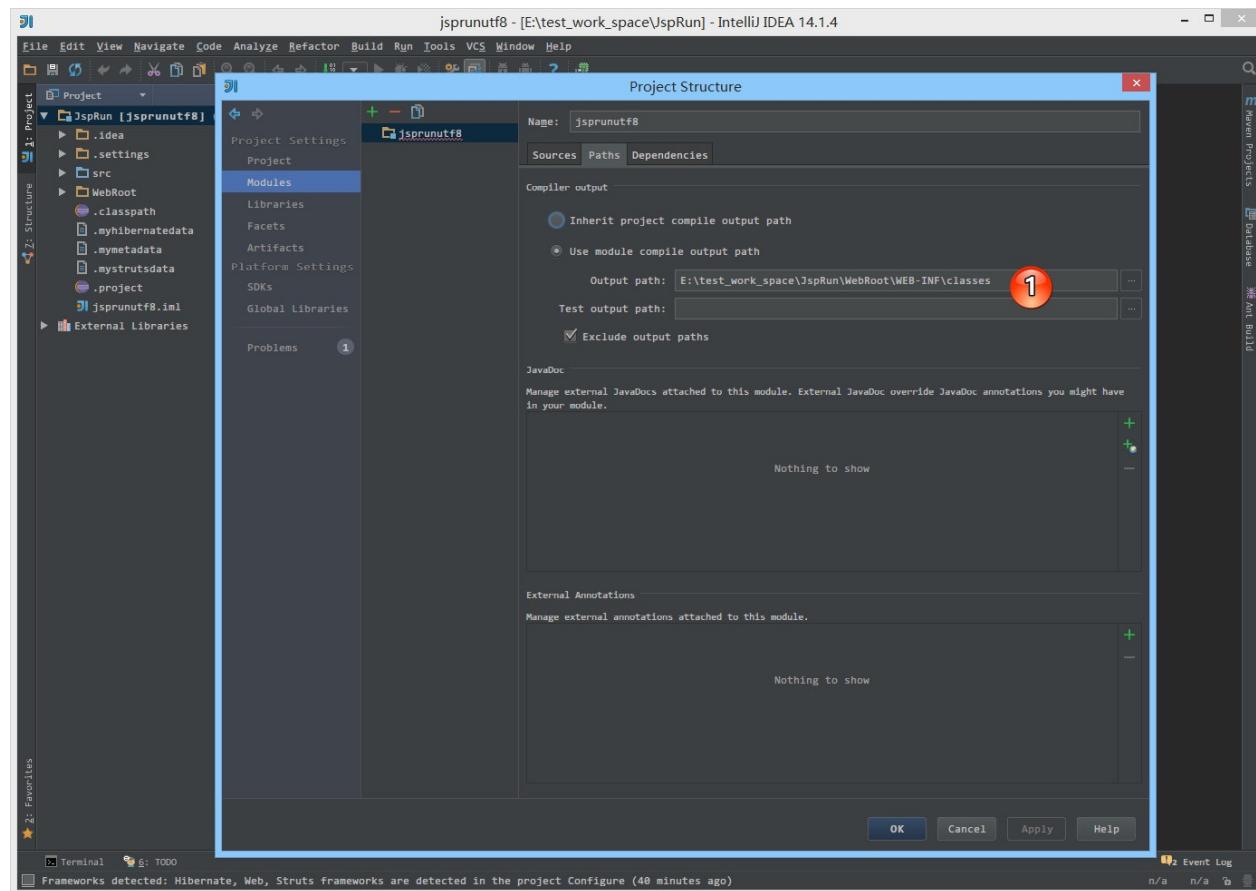


- 如上图标注 1 所示，因为这个项目是单 Module 的，所以我们这里在 `Project compiler output` 可以选择项目 `WEB-INF` 下的 `classes` 编译目录。

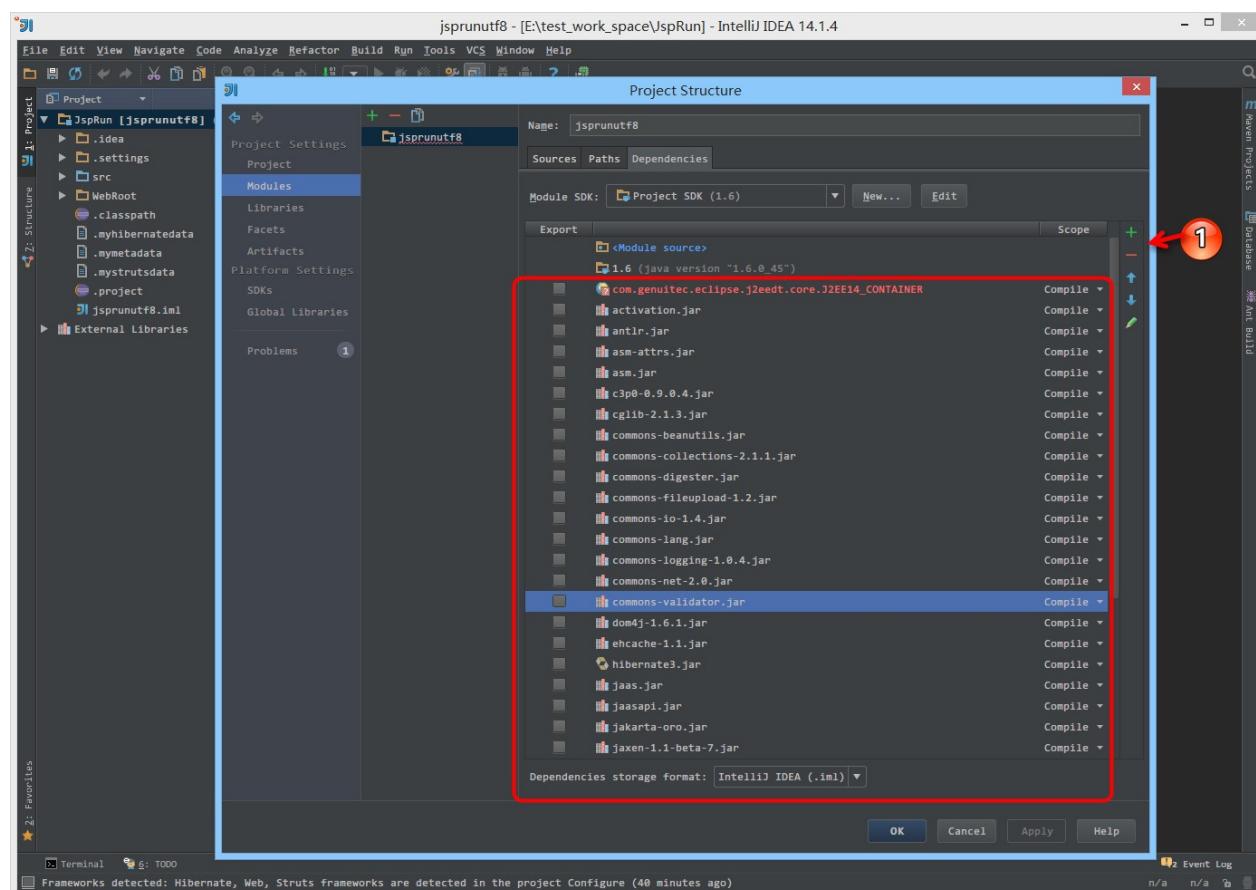


- 如上图红圈所示，我们可以根据对项目的任意目录进行这五种目录类型标注，这个知识点非常非常重要，必须会。

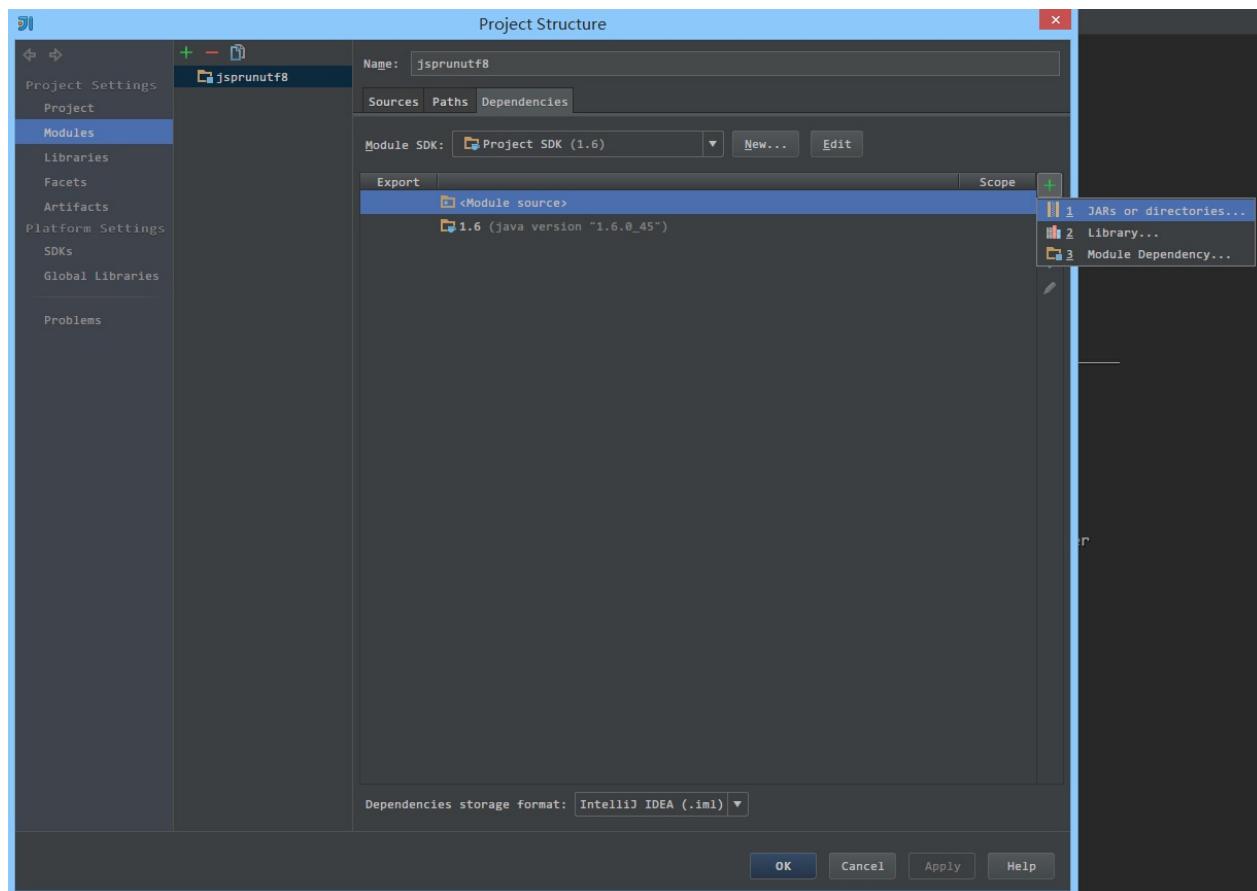
- Sources 一般用于标注类似 `src` 这种可编译目录。有时候我们不单单项目的 `src` 目录要可编译，还有其他一些特别的目录也许我们也要作为可编译的目录，就需要对该目录进行此标注。只有 `Sources` 这种可编译目录才可以新建 Java 类和包，这一点需要牢记。
 - Tests 一般用于标注可编译的单元测试目录。在规范的 maven 项目结构中，顶级目录是 `src`，maven 的 `src` 我们是不会设置为 `Sources` 的，而是在其子目录 `main` 目录下的 `java` 目录，我们会设置为 `Sources`。而单元测试的目录是 `src - test - java`，这里的 `java` 目录我们就会设置为 `Tests`，表示该目录是作为可编译的单元测试目录。一般这个和后面几个我们都是在 maven 项目下进行配置的，但是我这里还是会先说说。从这一点我们也可以看出 IntelliJ IDEA 对 maven 项目的支持是比较彻底的。
 - Resources 一般用于标注资源文件目录。在 maven 项目下，资源目录是单独划分出来的，其目录为：`src - main - resources`，这里的 `resources` 目录我们就会设置为 `Resources`，表示该目录是作为资源目录。资源目录下的文件是会被编译到输出目录下的。
 - Test Resources 一般用于标注单元测试的资源文件目录。在 maven 项目下，单元测试的资源目录是单独划分出来的，其目录为：`src - test - resources`，这里的 `resources` 目录我们就会设置为 `Test Resources`，表示该目录是作为单元测试的资源目录。资源目录下的文件是会被编译到输出目录下的。
 - Excluded 一般用于标注排除目录。被排除的目录不会被 IntelliJ IDEA 创建索引，相当于被 IntelliJ IDEA 废弃，该目录下的代码文件是不具备代码检查和智能提示等常规代码功能。
 - 通过上面的介绍，我们知道对于非 maven 项目我们只要会设置 `src` 即可。
- 如上图箭头所示，被标注的目录会在右侧有一个总的概括。其中 `classes` 虽然是 `Excluded` 目录，但是它有特殊性，可以不显示在这里。
 - 如上图标注 1 所示，如果要去掉目录的标记含义，可以点击打叉按钮进行删除。



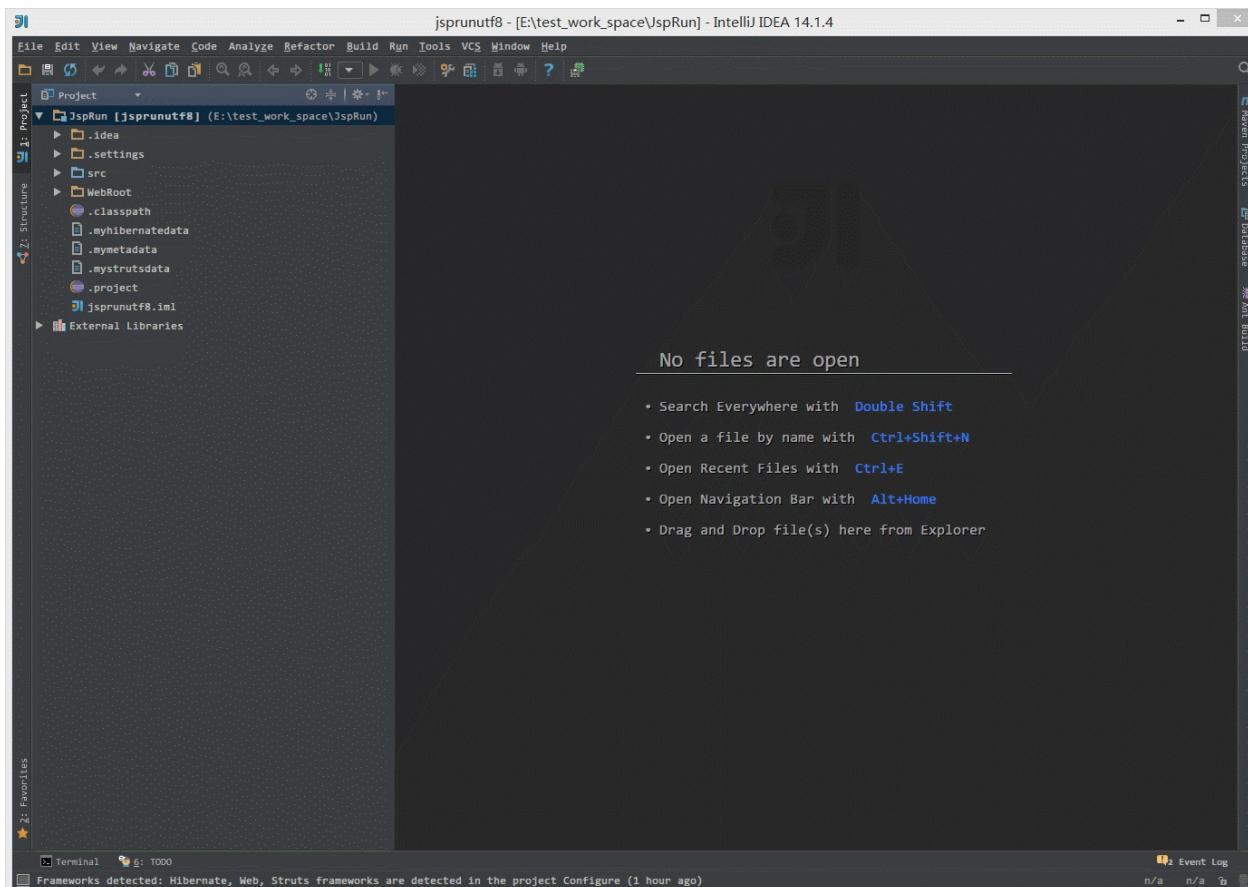
- 如上图标注 1 所示，这是一个单 Module 的项目，Module 的编译输出目录可以跟 Project 的项目输出目录一致，都是 WebRoot 下的 classes 目录。



- 如上图红圈所示，这里显示项目的所有依赖包，其中红色表示 IntelliJ IDEA 无法识别。IntelliJ IDEA 支持一个一个导入依赖包，也支持直接导入一个目录下的所有依赖包。上图的这种就是一个一个具体化的依赖包。除非你的项目各个依赖包有存在依赖顺序关系，不然不建议一个一个引入，这样比较麻烦。所以我这里会先把这些零散的依赖包全部去除掉，然后按目录来引入。
 - 如上图标注 1 所示，依赖包支持这五种操作。
- 加号，表示可以引入新依赖包。
 - 减号，表示可以去除对应的依赖包。
 - 向上箭头，表示依赖包可以向上移动位置。依赖包越上面的表示在项目加载的时候越是优先，所以对于同一个依赖包，不同版本，依赖顺序不同，结果可能也是很有区别的。
 - 向下箭头，表示依赖包可以向下移动位置，原因同上。
 - 笔，表示可以编辑依赖包的名称和路径。

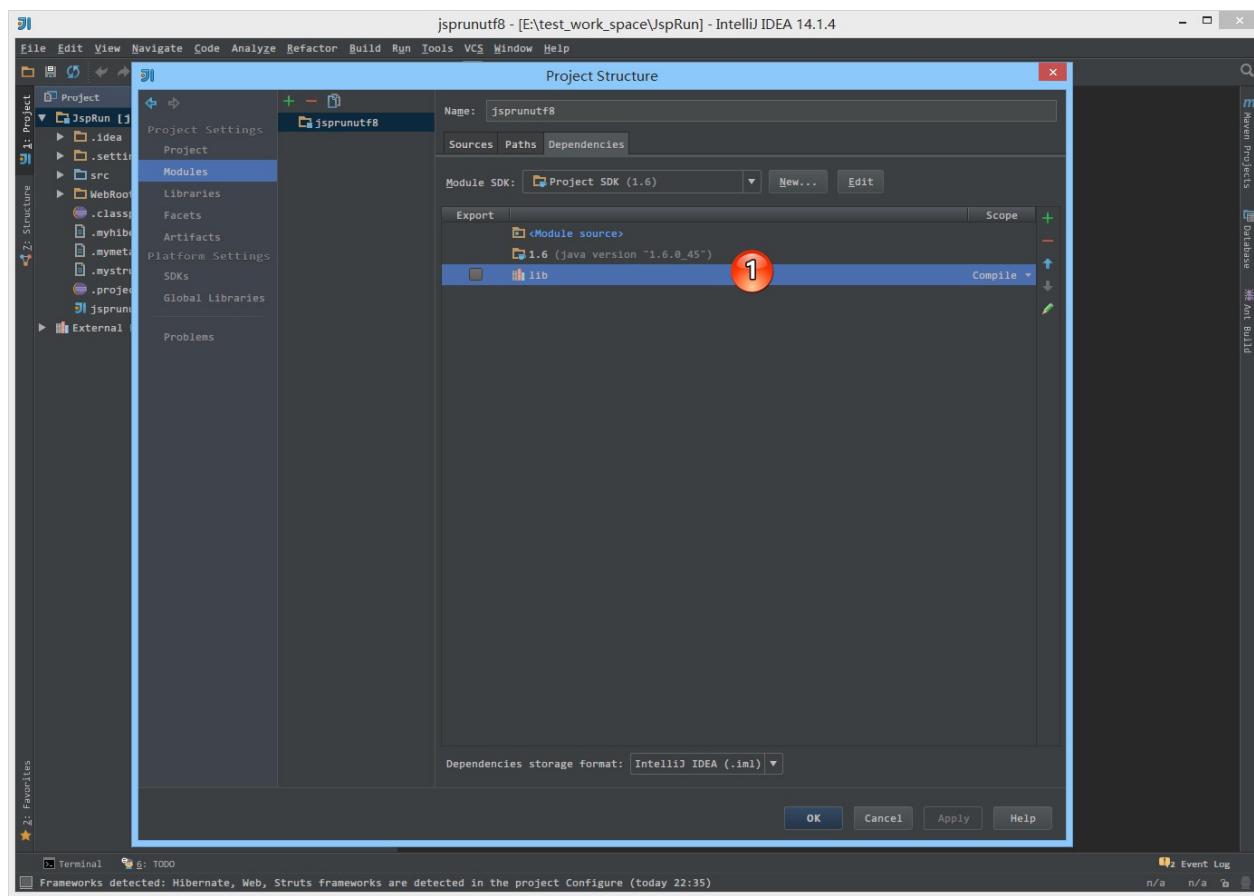


- 如上图所示，引入依赖包支持 jar 包和目录。以及已经导入项目的 Libraries 中的依赖包。多 Module 的项目还可以依赖其他 Module。

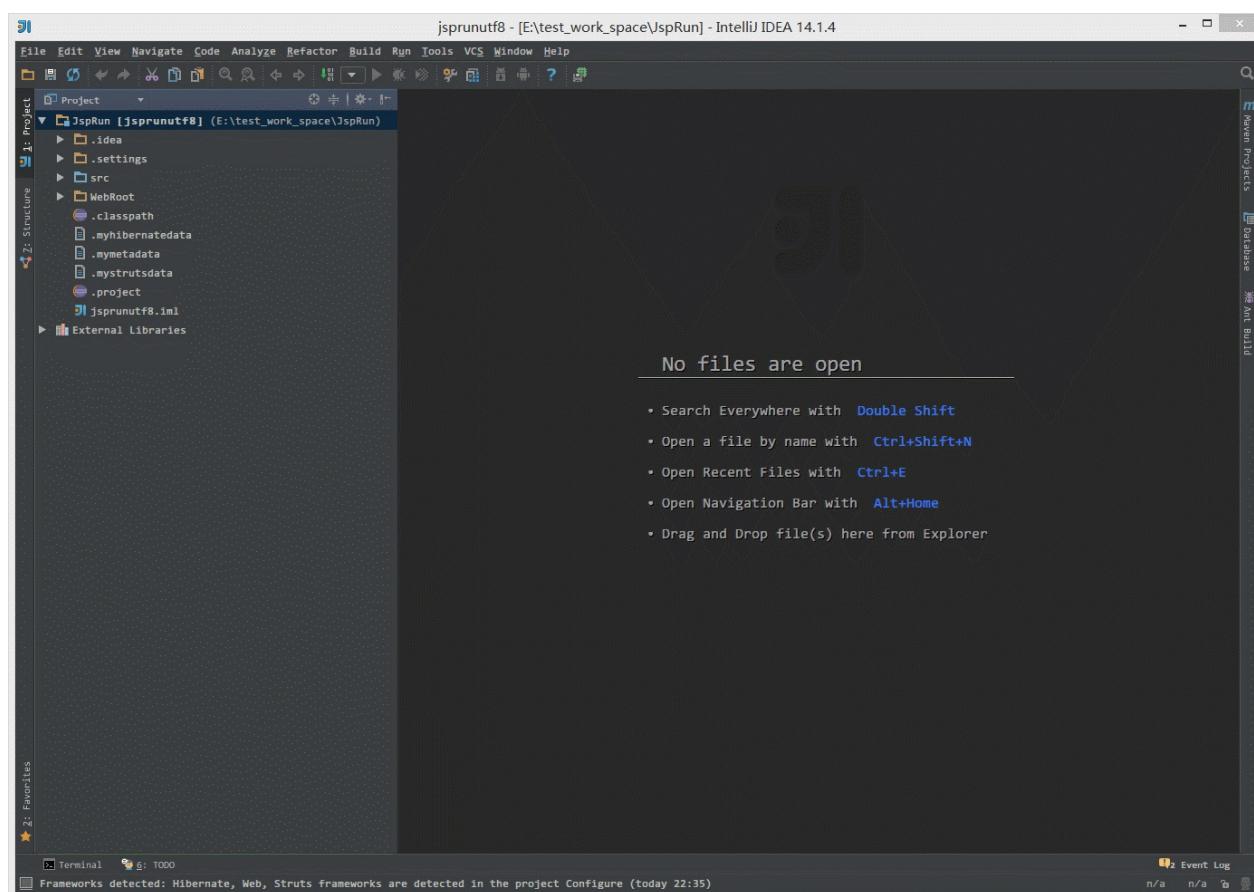


- 由于我们刚刚去掉了红圈中的所有依赖包，所以我们现在还需要导入依赖包。到导入之前我们先把项目的 lib 作为一个总的依赖包进行放置到项目 Libraries 中。如上图 Gif 演示，就是把 lib 目录转换成一个依赖包。这样的好处是，当我们项目有新添加依赖包，我们只要放置在 lib 目录下即可自动被项目引入，原因就是因为这里引的是目录，而不是一个一个依赖包。

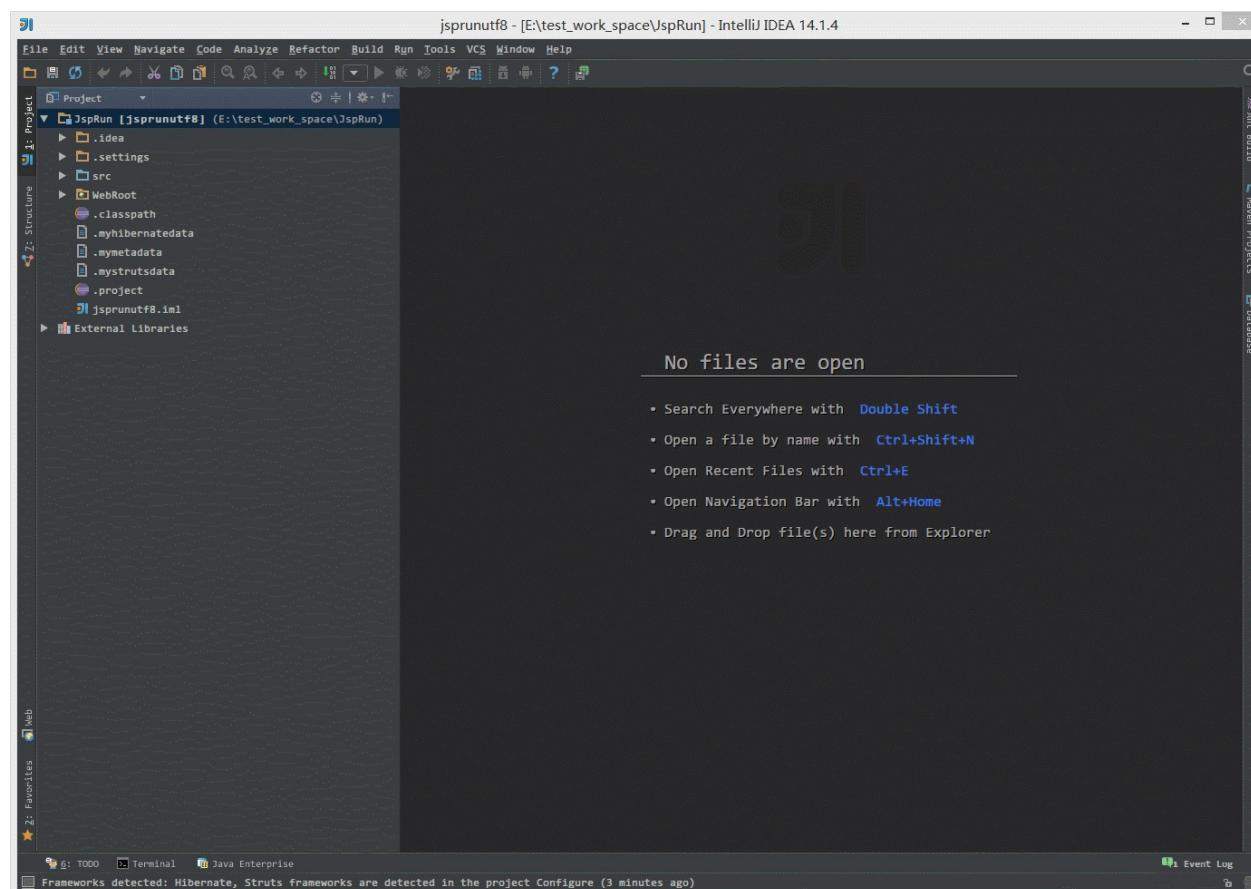
22. Eclipse 的 Java Web 项目环境搭建



- 如上图标注 1 所示，引入刚刚放置好的 Libraries 下 lib 依赖包。

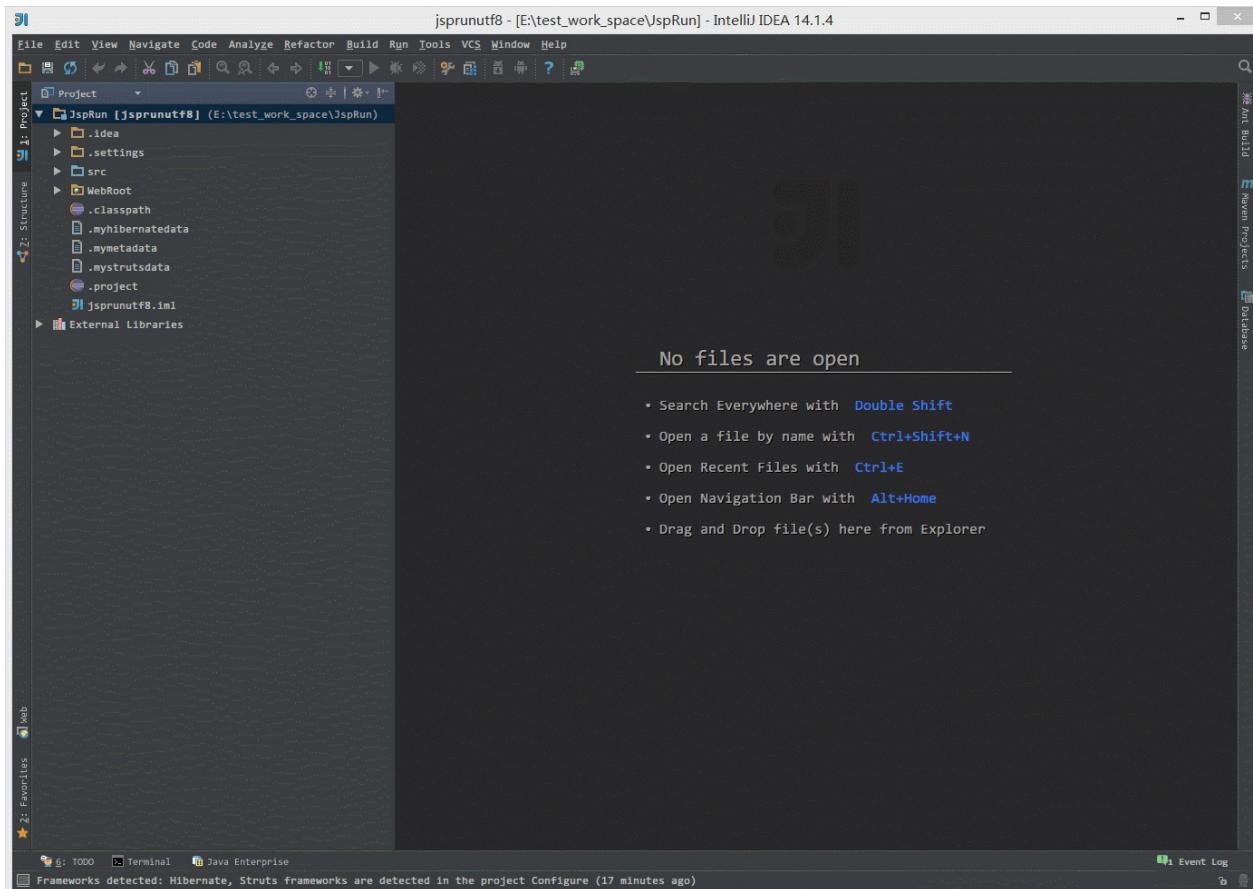


- 对于传统的 Java Web 项目，我们一般还需要指定配置 web.xml 位置。如上图 Gif 演示，这个一般在 Facts 中进行配置。Facts 可以理解为用于配置项目的框架区，在这里管理项目使用的框架配置文件。这个是 IntelliJ IDEA 特有的一个配置点。
- 除了 web.xml 一般我们要配置外，其他一些框架，即使我们不在这里配置也是不会影响项目的运行的，但是是有缺点的。比如我们项目中一般都是有 Spring 框架的，而 Spring 是有很多配置文件的，如果我们在那里进行了配置，那你会发现 IntelliJ IDEA 编辑区底部会多出现几个跟 Spring 项目的设置区，原因就是你告诉了 IntelliJ IDEA，你的项目使用了 Spring 框架，那 IntelliJ IDEA 就会出现其对应的配置功能给你。Hibernate 等其他框架道理一样。

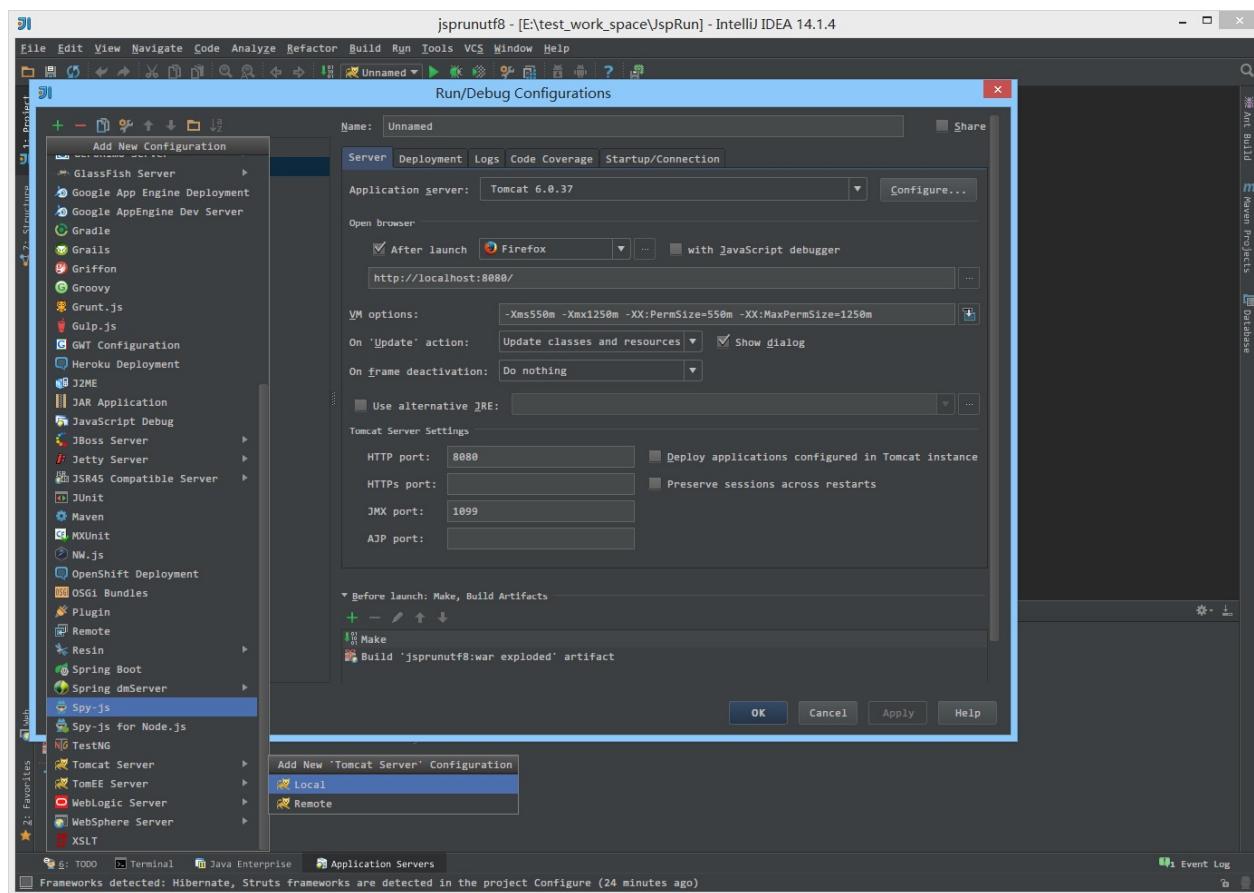


- Artifacts 也是 IntelliJ IDEA 特有的一个概念，我们可以理解这里是配置项目最终输出结果的地方。比如 Java Web 项目我们一般必备一个配置就是要配置成一个 war 包展开的方式，这样容器才能运行该项目。上图 Gif 演示的就是配置一个 war 包展开的输出结构。其结构是由前面的几项配置决定，所以如果前面的 Module 配置没有多大问题，这里可以省去一些配置步骤。但是有些时候我们也是需要做一些修改，比如此项目的输出目录默认生成的是错误，我改为了项目中的 webRoot 目录。
- 该 Artifacts 配置，等下在配置 Tomcat 的时候也会引用到，所以这里需要重点注意下。

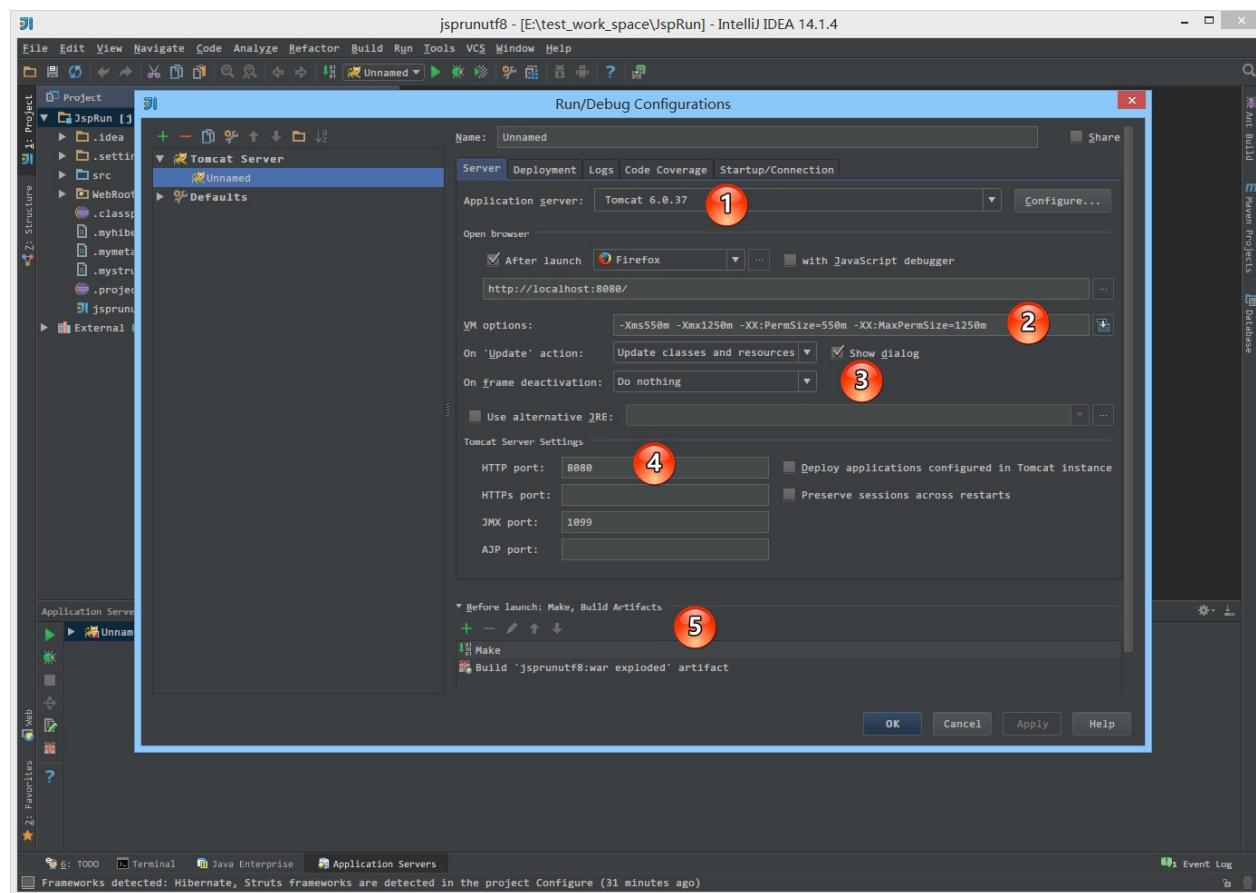
Tomcat 配置



- 如上图 Gif 所示，为项目整体的一个 Tomcat 配置过程，有些需要特别讲解的，我们将在下面进行分解。



- 如上图弹出层所示，IntelliJ IDEA 支持目前市场上主流的应用容器，所以基本上一个 IntelliJ IDEA 足够我们开发常见的项目。但是这里有一点需要提醒的，IntelliJ IDEA 支持这些容器但是不等同于帮我们自带了这些容器的文件，所以上面的 Gif 演示中，原本我只有引入 Tomcat 7，但是为了这个项目我又引入了 Tomcat 6，而引入的 Tomcat 6 我只是指定了其存放的目录位置 IntelliJ IDEA 自动会识别到。



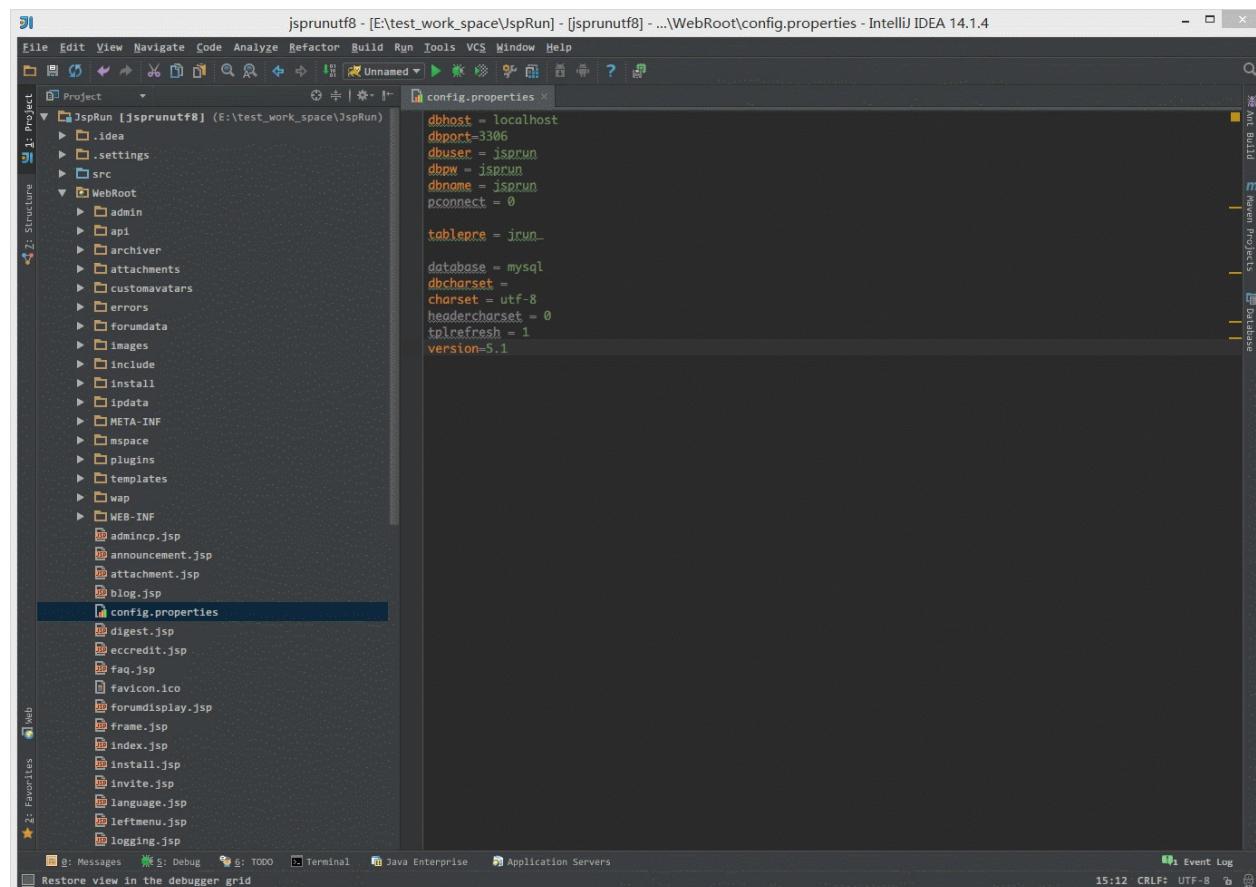
- 如上图标注 1 所示，我们可以切换随时为项目切换不同的容器。
- 如上图标注 2 所示，我们可以指定给运行的容器设置 VM 参数。
- 如上图标注 3 所示，这分别是在两种命令下的触发的事件。这个是 IntelliJ IDEA 特有的，也是重点。
- On Update action** 当我们按 `Ctrl + F10` 进行容器更新的时候，可以根据我们配置的这个事件内容进行容器更新。其中我选择的 `Update classes and resources` 事件是最常用的，表示我们在按 `Ctrl + F10` 进行容器更新的时候，我们触发更新编译的类和资源文件到容器中。在 Debug 模式下，这个也就是所谓的 `HotSwap`。这是这种热部署没有 JRebel 插件那样好用。
- On frame deactivation** 当我们切换 IntelliJ IDEA 到浏览器的时候进行指定事件更新，这个一般是因为 Web 开发的时候，我们需要经常在 IntelliJ IDEA 和各个浏览器之间来回切换测试，所以才有了这种需求。IntelliJ IDEA 是为了帮我们在做这种无聊切换的时候做一些指定事情。当然了，如果切换过于频繁，这个功能还是很耗资源的，所以我设置的是 `Do nothing` 表示切换的时候什么都不做。
- 如上图标注 4 所示，默认 Tomcat 的 HTTP 端口是 8080，如果你需要改其端口可以在这里设置。
- 如上图标注 5 所示，这个知识点在前面的文章已经有讲过了。这里表示在 Tomcat 容器运行前做什么事情，这里分别了：Make 和 Build Artifacts 操作。如上面 Gif 演示，这里的 `Build Artifacts` 是我们在 Deployment 选项卡中添加了 Artifact 之后自动出现的。

Tomcat 启动

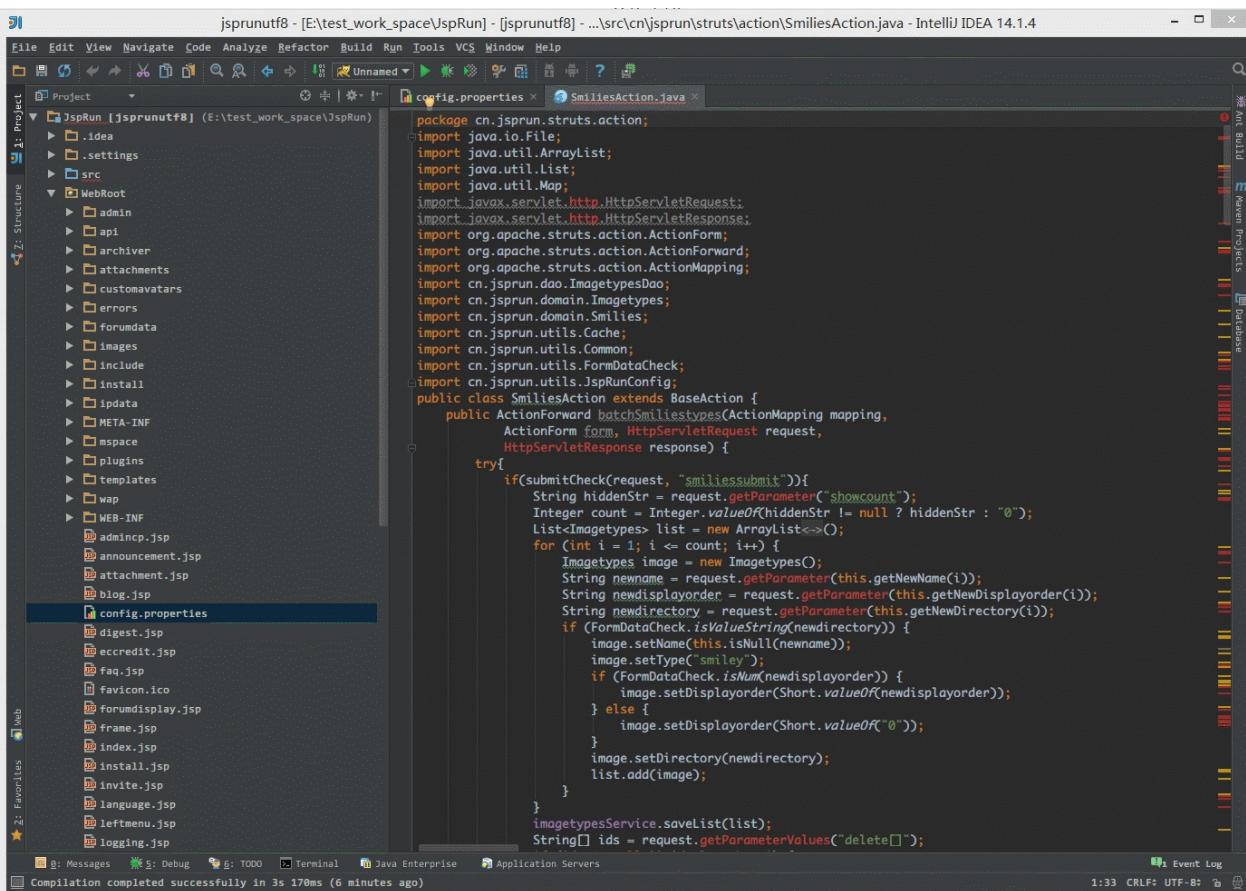
- 通过上面的配置，IntelliJ IDEA 环境配置算是配置好了，现在配置下该项目相关的。
- 打开你的 Mysql，执行下面脚本，创建一个新的数据库和用户：

```
CREATE DATABASE `jsprun` CHARACTER SET utf8;
CREATE USER 'jsprun'@'localhost' IDENTIFIED BY 'jsprun';
GRANT ALL PRIVILEGES ON jsprun.* TO 'jsprun'@'localhost';
FLUSH PRIVILEGES;
```

- 切换到上面新建的 `jsprun` 数据库中执行项目中这个数据脚本，文件位置：`JspRun\WebRoot\install\jsprun_zh_CN.sql`。
- 修改 `JspRun\WebRoot\config.properties` 文件中的几个属性为下面内容：
- `dbuser = jsprun`
- `dbpw = jsprun`

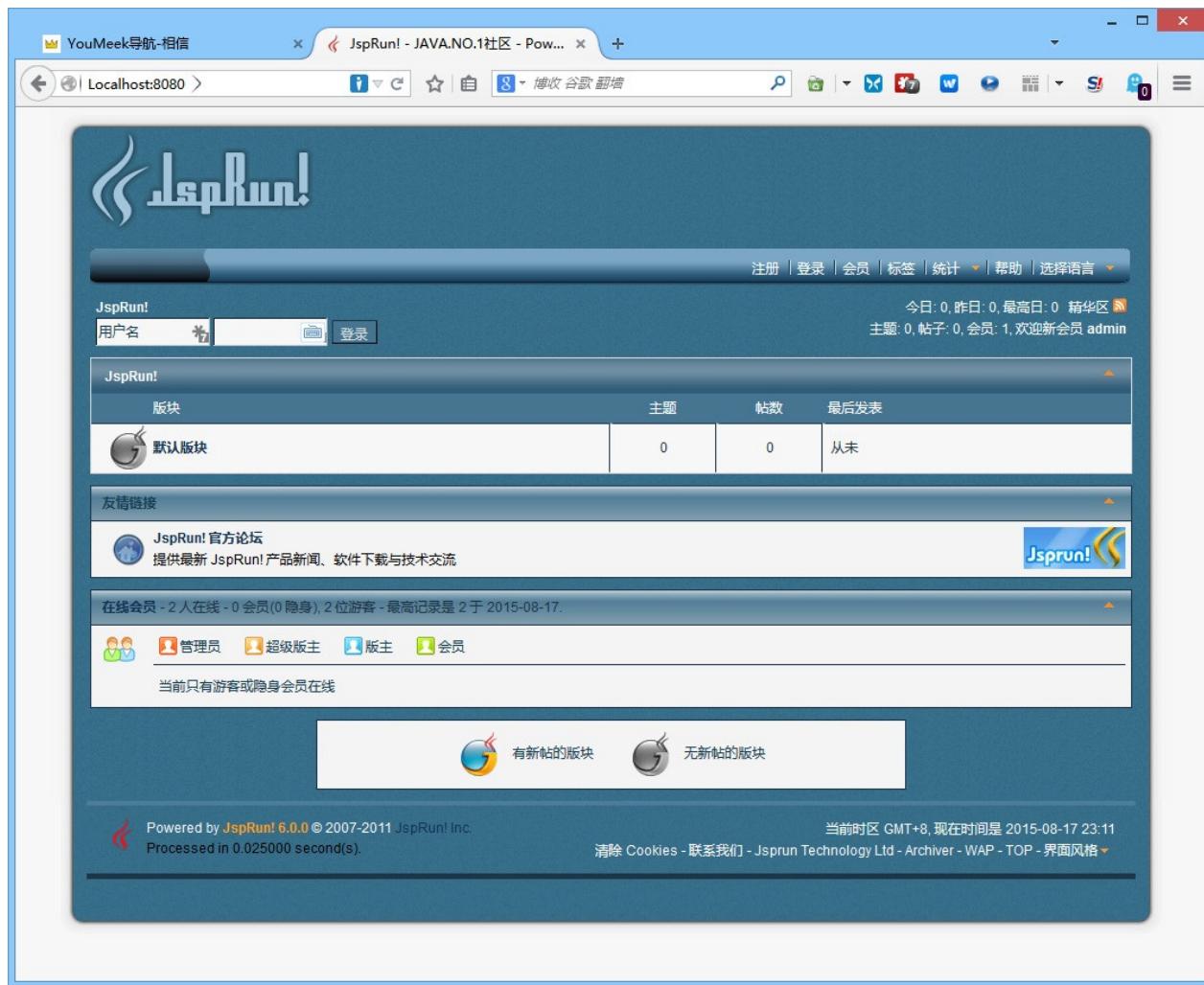


- 如上图 Gif 演示，我们缺少引入 Tomcat 的依赖包。



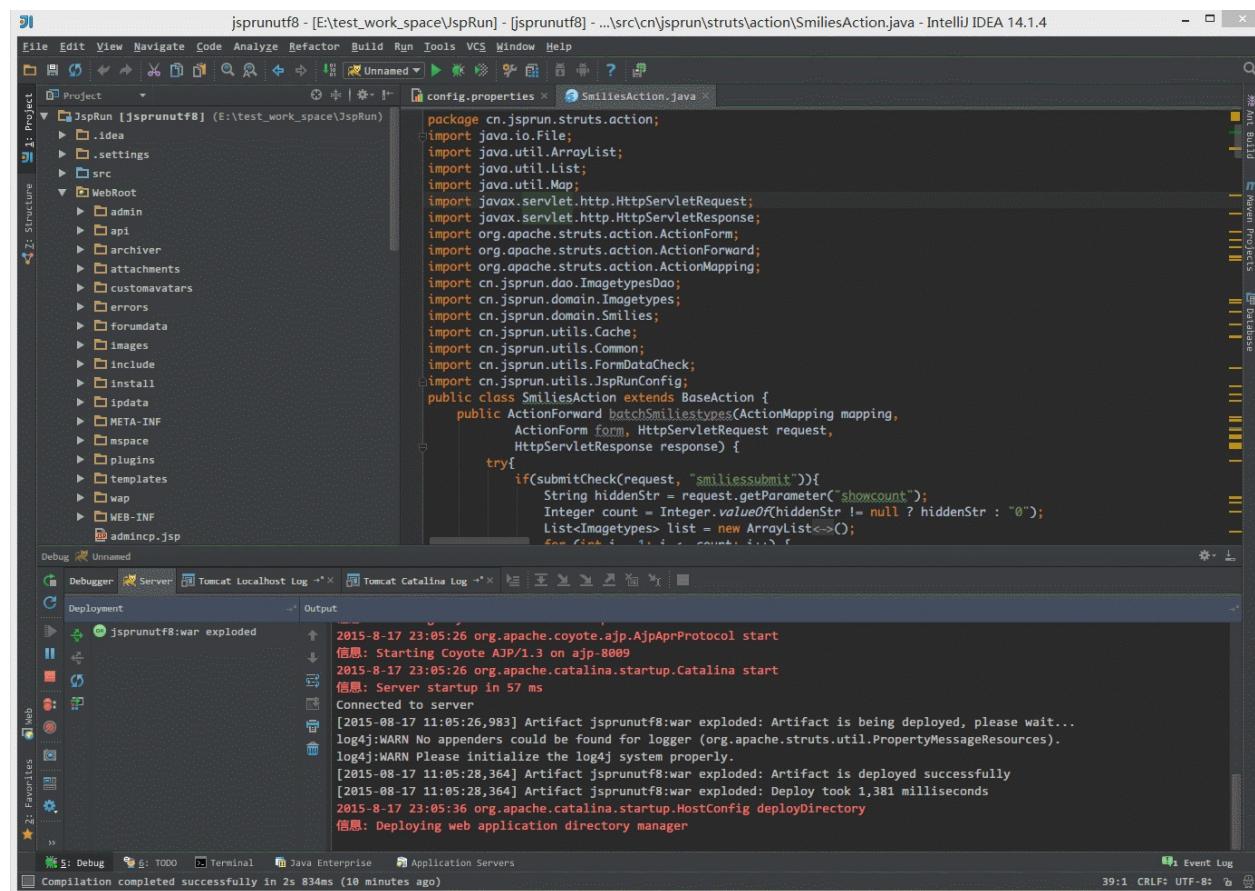
```
package cn.jsprun.struts.action;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import cn.jsprun.dao.ImagetypesDao;
import cn.jsprun.domain.Smilies;
import cn.jsprun.utils.Cache;
import cn.jsprun.utils.Common;
import cn.jsprun.utils.FormDataCheck;
import cn.jsprun.utils.JspRunConfig;
public class SmiliesAction extends BaseAction {
    public ActionForward bgetSmilietypesActionMapping(
        ActionForm form, HttpServletRequest request,
        HttpServletResponse response) {
        try{
            if(submitCheck(request, "smiliessubmit")){
                String hiddenStr = request.getParameter("showcount");
                Integer count = Integer.valueOf(hiddenStr != null ? hiddenStr : "0");
                List<Imagetypes> list = new ArrayList<>();
                for (int i = 1; i <= count; i++) {
                    Imagetypes image = new Imagetypes();
                    String newname = request.getParameter(this.getNewName(i));
                    String newdisplayorder = request.getParameter(this.getNewDisplayorder(i));
                    String newdirectory = request.getParameter(this.getNewDirectory(i));
                    if (FormDataCheck.isNotEmptyString(newdirectory)) {
                        image.setName(this.isNull(newname));
                        image.setType("smiley");
                        if (FormDataCheck.isNum(newdisplayorder)) {
                            image.setDisplayorder(Short.valueOf(newdisplayorder));
                        } else {
                            image.setDisplayorder(Short.valueOf("0"));
                        }
                        image.setDirectory(newdirectory);
                        list.add(image);
                    }
                }
                imagetypesService.saveList(list);
                String[] ids = request.getParameterValues("delete[]");
            }
        } catch (Exception e) {
            log.error(e.getMessage());
        }
    }
}
```

- 如上图 Gif 演示，我们引入 Tomcat 的依赖包之后，可以运行该项目。



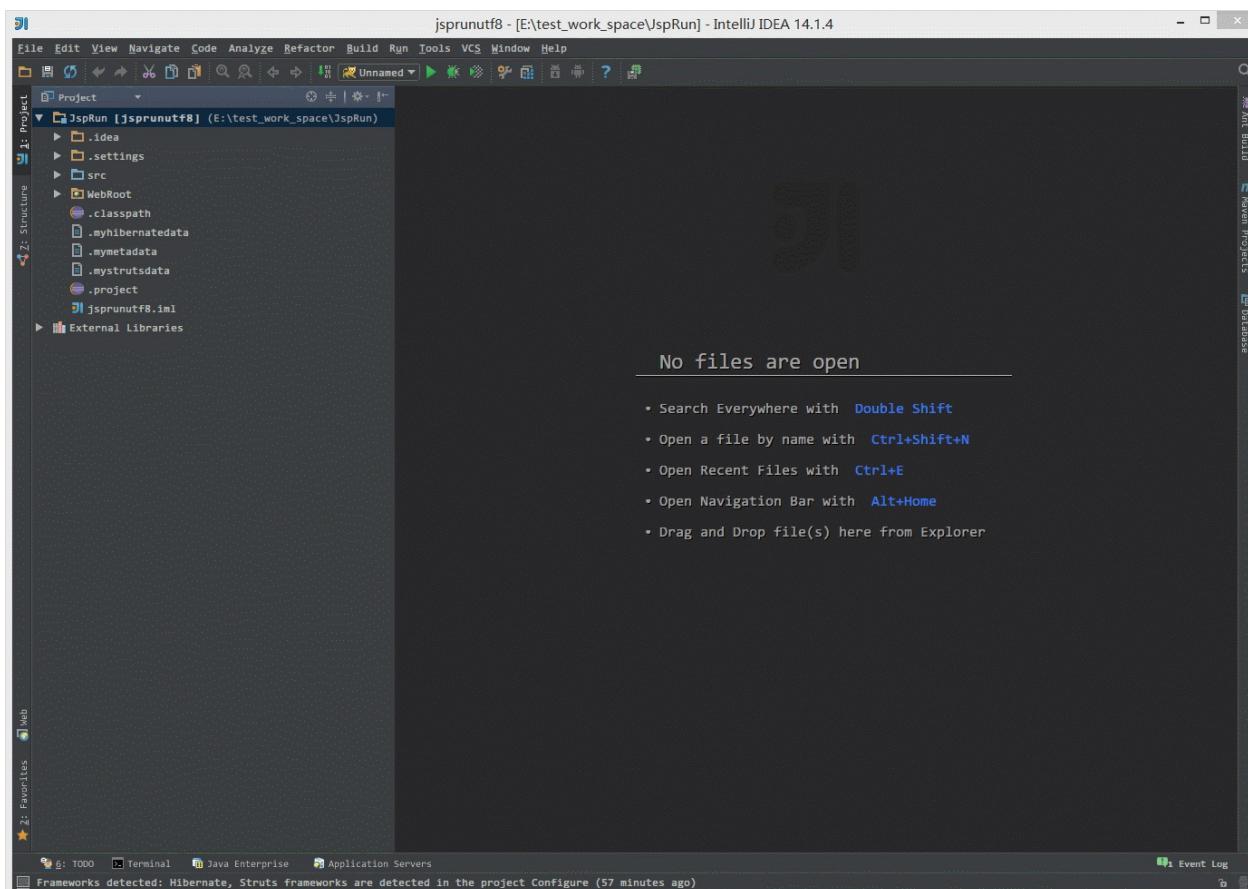
- 如上图所示，为最终项目运行效果图。

Tomcat 停止



- 如上图所示，停止按钮是要按两次，第一按完出现一个骷髅头并不是停止，需要再点击一次。
- 有时候即使点了两次，Tomcat 容器也不一定能完全停掉，这时候很容易出现端口被占用的操作，这时候你需要打开系统的资源管理器，手动 kill 系统上所有的 java 进程。

输出 war 压缩包



- 如上图 Gif 所示，除了在 Artifacts 中需要配置，还需要在容器中也跟着配置，这样在启动容器的时候才会输出一个 war 压缩包。
- 通过配置，我们也知道 war 的压缩包本质是根据展开的 war 输出包进行压缩得来。

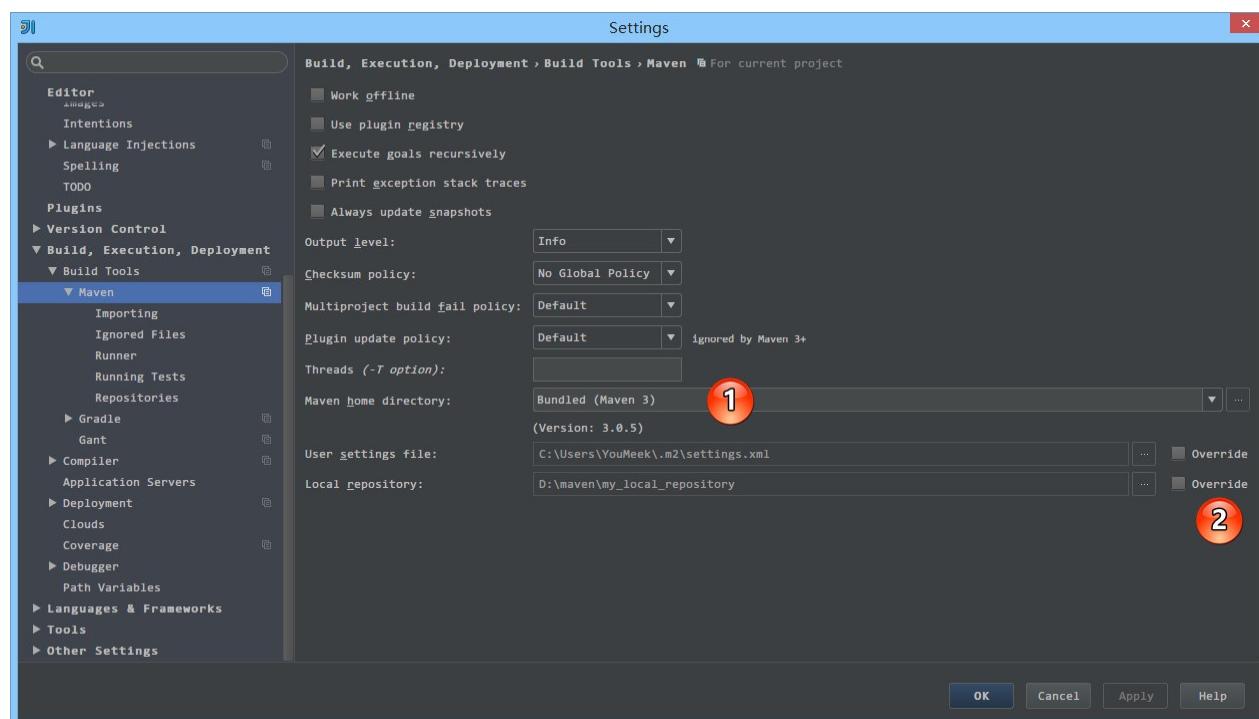
Maven 项目介绍

学习前提

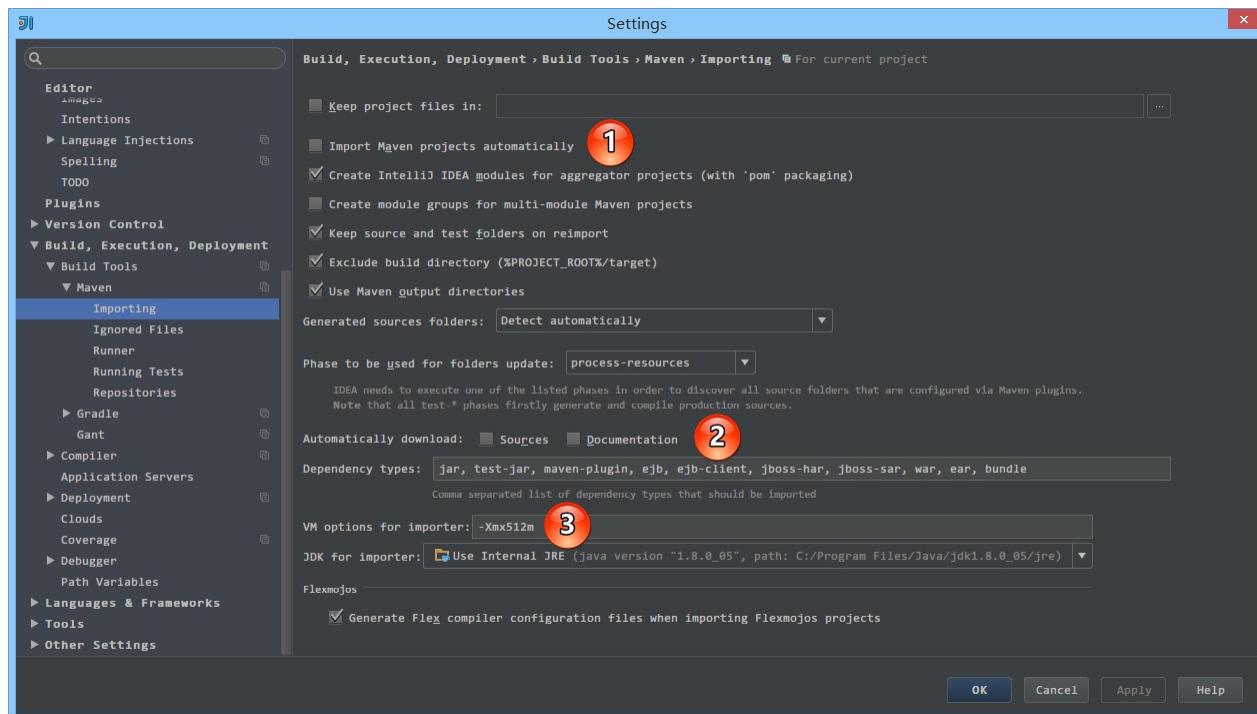
相对于传统的项目，Maven 下管理和构建的项目真的非常好用和简单，所以这里也强调下，尽量使用此类工具进行项目构建。

学习本讲还有一个前提：你必须会 Maven 相关知识点，Maven 相关知识点是不在本专题的讲解范围里面的，所以请自己私下进行学习。如果愿意你也可以看我过去整理的一份材料：<http://www.youmeek.com/intellij-idea-part-xviii-maven/>

Maven 常用设置介绍

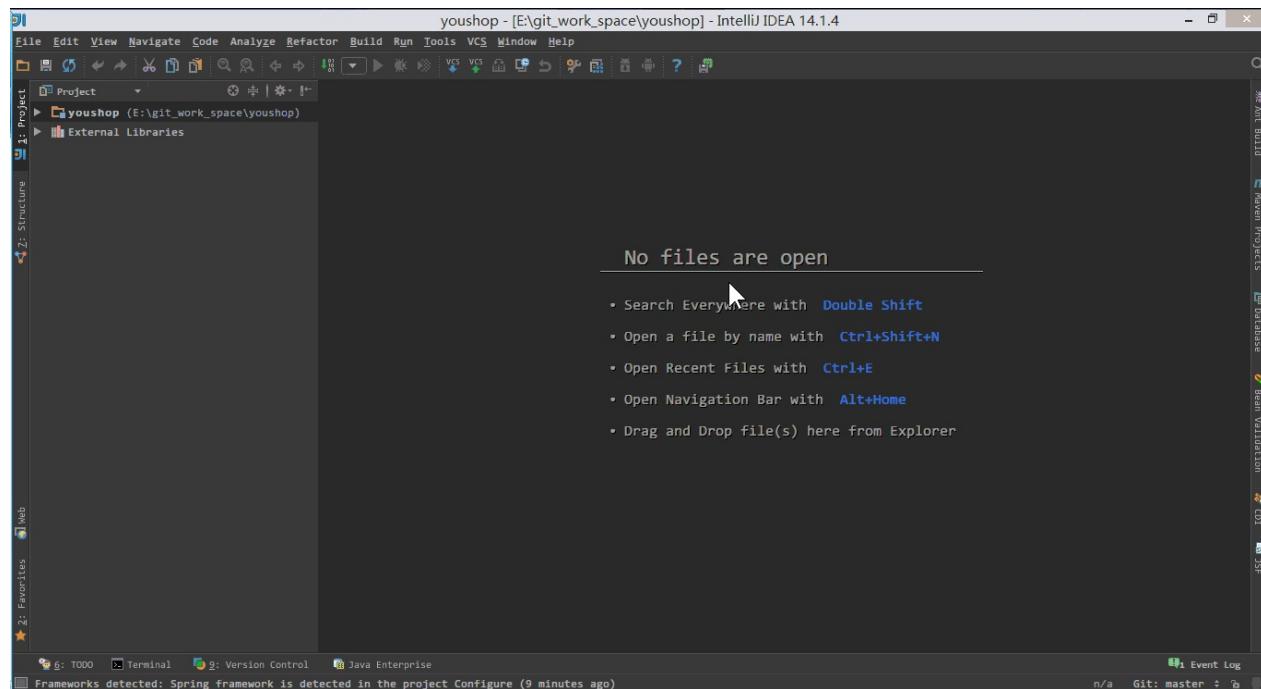


- 如上图标注 1 所示，我们可以指定我们本地 Maven 的安装目录所在，因为我已经配置了 `MAVEN_HOME` 系统参数，所以直接这样配置 IntelliJ IDEA 是可以找到的。但是假如你没有配置的话，这里可以选择你的 Maven 安装目录。
- 如上图标注 2 所示，我们还可以指定 Maven 的 `settings.xml` 位置和本地仓库位置。



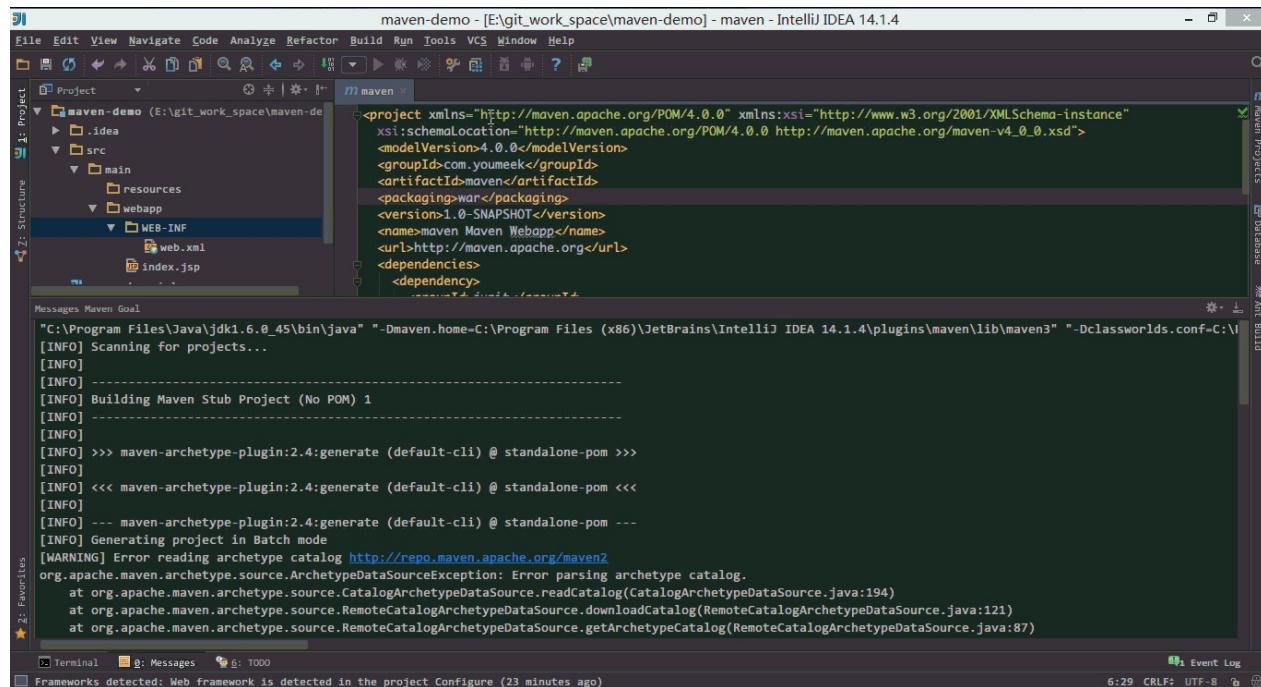
- 如上图标注 1 所示，`Import Maven projects automatically` 表示 IntelliJ IDEA 会实时监控项目的 `pom.xml` 文件，进行项目变动设置。
- 如上图标注 2 所示，在 Maven 导入依赖包的时候是否自动下载源码和文档。默认是没有勾选的，也不建议勾选，原因是这样可以加快项目从外网导入依赖包的速度，如果我们需要源码和文档的时候我们到时候再针对某个依赖包进行联网下载即可。IntelliJ IDEA 支持直接从公网下载源码和文档的。
- 如上图标注 3 所示，可以设置导入的 VM 参数。一般这个都不需要主动改，除非项目真的导入太慢了我们再增大此参数。

Maven 骨架创建 Java Web 项目



- 如上图 Gif 演示，根据已有的 Maven 骨架进行 Java Web 项目创建。其中需要特别注意的是，在创建项目过程中 Maven 会去外网中央仓库中下载对应的依赖或是组件，这个过程根据自身网络环境决定其快慢。如果出现无法下载的情况请自备 VPN 或者通过修改 Maven 配置文件 `settings.xml` 切换国内的中央仓库。
- 由于我已经试过多次了，所以 Gif 演示中我可以快速创建好，但是你那边不一定是这种情况的。

启动 Java Web 项目

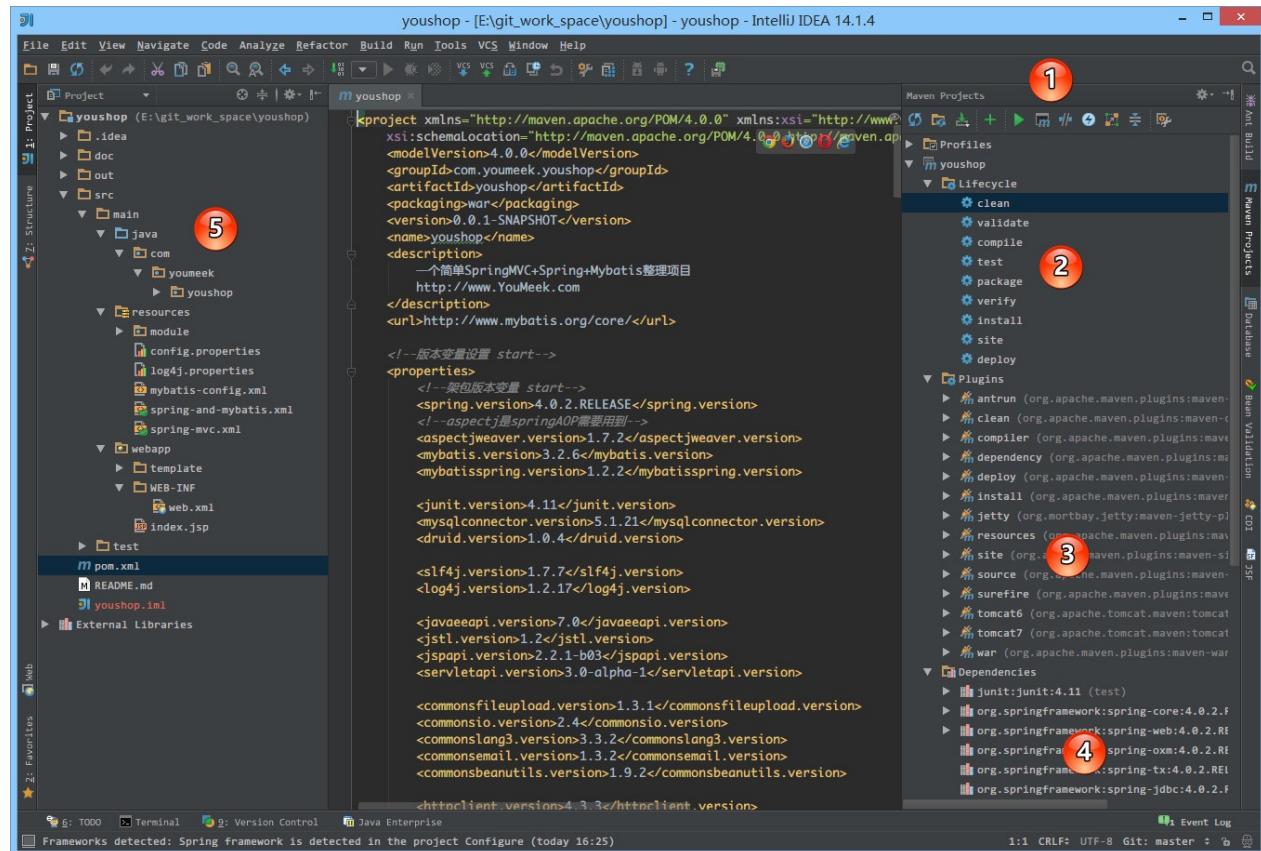


- 如上图 Gif 演示，配置好项目之后，启动 Tomcat 容器可以良好运行。

Maven 组件来管理项目

我们已经了解了如何通过 Maven 骨架生成一个最简单的 Java Web 项目，可是我们还是使用了 IntelliJ IDEA 的项目管理功能进行 Maven 项目的管理和构建。一般 Maven 的项目我们都可以脱离 IntelliJ IDEA 的项目配置功能进行独立的管理和构建的，接下来我们就讲如何通过 IntelliJ IDEA 提供的 Maven 管理工具进行项目的管理和构建。

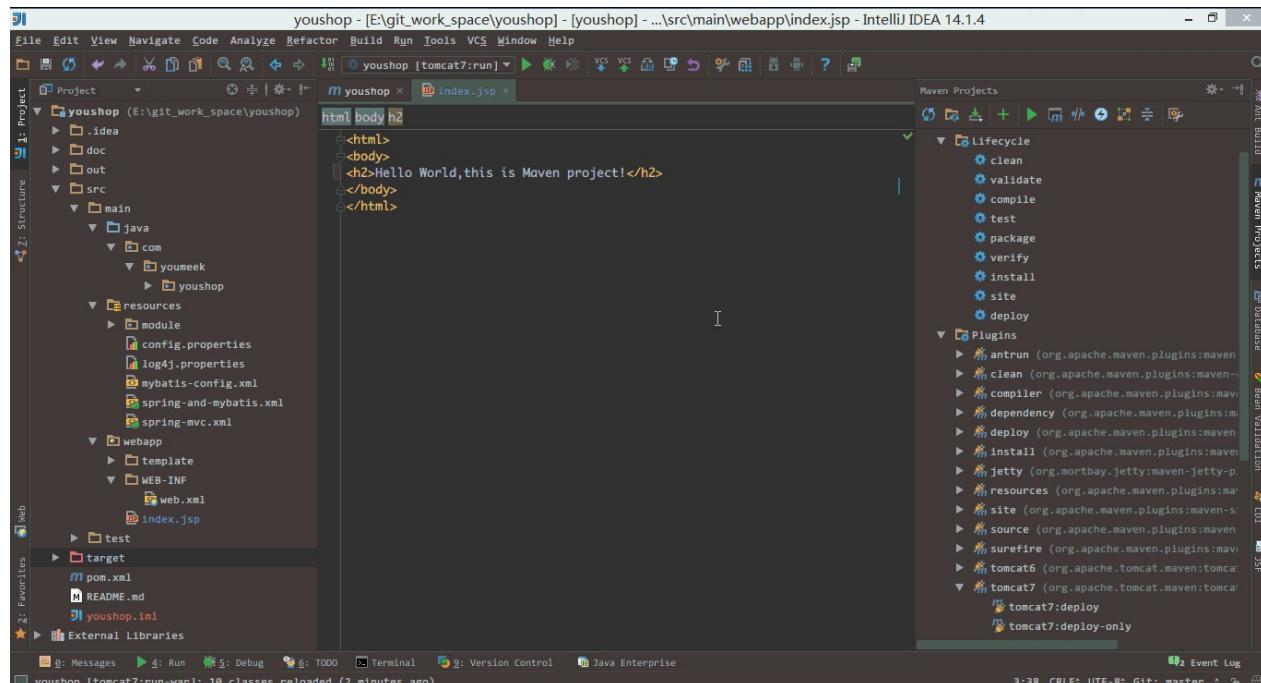
Maven 组件界面介绍



- 如上图标注 1 所示，为常用的 Maven 工具栏，其中最常用的有：

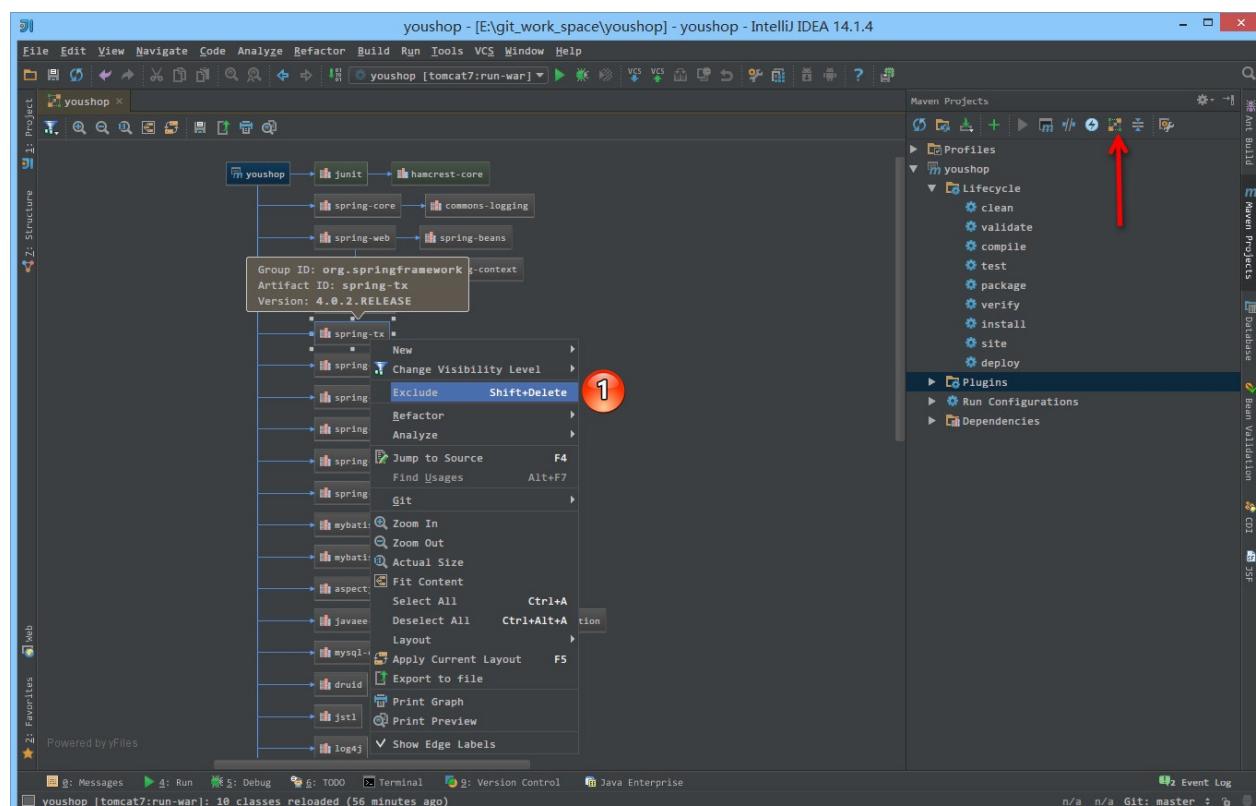
- 第一个按钮：Reimport All Maven Projects 表示根据 pom.xml 重新载入项目。一般当我们在 pom.xml 添加了依赖包或是插件的时候，发现标注 4 的依赖区中没有看到最新写的依赖的话，可以尝试点击此按钮进行项目的重新载入。
- 第六个按钮：Execute Maven Goal 弹出可执行的 Maven 命令的输入框。有些情况下我们需要通过书写某些执行命令来构建项目，就可以通过此按钮。
- 第九个按钮：Show Dependencies 显示项目依赖的结构图，可以方便我们直观项目的依赖包情况。这个功能有些具体的操作下面会专门进行讲解。
- 如上图标注 2 所示，常用的 Maven 生命周期的命令，通过双击对应的命令来执行项目编译、打包、部署等操作。
- 如上图标注 3 所示，为我们在 pom.xml 中配置的插件列表，方便调用插件。
- 如上图标注 4 所示，为我们在 pom.xml 中配置的依赖包列表。
- 如上图标注 5 所示，为常见的 Java Web 在 Maven 下的一个项目结构。

Maven 的 Tomcat 插件运行项目

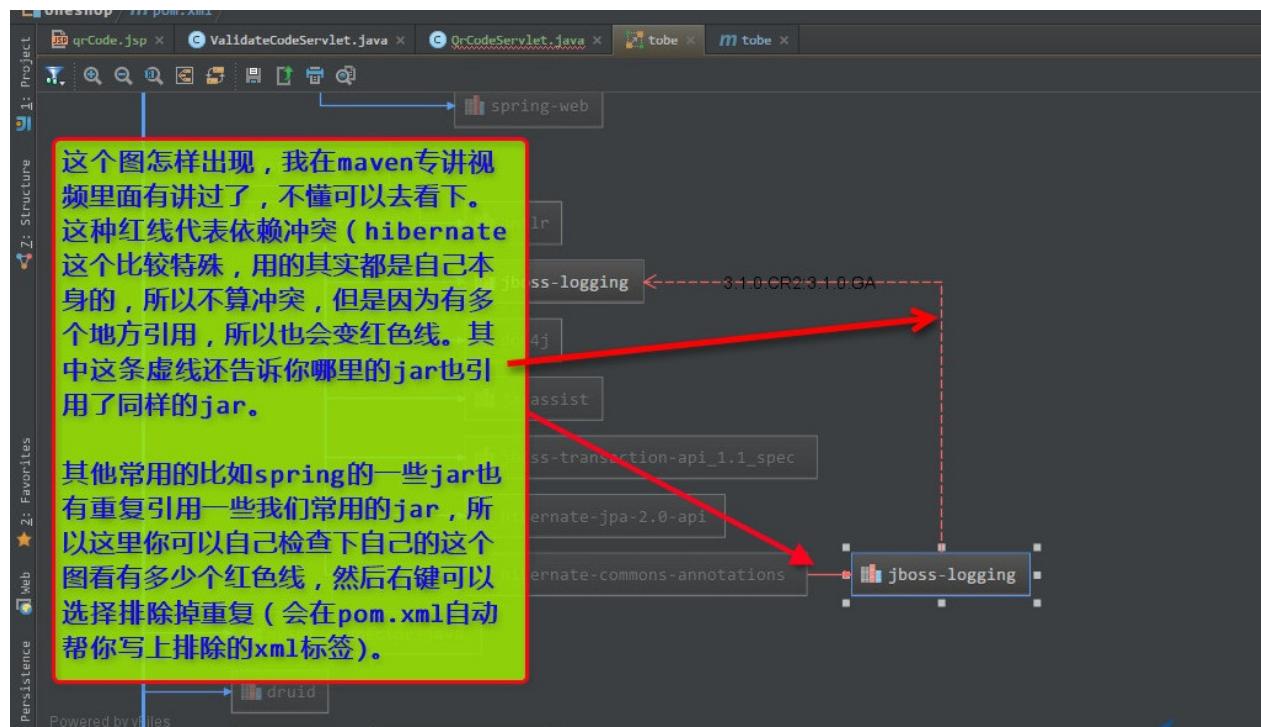


- 如上图 Gif 演示，通过 Maven 生命周期的命令进行编译和打包，及用 Maven 的 Tomcat 插件运行项目。
- 虽然我们可以通过 Maven 组件进行项目的管理，但是这并不等同于我们可以完全抛弃 IntelliJ IDEA 的项目设置，比如我们在 `pom.xml` 文件中设置了 JDK 编译版本是 1.7，但是在项目的 `ctrl + Shift + Alt + S` 配置中，我们配置的 JDK 是 1.8，那即使我们用 Maven 的编译工具或命令进行编译也是会调用 1.8 的。还有即使我们在 `Ctrl + Shift + Alt + S` 配置中没有配置 Artifacts，在我们运行 Maven 的 Tomcat 插件的时候也会自动帮我们生成的。

Maven 快速排查依赖包冲突



- 如上图箭头所示，点击此按钮会出现左边的项目依赖结构图，通过此图我们可以很好的观察项目的依赖依赖情况。
- 如上图标注 1 所示，我们可以直接在依赖结构图上编辑依赖，常用的就是此排除功能。



- 如上图描述，一般我们在出现红色线的时候是都要进行排除的，防止出现因为版本不同的依赖包造成代码无法编译。

Maven 的单模块 / 多模块之 Spring MVC + Spring + Mybatis 项目讲解

初衷

- 为了更加明了地展示 IntelliJ IDEA 的设置，本章教程为视频教程
- 本章展示 IntelliJ IDEA 高度集成化的效果，降低 IntelliJ IDEA 入门时间

视频下载

- 单模块的 Spring MVC + Spring + Mybatis 讲解（基于 IntelliJ IDEA）
 - 百度云盘：<http://pan.baidu.com/s/1dEuxWh7>
 - 360 云盘（6d49）：<https://yunpan.cn/cY444GphNgMe3>
- 多模块的 Spring MVC + Spring + Mybatis 讲解（基于 IntelliJ IDEA）
 - 百度云盘：<http://pan.baidu.com/s/1hr0x6sc>
 - 360 云盘（e319）：<https://yunpan.cn/cY4INmfJn8yvm>

开发环境

- JDK 7（理论上支持 JDK 6、JDK 7、JDK 8）
- Mysql 5.6
- Maven 3.1.1
- Tomcat 7
- Git 2.7.0.2-64-bit
- IntelliJ IDEA 15.0.4
- 所有编码：UTF-8

演示内容

- 环境相关：
 - Maven 环境说明：<http://code.youmeek.com/2016/03/09/2016/03/Maven/>
 - 我的 Maven 环境分享下载：<http://pan.baidu.com/s/1bnPZU2b>
 - 建议你也跟我一样直接解压在 D 盘根目录，这样其他就不需要设置了
 - Git 环境的说明：<http://code.youmeek.com/2016/02/28/2016/02/Hexo/>
 - IntelliJ IDEA 基础教程系列：<https://github.com/judasn/IntelliJ-IDEA-Tutorial>

- IntelliJ IDEA 设置：
 - Fork 单模块项目：<https://github.com/judasn/Basic-Single-Module-SSM>
 - Fork 多模块项目：<https://github.com/judasn/Basic-Multi-Module-SSM>
 - Checkout 项目并导入
 - IntelliJ IDEA Maven 设置
 - IntelliJ IDEA 文件编码设置
 - IntelliJ IDEA Mybatis 插件安装（该插件收费）：<https://plugins.jetbrains.com/plugin/7293?pr=fee>
-

- 项目设置：
 - 项目 JDK 设置
 - 项目 Facet 加入 Spring 配置
-

- 代码相关：
 - 简单讲解 pom.xml 文件
 - 用 IntelliJ IDEA 的 Database 初始化数据库
 - 单元测试
 - 启动 Tomcat 加上 Make Project 事件
 - 访问 Controller 演示 Debug
 - 讲解 Controller 中代码左侧的各个按钮效果
 - JSP 页面直接点击请求地址直接跳转到 Controller
 - 静态资源映射特别提醒下，比如你做图片上传等等，如果你没有映射好可能都会遇到 404
 - 查看 Druid 提供监控
 - 演示用 Mybatis 插件自动生成代码

Maven 的单模块 / 多模块之 Spring MVC + Spring + Spring Data JPA 项目（基于 IntelliJ IDEA）

初衷

- 本章展示 IntelliJ IDEA 高度集成化的效果，降低 IntelliJ IDEA 入门时间
- 欢迎来到 IntelliJ IDEA 世界

项目

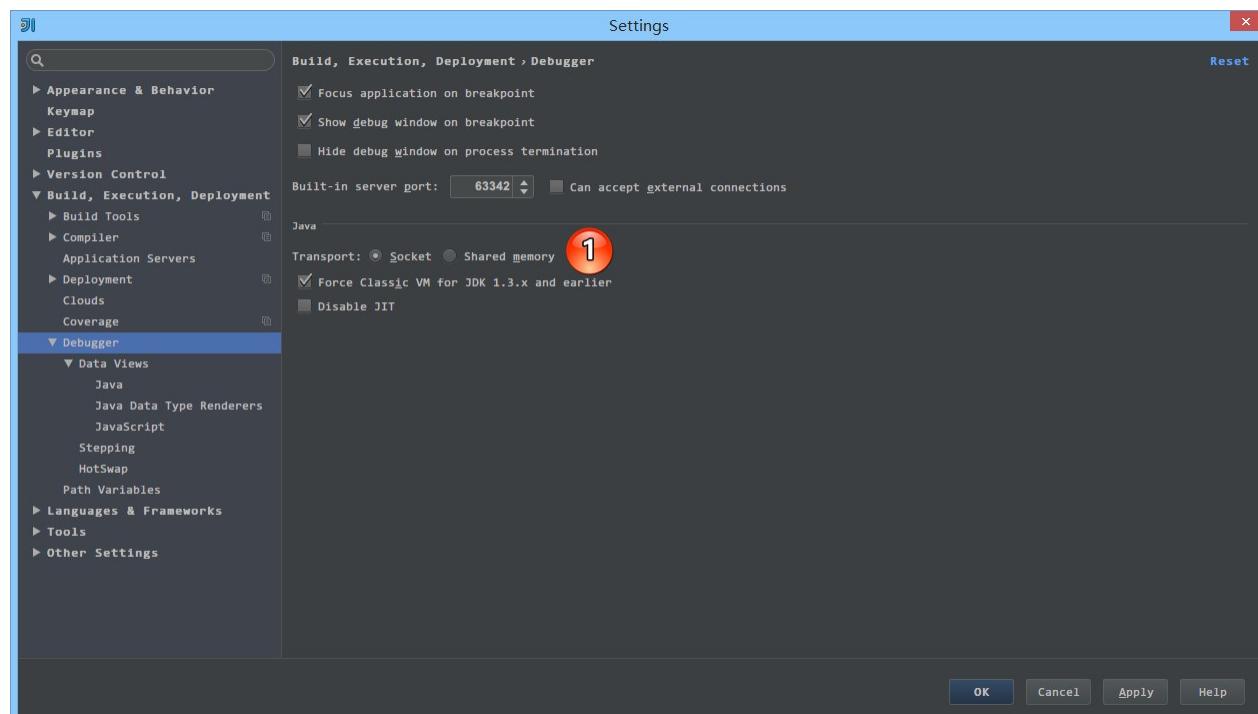
- 项目地址：<https://github.com/judasn/Basic-Single-Module-SSJPA>

开发环境

- JDK 7（理论上支持 JDK 6、JDK 7、JDK 8）
- Mysql 5.6
- Maven 3.1.1
- Tomcat 7
- Git 2.7.0.2-64-bit
- IntelliJ IDEA 2016.1.1
- 所有编码：UTF-8

Debug 介绍

Debug 设置



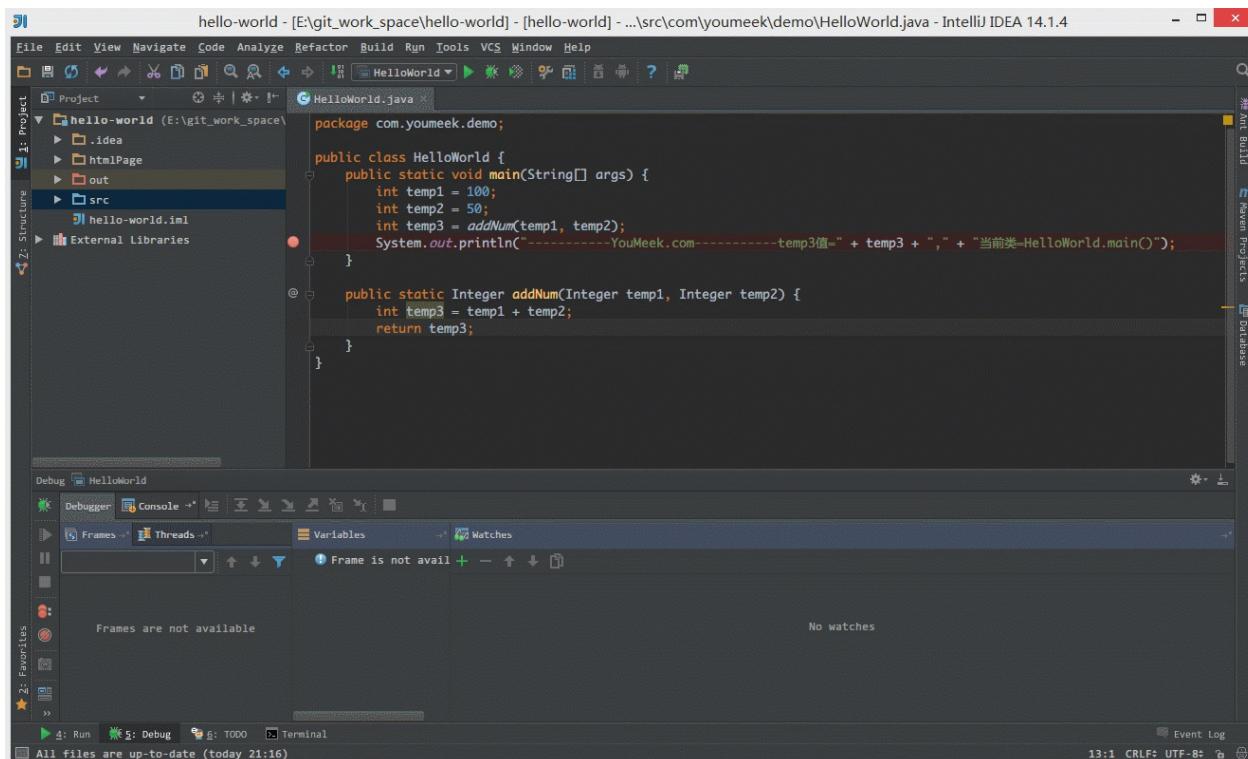
- 如上图标注 1 所示，表示设置 Debug 连接方式，默认是 `Socket`。`Shared memory` 是 Windows 特有的一个属性，一般在 Windows 系统下建议使用此设置，相对于 `Socket` 会快点。

Debug 常用快捷键

快捷键	介绍
F7	在 Debug 模式下，进入下一步，如果当前行断点是一个方法，则进入当前方法体内，如果该方法体还有方法，则不会进入该内嵌的方法中 必备
F8	在 Debug 模式下，进入下一步，如果当前行断点是一个方法，则不进入当前方法体内 必备
F9	在 Debug 模式下，恢复程序运行，但是如果该断点下面代码还有断点则停在下一个断点上 必备
Alt + F8	在 Debug 的状态下，选中对象，弹出可输入计算表达式调试框，查看该输入内容的调试结果 必备
Ctrl + F8	在 Debug 模式下，设置光标当前行为断点，如果当前已经是断点则去掉断点
Shift + F7	在 Debug 模式下，智能步入。断点所在行上有多个方法调用，会弹出进入哪个方法
Shift + F8	在 Debug 模式下，跳出，表现出来的效果跟 F9 一样
Ctrl + Shift + F8	在 Debug 模式下，指定断点进入条件
Alt + Shift + F7	在 Debug 模式下，进入下一步，如果当前行断点是一个方法，则进入当前方法体内，如果方法体还有方法，则会进入该内嵌的方法中，依此循环进入

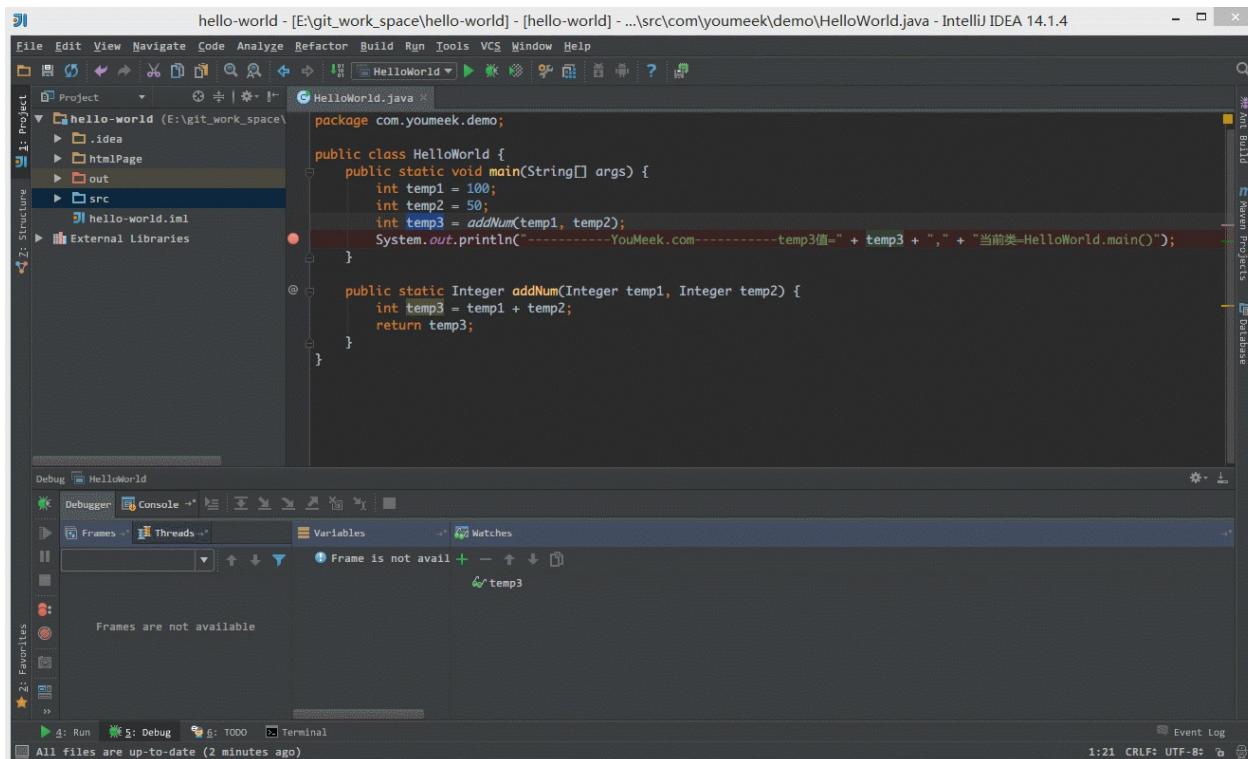
有时候我们可以这样粗鲁地认为 Debug 的使用就是等同于这几个快捷键的使用，所以上面的 必备 快捷键是我们必须牢记的，这些也是开发很常用的。

Debug 特殊技能使用



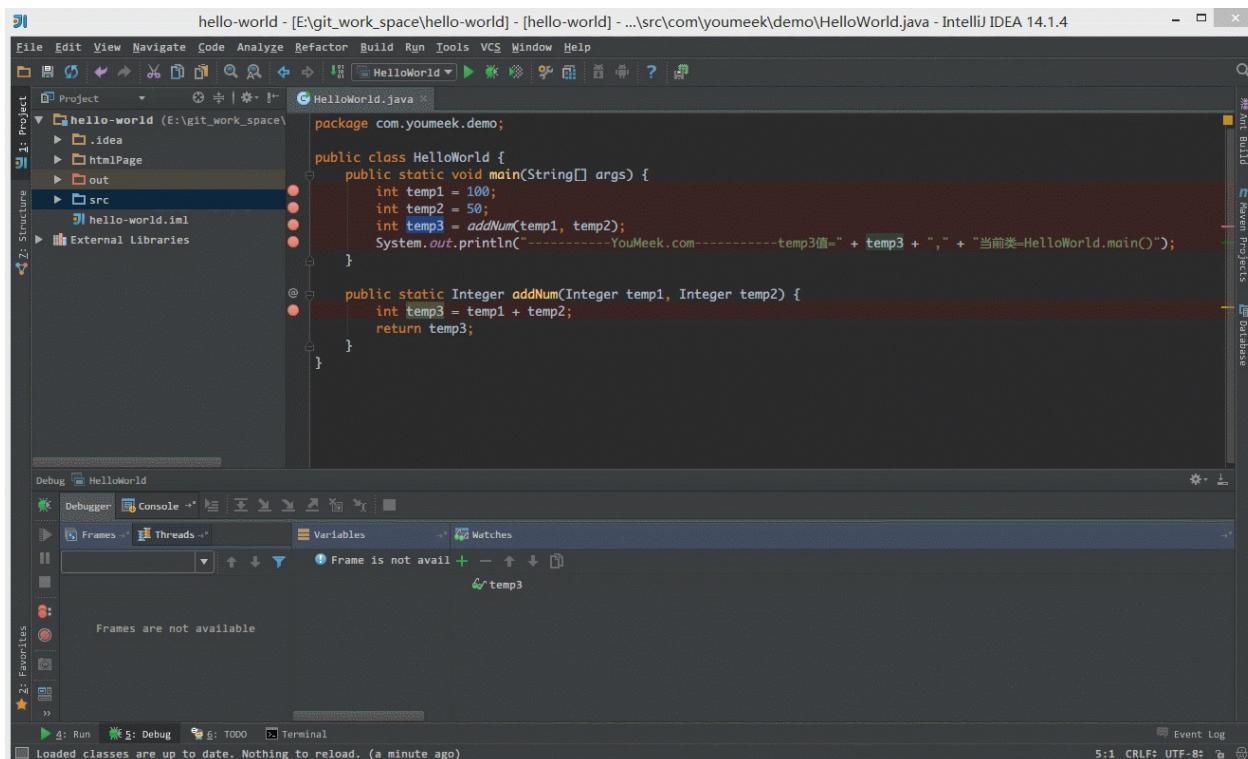
- 如上图 Gif 所示，查看所选对象的方法常用有三种方式：

- 选中对象后，使用快捷键 `Alt + F8`。
- 选中对象后，拖动对象到 `Watches`。
- 选中对象后，鼠标悬停在对象上 2 秒左右。

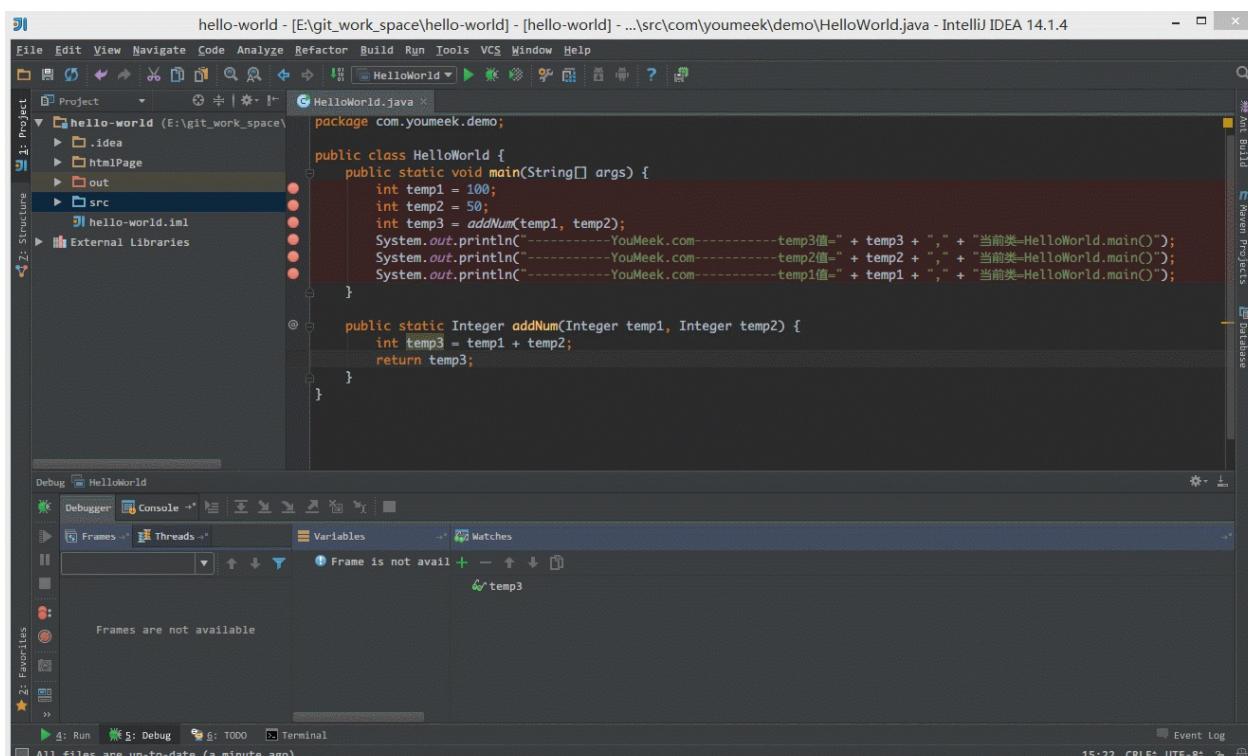


- 如上图 Gif 所示，在弹出表达式输入框中 IntelliJ IDEA 也是能帮我们智能提示。

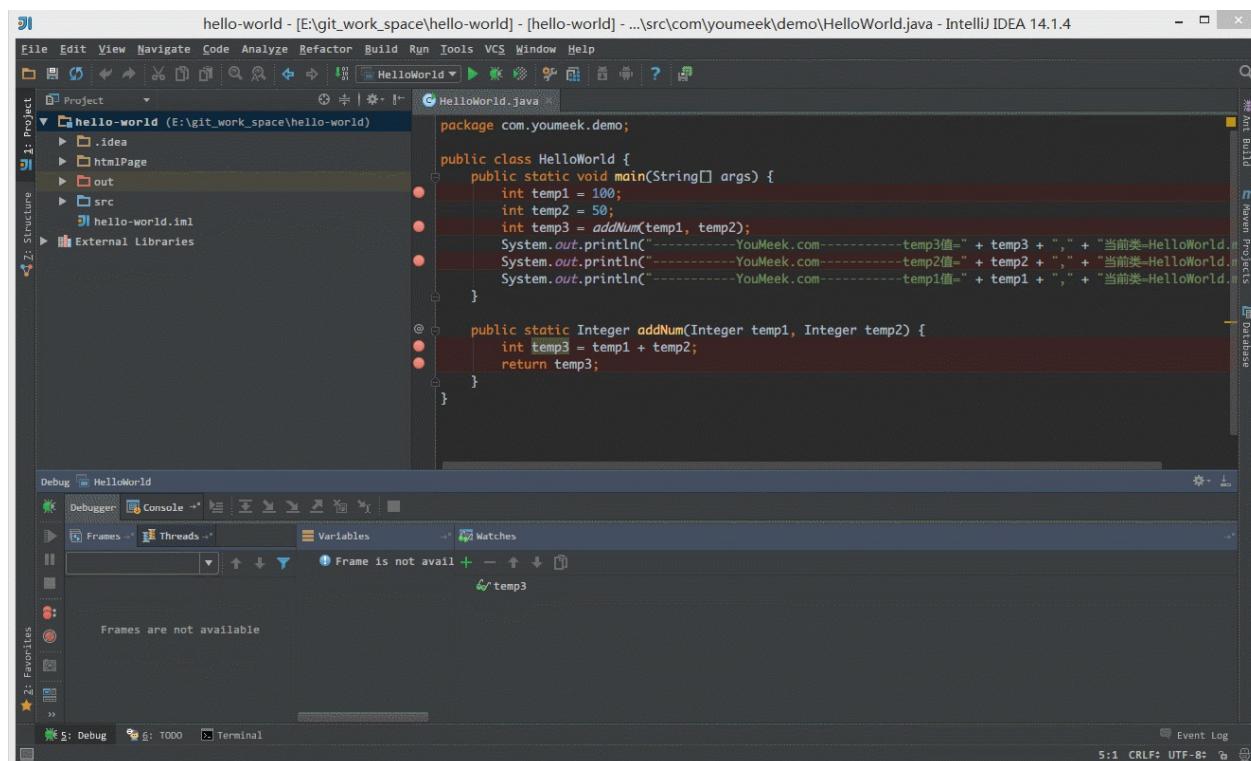
26.Debug 讲解



- 如上图 Gif 所示，当我们需要过掉后面的所有断点的时候，我们不需要去掉这些断点，只需要点击左下角那个小圆点，点击小圆点之后，所有断点变成灰色，然后我们再按快捷键 F9 即可过掉当前和后面所有的断点。



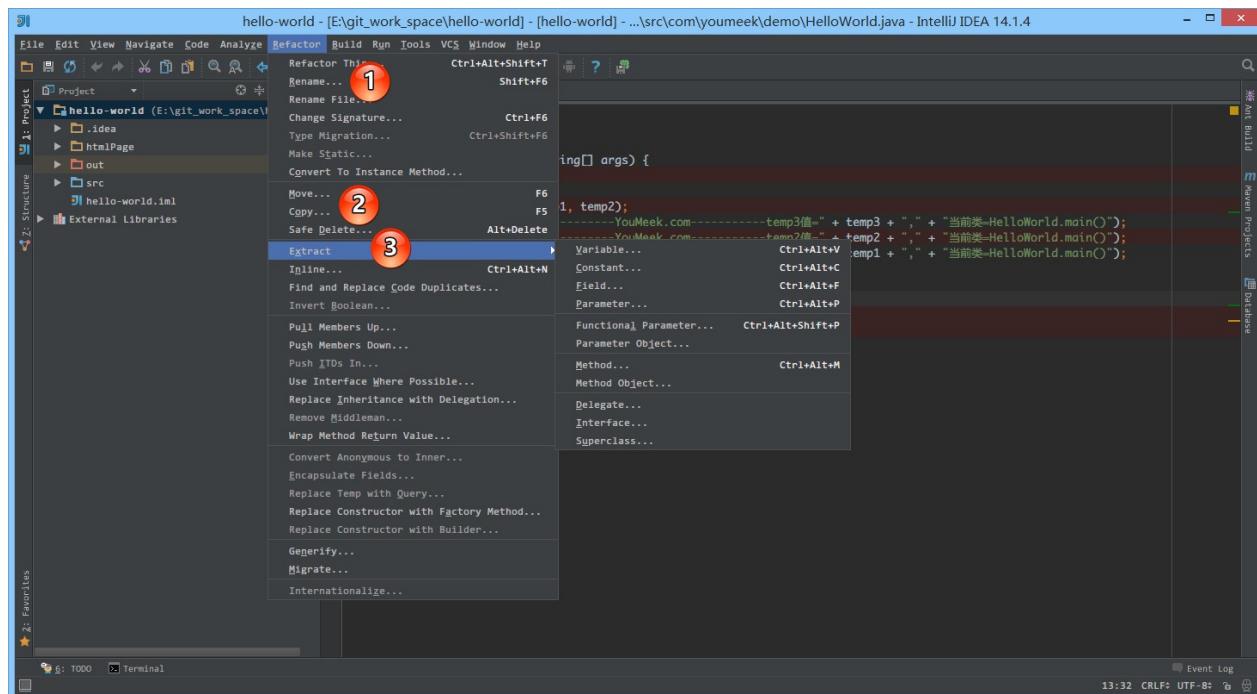
- 我们可以给断点设置进入的条件。如上图 Gif 所示，因为变量 temp3 不等于 200 所以该断点没有被进入直接跳过。



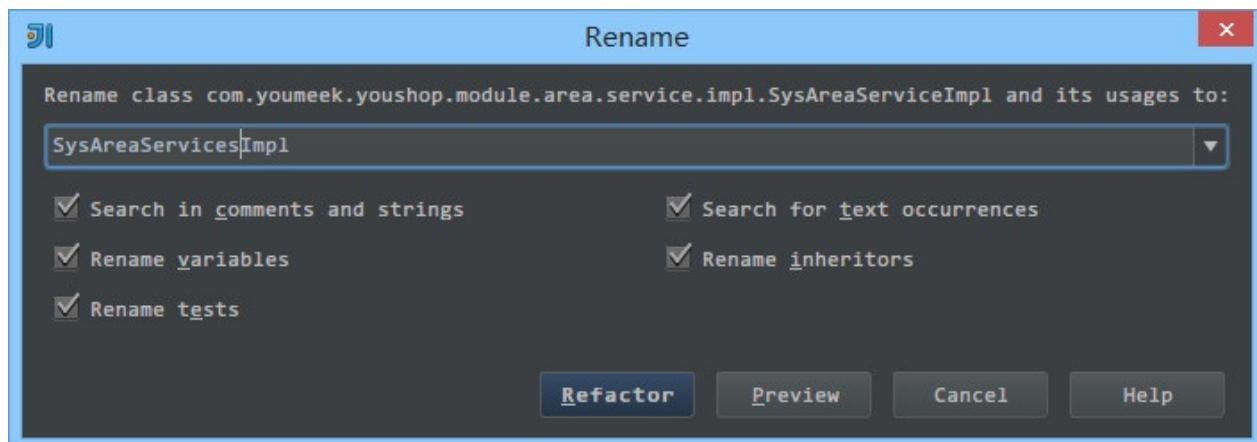
- 有时候当我们步入方法体之后，还想回退到方法体外。如 Gif 演示，断点进入 `addNum` 方法后，点击 `Drop Frame` 按钮之后，断点重新回到方法体之外。

重构讲解

重构的常用功能介绍



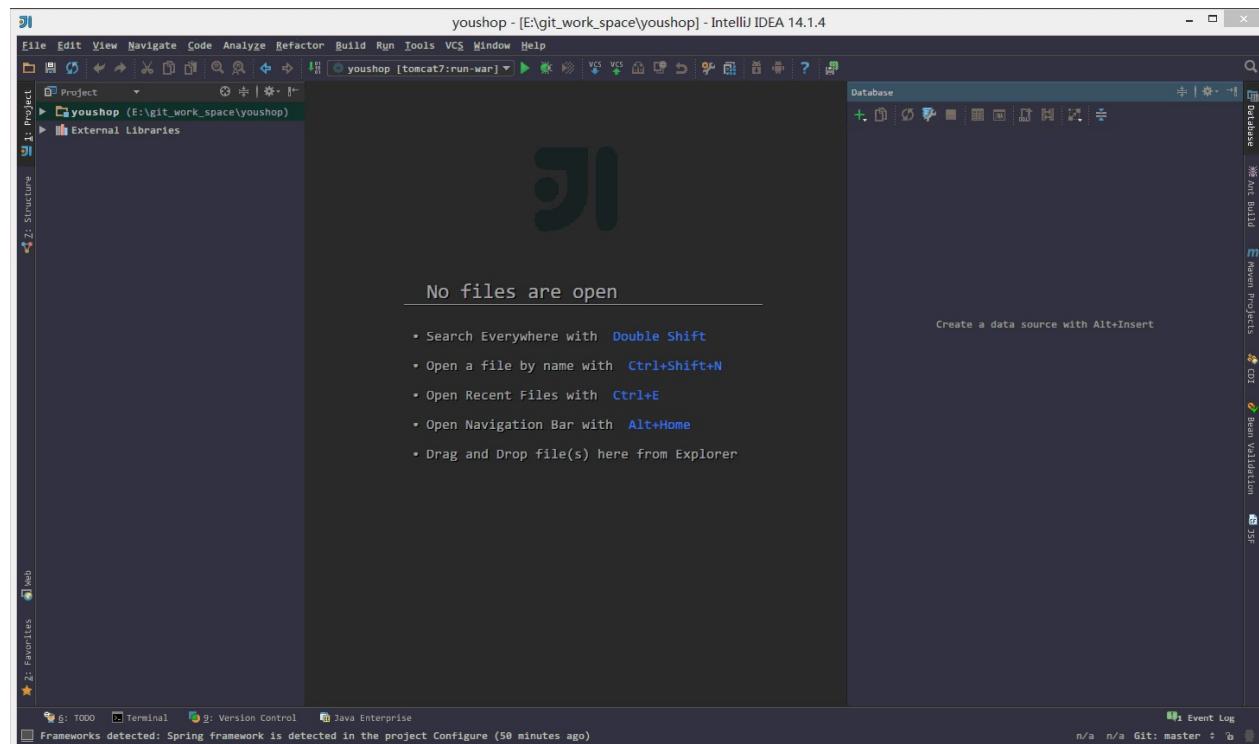
- 如上图标注所示，IntelliJ IDEA 的重构功能全部都在这个菜单上，所以我们也可以理解为这个菜单就是重构功能的体现，只是有时候我们通过快捷键的方式来加以体现而已。
- 如上图标注 1 所示，在实际开发中我们经常会对文件进行重命名，这里用的快捷键就是 `Shift + F6`。
- 如上图标注 2 所示，移动文件到其他包或是目录，我们可以通过快捷键也可以通过拖动对应的文件到其他目录进行重构。
- 如上图标注 3 所示，提取 Java 类代码也是经常遇到的优化类型的重构，这里不进行一一讲解，大家可以自己尝试下。



- 当我们要对一个类进行重命名的时候，会弹出该选项，IntelliJ IDEA 会帮我们自动扫描对应勾选项功能的地方进行重命名，这样可以省去我们很多工作。如果担心重构出错，可以点击 `Preview` 进行预览，IntelliJ IDEA 可以提示会修改哪些文件，修改哪一行。同时有一点是 IntelliJ IDEA 比较牛的地方，就是对于类关联的 `xml` 或是其他资源文件的重构，IntelliJ IDEA 都可以很好地做到识别，所以 IntelliJ IDEA 在开发 Java Web 项目上可以帮我们省去很多苦力活。
- 所以，如果单论 Java 类上的重构功能，跟其他 IDE 并不会有太明显的优势突出，但是在 Web 方向的重构，目前应该是所有市场上的 IDE 中最好的，包括 `HTML`、`CSS`、`JavaScript` 等相关文件都可以做到一些功能的重构。

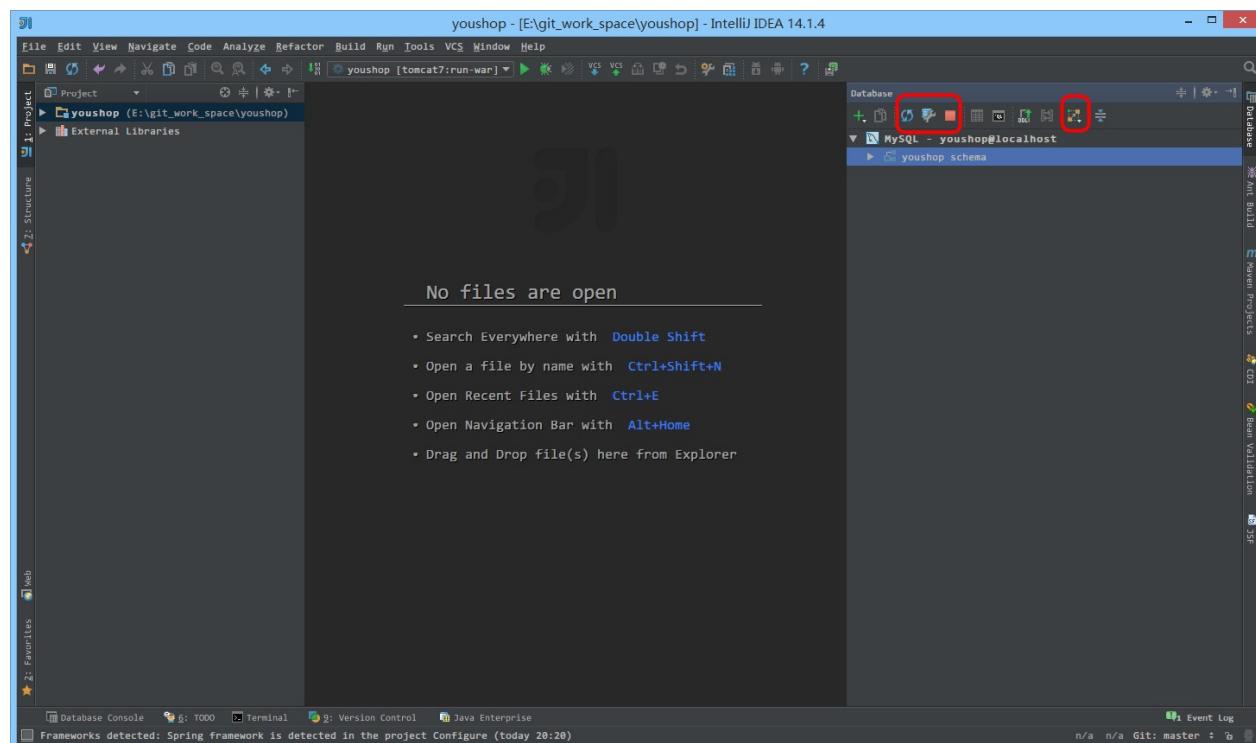
数据库管理工具介绍

配置 Database 组件的数据库连接



- 表面上很多人认为配置 Database 就是为了有一个 GUI 管理数据库功能，但是这并不是 IntelliJ IDEA 的 Database 最重要特性。数据库的 GUI 工具有很多，IntelliJ IDEA 的 Database 也没有太明显的优势。IntelliJ IDEA 的 Database 最大特性就是对于 Java Web 项目来讲，常使用的 ORM 框架，如 Hibernate、Mybatis 有很好的支持，比如配置好了 Database 之后，IntelliJ IDEA 会自动识别 domain 对象与数据表的关系，也可以通过 Database 的数据表直接生成 domain 对象等等。
- 如上图 Gif 所示，这是一个完成的配置 Database 过程，对于数据库需要的依赖包，IntelliJ IDEA 可以自动帮我们下载，所以我们只要配置对应的连接参数即可。

Database 设置



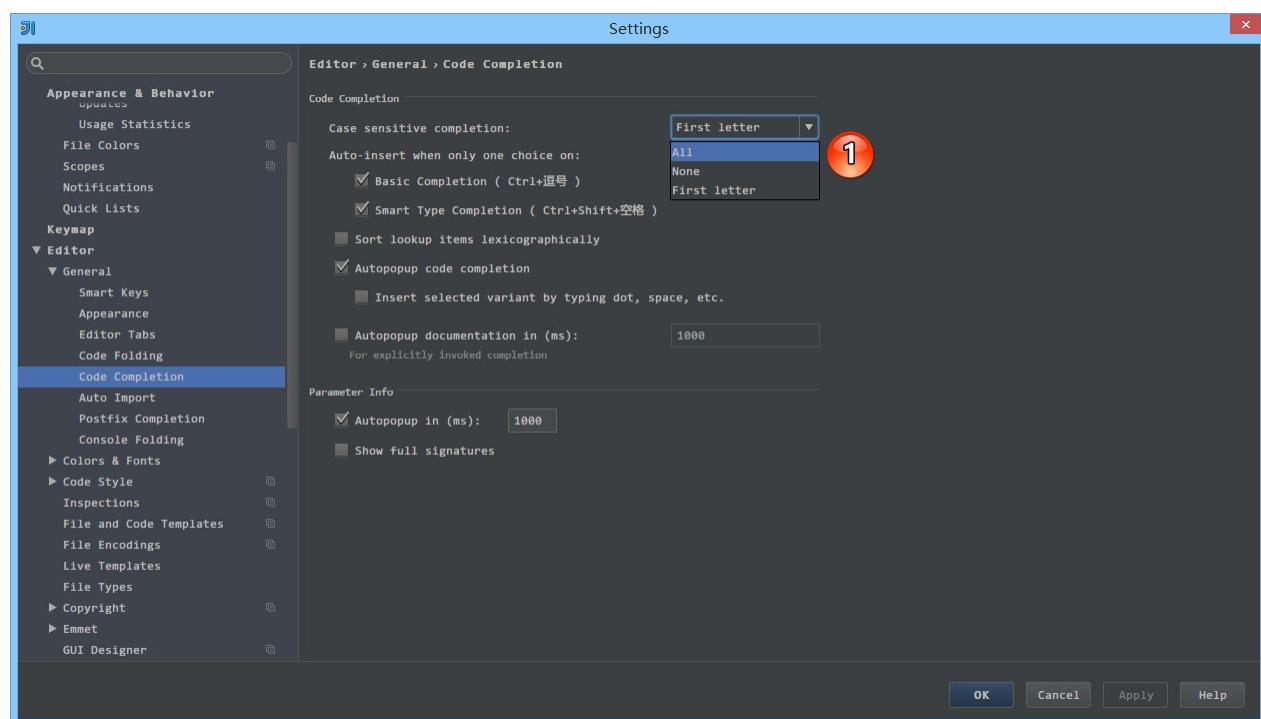
- 如上图标注的红圈所示，这是 Database 常用的四个操作。
- 第一个按钮是：同步当前数据库连接。这个是最重要的操作，有一些情况下，当我们配置好连接之后，没有显示数据表，那就是需要点击该按钮进行同步。还有一种情况就是我们在 IntelliJ IDEA 之外用其他工具操作数据库，比如新建表。而此时 IntelliJ IDEA 的 Database 如果没有同步到新表，也是需要点击此按钮进行同步的。
- 第二个按钮是：配置当前连接，跟我们首次设置连接的界面是一样的。
- 第三个按钮是：断开当前的连接。
- 第四个按钮是：查看当前所选对象的图标结构，比如我们当前选中的是整个数据库名，我们如果点击此按钮，则是显示该数据库下的所有数据表的图标结构图。

IntelliJ IDEA 常用设置讲解

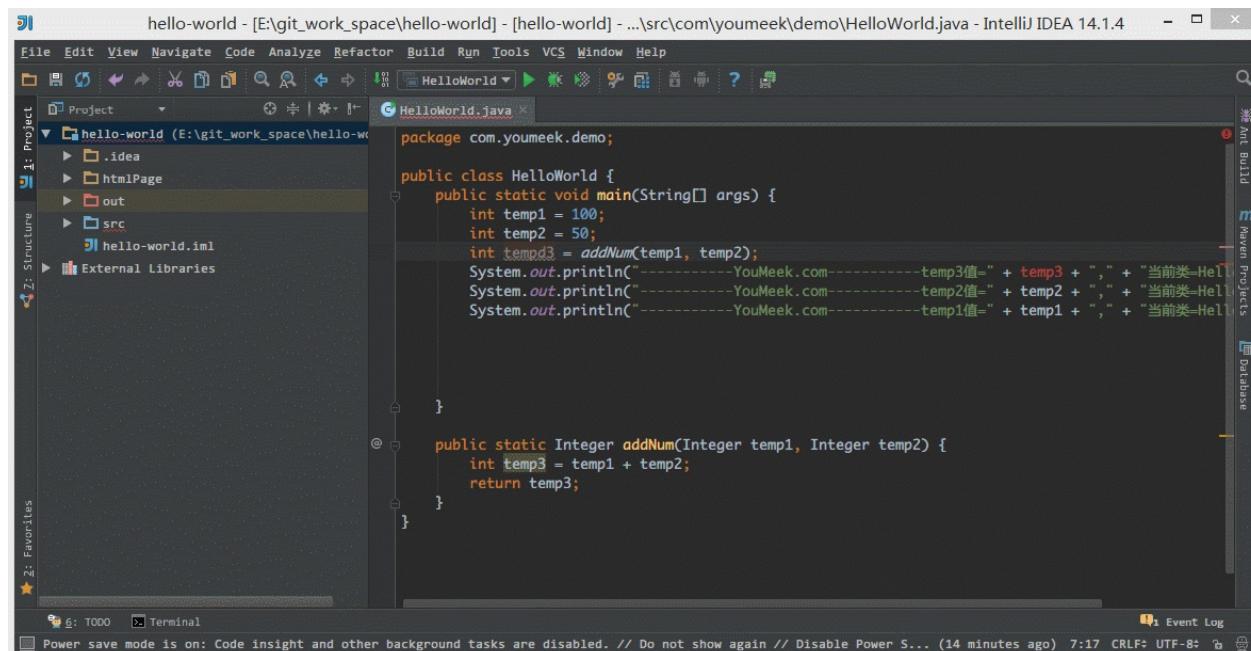
说明

IntelliJ IDEA 有很多人性化的设置我们必须单独拿出来讲解，也因为这些人性化的设置让我们这些 IntelliJ IDEA 死忠粉更加死心塌地使用它和分享它。

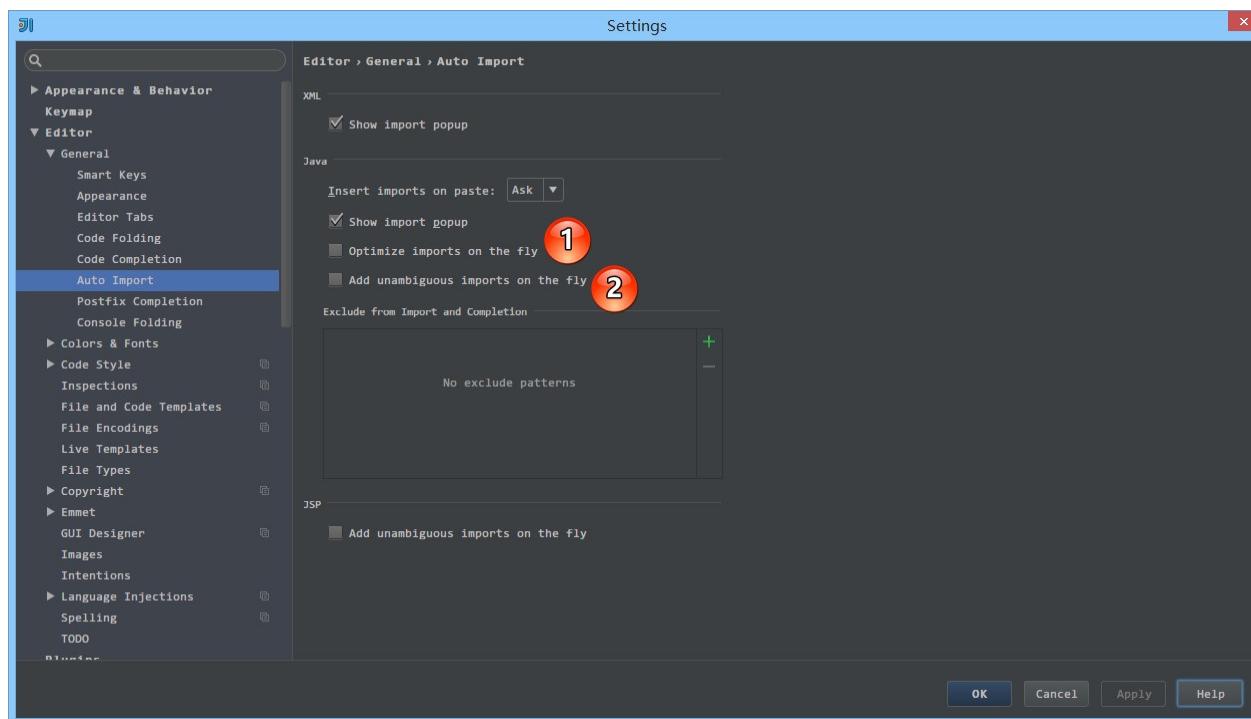
常用设置



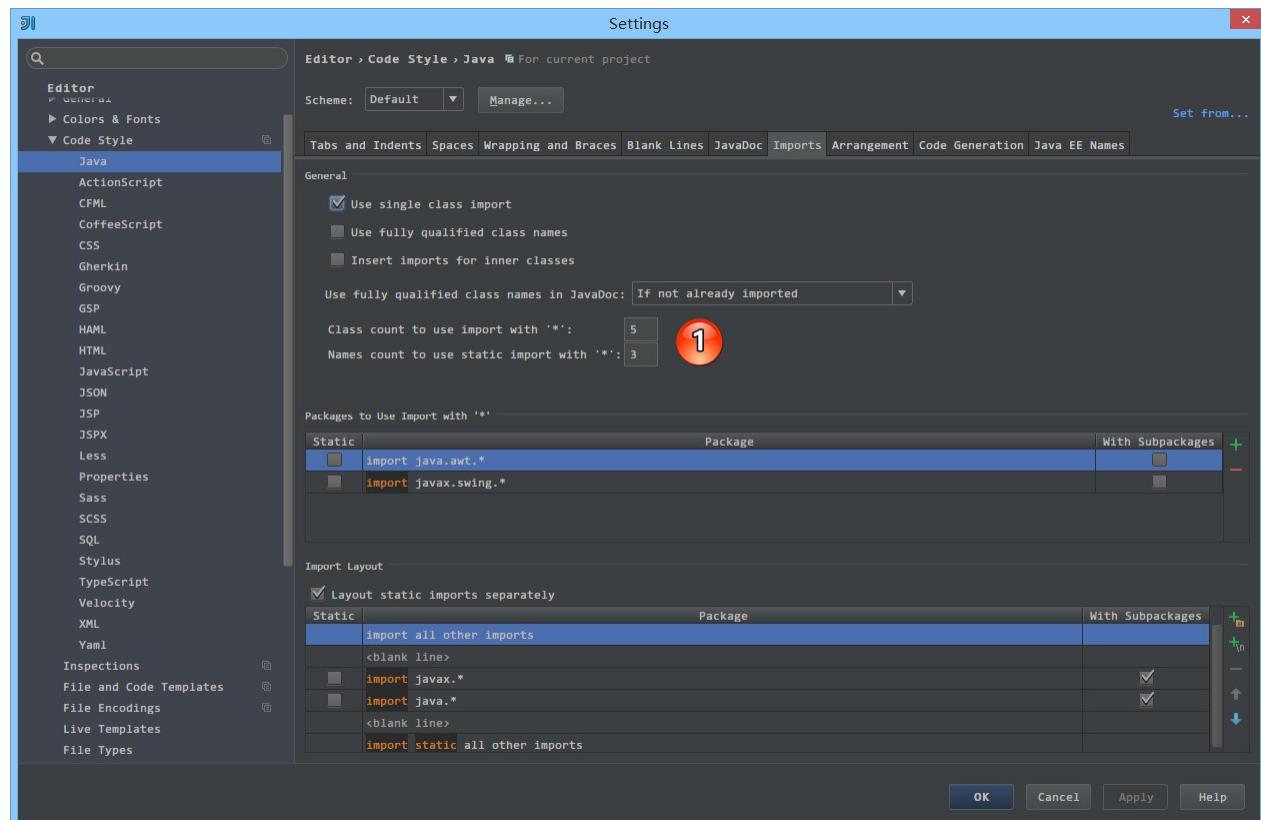
- IntelliJ IDEA 的代码提示和补充功能有一个特性：区分大小写。如上图标注 1 所示，默认就是 `First letter` 区分大小写的。
- 区分大小写的情况是这样的：比如我们在 Java 代码文件中输入 `stringBuffer` IntelliJ IDEA 是不会帮我们提示或是代码补充的，但是如果我们将输入改为 `StringBuffer` 就可以进行代码提示和补充。
- 如果想不区分大小写的话，改为 `None` 选项即可。



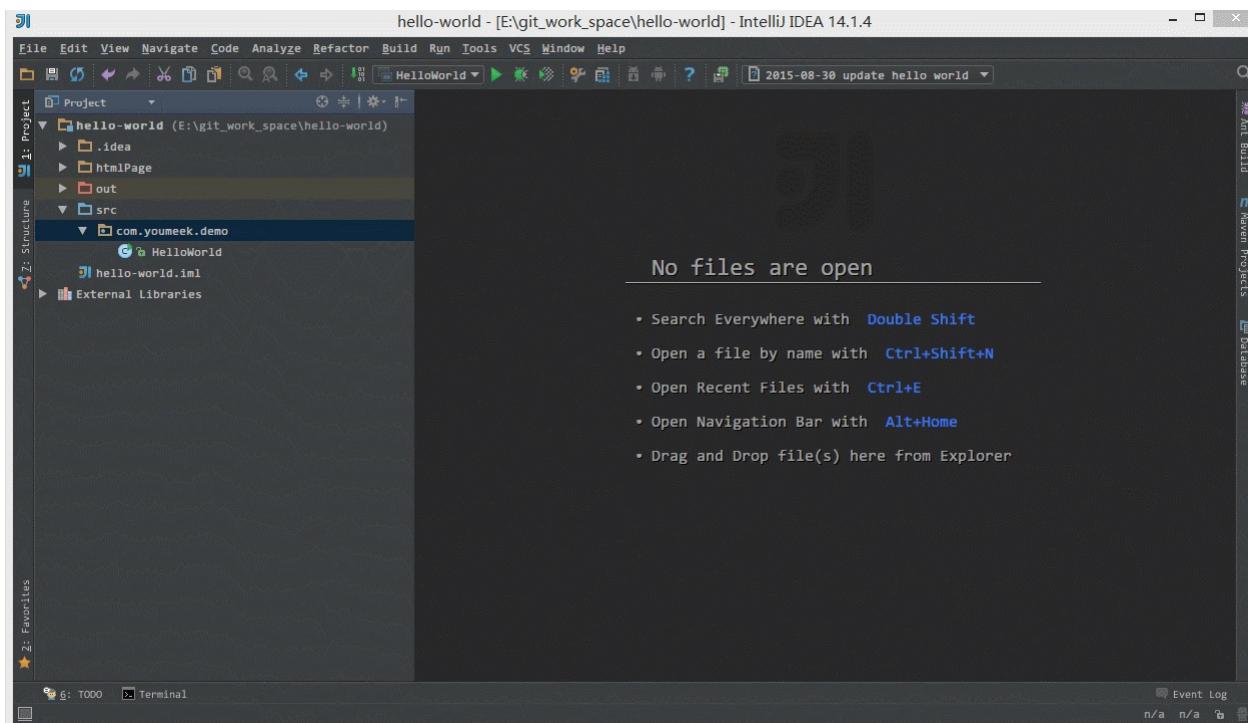
- 如上图 Gif 所示，该功能用来快速设置代码检查等级。我个人一般在编辑大文件的时候会使用该功能。IntelliJ IDEA 对于编辑大文件并没有太大优势，很卡，原因就是它有各种检查，这样是非常耗内存和 CPU 的，所以为了能加快大文件的读写，我一般会暂时性设置为 `None`。
- `Inspections` 为最高等级检查，可以检查单词拼写，语法错误，变量使用，方法之间调用等。
- `Syntax` 可以检查单词拼写，简单语法错误。
- `None` 不设置检查。



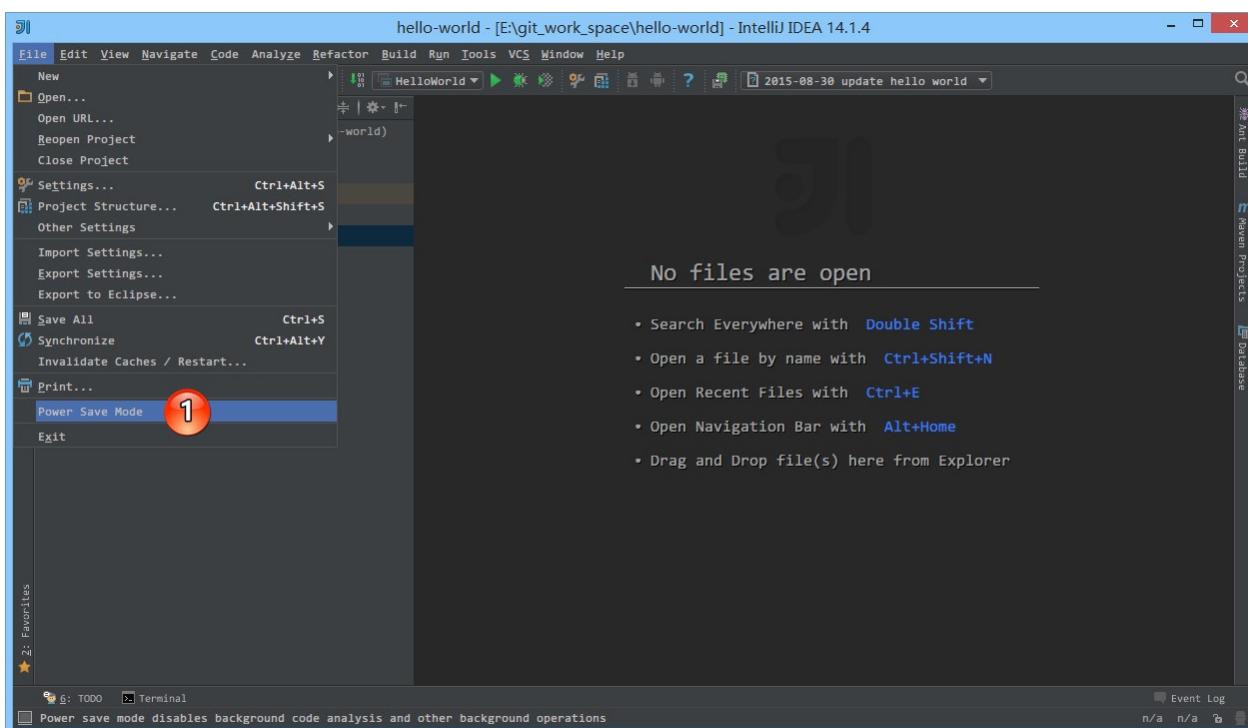
- 如上图标注 1 和 2 所示，默认 IntelliJ IDEA 是没有开启自动 import 包的功能。
- 勾选标注 1 选项，IntelliJ IDEA 将在我们书写代码的时候自动帮我们优化导入的包，比如自动去掉一些没有用到的包。
- 勾选标注 2 选项，IntelliJ IDEA 将在我们书写代码的时候自动帮我们导入需要用到的包。但是对于那些同名的包，还是需要手动 `Alt + Enter` 进行导入的，IntelliJ IDEA 目前还无法智能到替我们做判断。



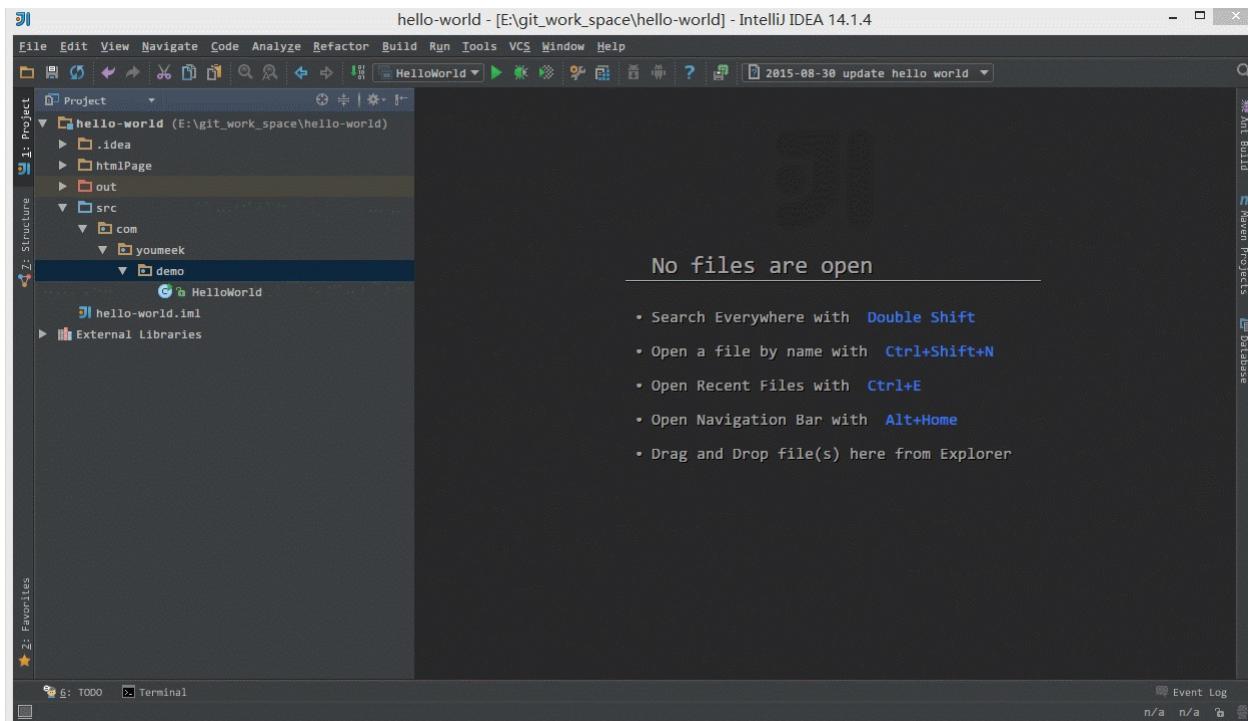
- 如上图标注 1 所示，当我们 Java 类中导入的某个包下类超过这里设置的指定个数，就会换成用 `*` 号来代替。



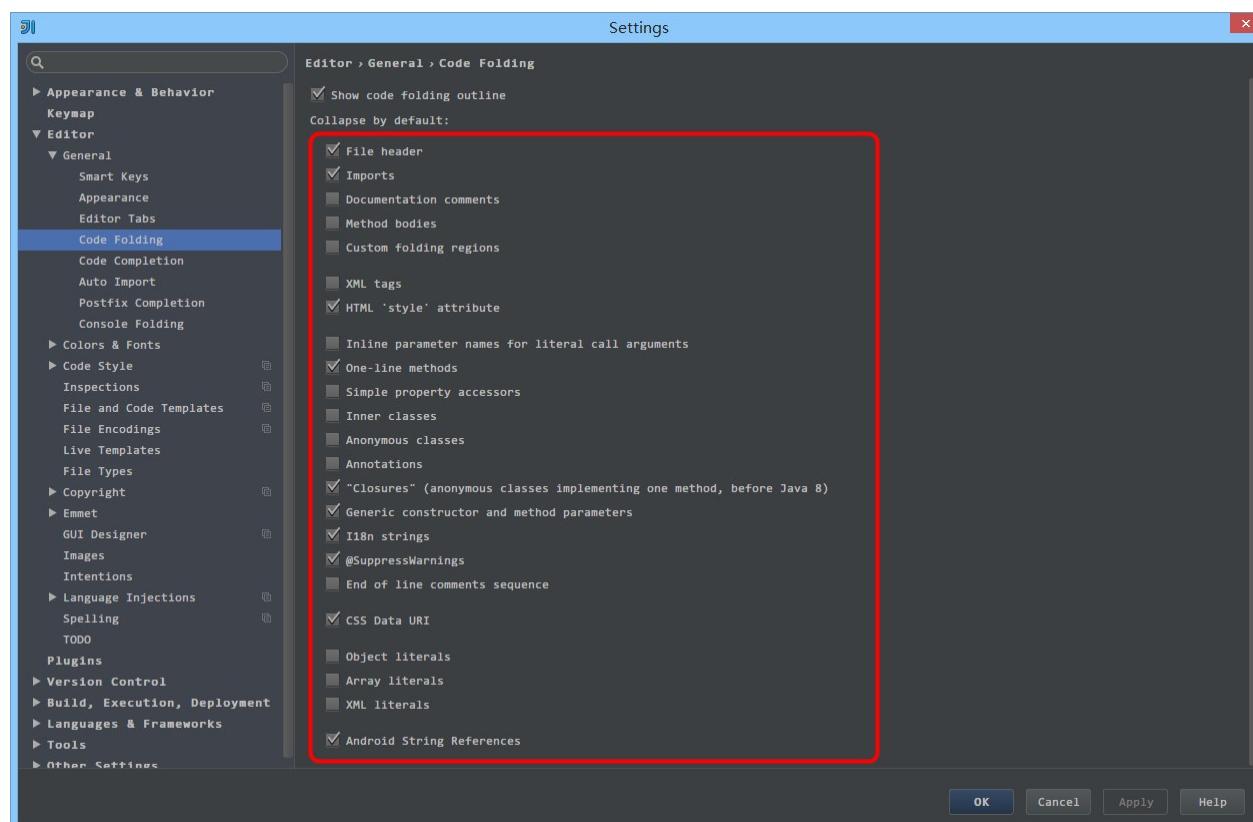
- 如上图 Gif 所示，IntelliJ IDEA 默认是会折叠空包的，这样就会出现包名连在一起的情况。但是有些人不喜欢这种结构，喜欢整个结构都是完整树状的，所以我们可以去掉演示中的勾选框即可。



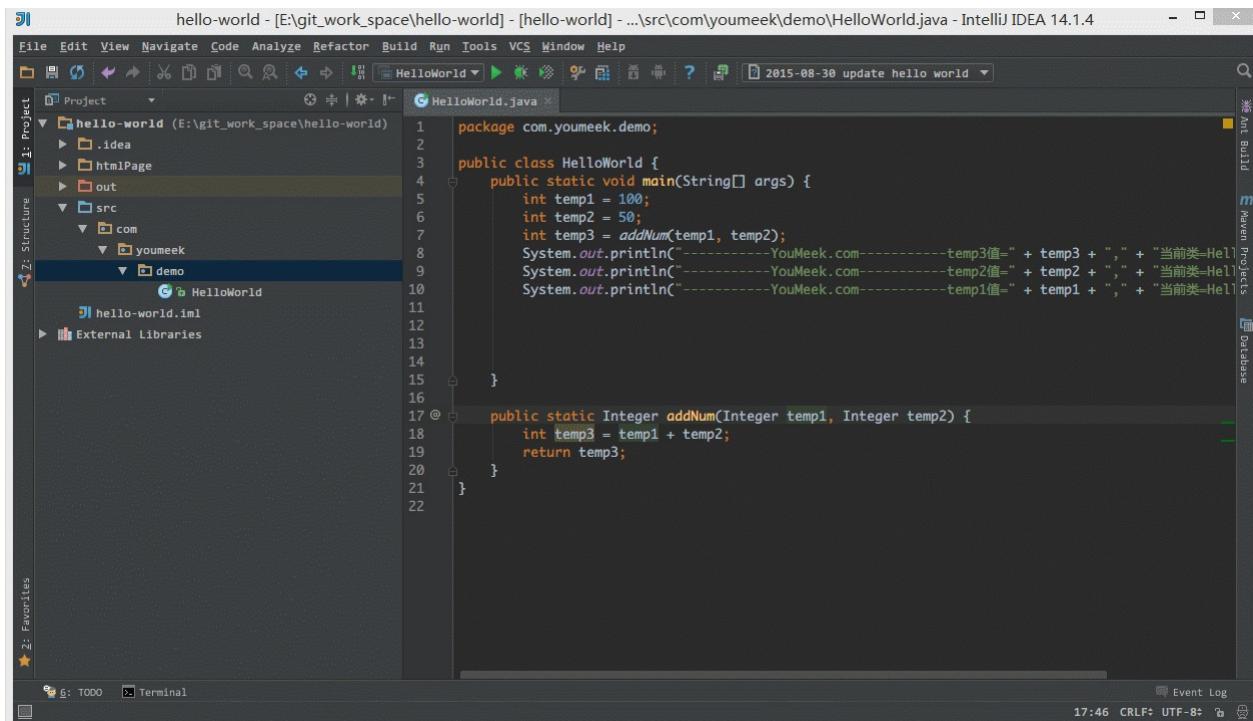
- 如上图标注 1 所示，IntelliJ IDEA 有一种叫做 省电模式 的状态，开启这种模式之后 IntelliJ IDEA 会关掉代码检查和代码提示等功能。所以一般我也会认为这是一种 阅读模式 ，如果你在开发过程中遇到突然代码文件不能进行检查和提示可以来看看这里是否有开启该功能。



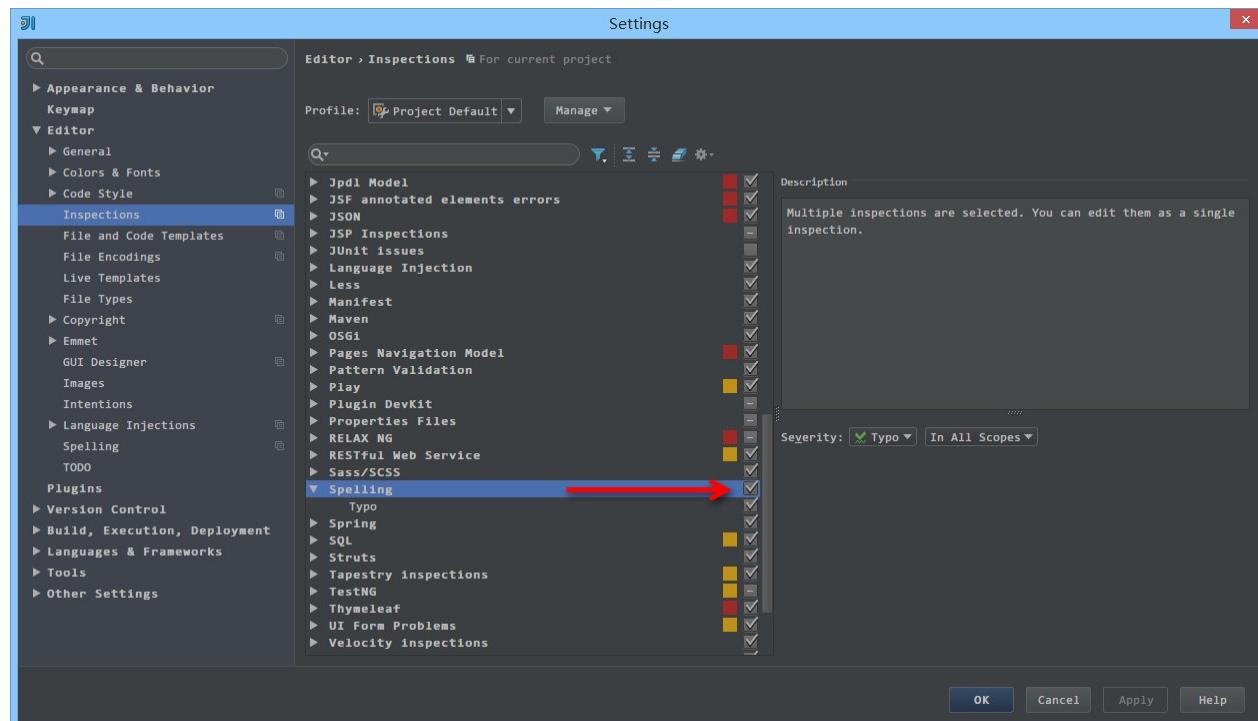
- 如上图 Gif 所示，在我们按 **Ctrl + Shift + N** 进行打开某个文件的时候，我们可以直接定位到改文件的行数上。一般我们在调 CSS，根据控制台找空指针异常的时候，使用该方法速度都会相对高一点。



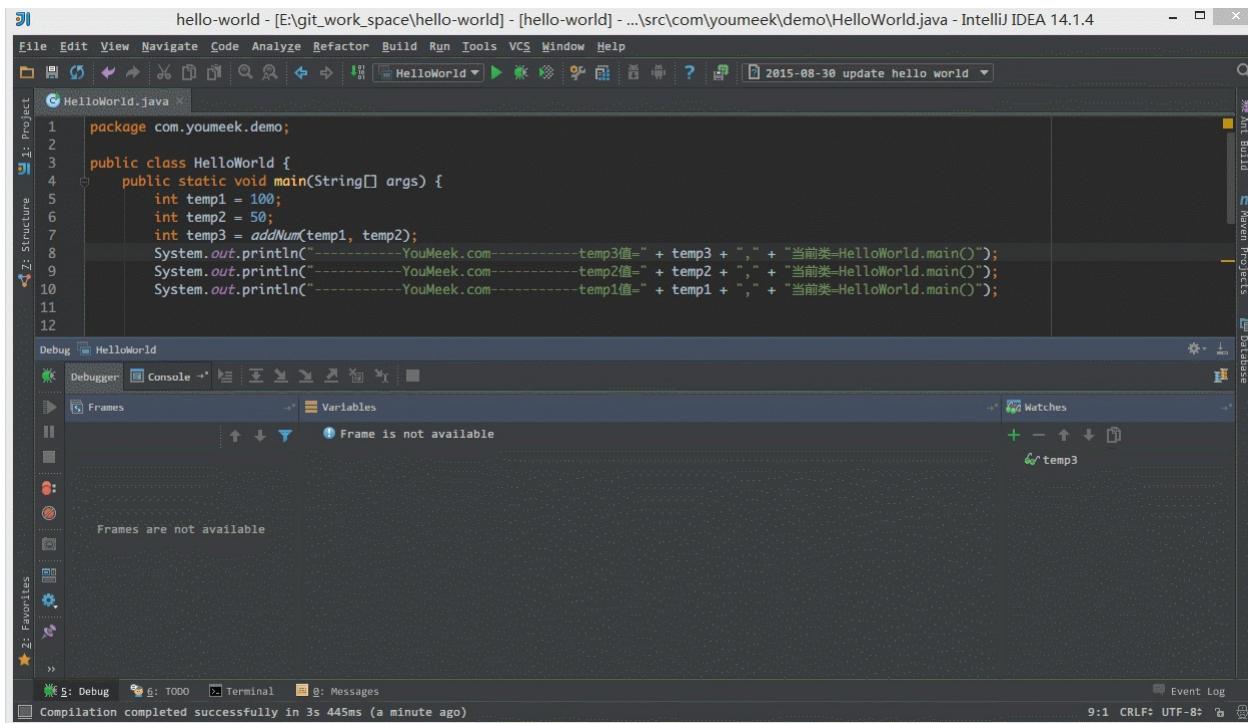
- 如上图标注红圈所示，我们可以对指定代码类型进行默认折叠或是展开的设置，勾选上的表示该类型的代码在文件被打开的时候默认是被折叠的，去掉勾选则反之。



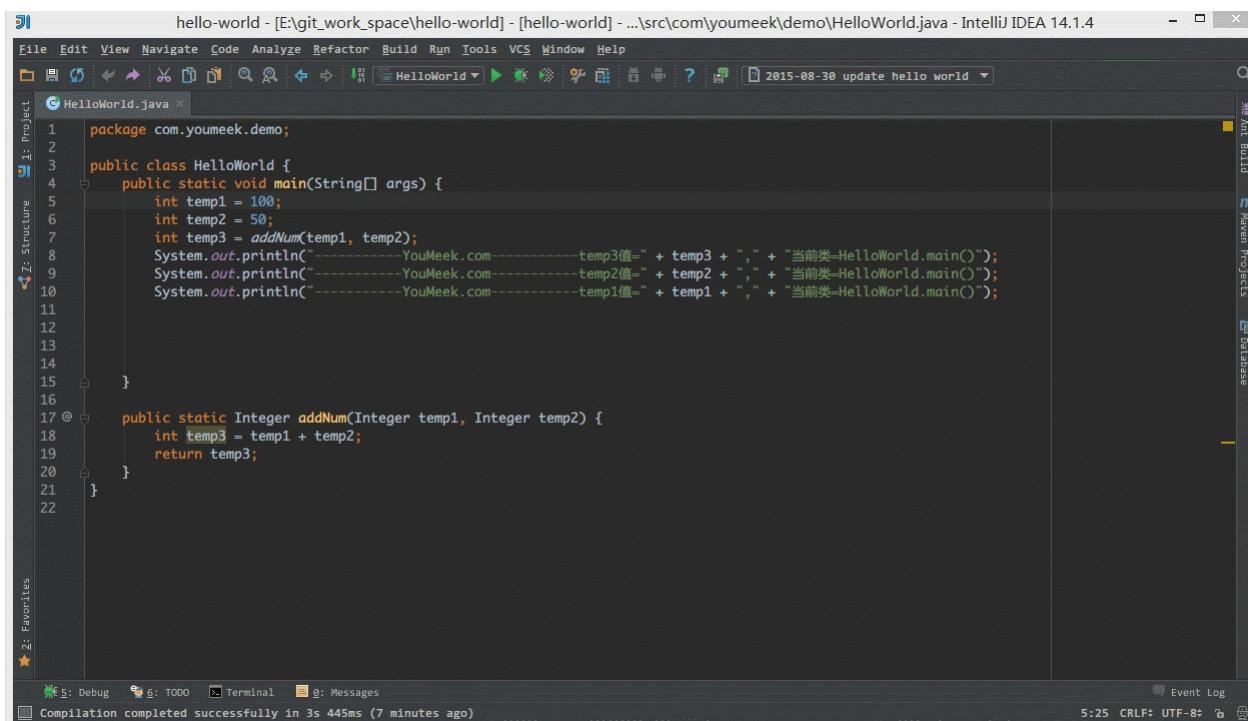
- 如上图 Gif 所示，IntelliJ IDEA 支持对代码进行垂直或是水平分组。一般在对大文件进行修改的时候，有些修改内容在文件上面，有些内容在文件下面，如果来回操作可能效率会很低，用此方法就可以好很多。当然了，前提是自己的显示器分辨率要足够高。



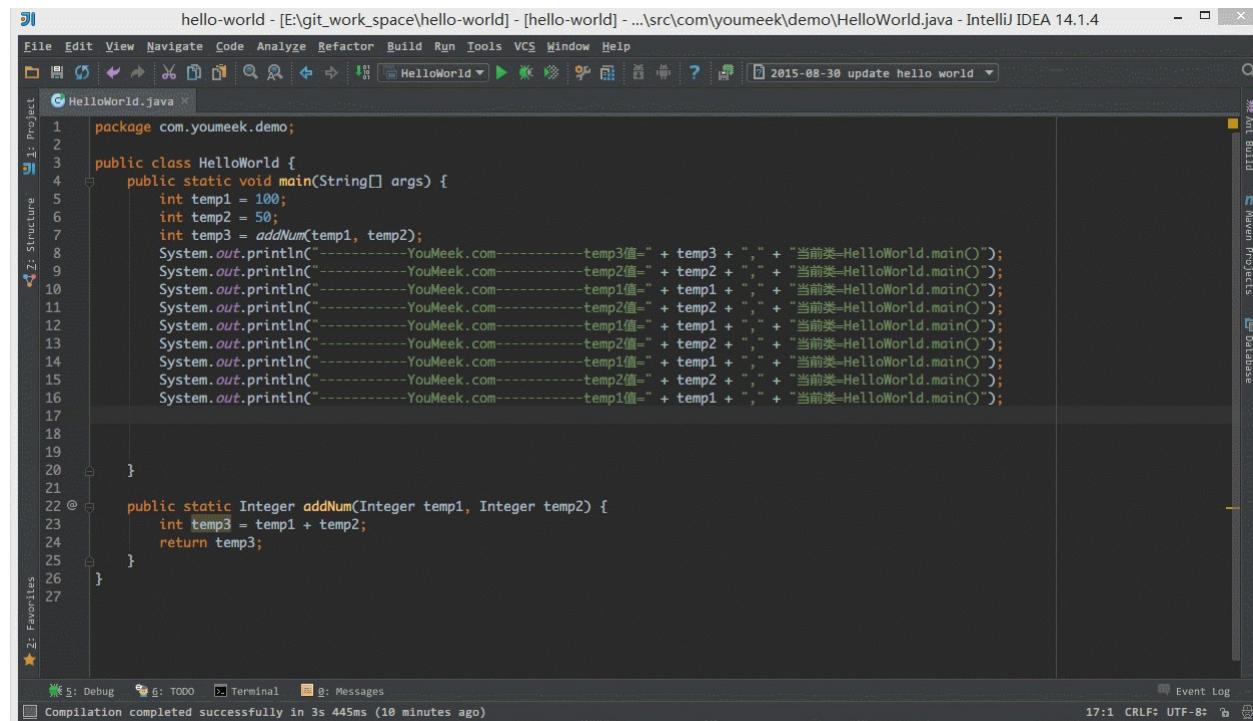
- 如上图箭头所示，IntelliJ IDEA 默认是开启单词拼写检查的，有些人可能有强迫症不喜欢看到单词下面有波浪线，就可以去掉该勾选。但是我个人建议这个还是不要关闭，因为拼写检查是一个很好的功能，当大家的命名都是标准话的时候，这可以在不时方便地帮我们找到代码因为拼写错误引起的 Bug。



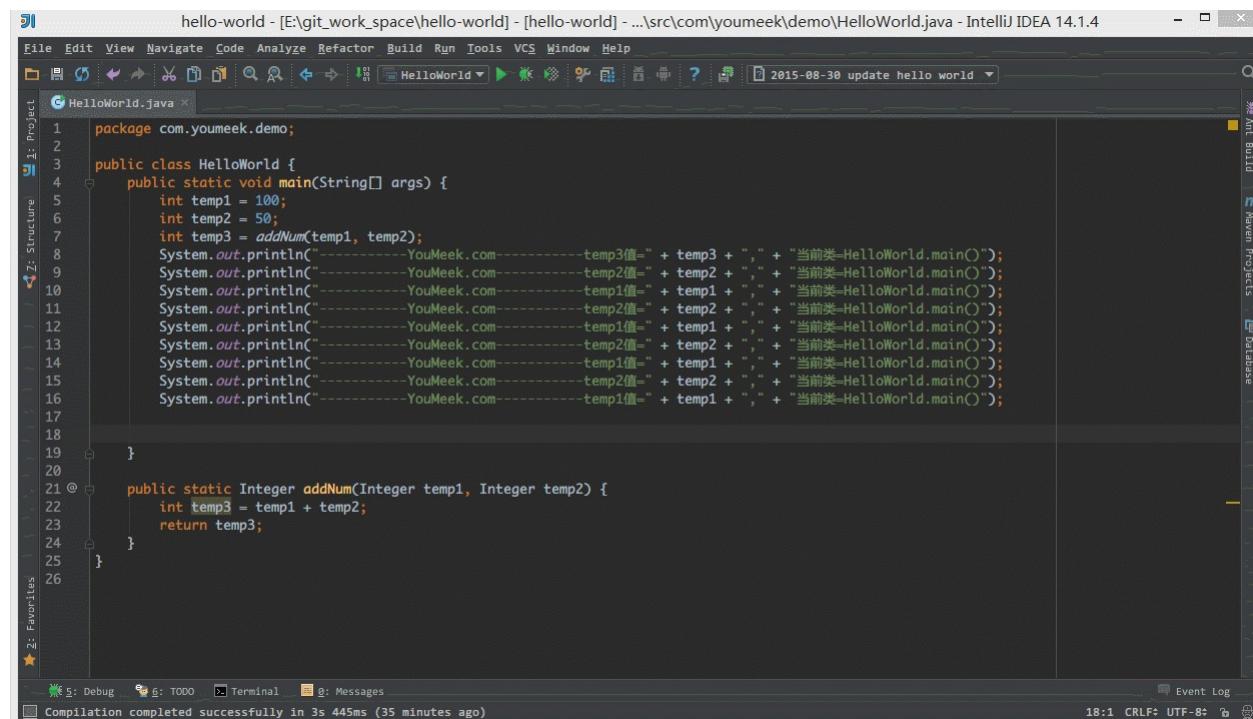
- 如上图 Gif 所示，我们可以对组件窗口的子窗口进行拖动移位，有时候设置过头或是效果不满意，那我们需要点击此按钮进行窗口还原。



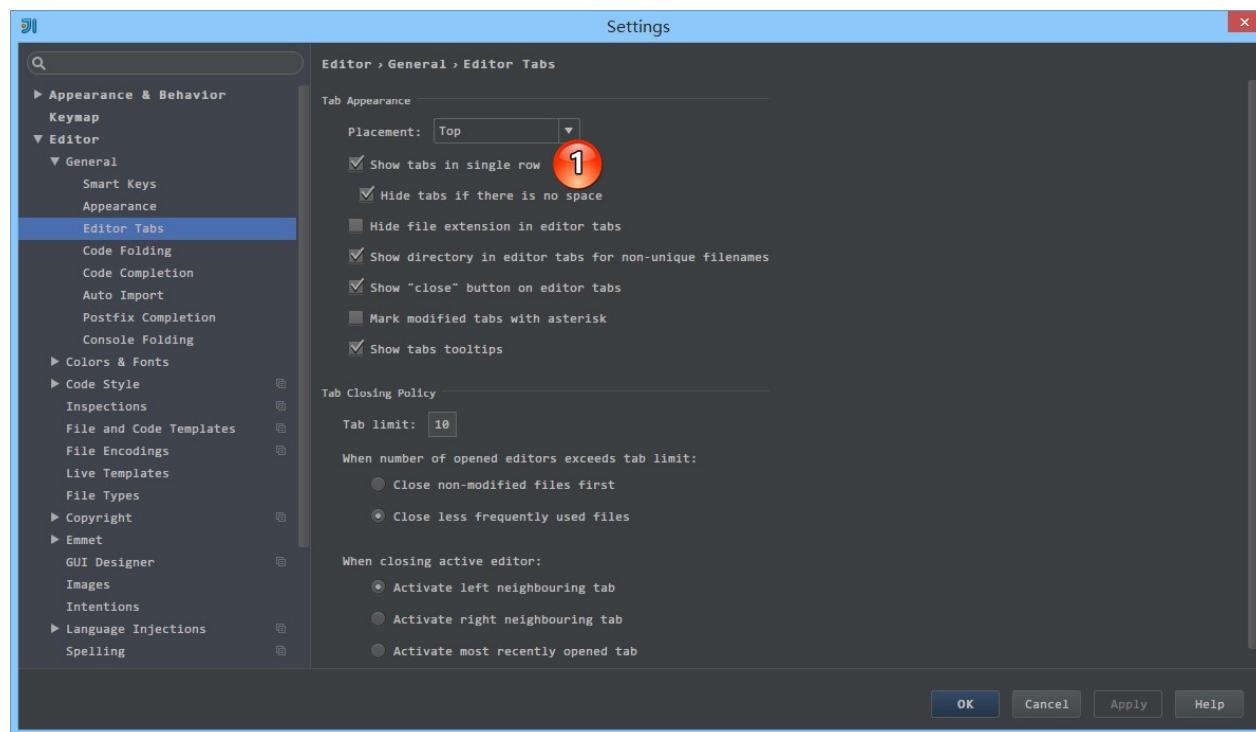
- 如上图 Gif 所示，在没有对 **Ctrl + D** 快捷键进行修改前，此快捷键将是用来复制并黏贴所选的内容的，但是黏贴的位置是补充在原来的位置后，我个人不喜欢这种风格，我喜欢复制所选的行数完整内容，所以进行了修改，修改后的效果如上图 Gif 演示。



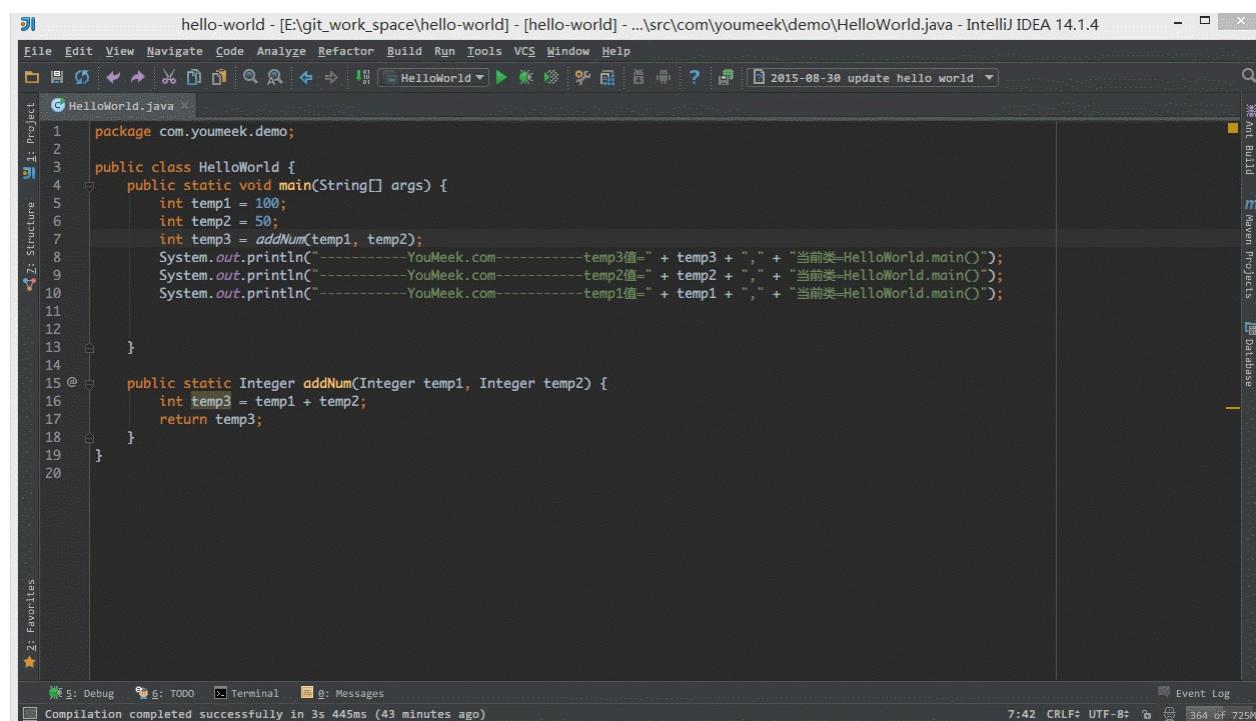
- 如上图 Gif 所示，默认 **Ctrl + 空格** 快捷键是基础代码提示、补充快捷键，但是由于我们中文系统基本这个快捷键都被输入法占用了，所以我们发现不管怎么按都是没有提示代码效果的，原因就是在此。我个人建议修改此快捷键为 **Ctrl + 逗号**。



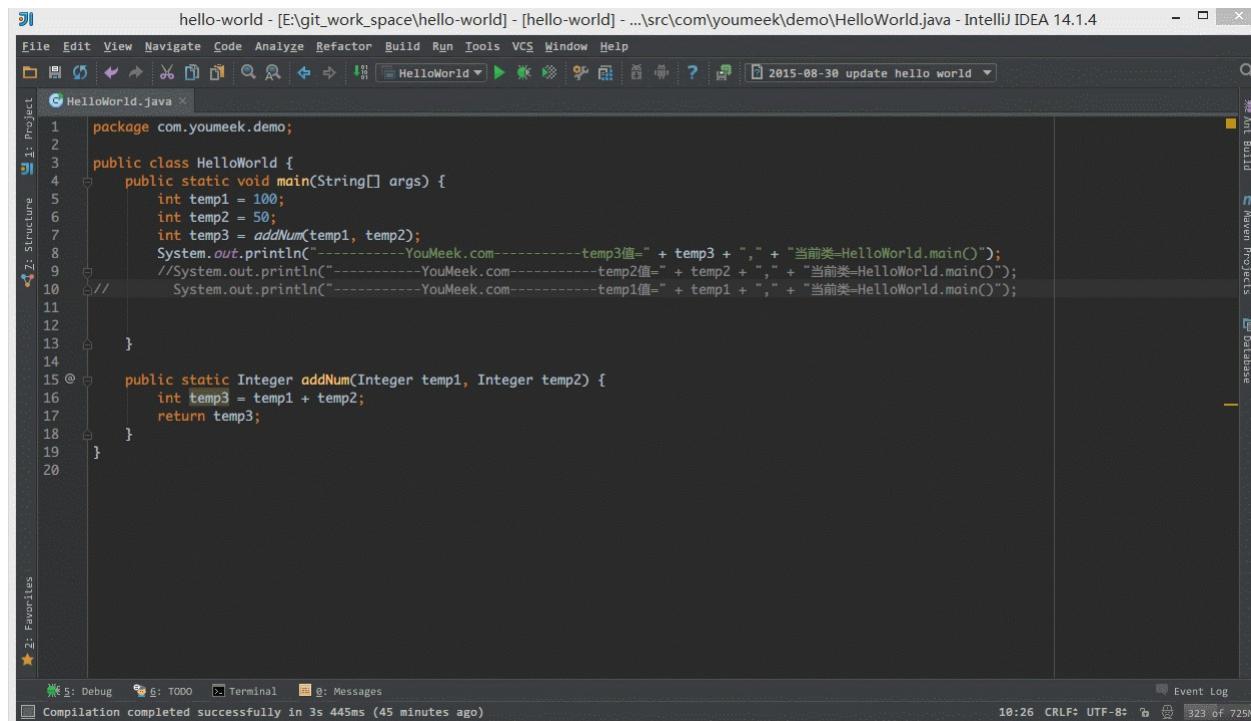
- 如上图 Gif 所示，IntelliJ IDEA 14 版本默认是不显示内存使用情况的，对于大内存的机器来讲不显示也无所谓，但是如果内存小的机器最好还是显示下。如上图演示，点击后可以进行部分内存的回收。



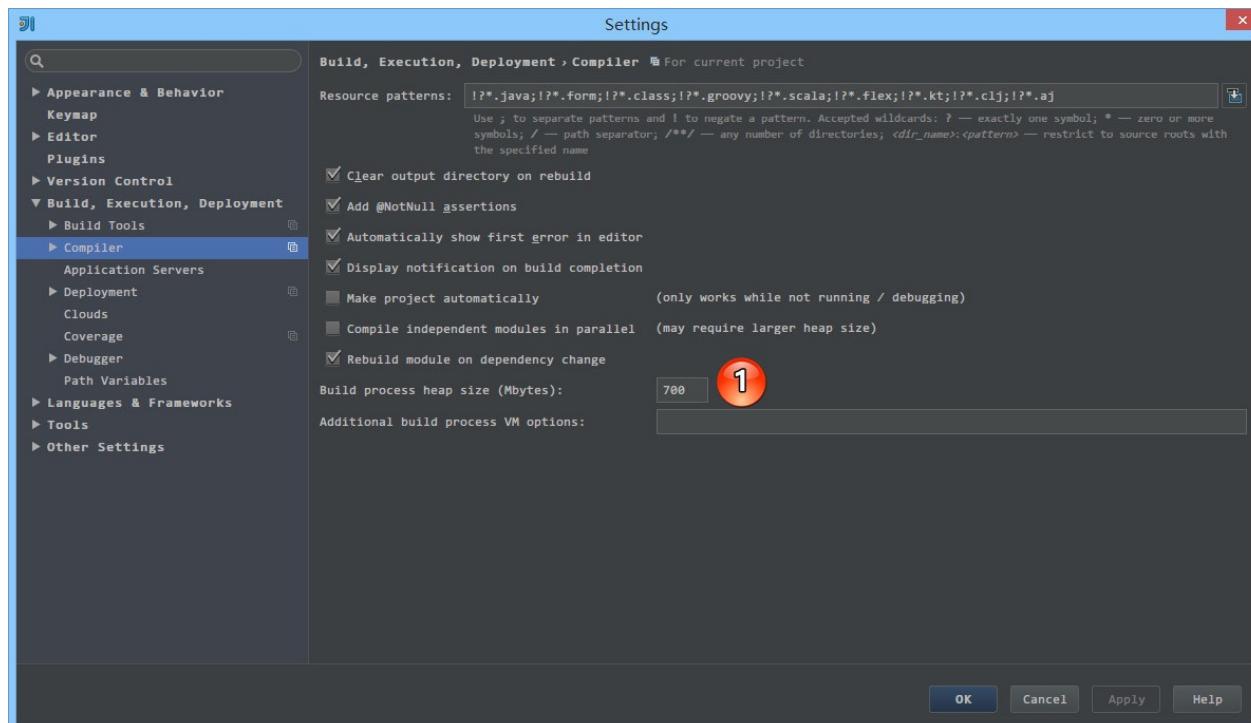
- 如上图标注 1 所示，在打开很多文件的时候，IntelliJ IDEA 默认是把所有打开的文件名 Tab 单行显示的。但是我个人现在的习惯是使用多行，多行效率比单行高，因为单行会隐藏超过界面部分 Tab，这样找文件不方便。



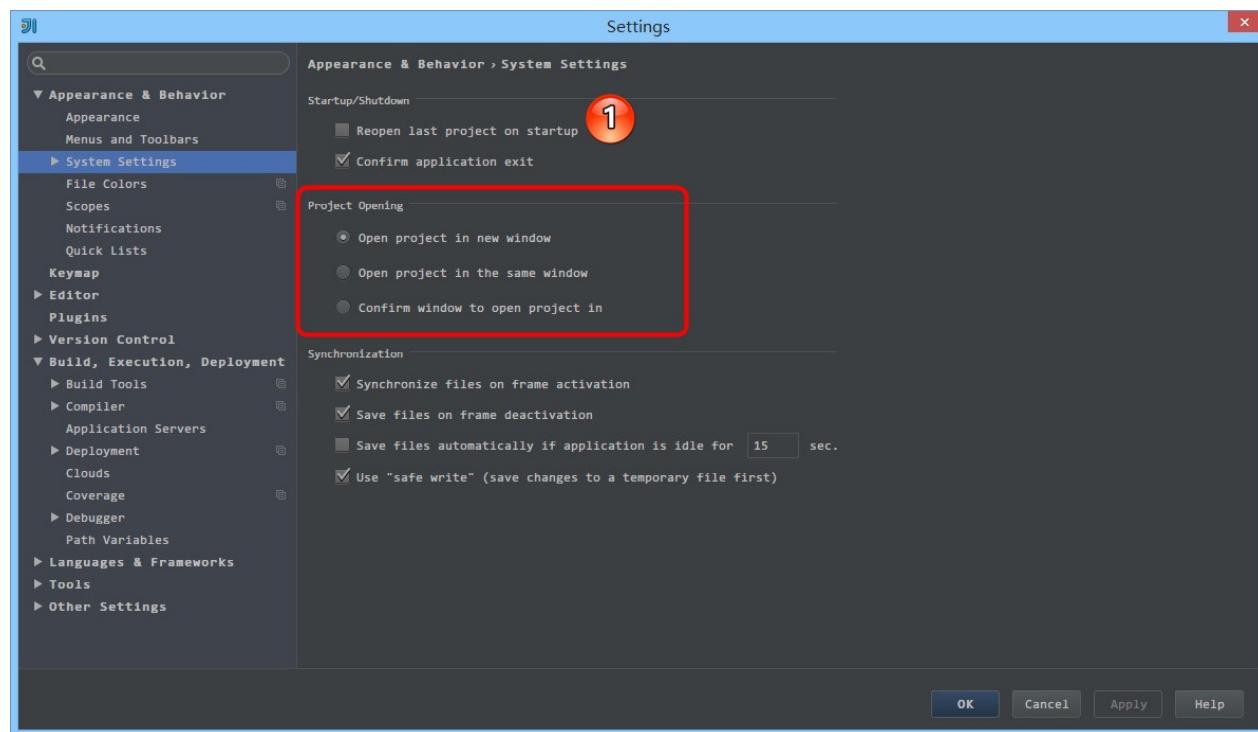
- 如上图 Gif 所示，默认 IntelliJ IDEA 对于 Java 代码的单行注释是把注释的斜杠放在行数的最开头，我个人觉得这样的单行注释非常丑，整个代码风格很难看，所以一般会设置为单行注释的两个斜杠跟随在代码的头部。



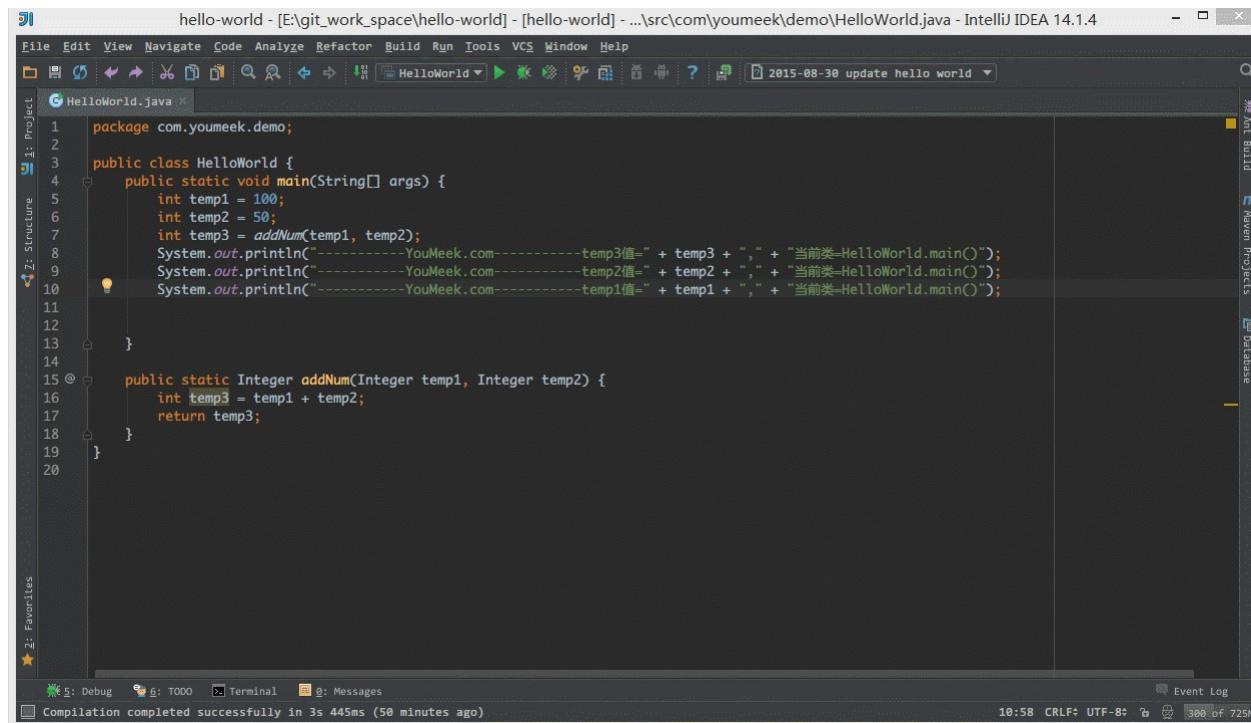
- 如上图 Gif 所示，默认 Java 代码的头个花括号是不换行的，但是有人喜欢对称结构的花括号，可以进行此设置。对此功能我倒是不排斥，我个人也是颇喜欢这种对称结构的，但是由于这种结构会占行，使得文件行数变多，所以虽然我个人喜欢，但是也不这样设置。



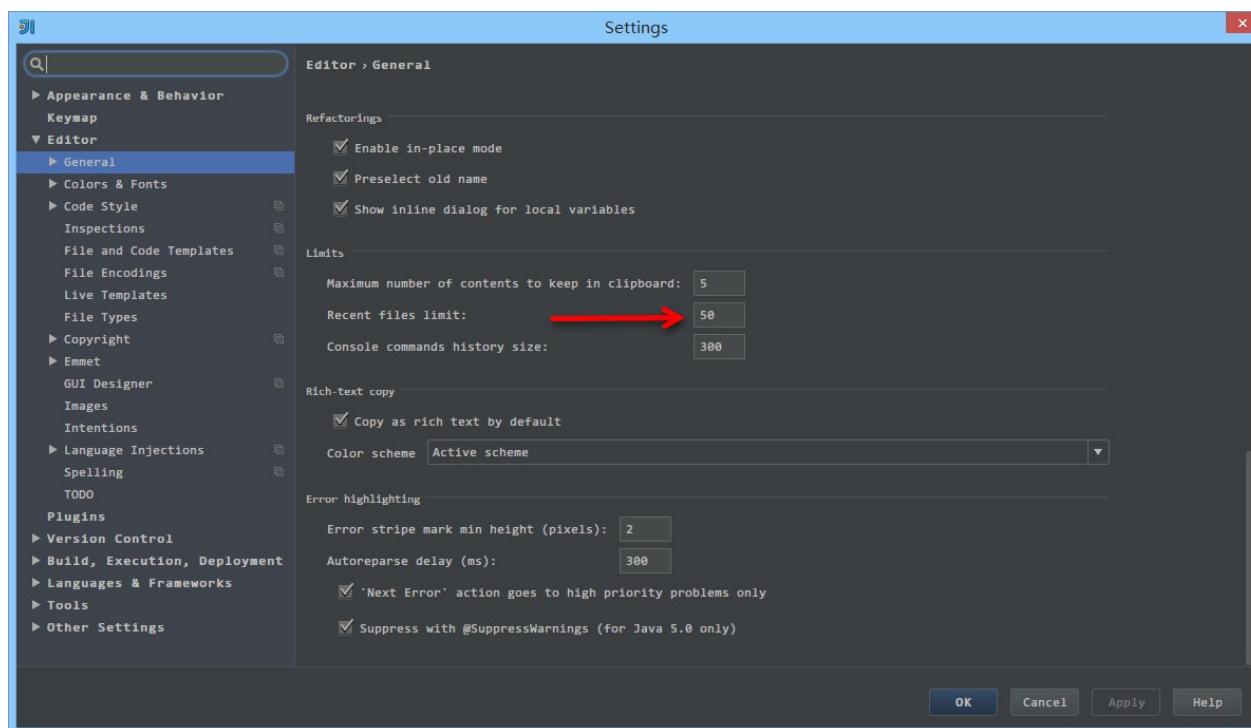
- 如上图标注 1 所示，如果在 make 或 rebuild 过程中很慢，可以增加此堆内存设置，一般大内存的机器设置 1500 以上都是不要紧的。



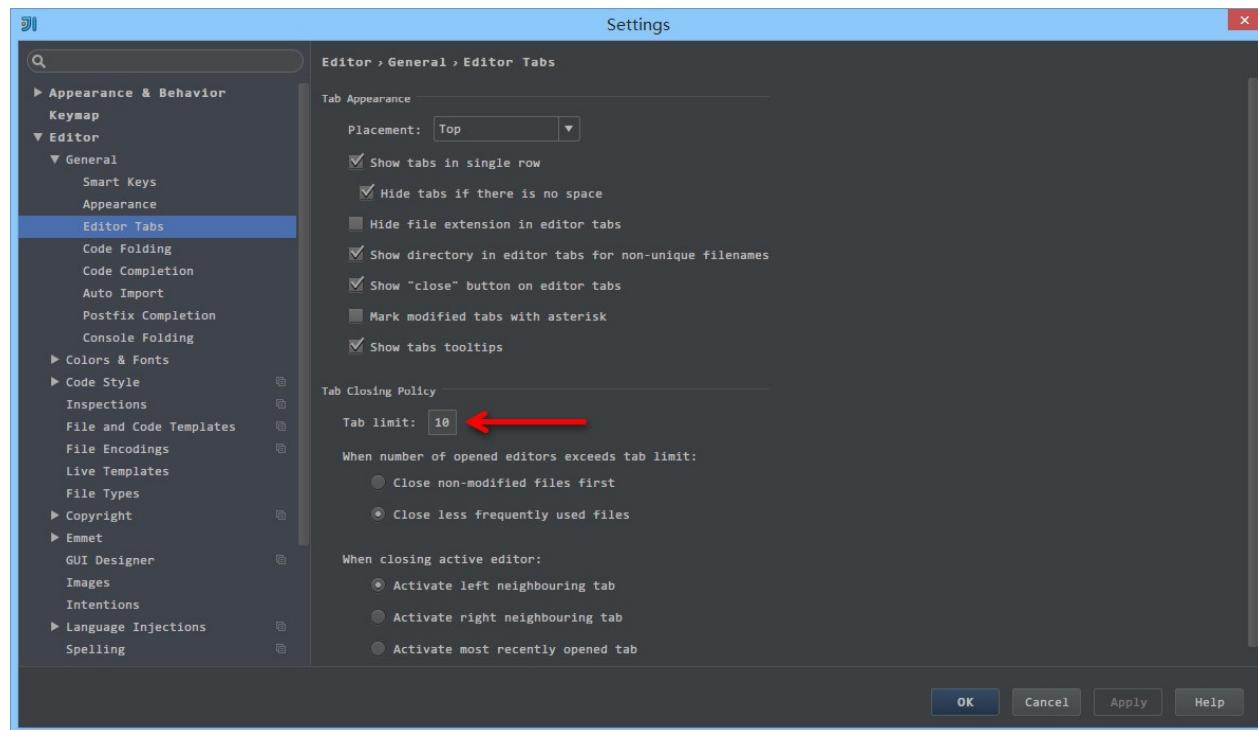
- 如上图标注 1 所示，勾选此选项后，启动 IntelliJ IDEA 的时候，默认会打开上次使用的项目。如果你只有一个项目的话，该功能还是很好用的，但是如果你有多个项目的话，建议还是关闭，这样启动 IntelliJ IDEA 的时候可以选择最近打开的某个项目。
- 如上图红圈所示，该选项是设置当我们已经打开一个项目窗口的时候，再打开一个项目窗口的时候是选择怎样的打开方式。
 - `Open project in new window` 每次都使用新窗口打开。
 - `Open project in the same window` 每次都替换当前已打开的项目，这样桌面上就只有一个项目窗口。
 - `Confirm window to open project in` 每次都弹出提示窗口，让我们选择用新窗口打开或是替换当前项目窗口。



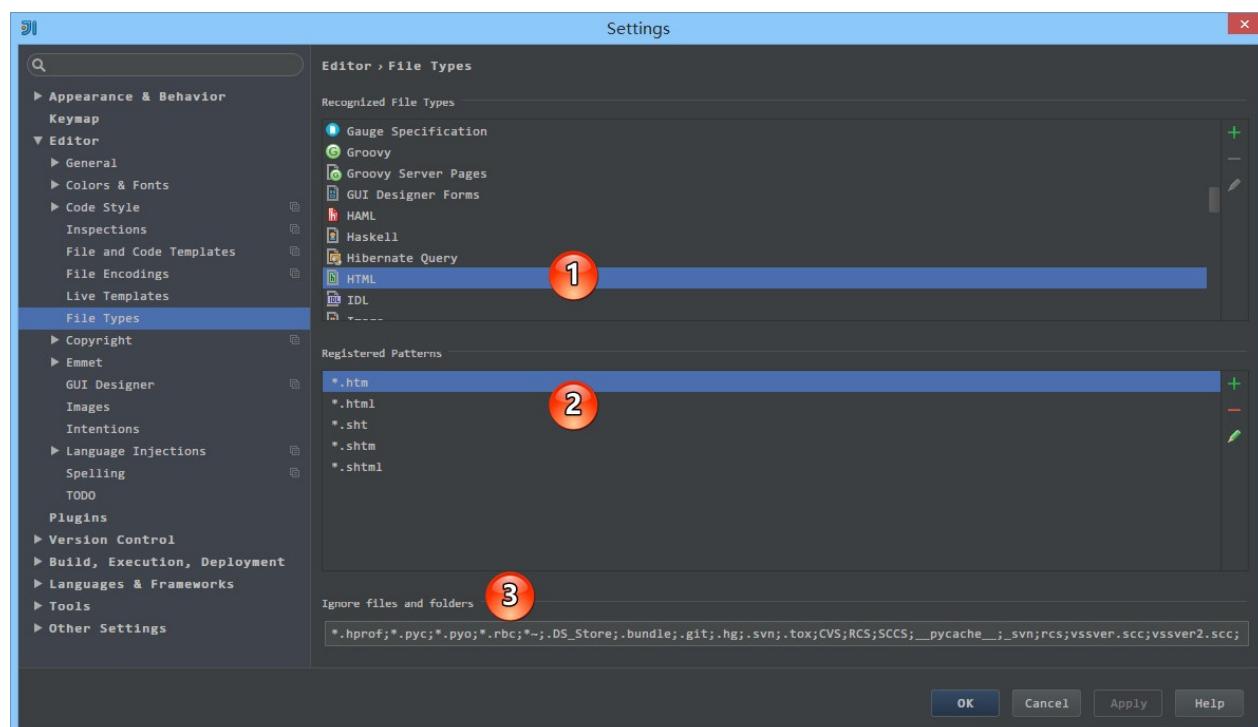
- 如上图 Gif 所示，对于横向太长的代码我们可以进行软分行查看。软分行引起分行效果是 IntelliJ IDEA 设置的，本质代码是没有真的分行的。



- 如上图箭头所示，该设置可以增加 **Ctrl + E** 弹出层显示的记录文件个数。



- 如上图箭头所示，该设置可以增加打开的文件 Tab 个数，当我们打开的文件超过该个数的时候，早打开的文件会被新打开的替换。



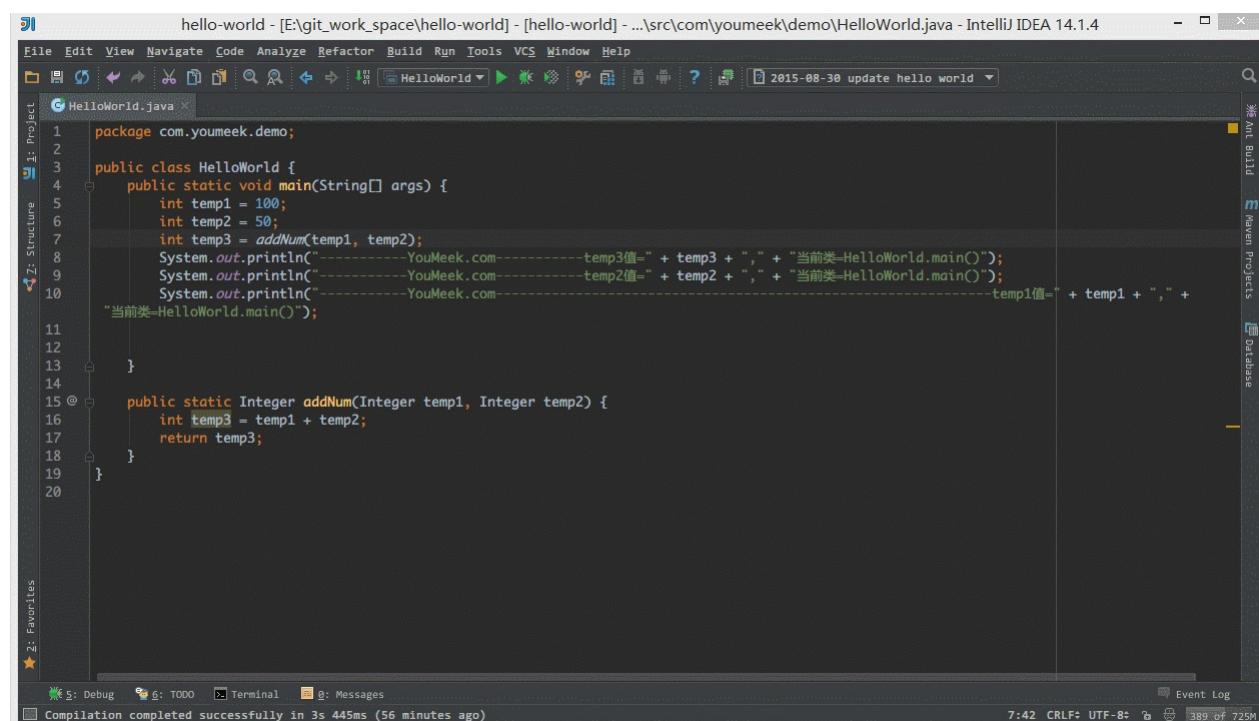
- 如上图标注 1 所示，该区域的后缀类型文件在 IntelliJ IDEA 中将以标注 2 的方式进行打开。
- 如上图标注 3 所示，我们可以在 IntelliJ IDEA 中忽略某些后缀的文件或是文件夹，比如我一般会把 .idea 这个文件夹忽略。

IntelliJ IDEA 常用设置讲解

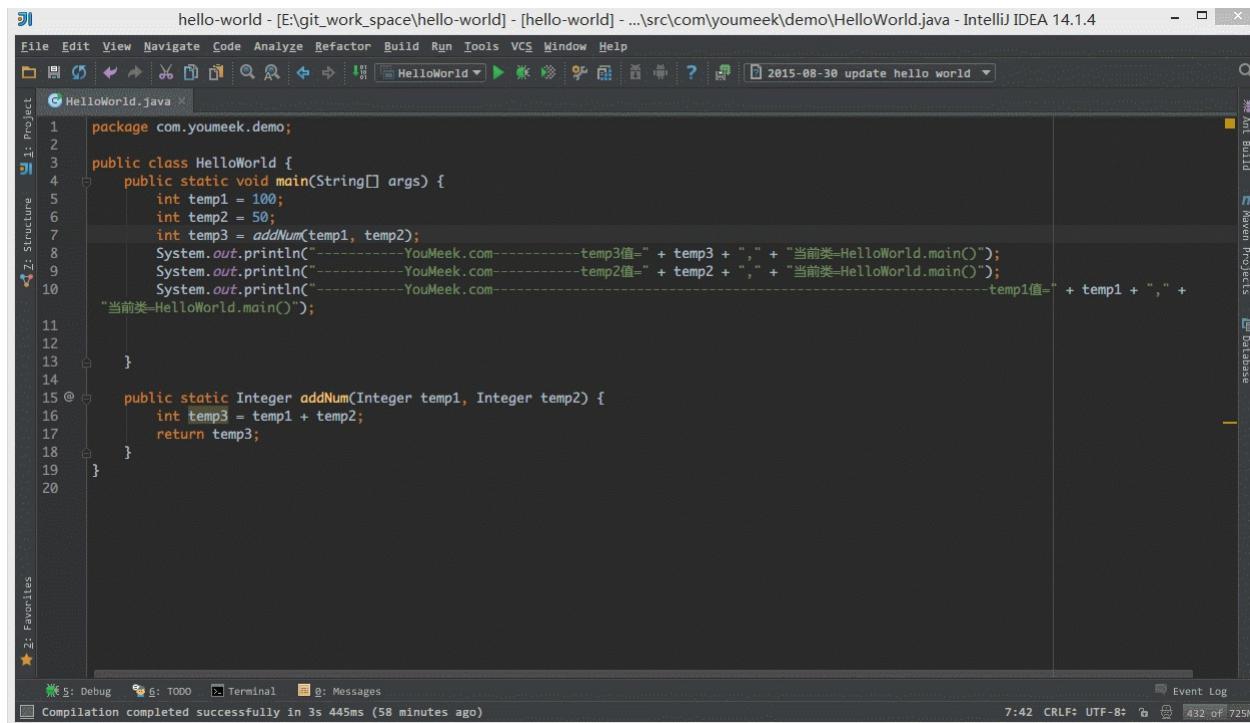
说明

IntelliJ IDEA 有很多人性化的设置我们必须单独拿出来讲解，也因为这些人性化的设置让我们这些 IntelliJ IDEA 死忠粉更加死心塌地使用它和分享它。

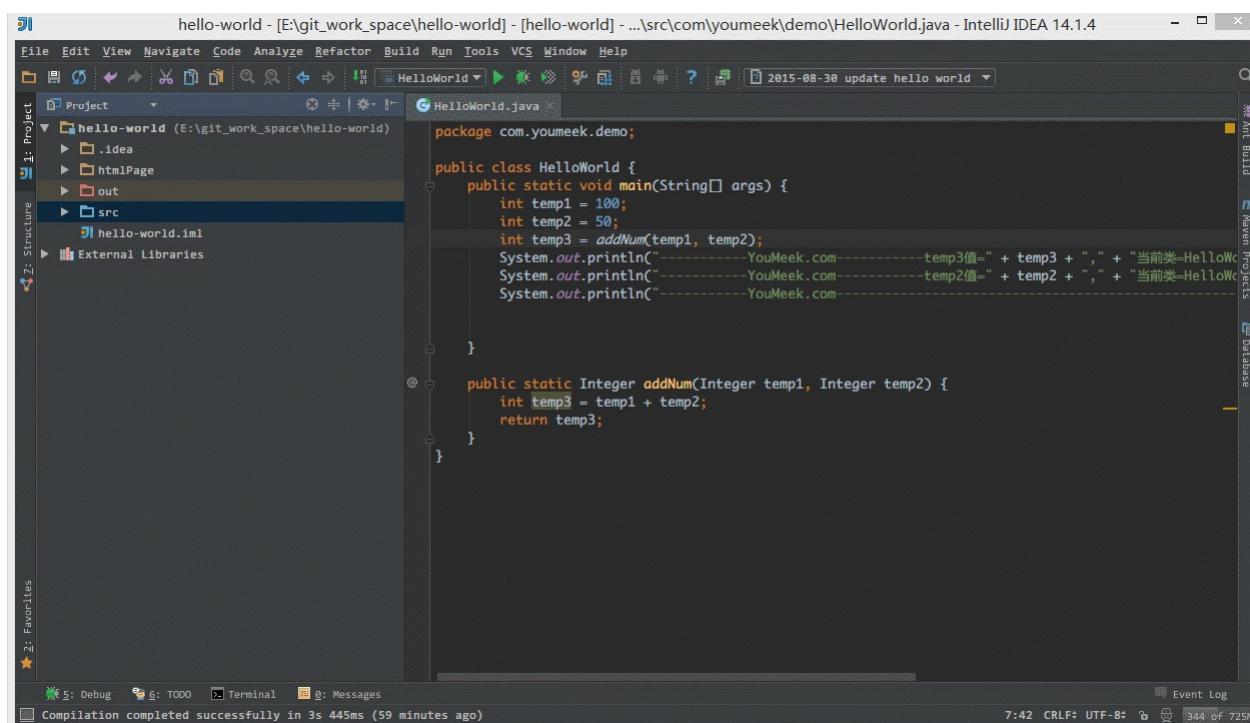
常用设置



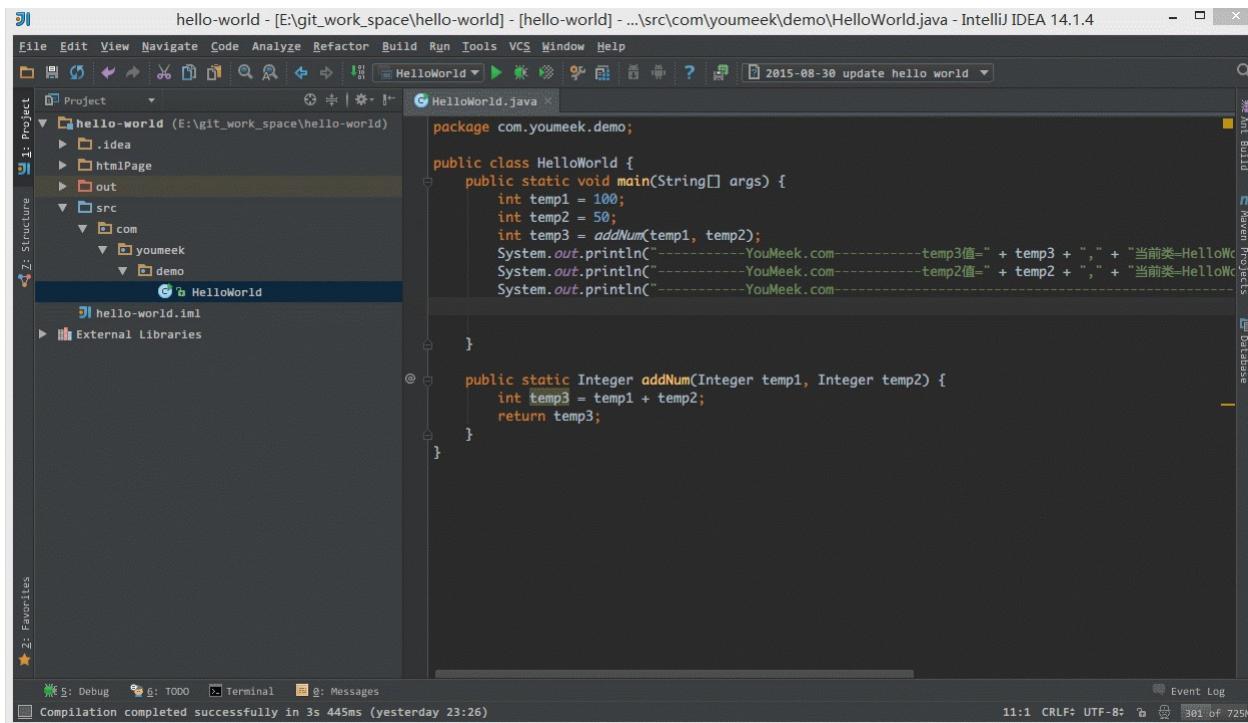
- 如上图 Gif 所示，当我们设置了组件窗口的 Pinned Mode 属性之后，在切换到其他组件窗口的时候，已设置该属性的窗口不会自动隐藏。



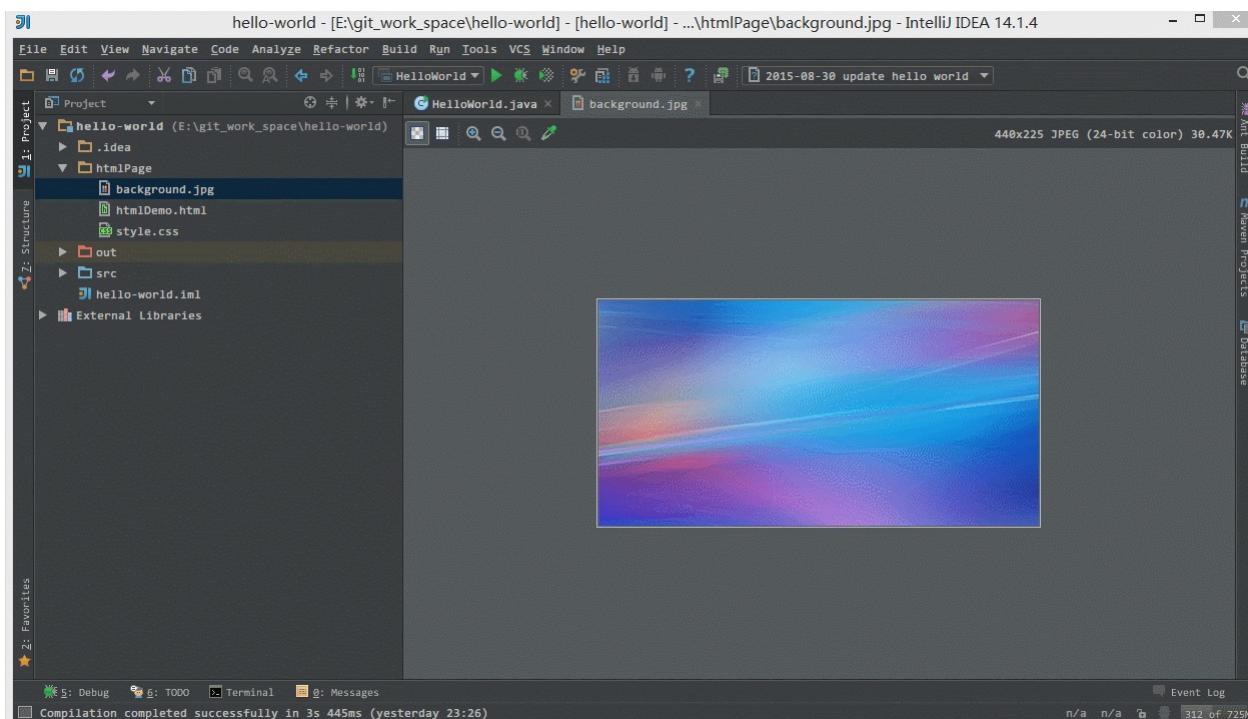
- 如上图 Gif 所示，我们可以对某些文件进行添加到收藏夹，然后在收藏夹组件窗口中可以查看到我们收藏的文件。



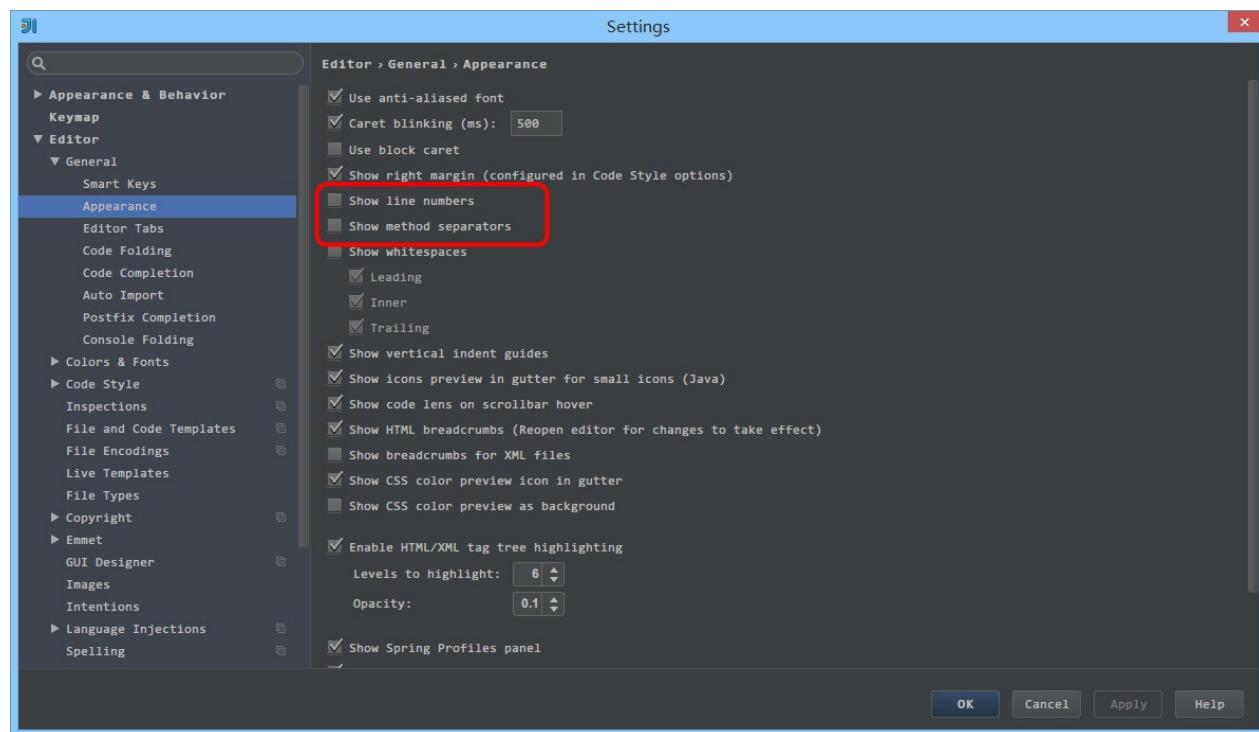
- 如上图 Gif 所示，我们可以通过 **Alt + F1 + 1** 快捷键来定位当前文件所在 Project 组件窗口中的位置。



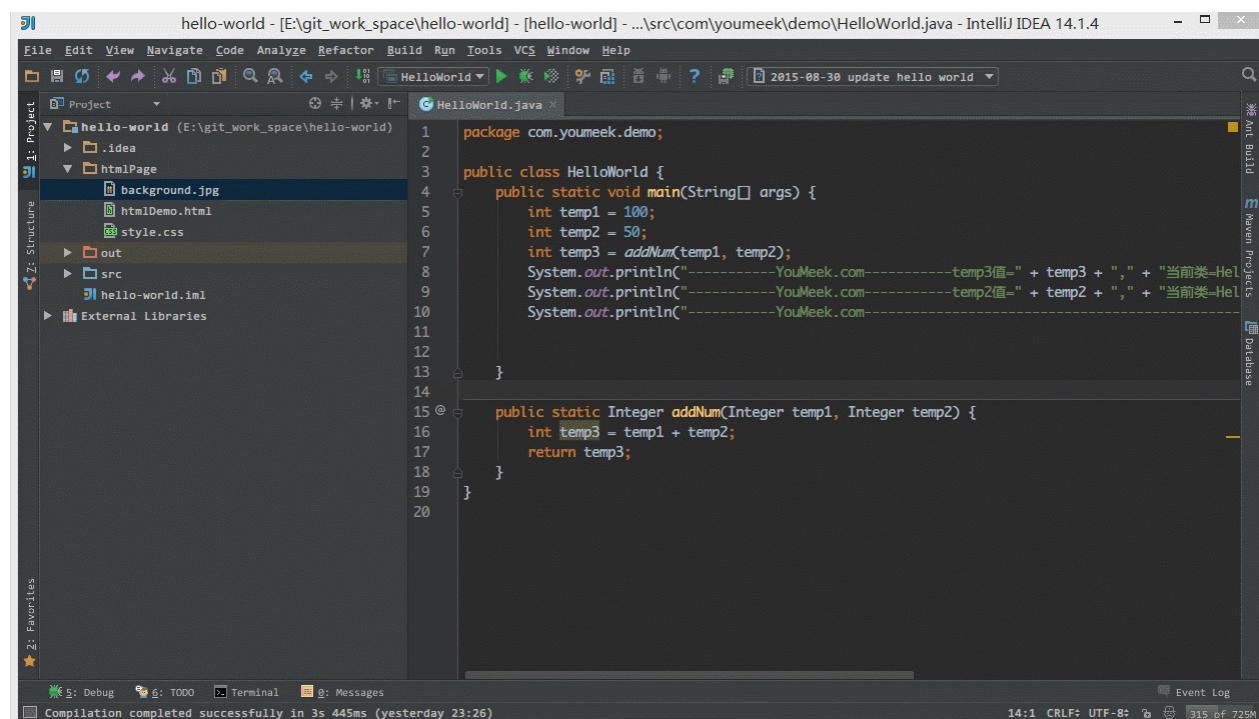
- 如上图 Gif 所示，我们可以勾选此设置后，增加 **Ctrl + 鼠标滚轮** 快捷键来控制代码字体大小显示。



- 如上图 Gif 所示，我们可以勾选此设置后，增加 **Ctrl + 鼠标滚轮** 快捷键来控制图片的大小显示。



- 如上图红圈所示，默认 IntelliJ IDEA 是没有勾选 Show line numbers 显示行数的，但是我建议一般这个要勾选上。
- 如上图红圈所示，默认 IntelliJ IDEA 是没有勾选 Show method separators 显示方法线的，这种线有助于我们区分开方法，所以也是建议勾选上的。



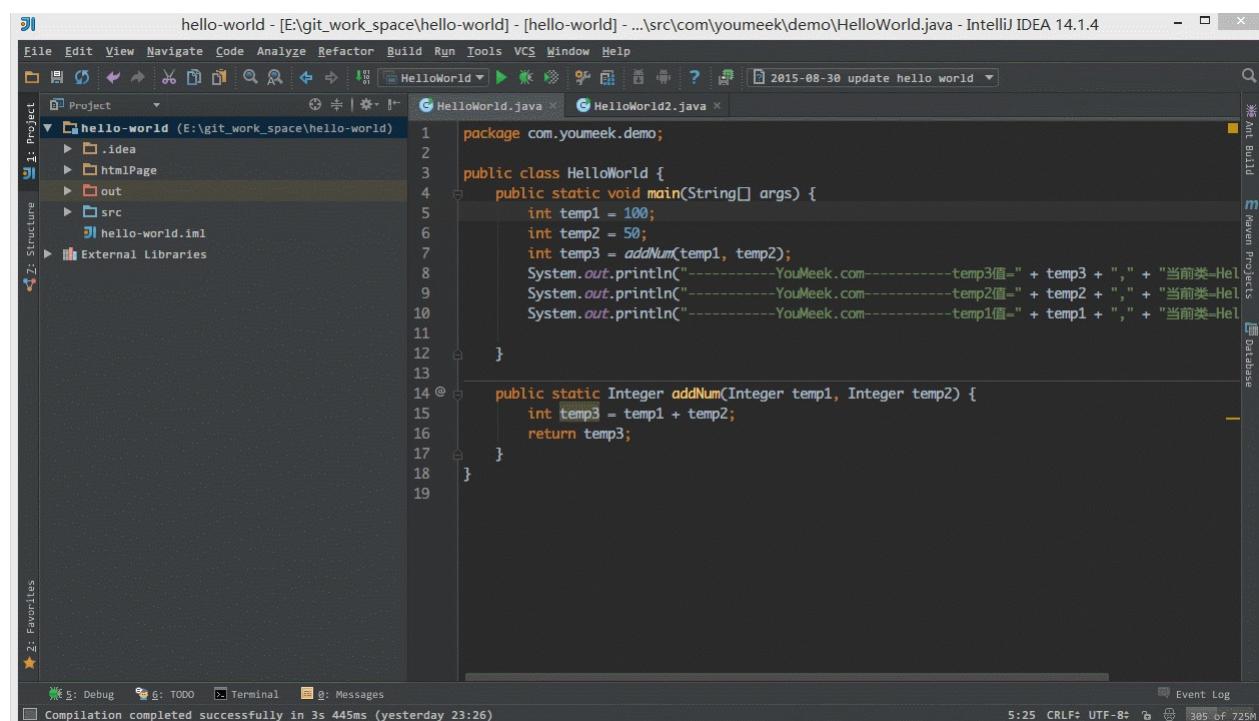
- 如上图 Gif 所示，我们选中要被折叠的代码按 Ctrl + Alt + T 快捷键，选择自定义折叠代码区域功能。

IntelliJ IDEA 常用设置讲解

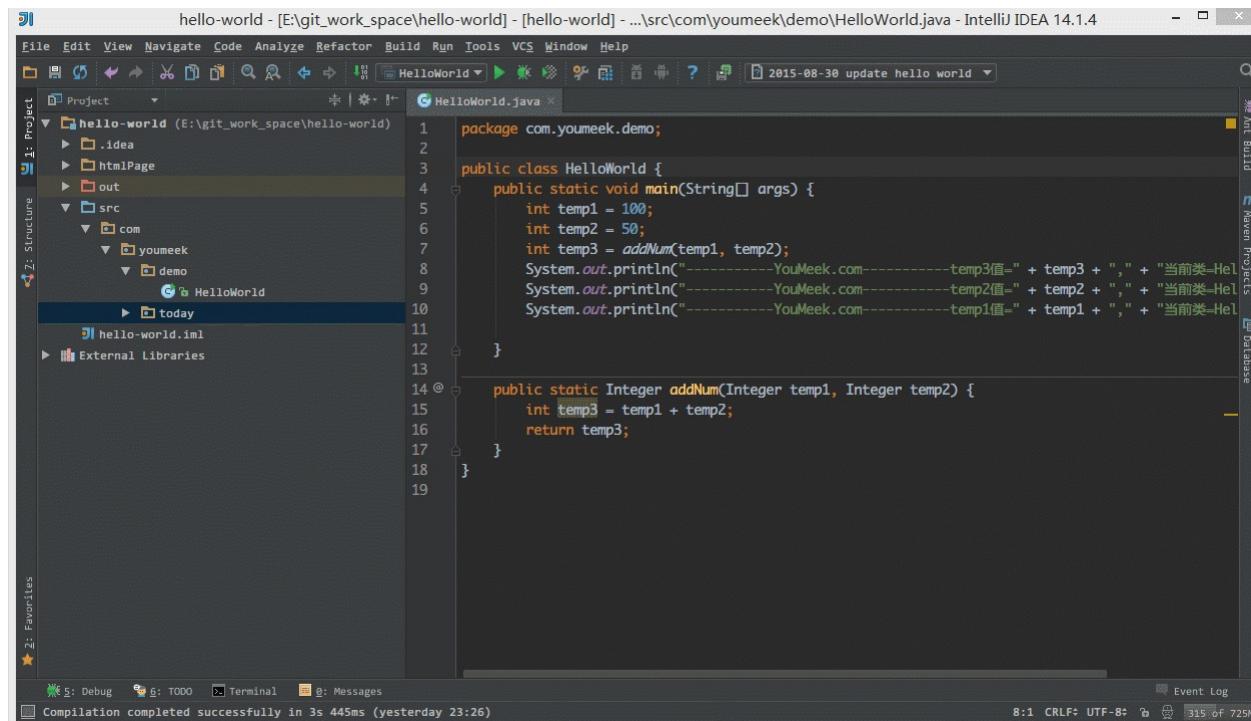
说明

IntelliJ IDEA 有很多人性化的设置我们必须单独拿出来讲解，也因为这些人性化的设置让我们这些 IntelliJ IDEA 死忠粉更加死心塌地使用它和分享它。

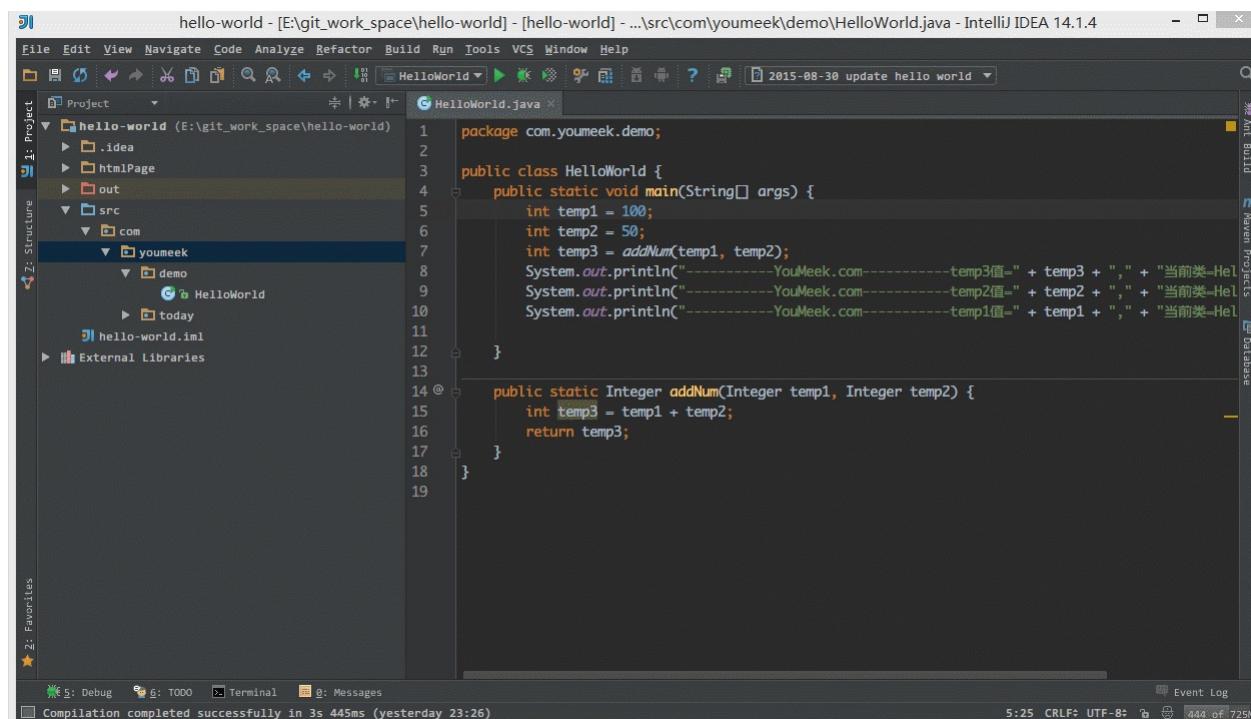
常用设置



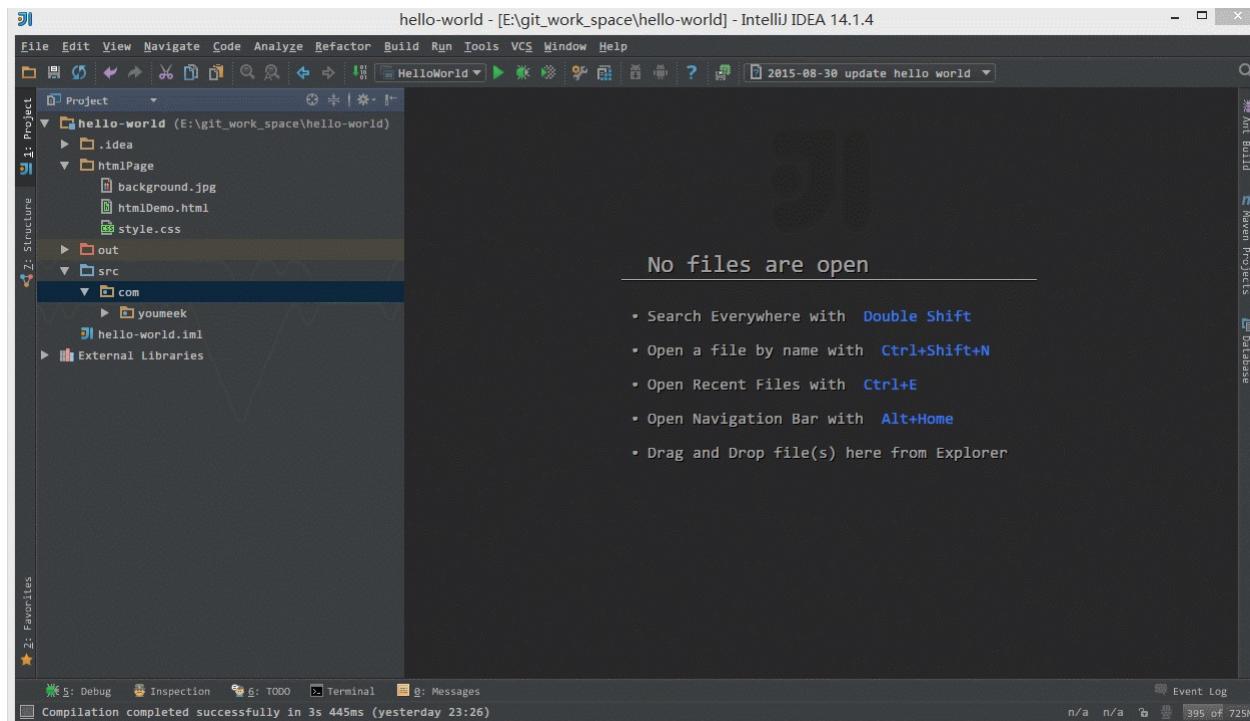
- 如上图 Gif 所示，当我们在编辑某个文件的时候，自动定位到当前文件所在的 Project 组件窗口位置。



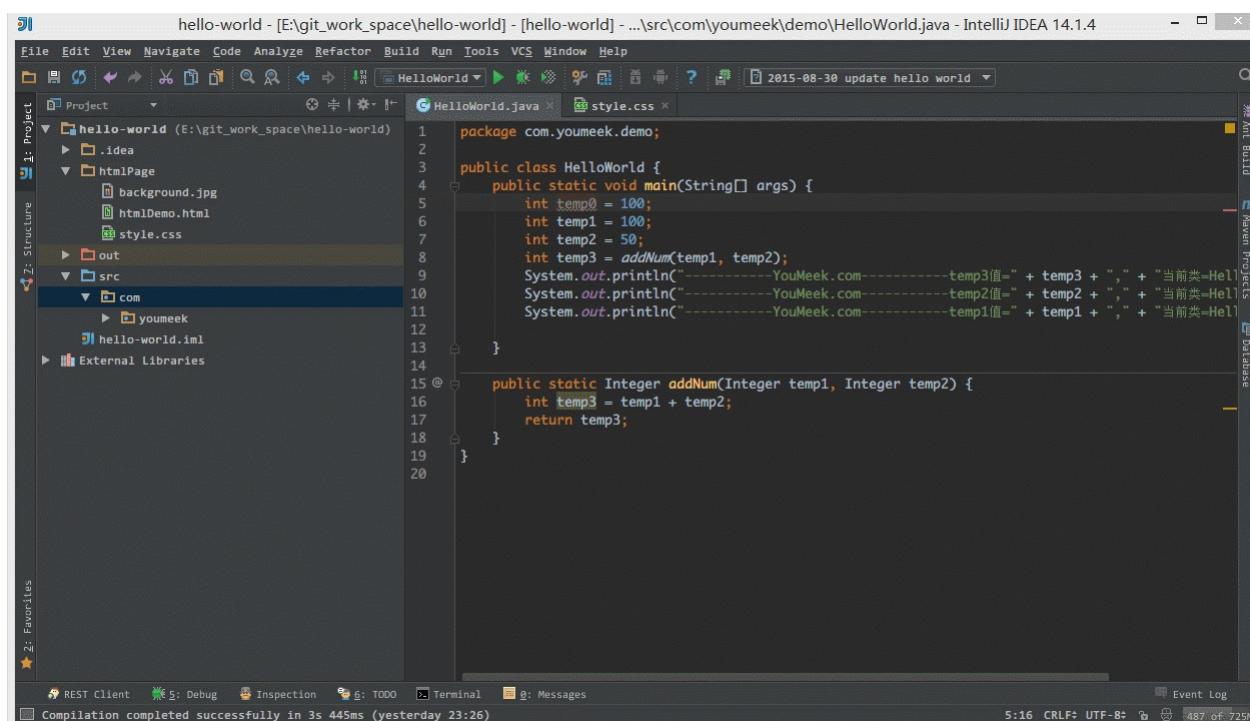
- 如上图 Gif 所示，即使我们项目没有使用版本控制功能，IntelliJ IDEA 也给我们提供了本地文件历史记录。除了简单的记录之外，我们还可以给当前版本加标签。



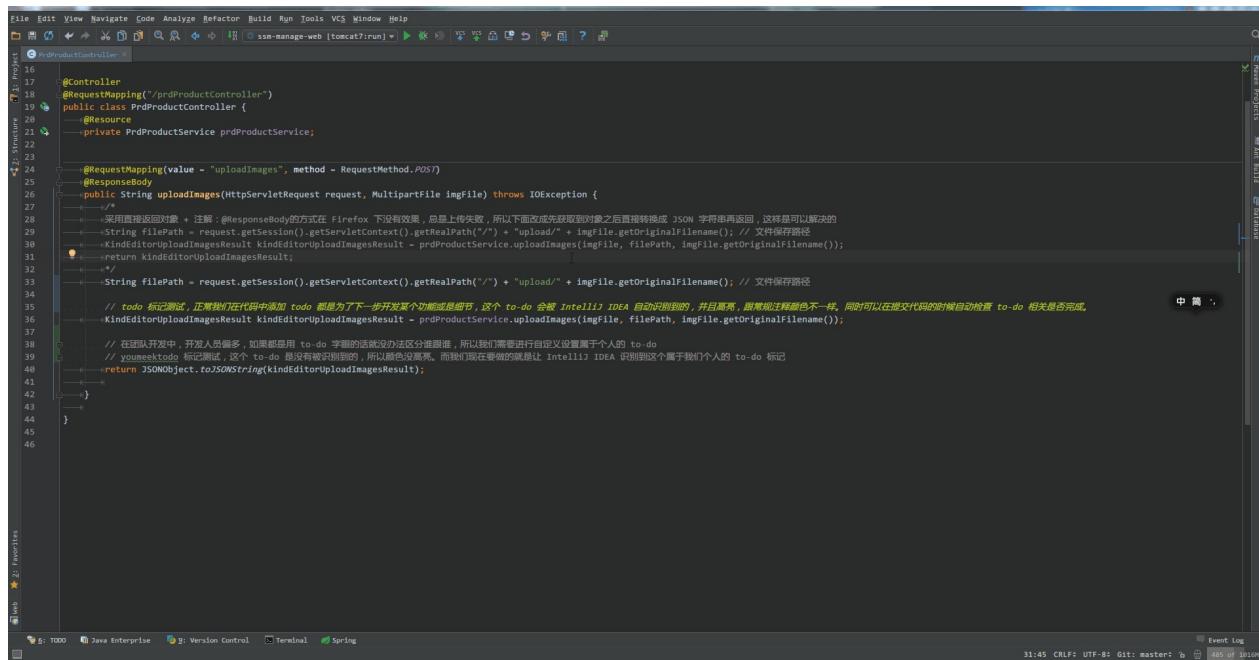
- 如上图 Gif 所示，我们还可以根据选择的代码，查看该段代码的本地历史，这样就省去了查看文件中其他内容的历史了。除了对文件可以查看历史，文件夹也是可以查看各个文件变化的历史。



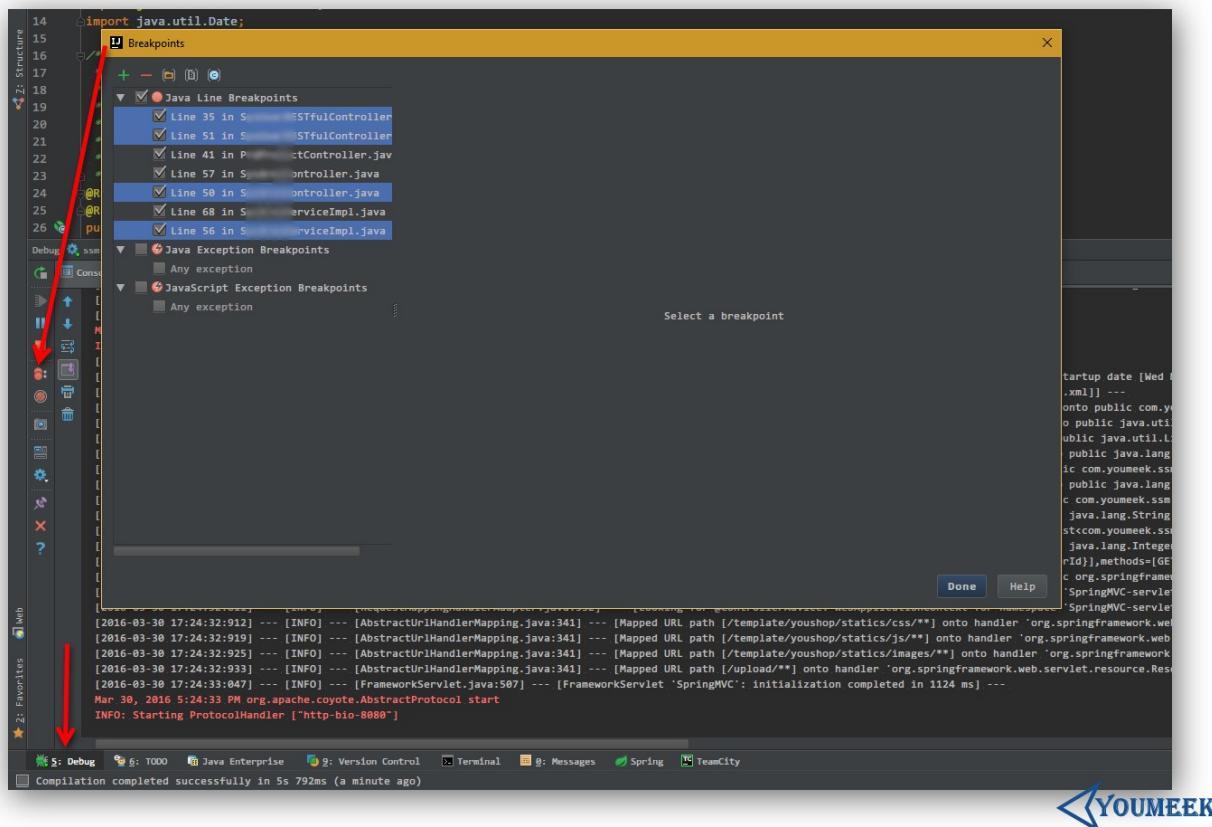
- 如上图 Gif 所示，IntelliJ IDEA 自带了代码检查功能，可以帮我们分析一些简单的语法问题和一些代码细节。



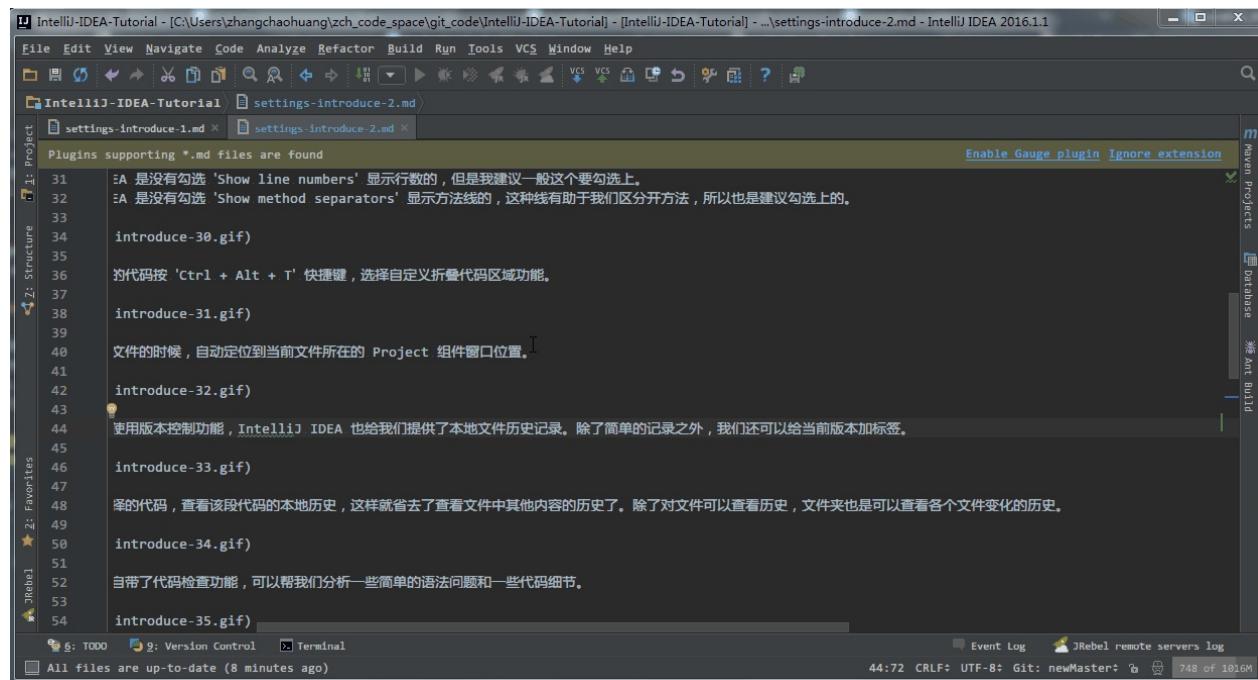
- 如上图 Gif 所示，IntelliJ IDEA 自带模拟请求工具 Rest Client，在开发时用来模拟请求是非常好用的。



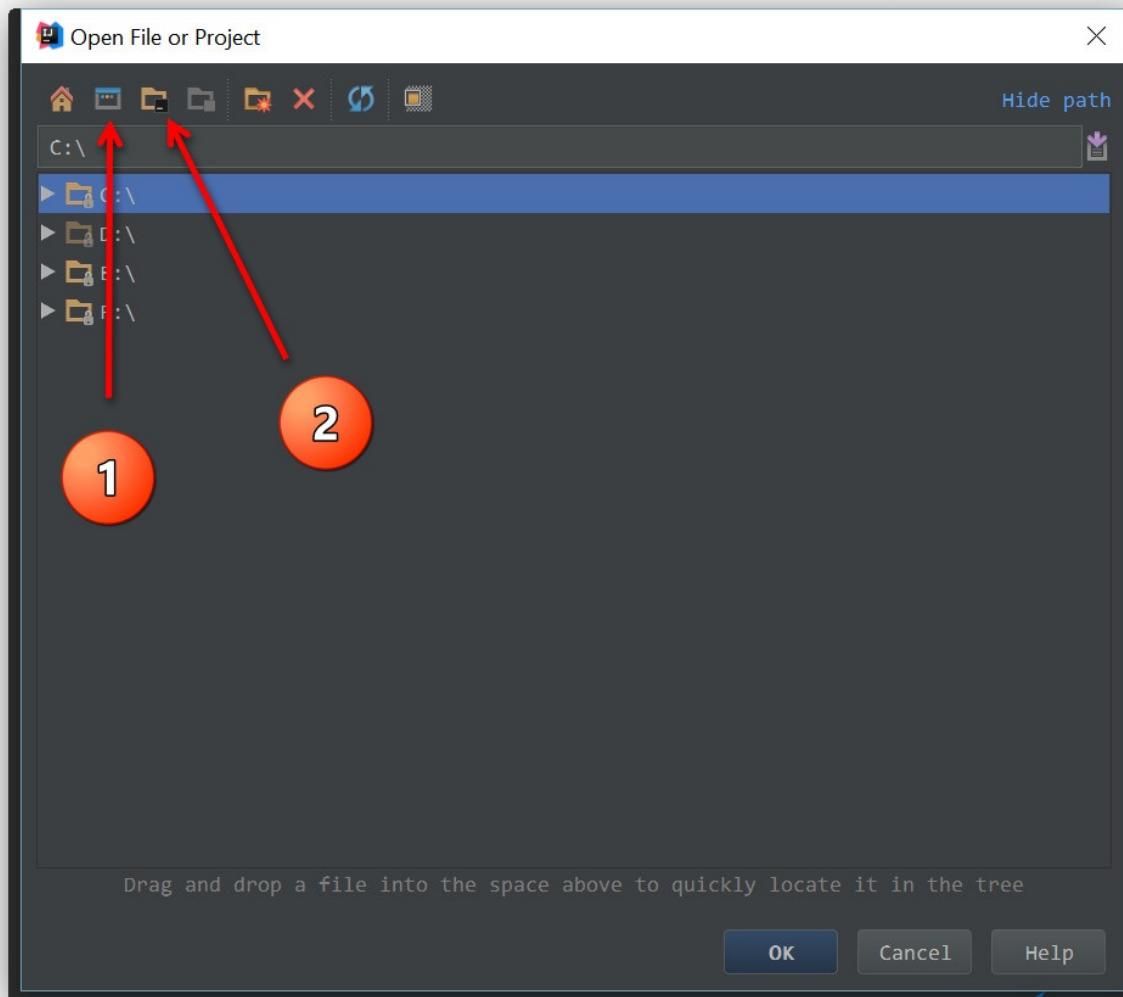
- 如上图 Gif 所示，IntelliJ IDEA 的自定义 TODO 功能非常好用，强烈建议平时开发要经常使用上。



- 如上图箭头所示，在 Debug 状态下，如果我们要批量删除断点，可以点击图上箭头所示的按钮，然后选中要删除的断点按断点上面的减号进行删除。

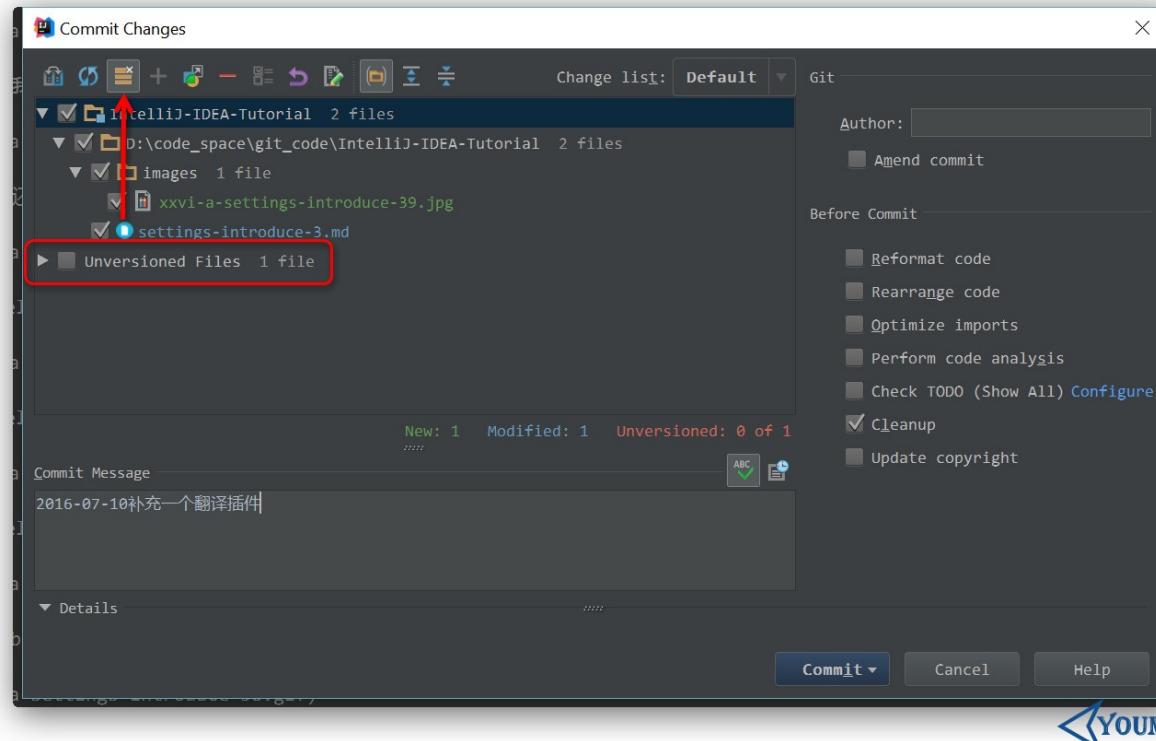


- 默认 IntelliJ IDEA 是有一套自己的 IDE 整体设置的，比如 IDE 设置中我们常修改的：默认编码、编译版本、Maven 本地库路径等等，这些其实是可以避免的。
- 按正常设置 Project 流程的话，我们在 Project 状态下进行 IDE 设置，这些设置并不会被当做一个标准的 IDE 设置模板保存起来，所以你下次打开一个新的 Project 还是要重新设置的，为了简便操作，你可以按上图 Gif 所示进行设置 `Default Settings`，这样下次打开新的 Project 就会以这个 IDE 设置进行。
- 需要注意的是：设置好配置之后，你需要重启 IntelliJ IDEA，重启之后的 IntelliJ IDEA 重新打开 Project 才能有效果。



< YOUMEEK

- 如上图所示，不管是用 IntelliJ IDEA 打开新文件，或是在安装本地插件，在弹出的窗口中，图 1 按钮支持快速定位到系统桌面目录，图 2 按钮支持快速定位到当前项目目录。



<YOUIMEEK

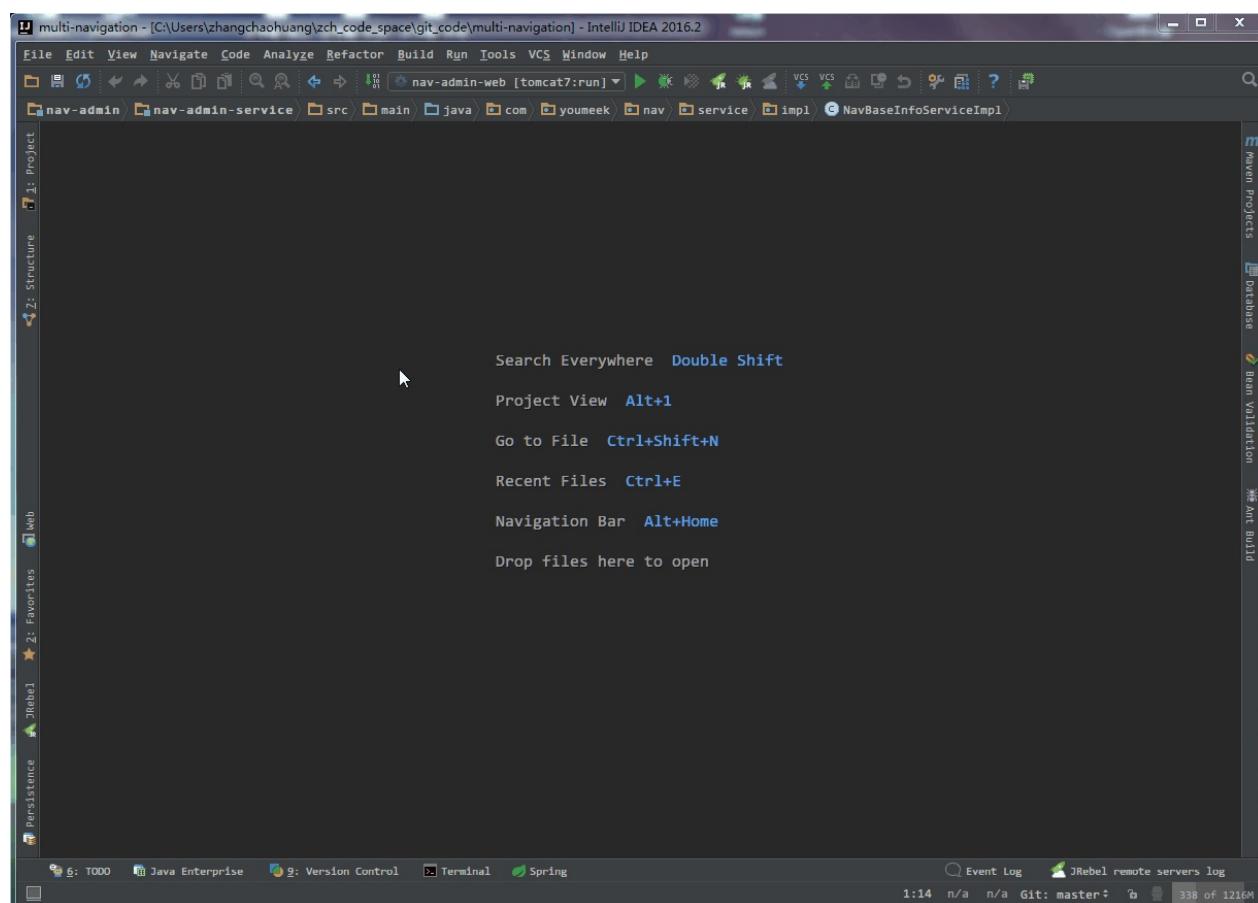
- 2016.2 版本新增箭头指向的按钮，在提交列表里可以显示项目中未加入到版本控制的文件，方便我们在提交的时候做好代码检查，以防漏掉某些文件未提交。

IntelliJ IDEA 常用设置讲解

说明

IntelliJ IDEA 有很多人性化的设置我们必须单独拿出来讲解，也因为这些人性化的设置让我们这些 IntelliJ IDEA 死忠粉更加死心塌地使用它和分享它。

常用设置



- 如上图 Gif 所示，这是一个 Maven 多模块项目，在开发多模块的时候，经常会改到其他模块的代码，而模块与模块之间是相互依赖，如果不进行 install 就没办法使用到最新的依赖。
- 所以，为了减少自己手动 install 的过程，可以把 install 过程放在项目启动之前，就像 Gif 所示那样。

快捷键

说明

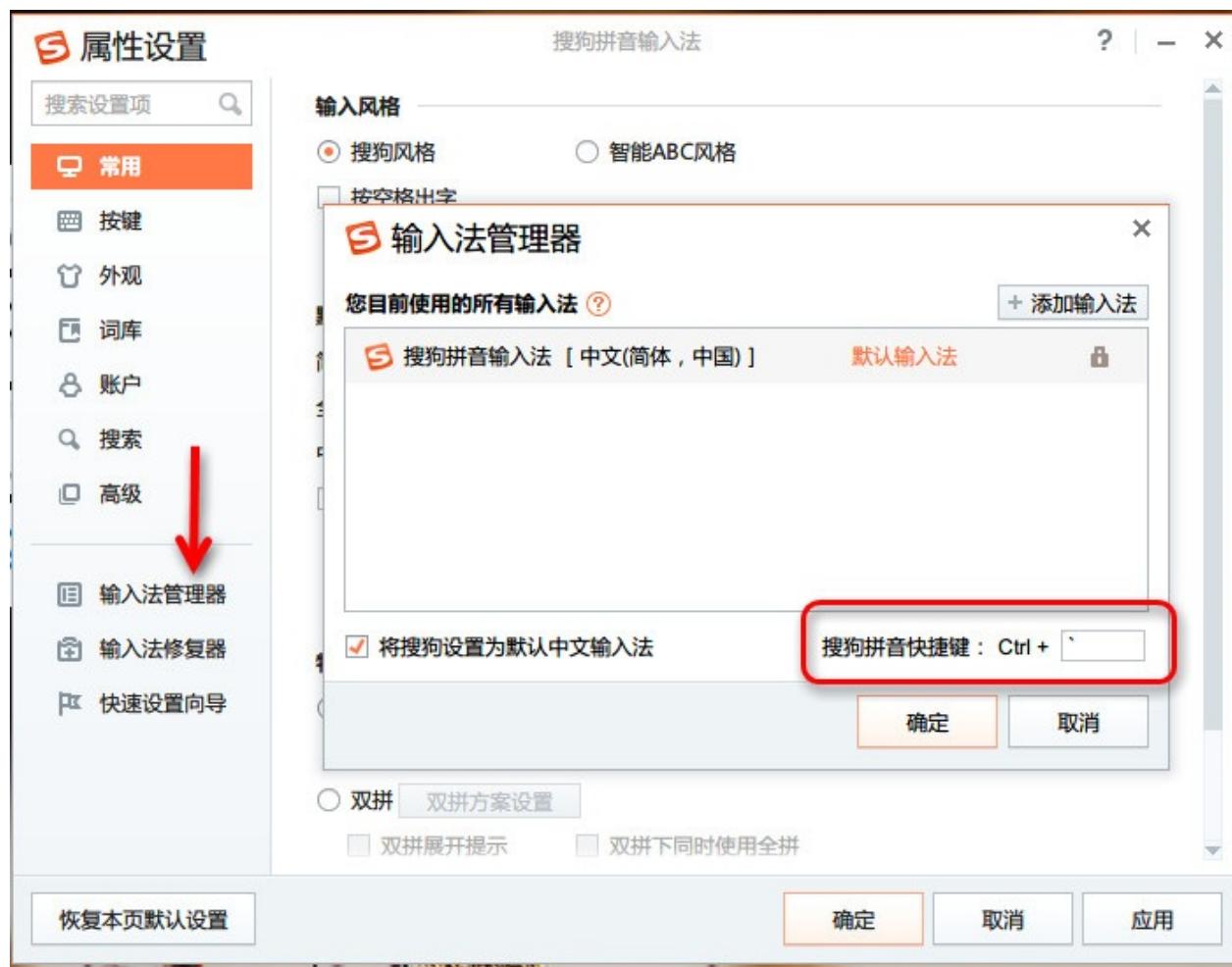
IntelliJ IDEA 的便捷操作性，快捷键的功劳占了一大半，对于各个快捷键组合请认真对待。 IntelliJ IDEA 本身的设计思维是提倡键盘优先于鼠标的，所以各种快捷键组合层出不穷，对于快捷键设置也有各种支持，对于其他 IDE 的快捷键组合也有预设模板进行支持。

关于各个快捷键的频率分类上可能每个人都有各自的看法，下面的整理也只是以我个人的使用习惯来划分的，而我应该是可以代表某一部分小众人员。但是我个人还是强烈建议你可以在我的基础上整理一份属于你的快捷键目录（删除掉多余的字眼，只保留快捷键内容），本篇文章也只是起到一个工具和引子的作用。

对于下面各个快捷键的使介绍描述也许用我个人语言翻译起来不够准确或是不全面，且在不同的文件类型上按出来的效果也可能结果不太一样，所以强烈建议你自己把各个快捷键都亲自操作下体会下各个快捷键的实际用法。

前提

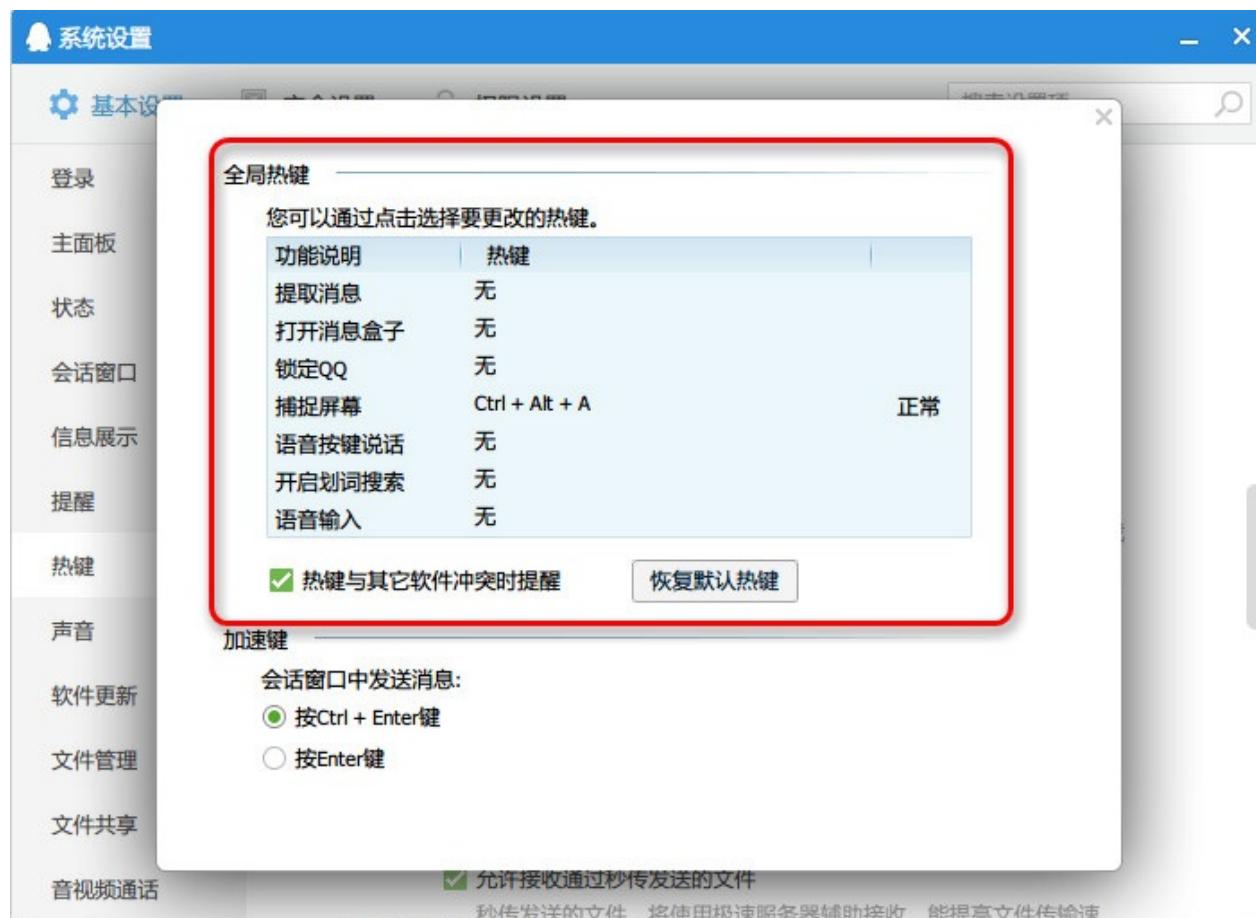
由于文化的不同，我们使用的电脑必备一个软件就是中文输入法，而目前大多数人都使用搜狗拼音输入法或是其他类似的。而这些输入法跟 IntelliJ IDEA 有一个万恶的冲突永恒不变：快捷键冲突。所以为了配合 IntelliJ IDEA，我们要去掉这些输入法下的所有快捷键。



- 如上图红色圈住内容所示，默认是 逗号 我改为了 ESC 键下的 波浪号 ， Ctrl + 逗号 这个快捷键适合做智能提示用，下面的快捷键列表会讲。



- 如上图红色圈住内容所示，这些快捷键很容易跟 IntelliJ IDEA 快捷键冲突，需要全部去掉。



- 如上图红色圈住内容所示，QQ 这些快捷键也很容易跟 IntelliJ IDEA 快捷键冲突，需要全部去掉，最多保持一个截图功能即可。

可能还有其他一些软件，比如网易云音乐、有道词典等等这些软件都可能存在快捷键冲突，所以为了 IntelliJ IDEA 这些软件的快捷键都是值得舍弃的，如果你在开发的时候。

Ctrl

快捷键	介绍
Ctrl + F	在当前文件进行文本查找 (必备)
Ctrl + R	在当前文件进行文本替换 (必备)
Ctrl + Z	撤销 (必备)
Ctrl + Y	删除光标所在行 或 删除选中的行 (必备)
Ctrl + X	剪切光标所在行 或 剪切选择内容
Ctrl + C	复制光标所在行 或 复制选择内容
Ctrl + D	复制光标所在行 或 复制选择内容，并把复制内容插入光标位置下面 (必备)
Ctrl + W	递进式选择代码块。可选中光标所在的单词或段落，连续按会在原有选中的基础上再扩展选中范围 (必备)

Ctrl + E	显示最近打开的文件记录列表 (必备)
Ctrl + N	根据输入的类名查找类文件 (必备)
Ctrl + G	在当前文件跳转到指定行处
Ctrl + J	插入自定义动态代码模板 (必备)
Ctrl + P	方法参数提示显示 (必备)
Ctrl + Q	光标所在的变量 / 类名 / 方法名等上面 (也可以在提示补充的时候按) , 显示文档内容
Ctrl + U	前往当前光标所在的方法的父类的方法 / 接口定义 (必备)
Ctrl + B	进入光标所在的方法/变量的接口或是定义处, 等效于 <code>Ctrl + 左键单击</code> (必备)
Ctrl + K	版本控制提交项目, 需要此项目有加入到版本控制才可用
Ctrl + T	版本控制更新项目, 需要此项目有加入到版本控制才可用
Ctrl + H	显示当前类的层次结构
Ctrl + O	选择可重写的方法
Ctrl + I	选择可继承的方法
Ctrl + +	展开代码
Ctrl + -	折叠代码
Ctrl + /	注释光标所在行代码, 会根据当前不同文件类型使用不同的注释符号 (必备)
Ctrl + [移动光标到当前所在代码的花括号开始位置
Ctrl +]	移动光标到当前所在代码的花括号结束位置
Ctrl + F1	在光标所在的错误代码处显示错误信息 (必备)
Ctrl + F3	调转到所选中的词的下一个引用位置 (必备)
Ctrl + F4	关闭当前编辑文件
Ctrl + F8	在 Debug 模式下, 设置光标当前行为断点, 如果当前已经是断点则去掉断点
Ctrl + F9	执行 Make Project 操作
Ctrl + F11	选中文件 / 文件夹, 使用助记符设定 / 取消书签 (必备)
Ctrl + F12	弹出当前文件结构层, 可以在弹出的层上直接输入, 进行筛选
Ctrl + Tab	编辑窗口切换, 如果在切换的过程又加按上 <code>delete</code> , 则是关闭对应选中的窗口
Ctrl + End	跳到文件尾
Ctrl + Home	跳到文件头

Ctrl + Space	基础代码补全，默认在 Windows 系统上被输入法占用，需要进行修改，建议修改为 Ctrl + 逗号 (必备)
Ctrl + Delete	删除光标后面的单词或是中文句 (必备)
Ctrl + BackSpace	删除光标前面的单词或是中文句 (必备)
Ctrl + 1,2,3...9	定位到对应数值的书签位置 (必备)
Ctrl + 左键单击	在打开的文件标题上，弹出该文件路径 (必备)
Ctrl + 光标定位	按 Ctrl 不要松开，会显示光标所在的类信息摘要
Ctrl + 左方向键	光标跳转到当前单词 / 中文句的左侧开头位置 (必备)
Ctrl + 右方向键	光标跳转到当前单词 / 中文句的右侧开头位置 (必备)
Ctrl + 前方向键	等效于鼠标滚轮向前效果 (必备)
Ctrl + 后方向键	等效于鼠标滚轮向后效果 (必备)

Alt

快捷键	介绍
Alt + `	显示版本控制常用操作菜单弹出层 (必备)
Alt + Q	弹出一个提示，显示当前类的声明 / 上下文信息
Alt + F1	显示当前文件选择目标弹出层，弹出层中有很多目标可以进行选择 (必备)
Alt + F2	对于前面页面，显示各类浏览器打开目标选择弹出层
Alt + F3	选中文本，逐个往下查找相同文本，并高亮显示
Alt + F7	查找光标所在的方法 / 变量 / 类被调用的地方
Alt + F8	在 Debug 的状态下，选中对象，弹出可输入计算表达式调试框，查看该输入内容的调试结果
Alt + Home	定位 / 显示到当前文件的 Navigation Bar
Alt + Enter	IntelliJ IDEA 根据光标所在问题，提供快速修复选择，光标放在的位置不同提示的结果也不同 (必备)
Alt + Insert	代码自动生成，如生成对象的 set / get 方法，构造函数，toString() 等 (必备)
Alt + 左方向键	切换当前已打开的窗口中的子视图，比如Debug窗口中有Output、Debugger等子视图，用此快捷键就可以在子视图中切换 (必备)
Alt + 右方向键	按切换当前已打开的窗口中的子视图，比如Debug窗口中有Output、Debugger等子视图，用此快捷键就可以在子视图中切换 (必备)
Alt + 前方向键	当前光标跳转到当前文件的前一个方法名位置 (必备)
Alt + 后方向键	当前光标跳转到当前文件的后一个方法名位置 (必备)
Alt + 1,2,3...9	显示对应数值的选项卡，其中 1 是 Project 用得最多 (必备)

Shift

快捷键	介绍
Shift + F1	如果有外部文档可以连接外部文档
Shift + F2	跳转到上一个高亮错误 或 警告位置
Shift + F3	在查找模式下，查找匹配上一个
Shift + F4	对当前打开的文件，使用新Windows窗口打开，旧窗口保留
Shift + F6	对文件 / 文件夹 重命名
Shift + F7	在 Debug 模式下，智能步入。断点所在行上有多个方法调用，会弹出进入哪个方法
Shift + F8	在 Debug 模式下，跳出，表现出来的效果跟 F9 一样
Shift + F9	等效于点击工具栏的 Debug 按钮
Shift + F10	等效于点击工具栏的 Run 按钮
Shift + F11	弹出书签显示层 (必备)
Shift + Tab	取消缩进 (必备)
Shift + ESC	隐藏当前 或 最后一个激活的工具窗口
Shift + End	选中光标到当前行尾位置
Shift + Home	选中光标到当前行头位置
Shift + Enter	开始新一行。光标所在行下空出一行，光标定位到新行位置 (必备)
Shift + 左键单击	在打开的文件名上按此快捷键，可以关闭当前打开文件 (必备)
Shift + 滚轮前后滚动	当前文件的横向滚动轴滚动 (必备)

Ctrl + Alt

快捷键	介绍
Ctrl + Alt + L	格式化代码，可以对当前文件和整个包目录使用 (必备)
Ctrl + Alt + O	优化导入的类，可以对当前文件和整个包目录使用 (必备)
Ctrl + Alt + I	光标所在行 或 选中部分进行自动代码缩进，有点类似格式化
Ctrl + Alt + T	对选中的代码弹出环绕选项弹出层 (必备)
Ctrl + Alt + J	弹出模板选择窗口，将选定的代码加入动态模板中
Ctrl + Alt + H	调用层次
Ctrl + Alt + B	在某个调用的方法名上使用会跳到具体的实现处，可以跳过接口
Ctrl + Alt + V	快速引进变量
Ctrl + Alt + Y	同步、刷新
Ctrl + Alt + S	打开 IntelliJ IDEA 系统设置 (必备)
Ctrl + Alt + F7	显示使用的地方。寻找被该类或是变量被调用的地方，用弹出框的方式找出来
Ctrl + Alt + F11	切换全屏模式
Ctrl + Alt + Enter	光标所在行上空出一行，光标定位到新行 (必备)
Ctrl + Alt + Home	弹出跟当前文件有关联的文件弹出层
Ctrl + Alt + Space	类名自动完成
Ctrl + Alt + 左方向键	退回到上一个操作的地方 (必备)
Ctrl + Alt + 右方向键	前进到上一个操作的地方 (必备)
Ctrl + Alt + 前方方向键	在查找模式下，跳到上个查找的文件
Ctrl + Alt + 后方方向键	在查找模式下，跳到下个查找的文件

Ctrl + Shift

快捷键	介绍
Ctrl + Shift + F	根据输入内容查找整个项目 或 指定目录内文件 (必备)
Ctrl + Shift + R	根据输入内容替换对应内容，范围为整个项目 或 指定目录内文件 (必备)
Ctrl + Shift +	自动将下一行合并到当前行末尾 (必备)

J	自动将下一行合并到当前行末尾 (必备)
Ctrl + Shift + Z	取消撤销 (必备)
Ctrl + Shift + W	递进式取消选择代码块。可选中光标所在的单词或段落，连续按会在原有选中的基础上再扩展取消选中范围 (必备)
Ctrl + Shift + N	通过文件名定位 / 打开文件 / 目录，打开目录需要在输入的内容后面多加一个正斜杠 (必备)
Ctrl + Shift + U	对选中的代码进行大 / 小写轮流转换 (必备)
Ctrl + Shift + T	对当前类生成单元测试类，如果已经存在的单元测试类则可以进行选择 (必备)
Ctrl + Shift + C	复制当前文件磁盘路径到剪贴板 (必备)
Ctrl + Shift + V	弹出缓存的最近拷贝的内容管理器弹出层
Ctrl + Shift + E	显示最近修改的文件列表的弹出层
Ctrl + Shift + H	显示方法层次结构
Ctrl + Shift + B	跳转到类型声明处 (必备)
Ctrl + Shift + I	快速查看光标所在的方法 或 类的定义
Ctrl + Shift + A	查找动作 / 设置
Ctrl + Shift + /	代码块注释 (必备)
Ctrl + Shift + [选中从光标所在位置到它的顶部中括号位置 (必备)
Ctrl + Shift +]	选中从光标所在位置到它的底部中括号位置 (必备)
Ctrl + Shift + +	展开所有代码 (必备)
Ctrl + Shift + -	折叠所有代码 (必备)
Ctrl + Shift + F7	高亮显示所有该选中文本，按Esc高亮消失 (必备)
Ctrl + Shift + F8	在 Debug 模式下，指定断点进入条件

Ctrl + Shift + F9	编译选中的文件 / 包 / Module
Ctrl + Shift + F12	编辑器最大化 (必备)
Ctrl + Shift + Space	智能代码提示
Ctrl + Shift + Enter	自动结束代码，行末自动添加分号 (必备)
Ctrl + Shift + Backspace	退回到上次修改的地方 (必备)
Ctrl + Shift + 1,2,3...9	快速添加指定数值的书签 (必备)
Ctrl + Shift + 左键单击	把光标放在某个类变量上，按此快捷键可以直接定位到该类中 (必备)
Ctrl + Shift + 左方向键	在代码文件上，光标跳转到当前单词 / 中文句的左侧开头位置，同时选中该单词 / 中文句 (必备)
Ctrl + Shift + 右方向键	在代码文件上，光标跳转到当前单词 / 中文句的右侧开头位置，同时选中该单词 / 中文句 (必备)
Ctrl + Shift + 前方向键	光标放在方法名上，将方法移动到上一个方法前面，调整方法排序 (必备)
Ctrl + Shift + 后方向键	光标放在方法名上，将方法移动到下一个方法前面，调整方法排序 (必备)

Alt + Shift

快捷键	介绍
Alt + Shift + N	选择 / 添加 task (必备)
Alt + Shift + F	显示添加到收藏夹弹出层 / 添加到收藏夹
Alt + Shift + C	查看最近操作项目的变化情况列表
Alt + Shift + I	查看项目当前文件
Alt + Shift + F7	在 Debug 模式下，下一步，进入当前方法体内，如果方法体还有方法，则会进入该内嵌的方法中，依此循环进入
Alt + Shift + F9	弹出 Debug 的可选择菜单
Alt + Shift + F10	弹出 Run 的可选择菜单
Alt + Shift + 左键双击	选择被双击的单词 / 中文句，按住不放，可以同时选择其他单词 / 中文句 (必备)
Alt + Shift + 前方向键	移动光标所在行向上移动 (必备)
Alt + Shift + 后方向键	移动光标所在行向下移动 (必备)

Ctrl + Shift + Alt

快捷键	介绍
Ctrl + Shift + Alt + V	无格式黏贴 (必备)
Ctrl + Shift + Alt + N	前往指定的变量 / 方法
Ctrl + Shift + Alt + S	打开当前项目设置 (必备)
Ctrl + Shift + Alt + C	复制参考信息

其他

快捷键	介绍
F2	跳转到下一个高亮错误或警告位置 (必备)
F3	在查找模式下，定位到下一个匹配处
F4	编辑源 (必备)
F7	在 Debug 模式下，进入下一步，如果当前行断点是一个方法，则进入当前方法体内，如果该方法体还有方法，则不会进入该内嵌的方法中
F8	在 Debug 模式下，进入下一步，如果当前行断点是一个方法，则不进入当前方法体内
F9	在 Debug 模式下，恢复程序运行，但是如果该断点下面代码还有断点则停在下一个断点上
F11	添加书签 (必备)
F12	回到前一个工具窗口 (必备)
Tab	缩进 (必备)
ESC	从工具窗口进入代码文件窗口 (必备)
连按两次 Shift	弹出 Search Everywhere 弹出层

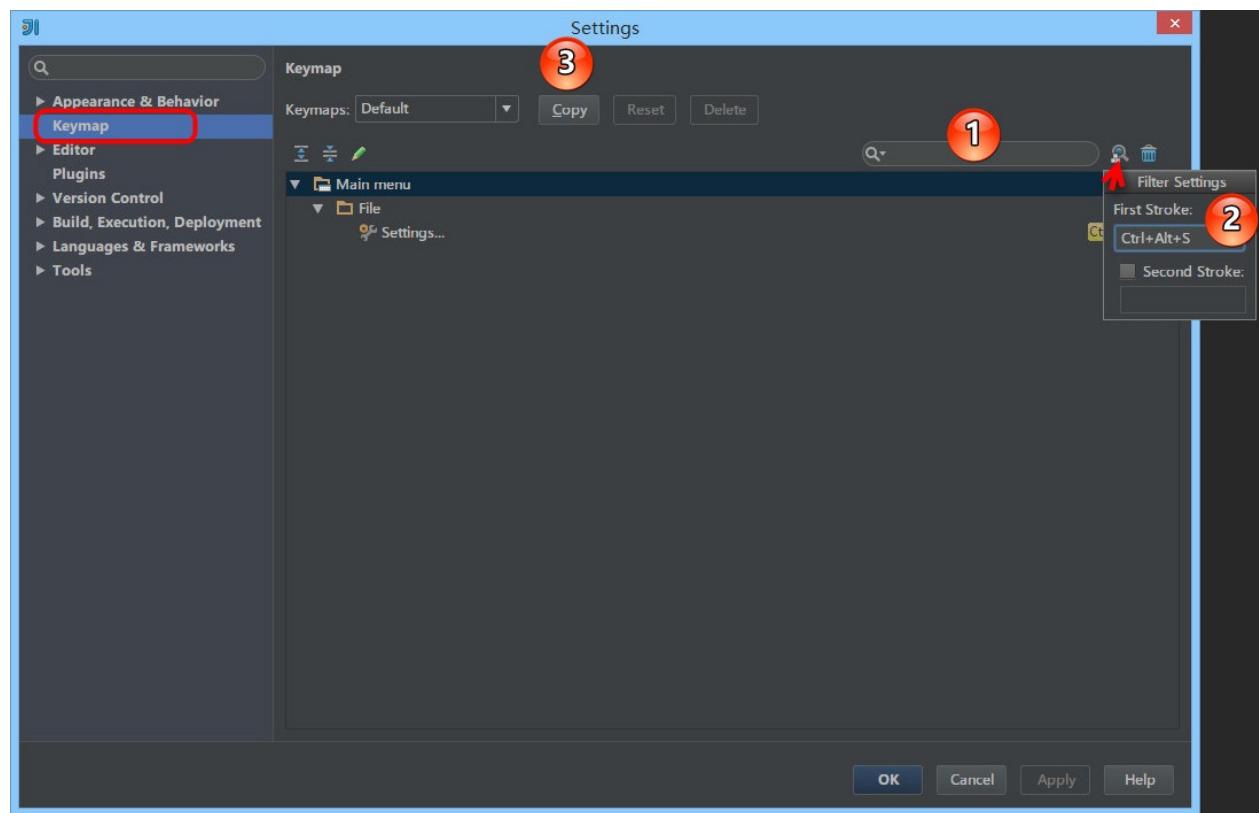
官网快捷键资料

- Windows /
Linux : https://www.jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard.pdf
- Mac OS
X : https://www.jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard_Mac.pdf

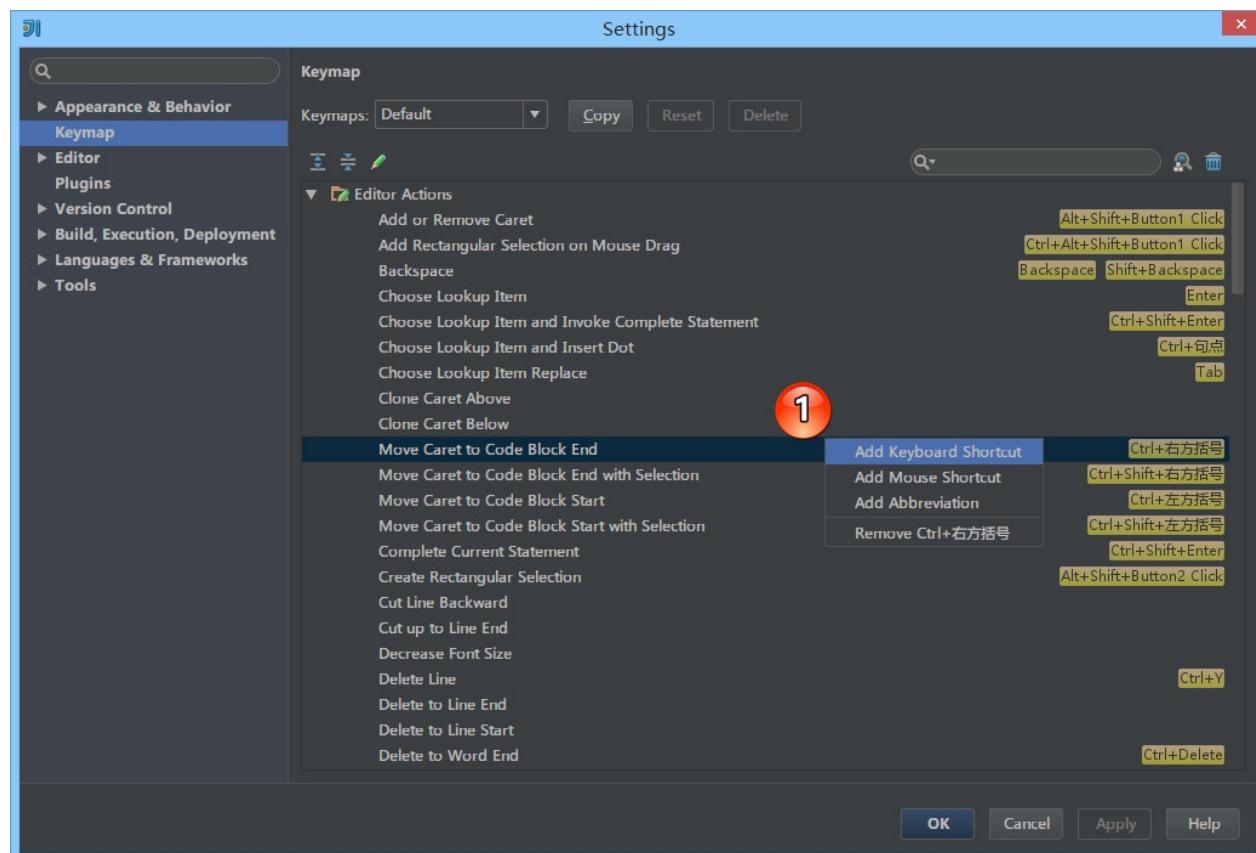
第三方快捷键资料

- 来自 eta02913 : <http://xinyuwu.iteye.com/blog/1005454>

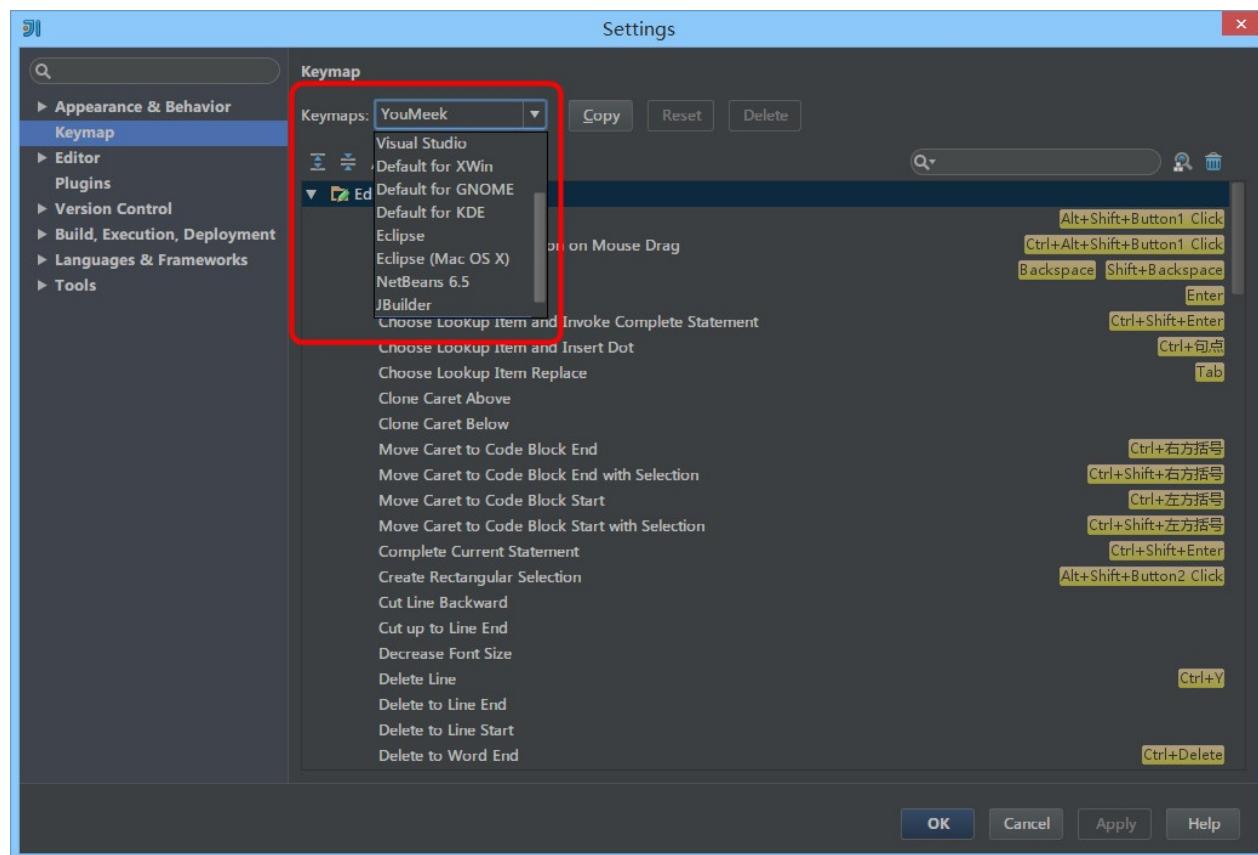
快捷键修改



- 按 `Ctrl + Alt + S` 弹出 IDE 设置，如上图选择左侧的 `Keymap`。
- IntelliJ IDEA 支持两种方式来筛选我们要找的快捷键，一种是上图标注 1 所示的，通过输入快捷键描述内容；一种是上图标注 2 所示的，通过按指定快捷键快捷键，这里需要再次强调的是，这个输入框是自动监听你当前按下的按键，而不是用来输入的。
- 上图标注 3 所示，初安装的 IntelliJ IDEA 使用的是 `Default` 的快捷键模板，IntelliJ IDEA 默认的快捷键模板都是不可修改的。如果你直接修改，当前这个位置 IntelliJ IDEA 会自动变成 `Default Copy`，建议你养成习惯，修改之前先点击 `Copy`，拷贝一套快捷键模板，然后输入自己的命名。

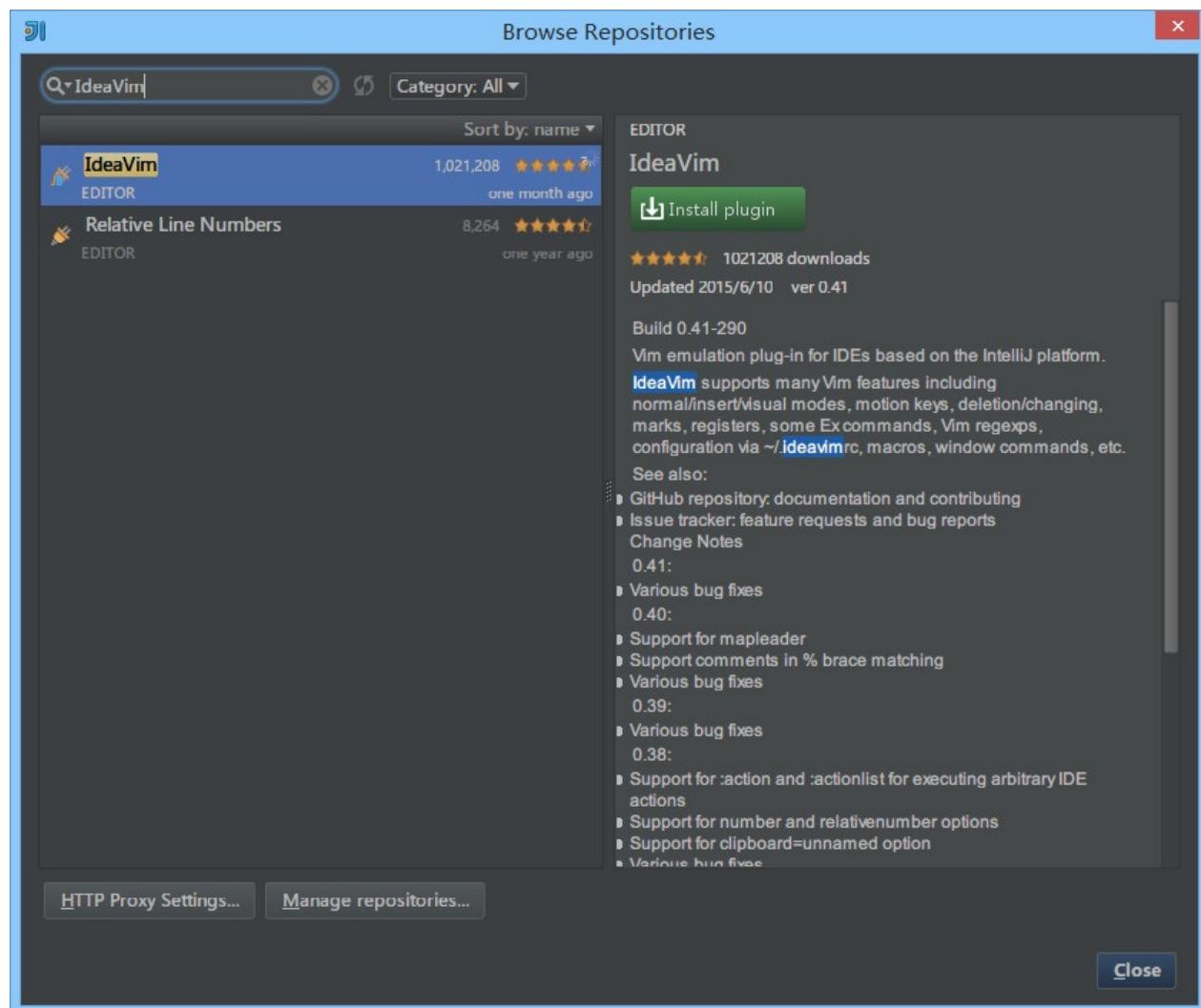


- IntelliJ IDEA 是支持一个操作命令同时设置多个快捷键组合，就如上图的 `Backspace`，同时支持 `Backspace` 和 `Shift + Backspace` 两组快捷键。
- 要修改某个快捷键，选中快捷键介绍内容，右键，就会弹出如上图标注 1 所示操作选择。
- 命令 `Add Keyboard Shortcut` 用来添加新纯键盘快捷键组合。
- 命令 `Add Mouse Shortcut` 用来添加新 键盘 + 鼠标 快捷键组合，比如设置 `Ctrl + 左键单击` 这类快捷组合。其中在弹出的添加面板中 `Click Pad` 是用来监听当前鼠标是左键单击还是右键单击。
- 命令 `Add Abbreviation` 根据 IntelliJ IDEA 的版本文档解释，添加简称主要是为了方便 `Search Everywhere` 中使用，但是我尝试之后发现没办法根据我设置的简称搜索，暂时无法了解其作用。
- 命令 `Remove` 快捷键 移出当前操作命令已设置的快捷键组合，由于 IntelliJ IDEA 默认就占用了很多快捷键组合，所以如果你要修改某个快捷键，建议还是删除掉旧的。



- IntelliJ IDEA 对其他 IDE 用户很友好，如上图对于其他主流的 IDE，快捷键上已经默认了有其过度快捷键模板了，但是我还是建议你专心使用 IntelliJ IDEA 的默认。

其他



- 如果你是一个 Vim 粉，IntelliJ IDEA 也为你准备了一个方案：如上图安装 IdeaVim 插件即可。

IntelliJ IDEA For Mac 快捷键

- 根据官方pdf翻
译：https://www.jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard_Mac.pdf
- 在 IntelliJ IDEA 中有两个 Mac 版本的快捷键，一个叫做：Mac OS X，一个叫做：Mac OS X 10.5+
- 目前都是用：Mac OS X 10.5+
- 有两套的原因：<https://intellij-support.jetbrains.com/hc/en-us/community/posts/206159109-Updated-Mac-OS-X-keymap-Feedback-needed>

建议将 Mac 系统中与 IntelliJ IDEA 冲突的快捷键取消或更改，不建议改 IntelliJ IDEA 的默认快捷键。

Mac 键盘符号和修饰键说明

- ⌘ Command
- ⇧ Shift
- ⌥ Option
- ⌃ Control
- ⤵ Return/Enter
- ⌫ Delete
- ⌦ 向前删除键 (Fn+Delete)
- ↑ 上箭头
- ↓ 下箭头
- ← 左箭头
- 右箭头
- ⇞ Page Up (Fn+↑)
- ⇟ Page Down (Fn+↓)
- Home Fn + ←
- End Fn + →
- ⇥ 右制表符 (Tab键)
- ⇤ 左制表符 (Shift+Tab)
- ⌫ Escape (Esc)

一、Editing (编辑)

- Control + Space 基本的代码补全（补全任何类、方法、变量）
- Control + Shift + Space 智能代码补全（过滤器方法列表和变量的预期类型）
- Command + Shift + Enter 自动结束代码，行末自动添加分号

- Command + P 显示方法的参数信息
- Control + J 快速查看文档
- Shift + F1 查看外部文档（在某些代码上会触发打开浏览器显示相关文档）
- Command + 鼠标放在代码上 显示代码简要信息
- Command + F1 在错误或警告处显示具体描述信息
- Command + N, Control + Enter, Control + N 生成代码（getter、setter、构造函数、hashCode>equals,toString）
- Control + O 覆盖方法（重写父类方法）
- Control + I 实现方法（实现接口中的方法）
- Command + Option + T 包围代码（使用if..else, try..catch, for, synchronized等包围选中的代码）
- Command + / 注释/取消注释与行注释
- Command + Option + / 注释/取消注释与块注释
- Option + 方向键上 连续选中代码块
- Option + 方向键下 减少当前选中的代码块
- Control + Shift + Q 显示上下文信息
- Option + Enter 显示意向动作和快速修复代码
- Command + Option + L 格式化代码
- Control + Option + O 优化import
- Control + Option + I 自动缩进线
- Tab / Shift + Tab 缩进代码 / 反缩进代码
- Command + X 剪切当前行或选定的块到剪贴板
- Command + C 复制当前行或选定的块到剪贴板
- Command + V 从剪贴板粘贴
- Command + Shift + V 从最近的缓冲区粘贴
- Command + D 复制当前行或选定的块
- Command + Delete 删除当前行或选定的块的行
- Control + Shift + J 智能的将代码拼接成一行
- Command + Enter 智能的拆分拼接的行
- Shift + Enter 开始新的一行
- Command + Shift + U 大小写切换
- Command + Shift +] / Command + Shift + [选择直到代码块结束/开始
- Option + Fn + Delete 删除到单词的末尾
- Option + Delete 删除到单词的开头
- Command + 加号 / Command + 减号 展开 / 折叠代码块
- Command + Shift + 加号 展开所有代码块
- Command + Shift + 减号 折叠所有代码块
- Command + W 关闭活动的编辑器选项卡

二、Search/Replace（查询/替换）

- Double Shift 查询任何东西
- Command + F 文件内查找
- Command + G 查找模式下，向下查找
- Command + Shift + G 查找模式下，向上查找
- Command + R 文件内替换
- Command + Shift + F 全局查找（根据路径）
- Command + Shift + R 全局替换（根据路径）
- Command + Shift + S 查询结构（Ultimate Edition 版专用，需要在Keymap中设置）
- Command + Shift + M 替换结构（Ultimate Edition 版专用，需要在Keymap中设置）

三、Usage Search（使用查询）

- Option + F7 / Command + F7 在文件中查找用法 / 在类中查找用法
- Command + Shift + F7 在文件中突出显示的用法
- Command + Option + F7 显示用法

四、Compile and Run（编译和运行）

- Command + F9 编译Project
- Command + Shift + F9 编译选择的文件、包或模块
- Control + Option + R 弹出 Run 的可选择菜单
- Control + Option + D 弹出 Debug 的可选择菜单
- Control + R 运行
- Control + D 调试
- Control + Shift + R, Control + Shift + D 从编辑器运行上下文环境配置

五、Debugging（调试）

- F8 进入下一步，如果当前行断点是一个方法，则不进入当前方法体内
- F7 进入下一步，如果当前行断点是一个方法，则进入当前方法体内，如果该方法体还有方法，则不会进入该内嵌的方法中
- Shift + F7 智能步入，断点所在行上有多个方法调用，会弹出进入哪个方法
- Shift + F8 跳出
- Option + F9 运行到光标处，如果光标前有其他断点会进入到该断点
- Option + F8 计算表达式（可以更改变量值使其生效）
- Command + Option + R 恢复程序运行，如果该断点下面代码还有断点则停在下一个断点上
- Command + F8 切换断点（若光标当前行有断点则取消断点，没有则加上断点）
- Command + Shift + F8 查看断点信息

六、Navigation（导航）

- Command + O 查找类文件
- Command + Shift + O 查找所有类型文件、打开文件、打开目录，打开目录需要在输入的内容前面或后面加一个反斜杠 /
- Command + Option + O 前往指定的变量 / 方法
- Control + 方向键左 / Control + 方向键右 左右切换打开的编辑tab页
- F12 返回到前一个工具窗口
- Esc 从工具窗口进入代码文件窗口
- Shift + Esc 隐藏当前或最后一个活动的窗口，且光标进入代码文件窗口
- Command + Shift + F4 关闭活动run/messages/find/... tab
- Command + L 在当前文件跳转到某一行的指定处
- Command + E 显示最近打开的文件记录列表
- Option + 方向键左 / Option + 方向键右 光标跳转到当前单词 / 中文句的左 / 右侧开头位置
- Command + Option + 方向键左 / Command + Option + 方向键右 退回 / 前进到上一个操作的地方
- Command + Shift + Delete 跳转到最后一个编辑的地方
- Option + F1 显示当前文件选择目标弹出层，弹出层中有很多目标可以进行选择(如在代码编辑窗口可以选择显示该文件的Finder)
- Command + B / Command + 鼠标点击 进入光标所在的方法/变量的接口或是定义处
- Command + Option + B 跳转到实现处，在某个调用的方法名上使用会跳到具体的实现处，可以跳过接口
- Option + Space, Command + Y 快速打开光标所在方法、类的定义
- Control + Shift + B 跳转到类型声明处
- Command + U 前往当前光标所在方法的父类的方法 / 接口定义
- Control + 方向键下 / Control + 方向键上 当前光标跳转到当前文件的前一个/后一个方法名位置
- Command +] / Command + [移动光标到当前所在代码的花括号开始/结束位置
- Command + F12 弹出当前文件结构层，可以在弹出的层上直接输入进行筛选（可用于搜索类中的方法）
- Control + H 显示当前类的层次结构
- Command + Shift + H 显示方法层次结构
- Control + Option + H 显示调用层次结构
- F2 / Shift + F2 跳转到下一个/上一个突出错误或警告的位置
- F4 / Command + 方向键下 编辑/查看代码源
- Option + Home 显示到当前文件的导航条
- F3 选中文件/文件夹/代码行，添加/取消书签
- Option + F3 选中文件/文件夹/代码行，使用助记符添加/取消书签
- Control + 0...Control + 9 定位到对应数值的书签位置
- Command + F3 显示所有书签

七、Refactoring（重构）

- F5 复制文件到指定目录
- F6 移动文件到指定目录
- Command + Delete 在文件上为安全删除文件，弹出确认框
- Shift + F6 重命名文件
- Command + F6 更改签名
- Command + Option + N 一致性
- Command + Option + M 将选中的代码提取为方法
- Command + Option + V 提取变量
- Command + Option + F 提取字段
- Command + Option + C 提取常量
- Command + Option + P 提取参数

八、VCS/Local History（版本控制/本地历史记录）

- Command + K 提交代码到版本控制器
- Command + T 从版本控制器更新代码
- Option + Shift + C 查看最近的变更记录
- Control + C 快速弹出版本控制器操作面板

九、Live Templates（动态代码模板）

- Command + Option + J 弹出模板选择窗口，将选定的代码使用动态模板包围
- Command + J 插入自定义动态代码模板

十、General（通用）

- Command + 1...Command + 9 打开相应编号的工具窗口
- Command + S 保存所有
- Command + Option + Y 同步、刷新
- Control + Command + F 切换全屏模式
- Command + Shift + F12 切换最大化编辑器
- Option + Shift + F 添加到收藏夹
- Option + Shift + I 检查当前文件与当前的配置文件
- Control + 快速切换当前的scheme（切换主题、代码样式等）
- Command + , 打开IDEA系统设置
- Command + ; 打开项目结构对话框
- Shift + Command + A 查找动作（可设置相关选项）
- Control + Shift + Tab 编辑窗口标签和工具窗口之间切换（如果在切换的过程加按上 delete，则是关闭对应选中的窗口）

十一、**Other**（一些官方文档上没有体现的快捷键）

- Command + Shift +8 竖编辑模式

从 Windows 过度到 Mac 必备快捷键对照表

Mac 键盘符号说明

- ⌘ == Command
- ⌥ == Shift
- ⇤ == Caps Lock
- ⇣ == Option
- ⌂ == Control
- ↵ == Return/Enter
- ⌫ == Delete
- ⌋ == 向前删除键 (Fn+Delete)
- ↑ == 上箭头
- ↓ == 下箭头
- ← == 左箭头
- → == 右箭头
- ⌊ == Page Up (Fn+↑)
- ⌋ == Page Down (Fn+↓)
- Home == Fn + ←
- End == Fn + →
- → == 右制表符 (Tab键)
- ← == 左制表符 (Shift+Tab)
- ⌄ == Escape (Esc)
- ▲ == 电源开关键

Ctrl

Win 快捷键	Mac 快捷键	介绍
Ctrl + F	Command + F	在当前文件进行文本查找
Ctrl + R	Command + R	在当前文件进行文本替换
Ctrl + Z	Command + Z	撤销
Ctrl + Y	Command + Delete	删除光标所在行 或 删除选中的行

Ctrl + D	Command + D	复制光标所在行 或 复制选择内容，并把复制内容插入光标位置下面
Ctrl + W	Option + 方向键上	递进式选择代码块。可选中光标所在的单词或段落，连续按会在原有选中的基础上再扩展选中范围
Ctrl + E	Command + E	显示最近打开的文件记录列表
Ctrl + N	Command + O	根据输入的 类名 查找类文件
Ctrl + J	Command + J	插入自定义动态代码模板
Ctrl + P	Command + P	方法参数提示显示
Ctrl + U	Command + U	前往当前光标所在的方法的父类的方法 / 接口定义
Ctrl + B	Command + B	进入光标所在的方法/变量的接口或是定义处，等效于 Ctrl + 左键单击
Ctrl + /	Command + /	注释光标所在行代码，会根据当前不同文件类型使用不同的注释符号
Ctrl + F1	Command + F1	在光标所在的错误代码处显示错误信息
Ctrl + F11	Option + F3	选中文件 / 文件夹，使用助记符设定 / 取消书签
Ctrl + Space	Control + Space	基础代码补全，默认在 Windows 系统上被输入法占用，需要进行修改，建议修改为 Ctrl + 逗号
Ctrl + Delete	Option + Fn + Delete	删除光标后面的单词或是中文句
Ctrl + BackSpace	Option + Delete	删除光标前面的单词或是中文句
Ctrl + 1,2,3...9	Control + 1,2,3...9	定位到对应数值的书签位置
Ctrl + 加号	Command + 加号	展开代码
Ctrl + 减号	Command + 减号	折叠代码
Ctrl + 左键单击	Control + 左键单击	在打开的文件标题上，弹出该文件路径
Ctrl + 左方向键	Option + 左方向键	光标跳转到当前单词 / 中文句的左侧开头位置

Ctrl + 右方向键	Option + 右方向键	光标跳转到当前单词 / 中文句的右侧开头位置
Ctrl + 前方向键	预设中没有该快捷键	等效于鼠标滚轮向前效果
Ctrl + 后方向键	预设中没有该快捷键	等效于鼠标滚轮向后效果

Alt

Win 快捷键	Mac 快捷键	介绍
Alt + `	Control + V	显示版本控制常用操作菜单弹出层
Alt + F1	Option + F1	显示当前文件选择目标弹出层，弹出层中有很多目标可以选择
Alt + F7	Option + F7	查询所选对象/变量被引用
Alt + Enter	Option + Enter	IntelliJ IDEA 根据光标所在问题，提供快速修复选择，光标放在的位置不同提示的结果也不同
Alt + Insert	Command + N	代码自动生成，如生成对象的 set / get 方法，构造函数， <code>toString()</code> 等
Alt + 左方向键	Control + 左方向键	切换当前已打开的窗口中的子视图，比如 Debug 窗口中有 Output、Debugger 等子视图，用此快捷键就可以在子视图中切换
Alt + 右方向键	Control + 右方向键	切换当前已打开的窗口中的子视图，比如 Debug 窗口中有 Output、Debugger 等子视图，用此快捷键就可以在子视图中切换
Alt + 前方向键	Control + 前方向键	当前光标跳转到当前文件的前一个方法名位置
Alt + 后方向键	Control + 后方向键	当前光标跳转到当前文件的后一个方法名位置
Alt + 1,2,3...9	Command + 1,2,3...9	显示对应数值的选项卡，其中 1 是 Project 用得最多

Shift

Win 快捷键	Mac 快捷键	介绍
Shift + F11	Command + F3	弹出书签显示层
Shift + Tab	Shift + Tab	取消缩进
Shift + Enter	Shift + Enter	开始新一行。光标所在行下空出一行，光标定位到新行位置
Shift + 左键单击	Shift + 左键单击	在打开的文件名上按此快捷键，可以关闭当前打开文件

Ctrl + Alt

Win 快捷键	Mac 快捷键	介绍
Ctrl + Alt + L	Command + Option + L	格式化代码，可以对当前文件和整个包目录使用
Ctrl + Alt + O	Control + Option + O	优化导入的类，可以对当前文件和整个包目录使用
Ctrl + Alt + T	Command + Option + T	对选中的代码弹出环绕选项弹出层
Ctrl + Alt + S	Command + 逗号	打开 IntelliJ IDEA 系统设置
Ctrl + Alt + Enter	Command + Option + Enter	光标所在行上空出一行，光标定位到新行
Ctrl + Alt + 左方向键	Command + Option + 左方向键	退回到上一个操作的地方
Ctrl + Alt + 右方向键	Command + Option + 右方向键	前进到上一个操作的地方

Ctrl + Shift

Win 快捷键	Mac 快捷键	介绍
Ctrl + Shift + F	Command + Shift + F	根据输入内容查找整个项目 或 指定目录内文件
Ctrl + Shift + R	Command + Shift + F	根据输入内容替换对应内容，范围为整个项目 或 指定目录内文件
Ctrl + Shift + J	Control + Shift + J	自动将下一行合并到当前行末尾
Ctrl + Shift + Z	Command + Shift + Z	取消撤销

Ctrl + Shift + W	Option + 方向键下	递进式取消选择代码块。可选中光标所在的单词或段落，连续按会在原有选中的基础上再扩展取消选中范围
Ctrl + Shift + N	Command + Shift + O	通过文件名定位 / 打开文件 / 目录，打开目录需要在输入的内容后面多加一个正斜杠
Ctrl + Shift + U	Command + Shift + U	对选中的代码进行大 / 小写轮流转换
Ctrl + Shift + T	Command + Shift + T	对当前类生成单元测试类，如果已经存在的单元测试类则可以进行选择
Ctrl + Shift + C	Command + Shift + C	复制当前文件磁盘路径到剪贴板
Ctrl + Shift + B	Control + Shift + B	跳转到类型声明处
Ctrl + Shift + /	Command + Option + /	代码块注释
Ctrl + Shift + [Command + Shift + [选中从光标所在位置到它的顶部中括号位置
Ctrl + Shift +]	Command + Shift +]	选中从光标所在位置到它的底部中括号位置
Ctrl + Shift + 加号	Command + Shift + 加号	展开所有代码
Ctrl + Shift + 减号	Command + Shift + 减号	折叠所有代码
Ctrl + Shift + F7	Command + Shift + F7	高亮显示所有该选中文本，按Esc高亮消失
Ctrl + Shift + F12	Command + Shift + F12	编辑器最大化
Ctrl + Shift + Enter	Command + Shift + Enter	自动结束代码，行末自动添加分号
Ctrl + Shift + Backspace	Ctrl + Shift + Backspace	退回到上次修改的地方
Ctrl + Shift + 1,2,3...9	Control + Shift + 1,2,3...9	快速添加指定数值的书签
Ctrl + Shift + 左键单击	Command + Shift + 左键单击	把光标放在某个类变量上，按此快捷键可以直接定位到该类中
Ctrl + Shift + 左方向键	Option + Shift + 左方向键	在代码文件上，光标跳转到当前单词 / 中文句的左侧开头位置，同时选中该单词 / 中文句
Ctrl + Shift	Option + Shift	在代码文件上，光标跳转到当前单词 / 中文句的右侧开

+ 右方向键	+ 右方向键	头位置，同时选中该单词 / 中文句
Ctrl + Shift + 前方向键	Command + Shift + 前方向键	光标放在方法名上，将方法移动到上一个方法前面，调整方法排序
Ctrl + Shift + 后方向键	Command + Shift + 后方向键	光标放在方法名上，将方法移动到下一个方法前面，调整方法排序

Alt + Shift

Win 快捷键	Mac 快捷键	介绍
Alt + Shift + N	Option + Shift + B	选择 / 添加 task
Alt + Shift + 左键双击	Option + Shift + 左键双击	选择被双击的单词 / 中文句，按住不放，可以同时选择其他单词 / 中文句
Alt + Shift + 前方向键	Option + Shift + 前方向键	移动光标所在行向上移动
Alt + Shift + 后方向键	Option + Shift + 后方向键	移动光标所在行向下移动

Ctrl + Shift + Alt

Win 快捷键	Mac 快捷键	介绍
Ctrl + Shift + Alt + V	Command + Shift + Option + V	无格式黏贴
Ctrl + Shift + Alt + S	Command + ;	打开当前项目设置

其他

Win 快捷键	Mac 快捷键	介绍
F2	F2	跳转到下一个高亮错误 或 警告位置
F4	F4	编辑源
F11	F3	添加书签
F12	F12	回到前一个工具窗口
Tab	Tab	缩进
ESC	ESC	从工具窗口进入代码文件窗口

IntelliJ IDEA 的 Java 热部署插件 JRebel 安装及使用

JRebel 介绍

在 Java Web 开发中，一般更新了 Java 文件后要手动重启 Tomcat 服务器，才能生效，浪费不少生命啊，自从有了 JRebel 这神器的出现，不论是更新 class 类还是更新 Spring 配置文件都能做到立马生效，大大提高开发效率。

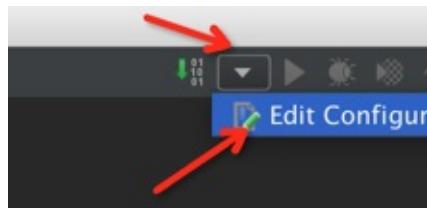
JRebel 安装

JRebel 的安装方法有两种，一种是直接在 Tomcat 启动参数上面加上 JRebel 的参数，另外一种是以插件的形式装到 IntelliJ IDEA 上，比较推荐后者。

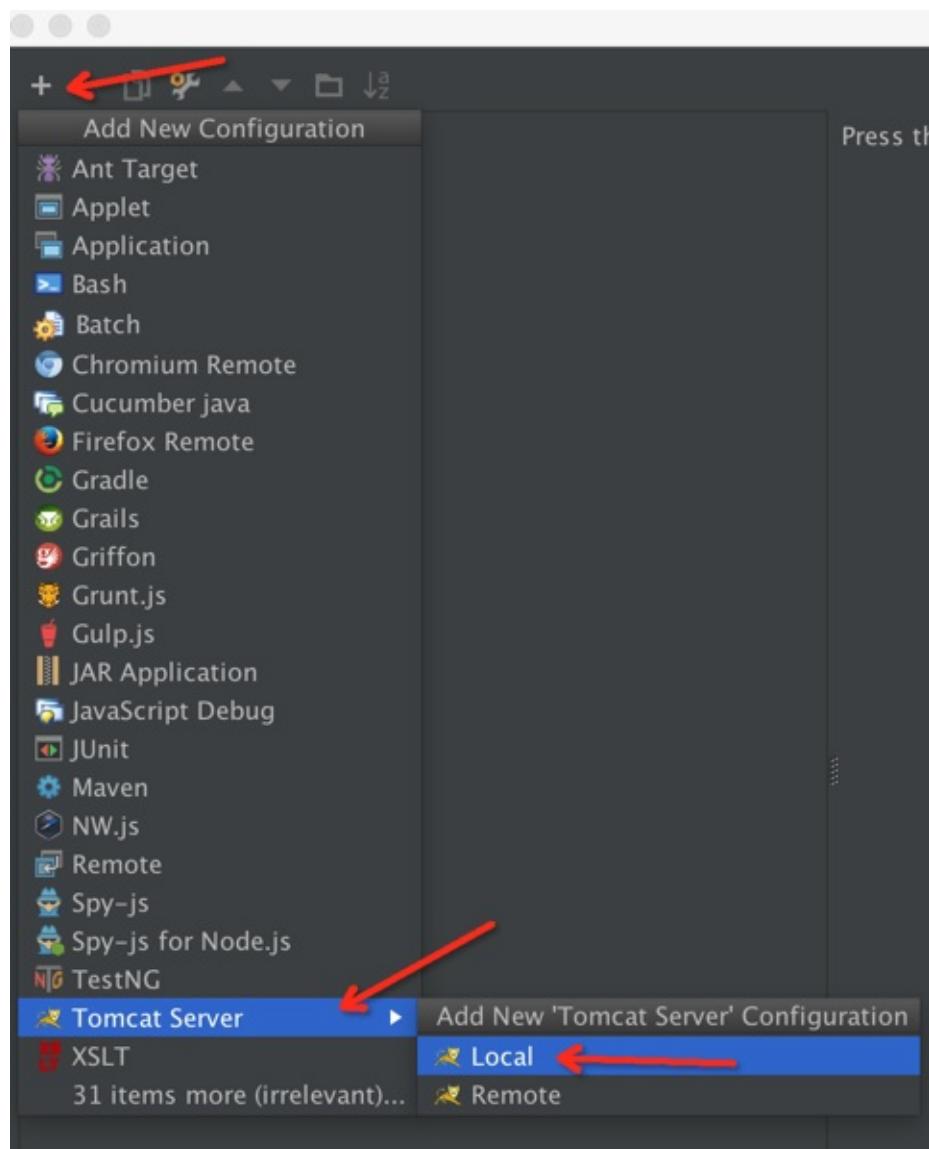
第一种安装方法：在 Tomcat 启动参数上加参数

首先先介绍第一种安装方法，先在硬盘某个位置把 JRebel 解压出来

然后配置 IntelliJ IDEA 的 Tomcat

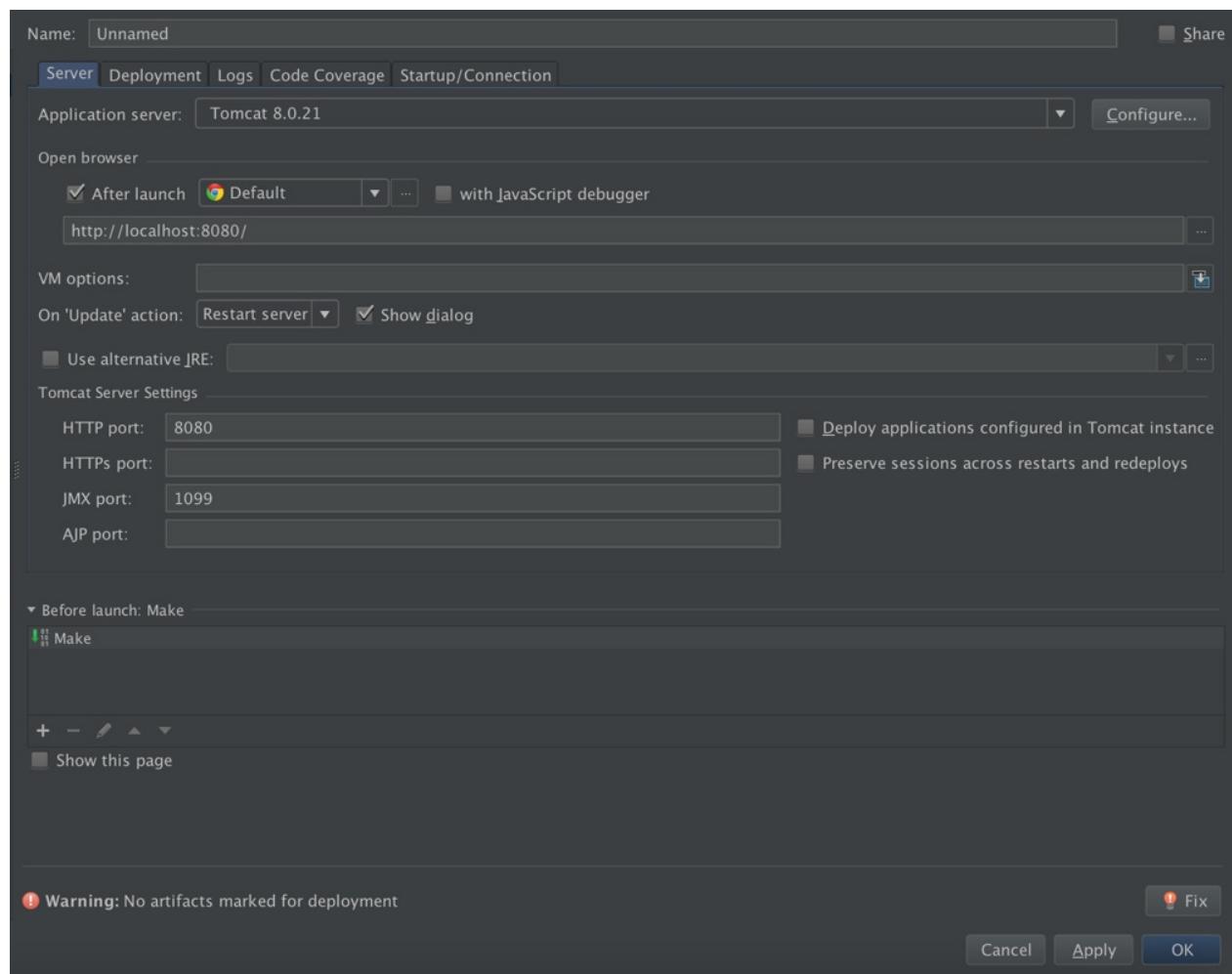


点 + 号选择 Tomcat Server -> Local

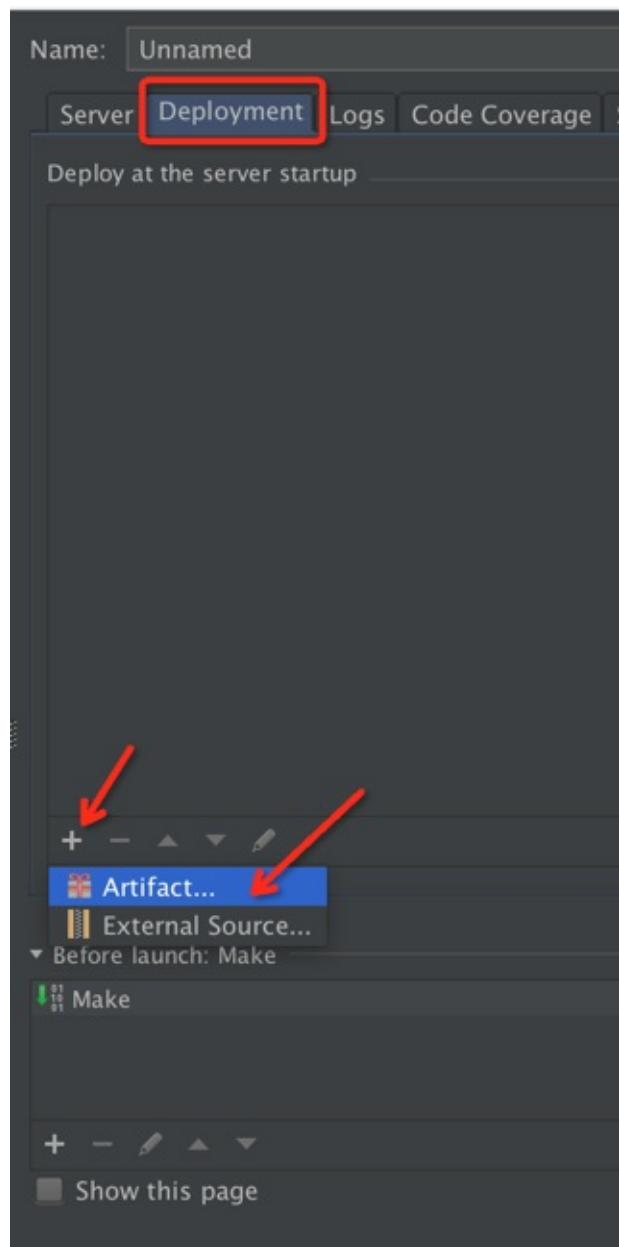


默认显示如图

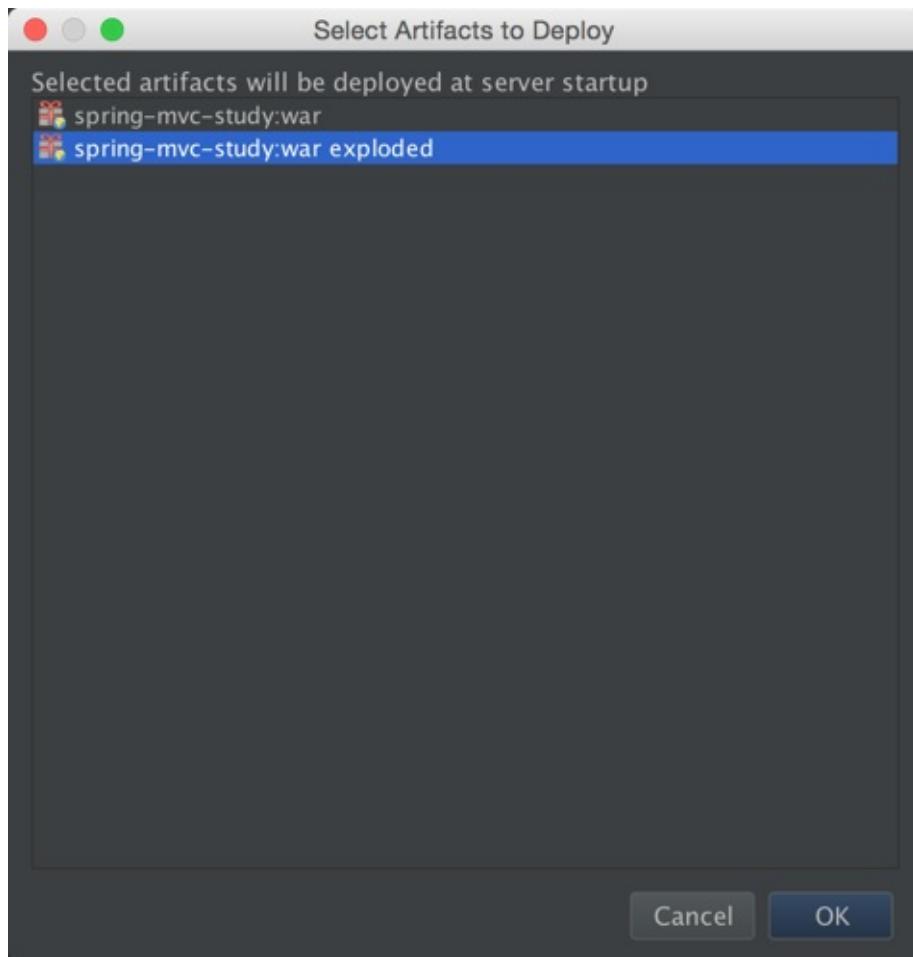
36. IntelliJ IDEA 的 Java 热部署插件 JRebel 安装及使用



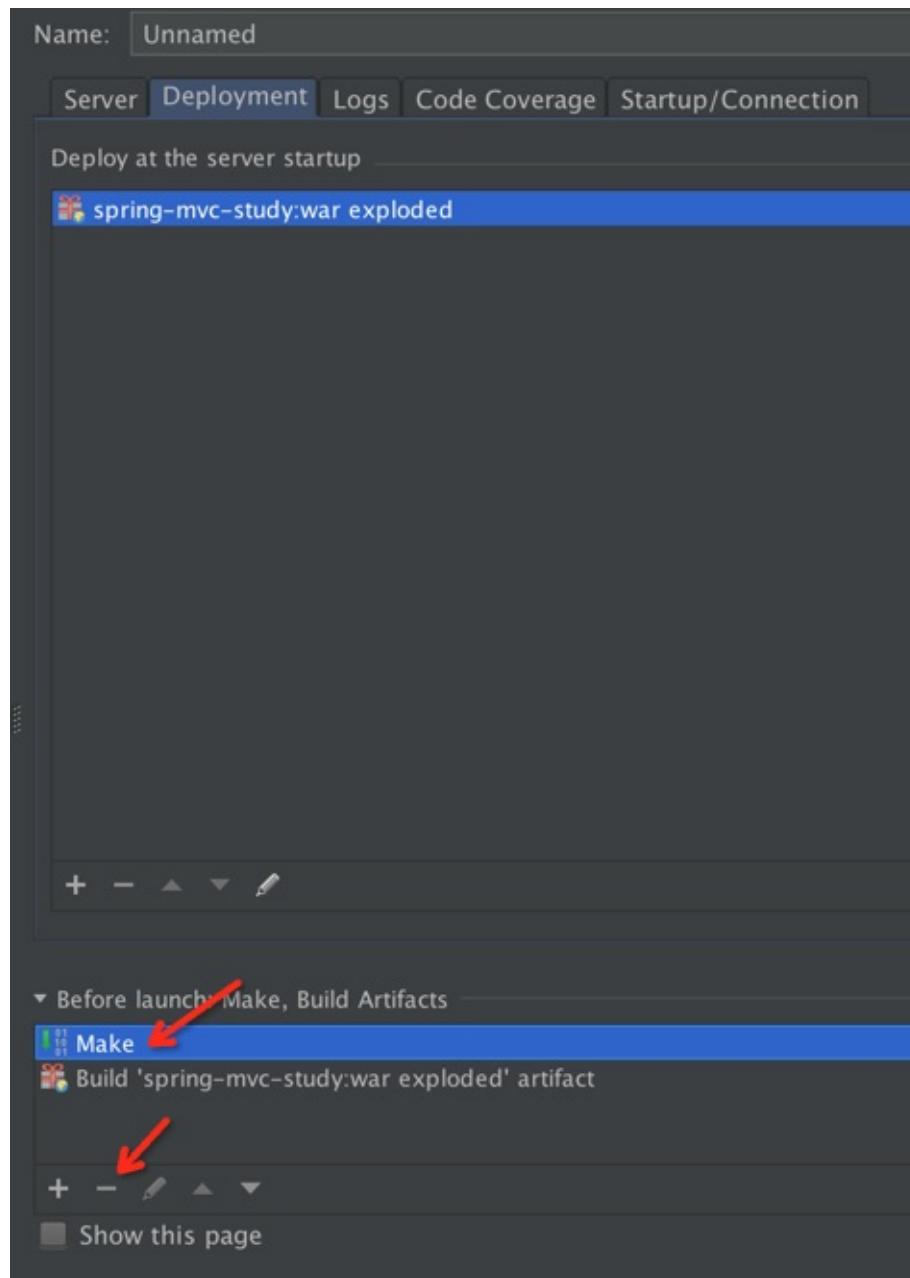
几个关键的地方需要注意的，就是首先要选择 Deployment 这个选项卡



选择自己的项目，建议选择带 exploded ，这个相当于改 Tomcat 的 CATALINA_HOME，效率比较高

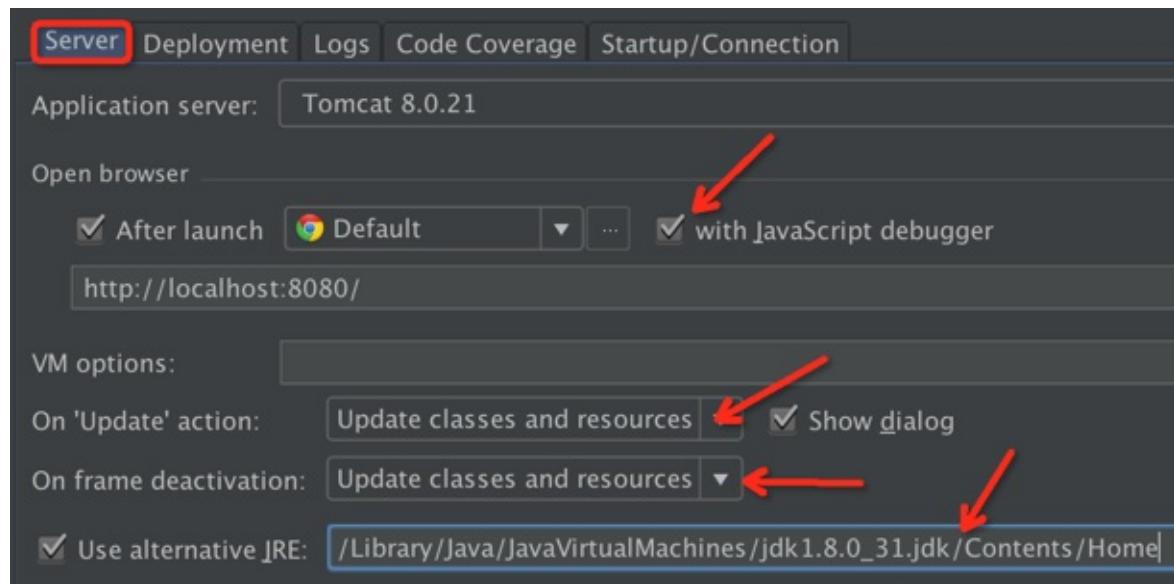


选择好后，删掉默认的Make，提高效率



接下来返回 Server 选项卡，会发现多了一项 On frame deactivation，如果你刚才没有配置 Deployment 选项卡的话的这项是不会出现的

按如图所示的来配置，特别需要注意的是 On 'Update' action 和 On frame deactivation 这两项项目一定要选择 update classes and resources，否则类修改热部署不生效，或者第三方模版框架例如 Freemarker 热部署不生效

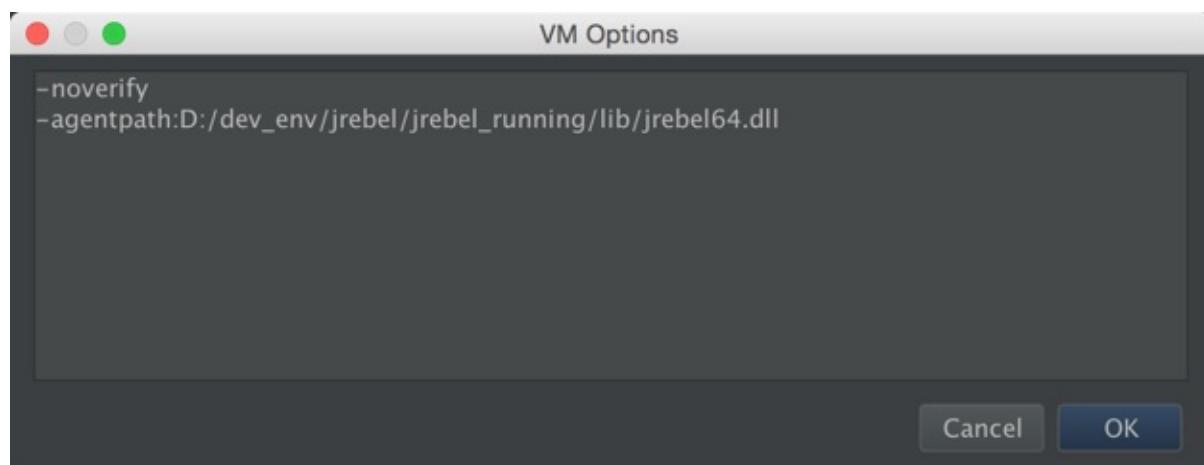


接下来就是很关键的需要引入 JRebel 的地方了，在 VM options 的最右边有个箭头，点进去



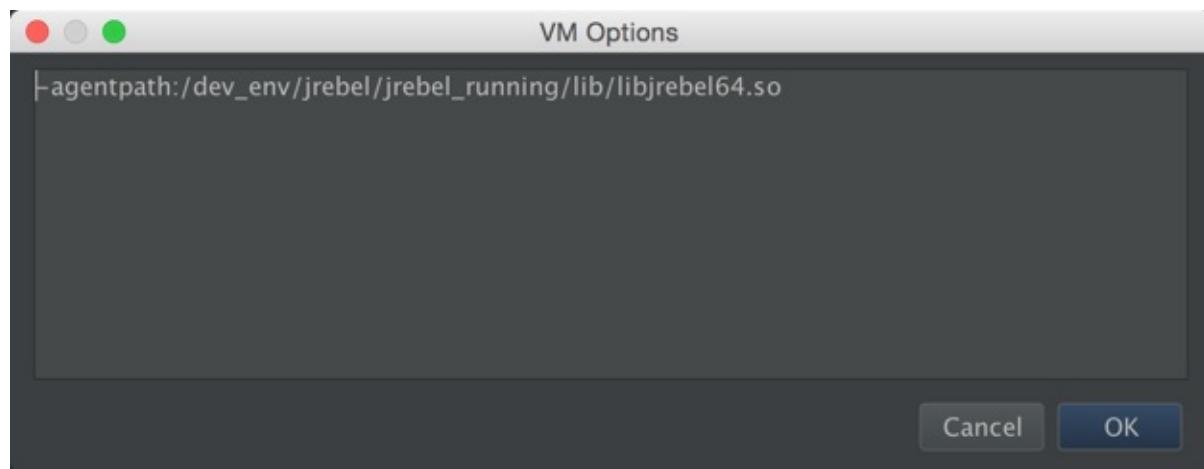
Windows 输入：

```
-noverify
-agentpath:D:/dev_env/jrebel/jrebel_running/lib/jrebel64.dll
```



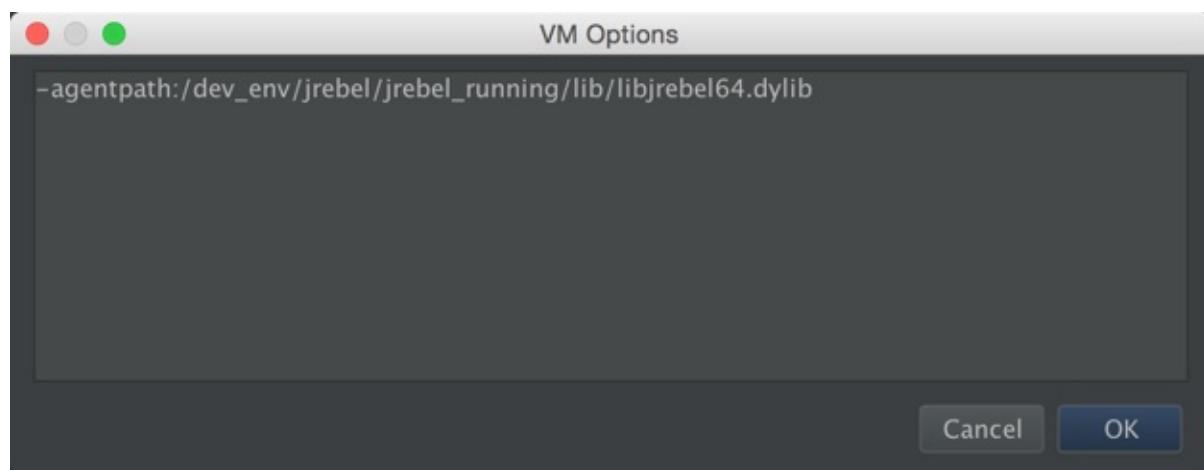
Linux 用这个：

```
-agentpath:/dev_env/jrebel/jrebel_running/lib/libjrebel64.so
```



Mac OS 用这个：

```
-agentpath:/dev_env/jrebel/jrebel_running/lib/libjrebel64.dylib
```

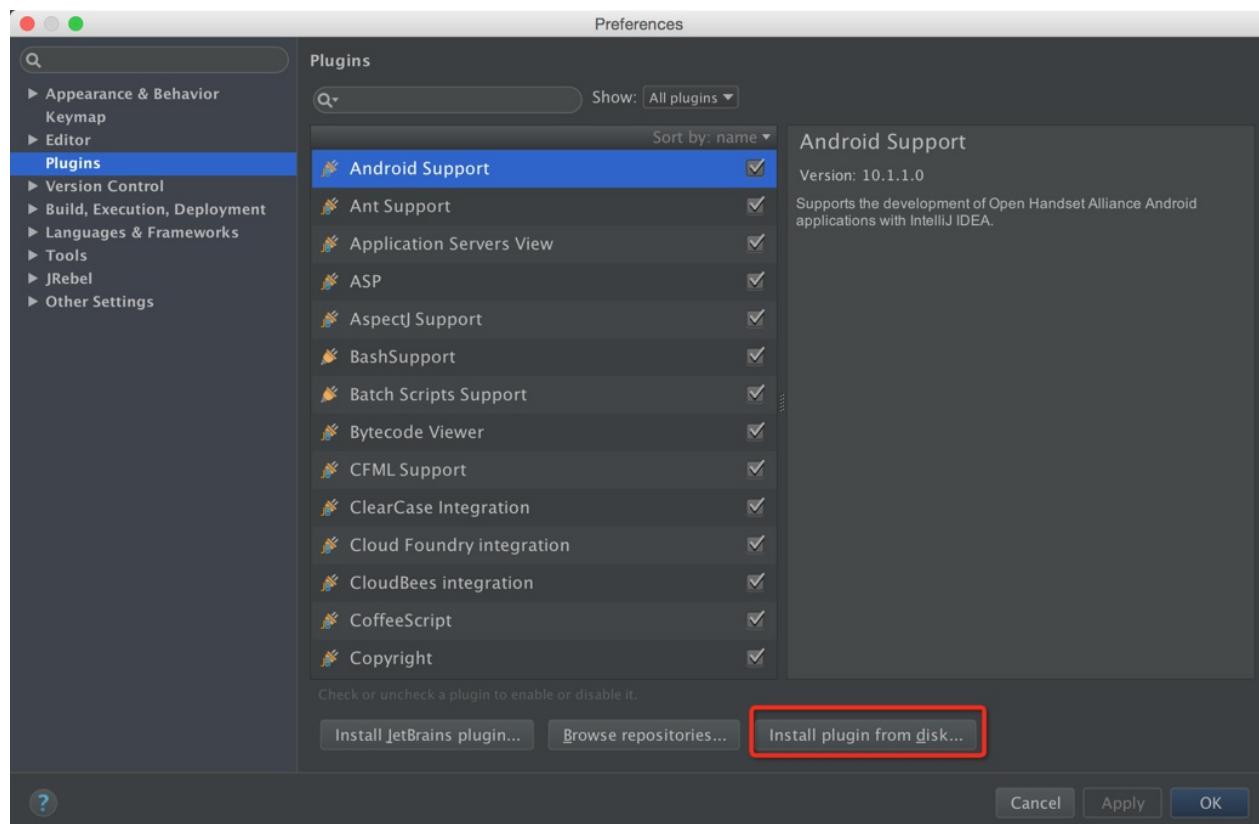


配置完成，直接启动 Tomcat 即可，不过此方法麻烦，每次新建项目都要从新配置

第二种安装方法：使用 **IntelliJ IDEA** 插件

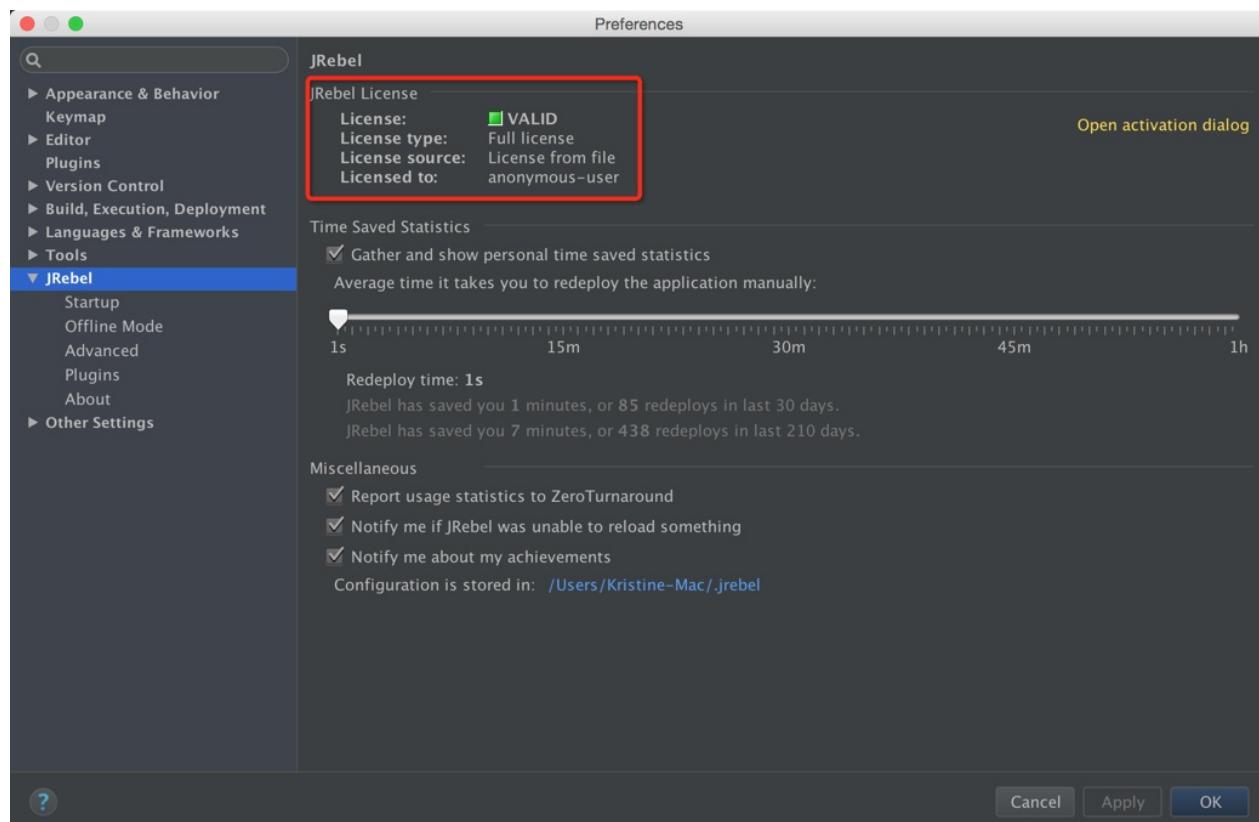
接下来介绍使用 IntelliJ IDEA 插件的方式启动 JRebel

首先是安装 JRebel 的插件，安装方法和其他插件安装方法一样，不过这里不采用在线安装，直接选择本地安装，直接选择插件安装即可



安装好后在设置里面会多出一项JRebel的配置

查看一下插件是否有效

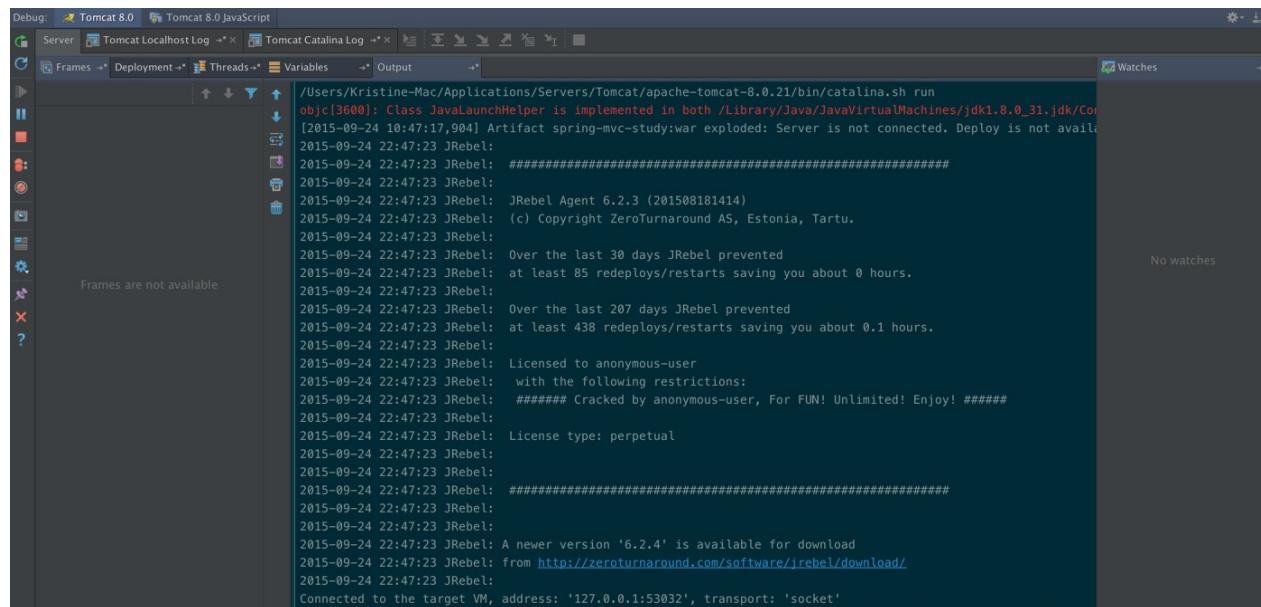


绿色的 **VALID** 表示是有效的

在原来运行项目的按钮边上会多出两个绿色的按钮，如图，前面那个是 Run，后面那个是 Debug



配置 Tomcat 的方法和直接上面说的直接调用配置方法一样，同样需要注意的是 `On 'Update' action` 和 `on frame deactivation` 这两项一定要选择 `Update classes and resources`，唯一不同的是 `VM options` 这项不需要填，放空就好。接下来直接启动项目，一般选择后面那个 Debug 按钮。



看到 Log 有 JRebel 输出的版本信息，没有报错就是表示成功执行了，随便改一个类试试吧。

JRebel 官网有免费激活服务，到官网注册领取，请支持正版：

- 官网注册，注册完就会显示一串密钥了：<https://zeroturnaround.com/software/jrebel/trial/>

IntelliJ IDEA 远程调试 Tomcat

准备工作

- 明确远程服务器的 IP 地址，比如我是：192.168.92.128
- 关掉服务器防火墙：`service iptables stop`

本地 Remote Server 配置

- 添加 Remote Server，如下图

◦

- 复制 Remote Server 自动生成的 JVM 参数，等下有用，如下图，比如我的是：
`-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005`
- 如下图，在 Host 添加服务器的 IP 地址：192.168.92.128。

-
- 把刚刚复制参数加个前缀，变成：
 - Linux（有单引号）：

```
export JAVA_OPTS='-
agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005'
```
 - Windows（没有单引号）：

```
set JAVA_OPTS=-
agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005
```

服务器 Tomcat 配置

- 以 Linux 环境为例
- Tomcat 安装在 /usr/program/tomcat7
- Tomcat 的执行程序：/usr/program/tomcat7/bin/catalina.sh
- 编辑 Tomcat 执行程序：`vim /usr/program/tomcat7/bin/catalina.sh`（Windows 是编辑：catalina.bat）
 - 在该文件的最上面，添加我们刚刚复制的那句话：

```
export JAVA_OPTS='-
agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005'
```

◦
 - 如果你的项目有特殊 JVM 参数，那你就把你的那部分参数和这部分参数合并在一起。
 - 如下图所示：

◦

服务器 Jetty 配置

- 同样是 Linux 环境
- jetty 不像 Tomcat 那样需要安装，只要有 jetty 的 jar 包就可以启动我们想要启动的应用。
- 在启动应用的时候加入之前上边我们 copy 的 -

```
agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005 就可以了。
```

- 就像这样： java -

```
agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005 -jar {your jetty  
path} {your port} --path {your war} 1>/dev/null 2>&1 &
```

开始调试

- 启动服务器 Tomcat
- 启动本地 Remote Server
- 如果可以看到如下图效果，表示已经连接成功了，接下里就是跟往常一样，在本地代码上设置断点，然后你访问远程的地址，触发到该代码自动就会在本地停住。

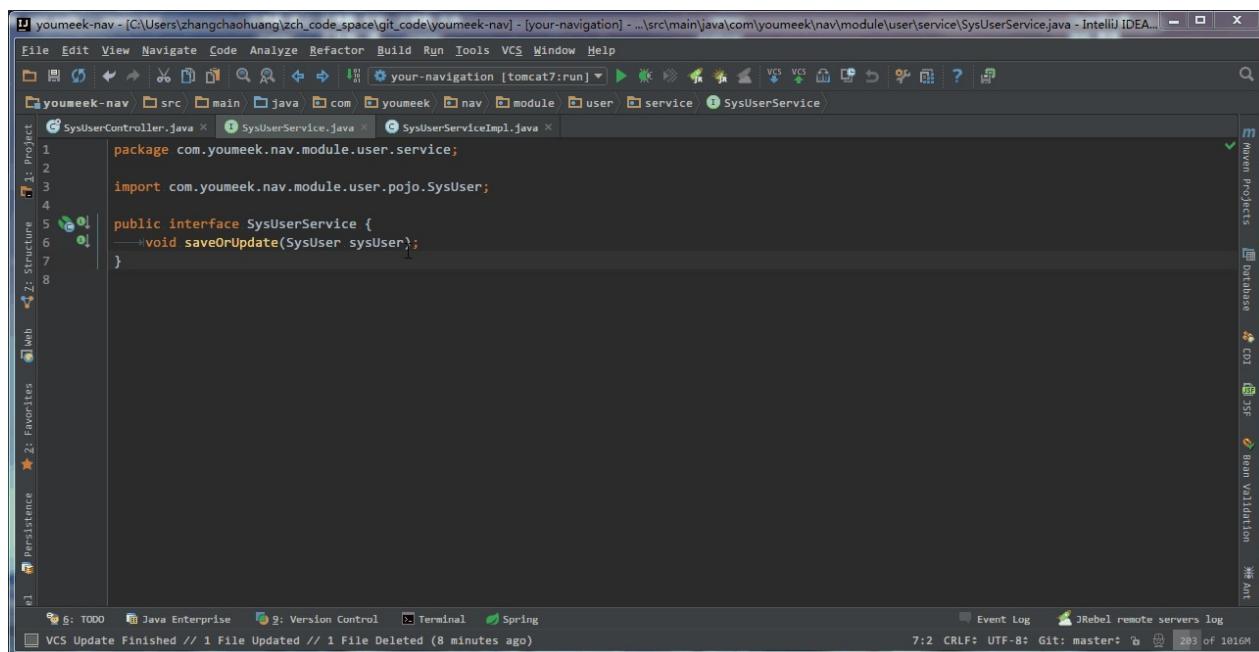
-
- 如下图，可以看到调试效果

最特殊的快捷键 Alt + Enter 介绍

说明

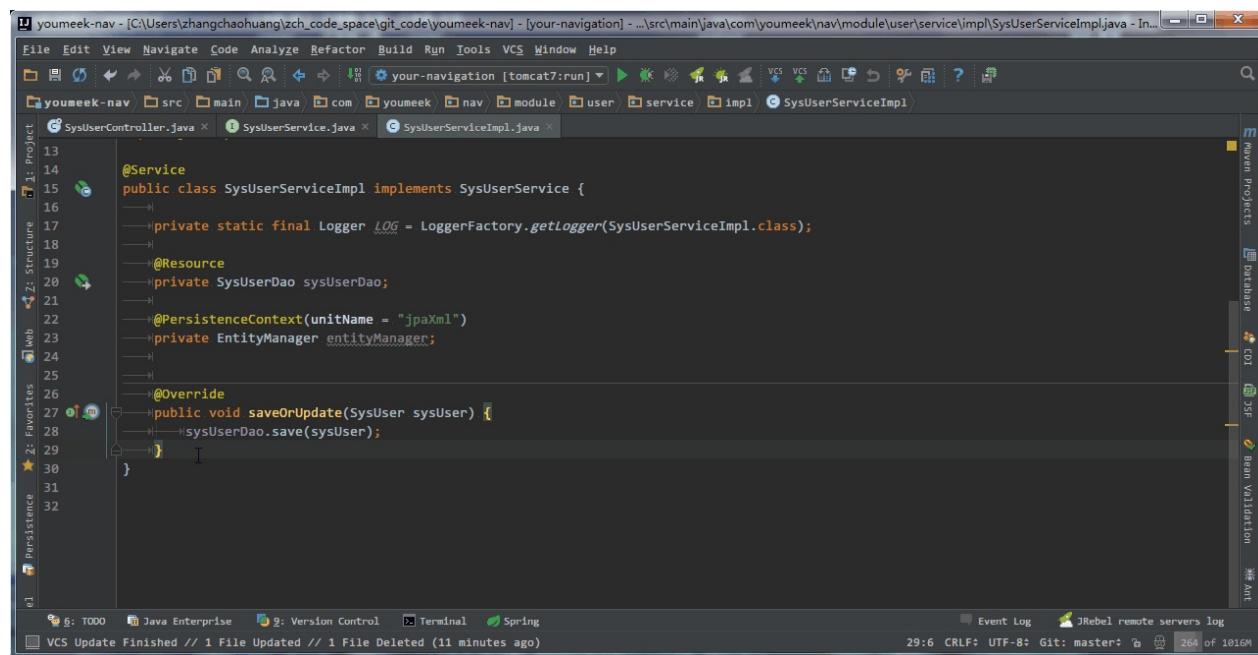
- 这是一个非常特殊的快捷键，有必要拿出来单独讲。
- 强烈注意：此快捷键跟光标所在位置有着很严重关联关系，光标放的位置不同，使用此快捷键出来的菜单选项完全不一样。
- 可以从几个思路：Java 类、JSP、HTML、JavaScript、CSS、SQL 等文件类型
- 下面演示的各个功能是基于：IntelliJ IDEA 2016.1.1，如果你使用早期版本，可能不一定有对应的功能。

智能辅助

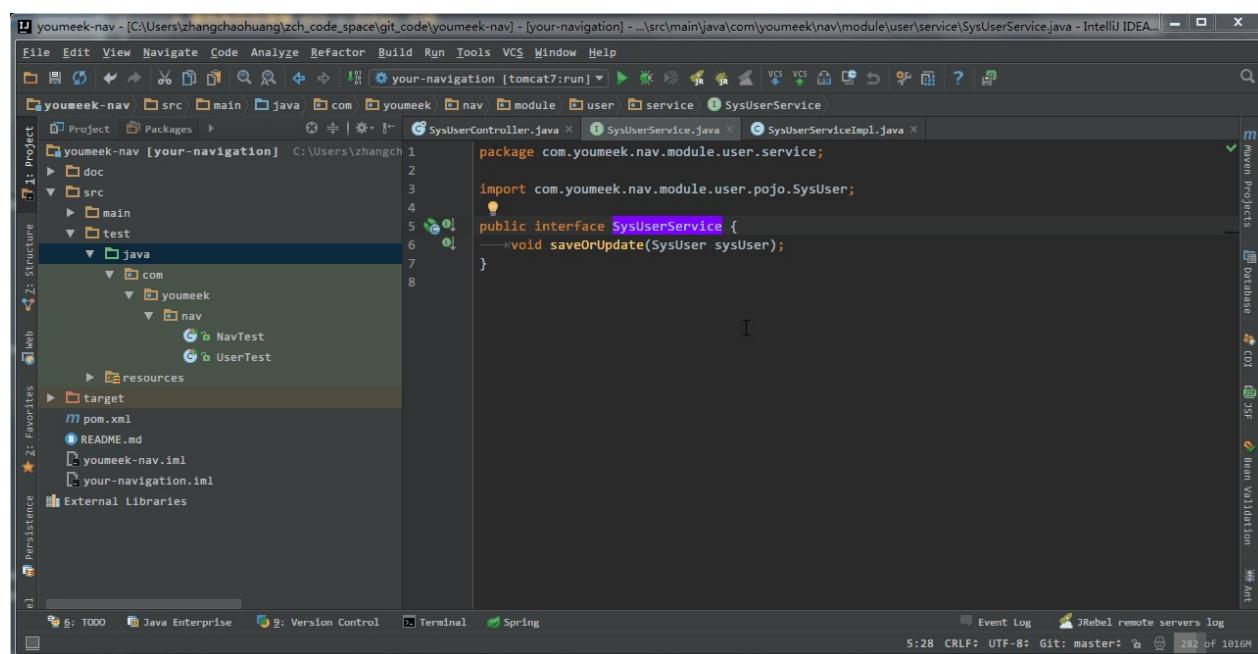


- 在 接口类 中，如果光标当前所在的方法，已经在 接口实现类 中生成了，则此快捷键的效果是跳转。
- 在 接口类 中添加一个方法后，让该 接口实现类 也跟着生成

38. 最特殊的快捷键 Alt + Enter 介绍（新用户必看）

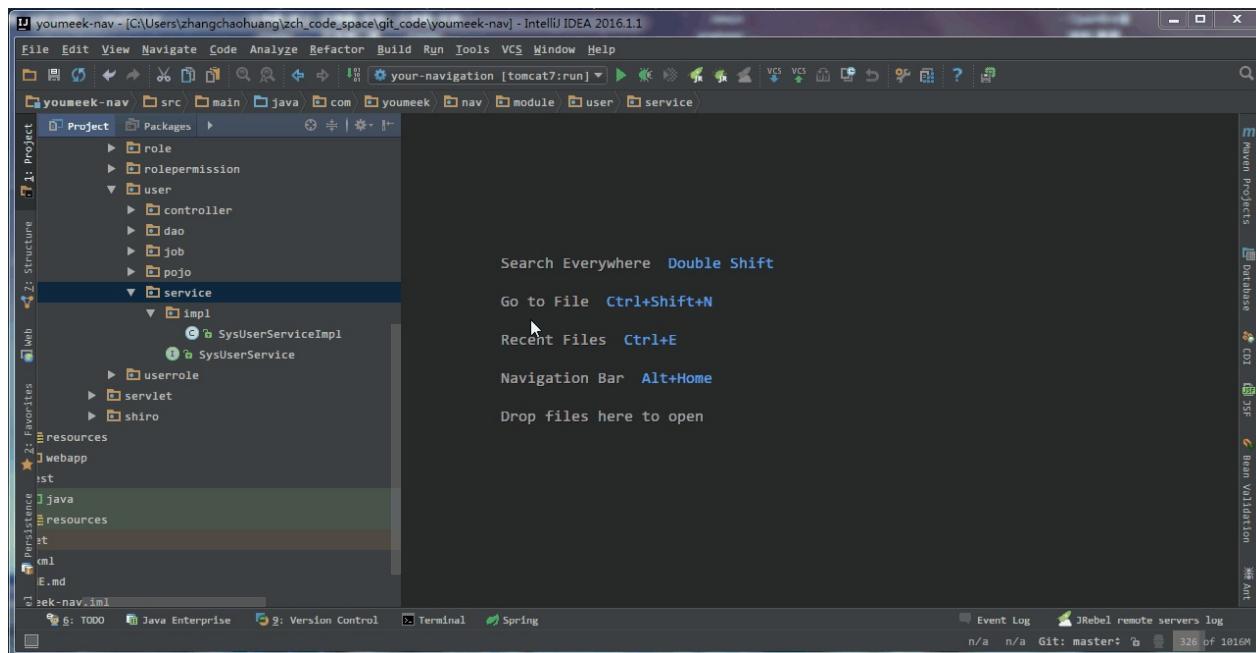


- 在接口实现类中添加一个方法后，让该接口类也跟着生成

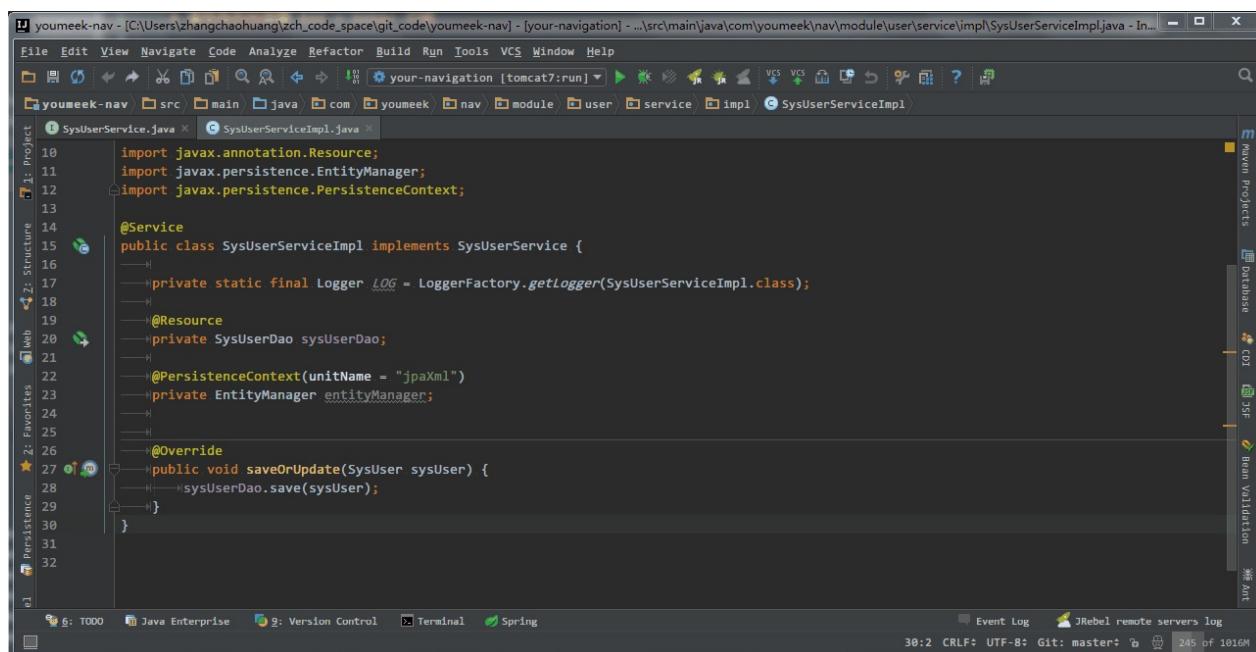


- 对当前光标所在类，生成单元测试类

38. 最特殊的快捷键 Alt + Enter 介绍 (新用户必看)



- 对当前光标所在类，创建子类，常用在对接口生成接口实现类



- 移除未使用的变量、对象等元素

38. 最特殊的快捷键 Alt + Enter 介绍（新用户必看）

The screenshot shows the IntelliJ IDEA interface with the code editor open. The code being edited is a Java class named SysUser.java. The cursor is positioned on the '@Column' annotation on line 46. A context menu is displayed, likely triggered by the Alt+Enter key combination. The menu options visible include 'Create Getter', 'Create Setter', 'Create Default Value', 'Create Validation', and 'Create Documentation'. The code editor shows several annotations like @NotBlank, @Column, and @Temporal.

```
34     * @NotBlank(message = "用户名不能为空！")
35     * @Column(name = "login_name", nullable = false)
36     * private String loginName;
37
38     * @NotBlank(message = "密码不能为空！")
39     * @Column(name = "password", nullable = false)
40     * private String password;
41
42     * @Column(name = "salt", nullable = false)
43     * private String salt;
44
45     * @Column(name = "available_enum", nullable = false, columnDefinition = "tinyint(4)")
46     * private Integer availableEnum;
47
48     * @Column(name = "create_datetime", nullable = false)
49     * @Temporal(TemporalType.TIMESTAMP)
50     * private Date createDate;
```

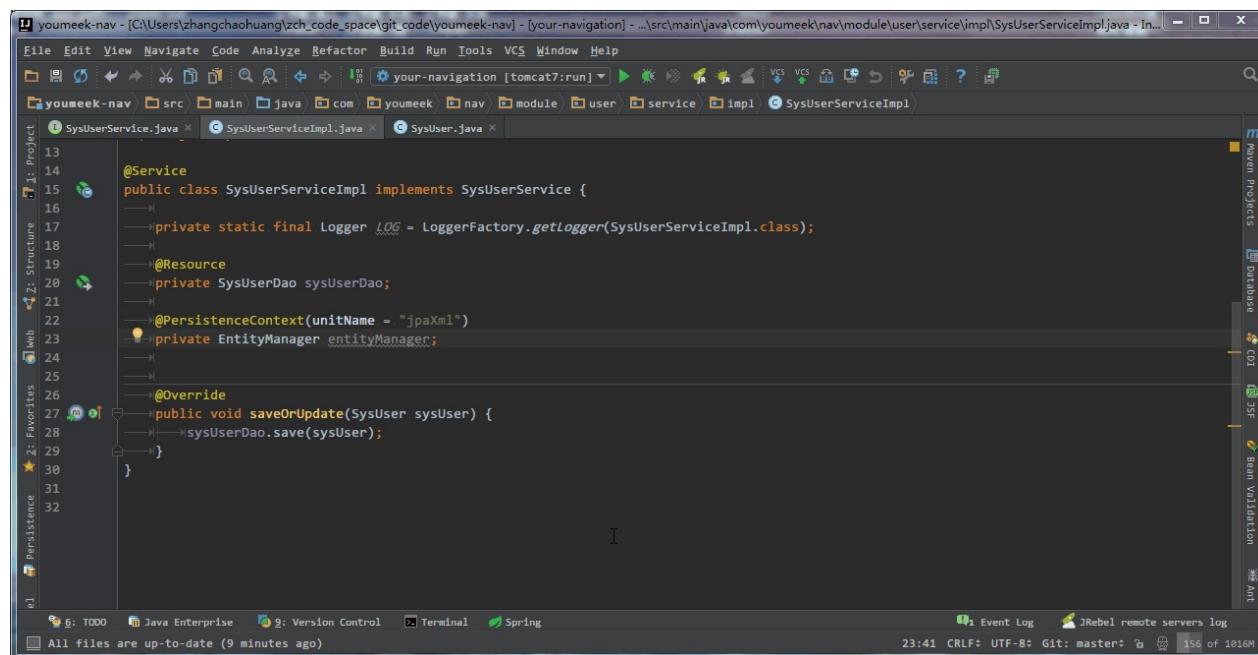
- 对属性创建 set、get 方法

The screenshot shows the IntelliJ IDEA interface with the code editor open. The code being edited is a Java class named SysUserService.java. The cursor is positioned on the 'public void saveOrUpdate(SysUser sysUser)' method on line 27. A context menu is displayed, likely triggered by the Alt+Enter key combination. The menu options visible include 'Create Getter', 'Create Setter', 'Create Default Value', 'Create Validation', and 'Create Documentation'. The code editor shows annotations like @Service, @Resource, and @Override.

```
13
14
15     @Service
16     public class SysUserServiceImpl implements SysUserService {
17
18         * private static final Logger LOG = LoggerFactory.getLogger(SysUserServiceImpl.class);
19
20         * @Resource
21         * private SysUserDao sysUserDao;
22
23         * @PersistenceContext(unitName = "jpaXml")
24         * private EntityManager entityManager;
25
26         * @Override
27         * public void saveOrUpdate(SysUser sysUser) {
28             * sysUserDao.save(sysUser);
29         }
30
31     }
32
```

- 添加 doc，只能把光标放在方法名或是变量名等这类元素上才会有

38. 最特殊的快捷键 Alt + Enter 介绍 (新用户必看)

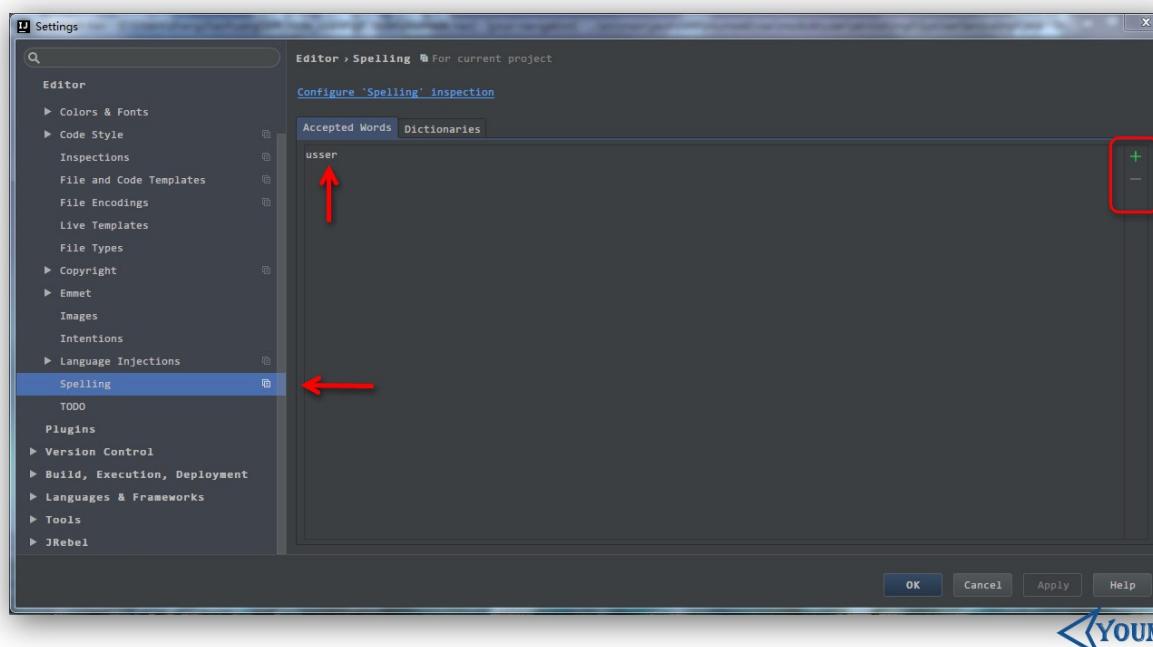


A screenshot of the Youmeek-Nav IDE interface. The main window shows a Java file named `SysUserService.java` with the following code:

```
13
14     @Service
15     public class SysUserServiceImpl implements SysUserService {
16
17         private static final Logger LOG = LoggerFactory.getLogger(SysUserServiceImpl.class);
18
19         @Resource
20         private SysUserDao sysUserDao;
21
22         @PersistenceContext(unitName = "jpaXml")
23         private EntityManager entityManager;
24
25
26         @Override
27         public void saveOrUpdate(SysUser sysUser) {
28             sysUserDao.save(sysUser);
29         }
30     }
```

The IDE has a toolbar at the top with various icons for file operations, navigation, and tools. On the left, there's a project tree showing the `youmeek-nav` project structure. The bottom status bar shows the current time as 23:41, encoding as CRLF: UTF-8, and Git status as master. A note in the status bar says "All files are up-to-date (9 minutes ago)".

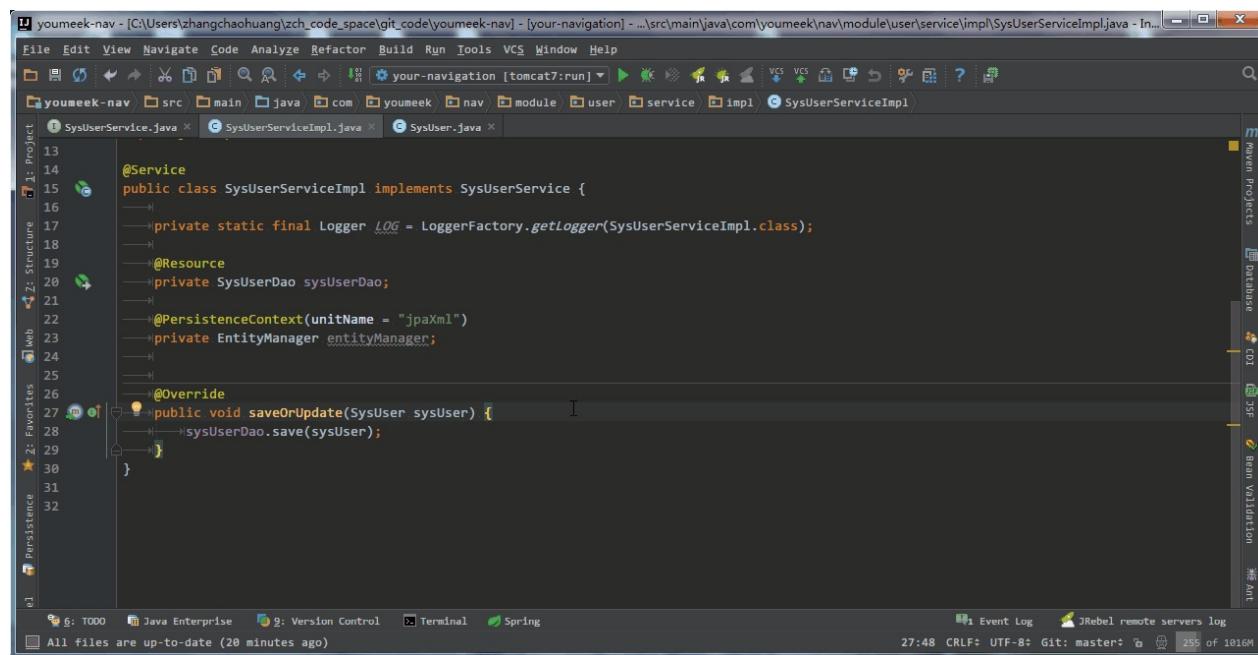
- 把自己造的单词加入词库中，让拼写单词检查错误的波浪线效果消失。



YOMEEK

- 自己造的词库在上图所示位置。

38. 最特殊的快捷键 Alt + Enter 介绍（新用户必看）

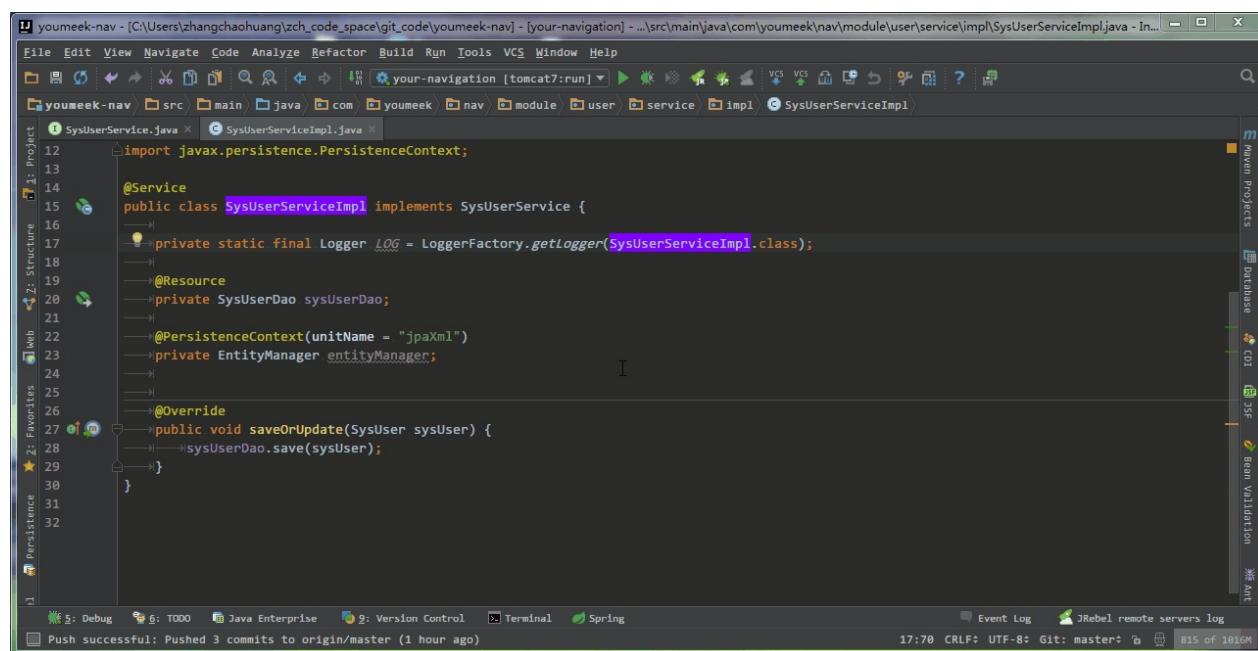


The screenshot shows the IntelliJ IDEA interface with the code editor open to `SysUserServiceImpl.java`. The cursor is positioned on the word `implements` in the class declaration. A context menu is open, with the option `Remove from interface` highlighted. The code editor displays the following Java code:

```
13
14     @Service
15     public class SysUserServiceImpl implements SysUserService {
16
17         private static final Logger LOG = LoggerFactory.getLogger(SysUserServiceImpl.class);
18
19         @Resource
20         private SysUserDao sysUserDao;
21
22         @PersistenceContext(unitName = "jpaXml")
23         private EntityManager entityManager;
24
25
26         @Override
27         public void saveOrUpdate(SysUser sysUser) {
28             sysUserDao.save(sysUser);
29         }
30     }
```

The status bar at the bottom indicates that all files are up-to-date (20 minutes ago) and shows the current time as 27:48.

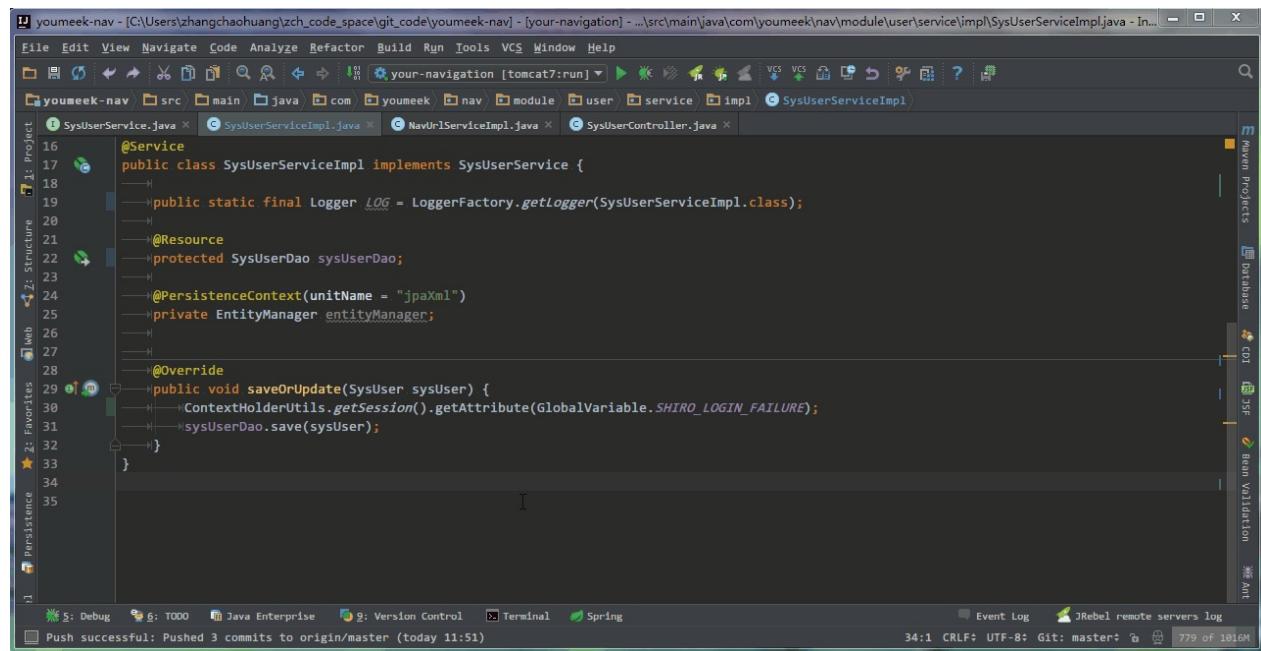
- 快速移除当前类所继承的接口，并且同时清空已经写好的该接口所有的 Override 方法。
- 光标只能方式 接口实现类 上的 接口对象单词 上才可以实现。



The screenshot shows the IntelliJ IDEA interface with the code editor open to `SysUserServiceImpl.java`. The cursor is now positioned on the word `implements` again, but the context menu is no longer open. The code editor displays the same Java code as the previous screenshot. The status bar at the bottom indicates a successful push to origin/master (1 hour ago) and shows the current time as 17:10.

- 修改光标当前元素的作用域

38. 最特殊的快捷键 Alt + Enter 介绍（新用户必看）

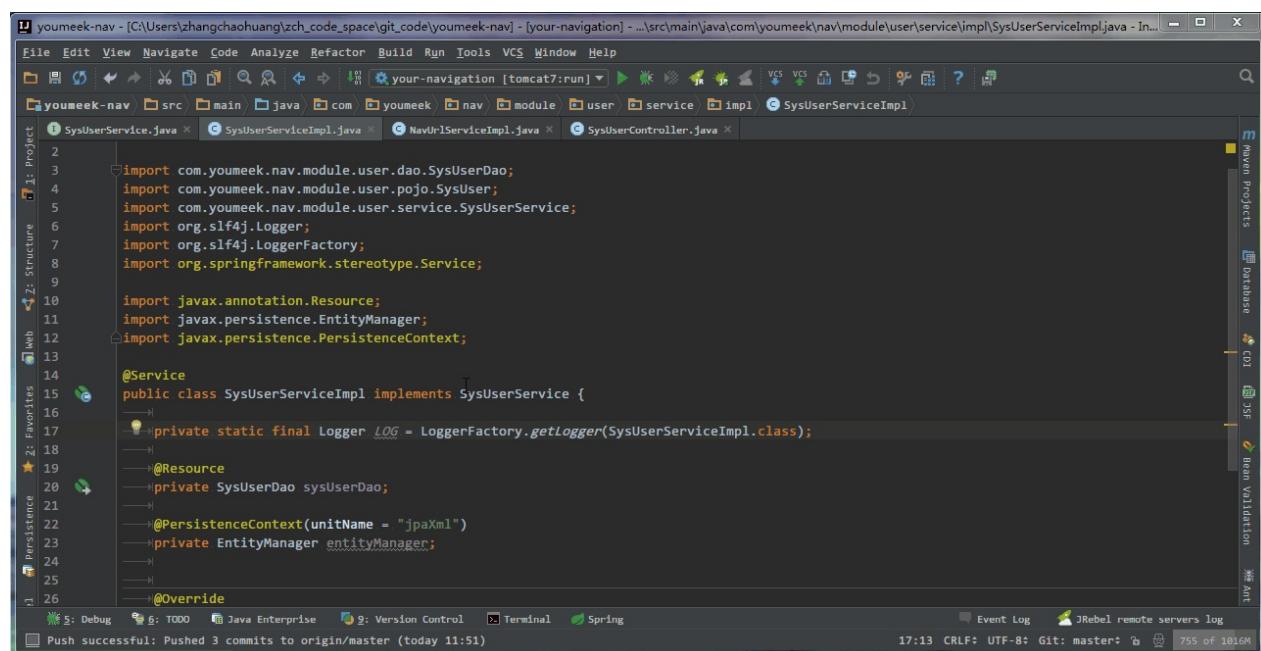


```
youmeek-nav - [C:\Users\zhangchaohuang\zch_code_space\git_code\youmeek-nav] - [your-navigation] - ...\\src\\main\\java\\com\\youmeek\\nav\\module\\user\\service\\impl\\SysUserServiceImpl.java - In...
```

```
16  @Service
17  public class SysUserServiceImpl implements SysUserService {
18
19      public static final Logger LOG = LoggerFactory.getLogger(SysUserServiceImpl.class);
20
21      @Resource
22      protected SysUserDao sysUserDao;
23
24      @PersistenceContext(unitName = "jpaXml")
25      private EntityManager entityManager;
26
27
28      @Override
29      public void saveOrUpdate(SysUser sysUser) {
30          ContextHolderUtils.getSession().getAttribute(GlobalVariable.SHIRO_LOGIN_FAILURE);
31          sysUserDao.save(sysUser);
32      }
33
34
35 }
```

The screenshot shows an IDE interface with the SysUserServiceImpl.java file open. The cursor is positioned on the 'return' keyword in the 'saveOrUpdate' method. A tooltip or context menu is likely displayed above the cursor, illustrating the 'Alt + Enter' feature.

- 给调用的方法生成返回值
- 根据返回值自动强转



```
youmeek-nav - [C:\Users\zhangchaohuang\zch_code_space\git_code\youmeek-nav] - [your-navigation] - ...\\src\\main\\java\\com\\youmeek\\nav\\module\\user\\service\\impl\\SysUserServiceImpl.java - In...
```

```
1  import com.youmeek.nav.module.user.dao.SysUserDao;
2  import com.youmeek.nav.module.user.pojo.SysUser;
3  import com.youmeek.nav.module.user.service.SysUserService;
4  import org.slf4j.Logger;
5  import org.slf4j.LoggerFactory;
6  import org.springframework.stereotype.Service;
7
8  import javax.annotation.Resource;
9  import javax.persistence.EntityManager;
10 import javax.persistence.PersistenceContext;
11
12
13
14  @Service
15  public class SysUserServiceImpl implements SysUserService {
16
17      private static final Logger LOG = LoggerFactory.getLogger(SysUserServiceImpl.class);
18
19      @Resource
20      private SysUserDao sysUserDao;
21
22      @PersistenceContext(unitName = "jpaXml")
23      private EntityManager entityManager;
24
25
26      @Override
27      public void saveOrUpdate(SysUser sysUser) {
28          ContextHolderUtils.getSession().getAttribute(GlobalVariable.SHIRO_LOGIN_FAILURE);
29          sysUserDao.save(sysUser);
30      }
31
32
33 }
```

This screenshot shows the same IDE interface as the previous one, but the cursor is now on the 'return' keyword in the 'saveOrUpdate' method. The tooltip or context menu is again likely displayed above the cursor.

- 对光标所在的对象进行包导入

38. 最特殊的快捷键 Alt + Enter 介绍（新用户必看）

The screenshot shows the IntelliJ IDEA interface with the code editor open. The code being edited is:

```
import javax.annotation.Resource;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import org.apache.commons.lang3.StringUtils;

@Service
public class SysUserServiceImpl implements SysUserService {
    ...
    @Override
    public void saveOrUpdate(SysUser sysUser, String youmeek) {
        if (StringUtil.isBlank(youmeek)) {
            ...
        }
        sysUserDao.save(sysUser);
    }
}
```

A tooltip from the code completion feature is displayed over the line `if (StringUtil.isBlank(youmeek)) {` with the text `Method 'isBlank' in type 'StringUtil' is not applicable for arguments [String]`. This indicates that the IDE is suggesting a static import for the `isBlank` method.

- 切换成静态导入

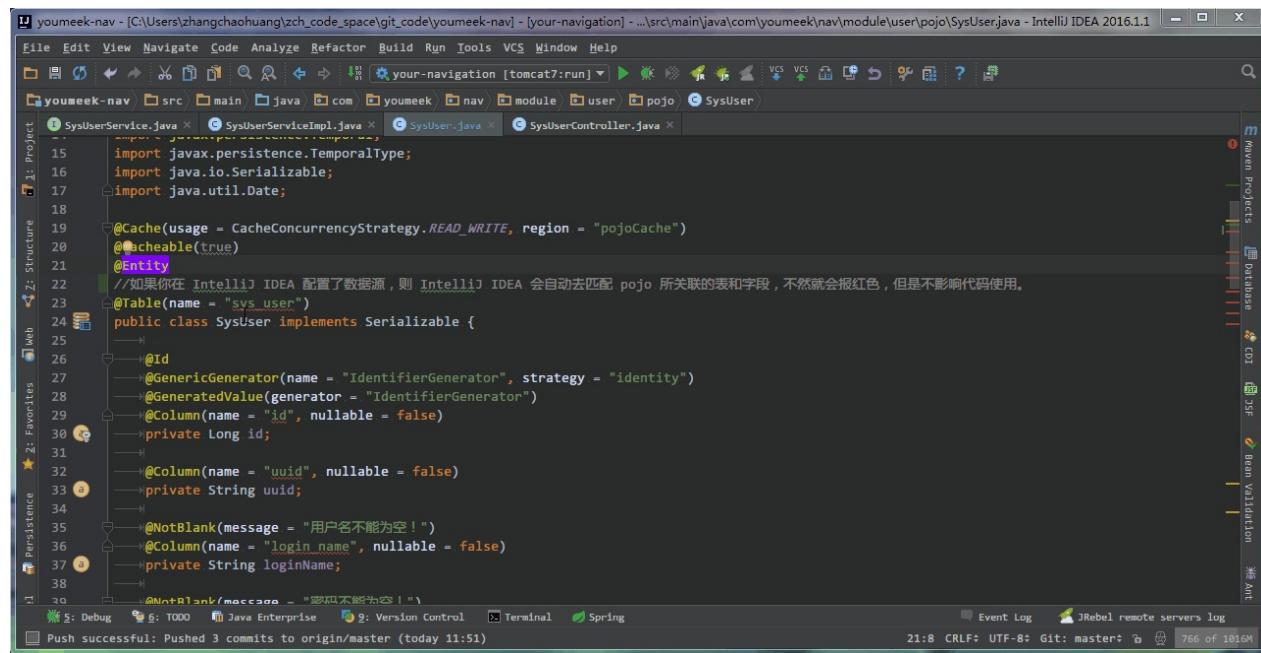
The screenshot shows the IntelliJ IDEA interface with the code editor open. The code has been modified to use static imports for `StringUtil.isBlank` and `List`:

```
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.util.ArrayList;
import java.util.List;

@Service
public class SysUserServiceImpl implements SysUserService {
    ...
    @Override
    public void saveOrUpdate(SysUser sysUser) {
        List<SysRole> sysRoleList = new ArrayList<SysRole>();
        ...
        sysUserDao.save(sysUser);
    }
}
```

- 根据 Language Level 级别不同，JDK 特性不同，给不同意见。Language Level 的含义在其他章节有讲过。

38. 最特殊的快捷键 Alt + Enter 介绍 (新用户必看)



```
15 import javax.persistence.TemporalType;
16 import java.io.Serializable;
17 import java.util.Date;
18
19 //如果你在 IntelliJ IDEA 配置了数据源，则 IntelliJ IDEA 会自动去匹配 pojo 所关联的表和字段，不然就会报红色，但是不影响代码使用。
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

Push successful: Pushed 3 commits to origin/master (today 11:51) 21:8 CRLF+ UTF-8: Git: master: 766 of 1016M

- 给 Hibernate 的 Entity 对象分配数据源，从而产生一系列智能功能

本系列教程结束

赠语

只希望本系列教程只是起到一个让你真正了解 IntelliJ IDEA 的一个引子作用，没有什么比 IntelliJ IDEA 自己提供的帮助文档更有意义和说服力。希望以后的学习中，你能多看 IntelliJ IDEA 自带的帮助文档和关注官网动态。

最后，感谢 JetBrains 公司做了如此优秀的产品，感谢你我对 IntelliJ IDEA 抱有兴趣，感谢极客学院对本教程的支持！

如果你发现本系列教程有错误或是表达不清的地方，请邮件（judas.n@qq.com）或下方微信联系，真心地万分感谢（鞠躬）。

