

# 基于 IntelliJ IDEA 高效开发分享

棒谷网络科技有限公司

张朝煌（架构组）



# 分享环境：

- 01. 基于 Spring Boot 项目：<https://github.com/judasn/spring-boot-showcase.git>
- 02. JDK 1.8
- 03. IntelliJ IDEA 2017.2
- 04. Windows 10 和 macOS Sierra

# 分享内容：

- 01. 跟踪 IntelliJ IDEA 动态渠道介绍
- 02. IntelliJ IDEA 对版本控制的支持 ( Git )
- 03. IntelliJ IDEA 的 Git Flow 流程
- 04. IntelliJ IDEA 代码 Debug 技巧
- 05. IntelliJ IDEA 实时代码模板使用场景
- 06. IntelliJ IDEA 文件代码模板使用场景
- 07. IntelliJ IDEA Postfix Completion 代码模板介绍
- 08. IntelliJ IDEA 辅助开发的各类插件推荐/使用场景
- 09. IntelliJ IDEA 提高效率的个性化设置
- 10. IntelliJ IDEA 常用快捷键使用场景 ( Windows + Mac )
- 11. IntelliJ IDEA Alt + Enter 特殊性

# 跟踪 IntelliJ IDEA 动态渠道介绍

# 关注渠道：

- 官方博客（支持 RSS）：<http://blog.jetbrains.com/idea/>
- IntelliJ IDEA 官方 community：<https://intellij-support.jetbrains.com/hc/en-us/community/topics>
- IntelliJ IDEA 官方 issue：<https://youtrack.jetbrains.com/issues/IDEA>
- YouTube：<https://www.youtube.com/user/intellijideavideo>
- Twitter：<https://twitter.com/IntelliJIDEA>
- Facebook：<https://www.facebook.com/IntelliJIDEA>
- JetBrains 新浪微博：<http://www.weibo.com/u/3220313942>
- JetBrains Google+：<https://plus.google.com/+jetbrains>
- IntelliJ IDEA Google+：<https://plus.google.com/+intellijidea>

# Spring Boot 项目 导入、运行、单元测试

# 项目资料：

- Github：<https://github.com/judasn/spring-boot-showcase>

## ## 环境

- JDK 版本：JDK 8
- IDE 版本：IntelliJ IDEA 2017.2
- JUnit 版本：JUnit 4 (测试方法无需前面写 test)

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.4.RELEASE</version>
  <relativePath/>
</parent>
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

```
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>druid</artifactId>
  <version>1.0.28</version>
</dependency>
<dependency>
  <groupId>com.github.drtrng</groupId>
  <artifactId>druid-spring-boot-starter</artifactId>
  <version>1.0.1</version>
</dependency>

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>

<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.6.1</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.6.1</version>
</dependency>

<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.3.2</version>
</dependency>

<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.31</version>
</dependency>
```

# 版本控制的支持（Git）



# 主流托管地：

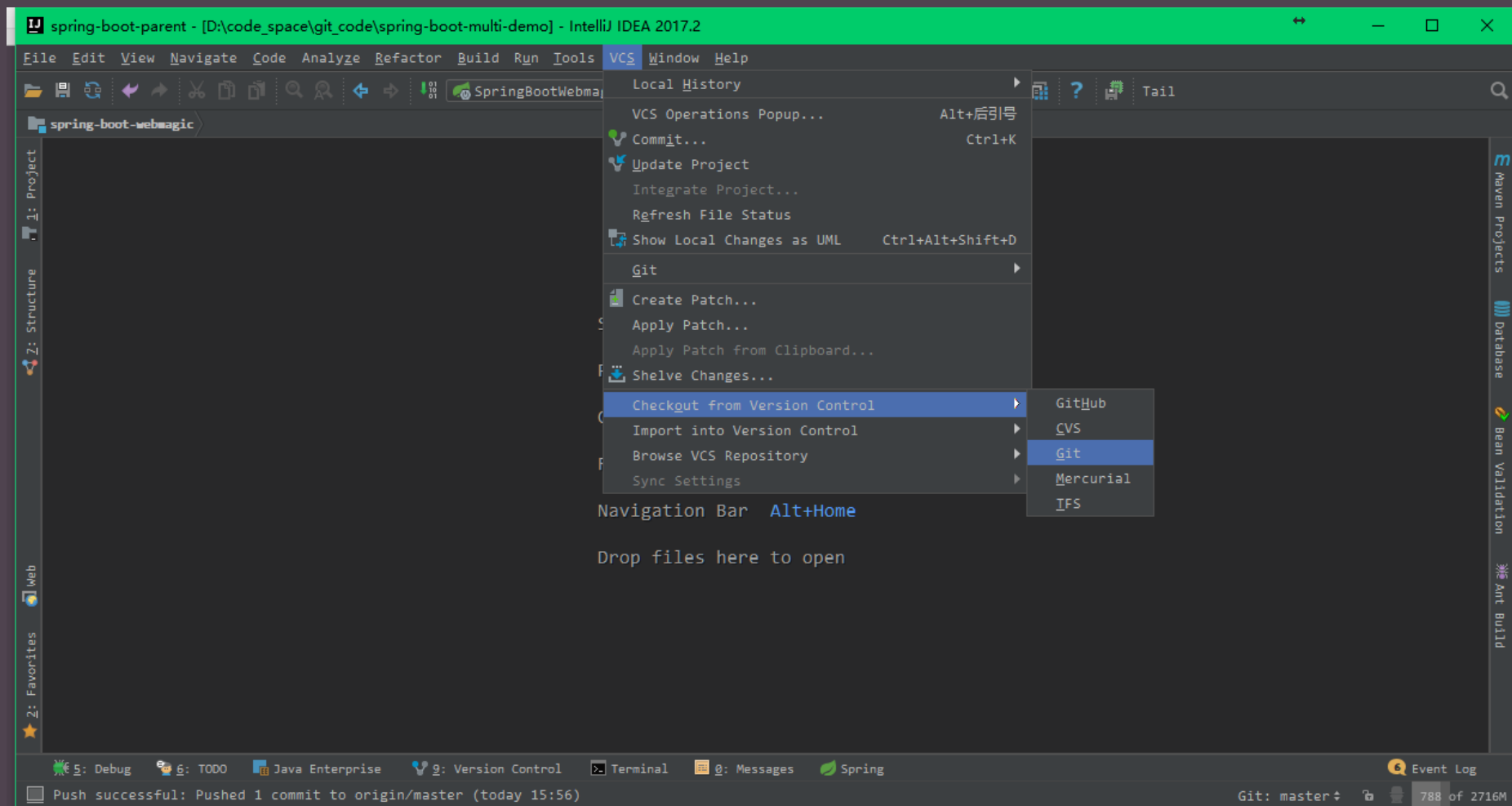
- Github：<https://github.com/>
- 码云：<http://git.oschina.net/>
- Coding：<https://coding.net/home.html>

## 对应的 IntelliJ IDEA 插件：

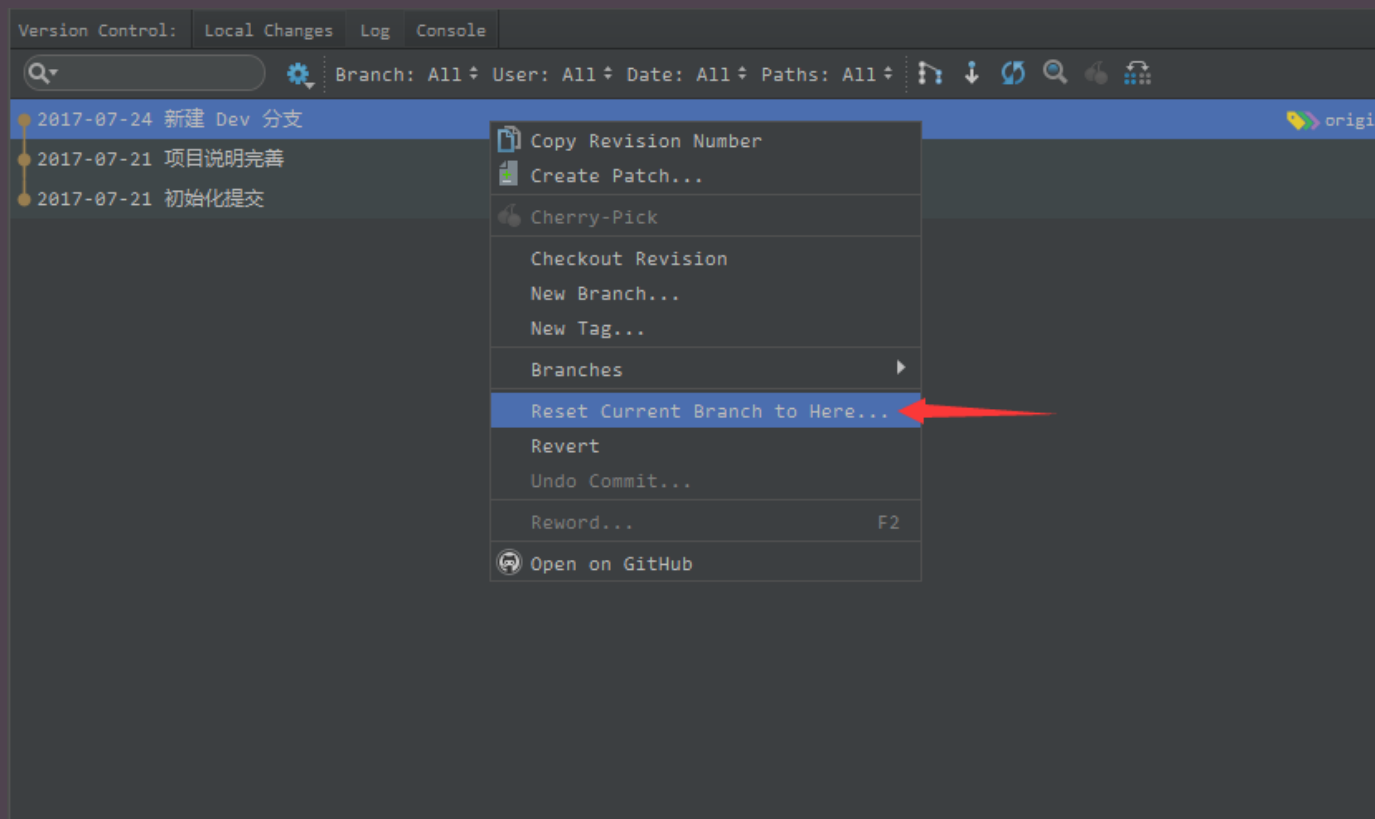
- Github：内置
- 码云：<https://plugins.jetbrains.com/plugin/8383-gitosc>
- Coding：<https://plugins.jetbrains.com/plugin/8565-coding-net>

# Clone 项目：

- 小项目和大项目采用不同方式



# 版本之间的回滚

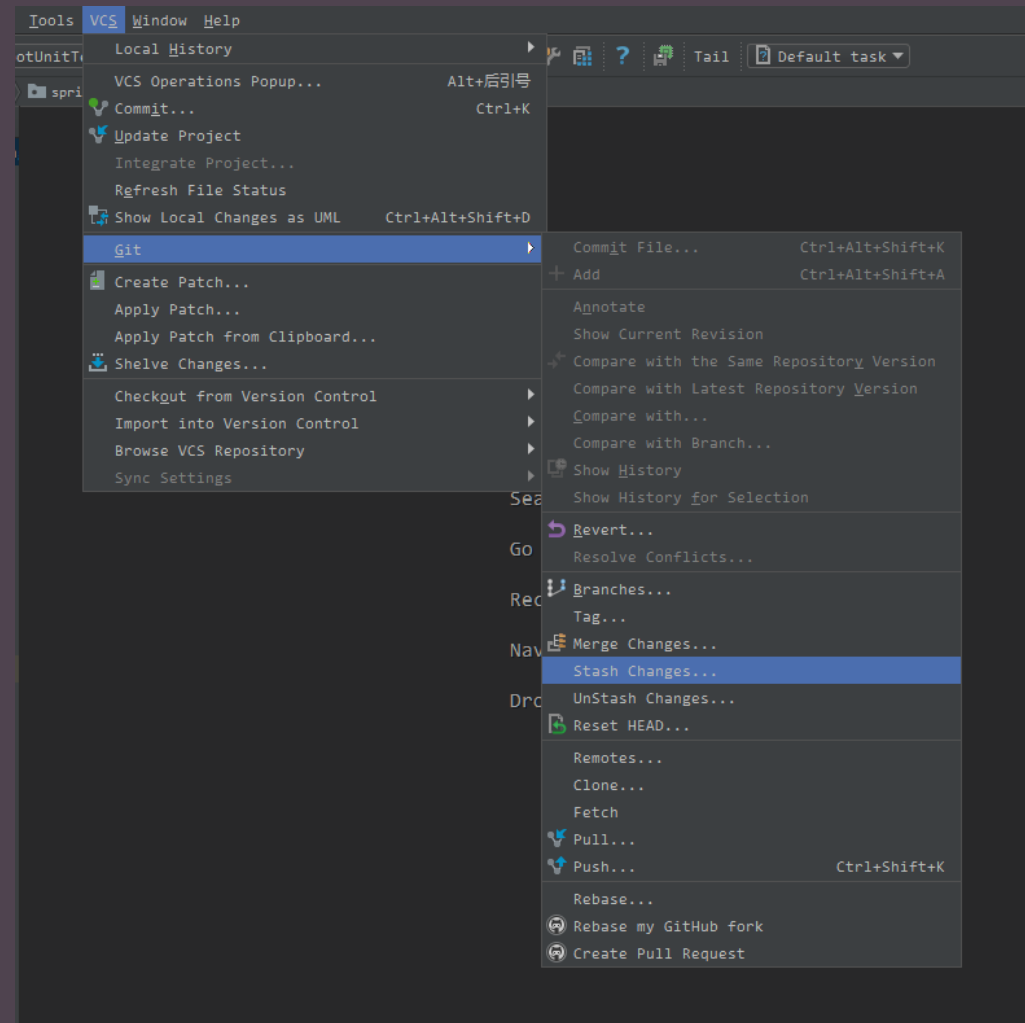
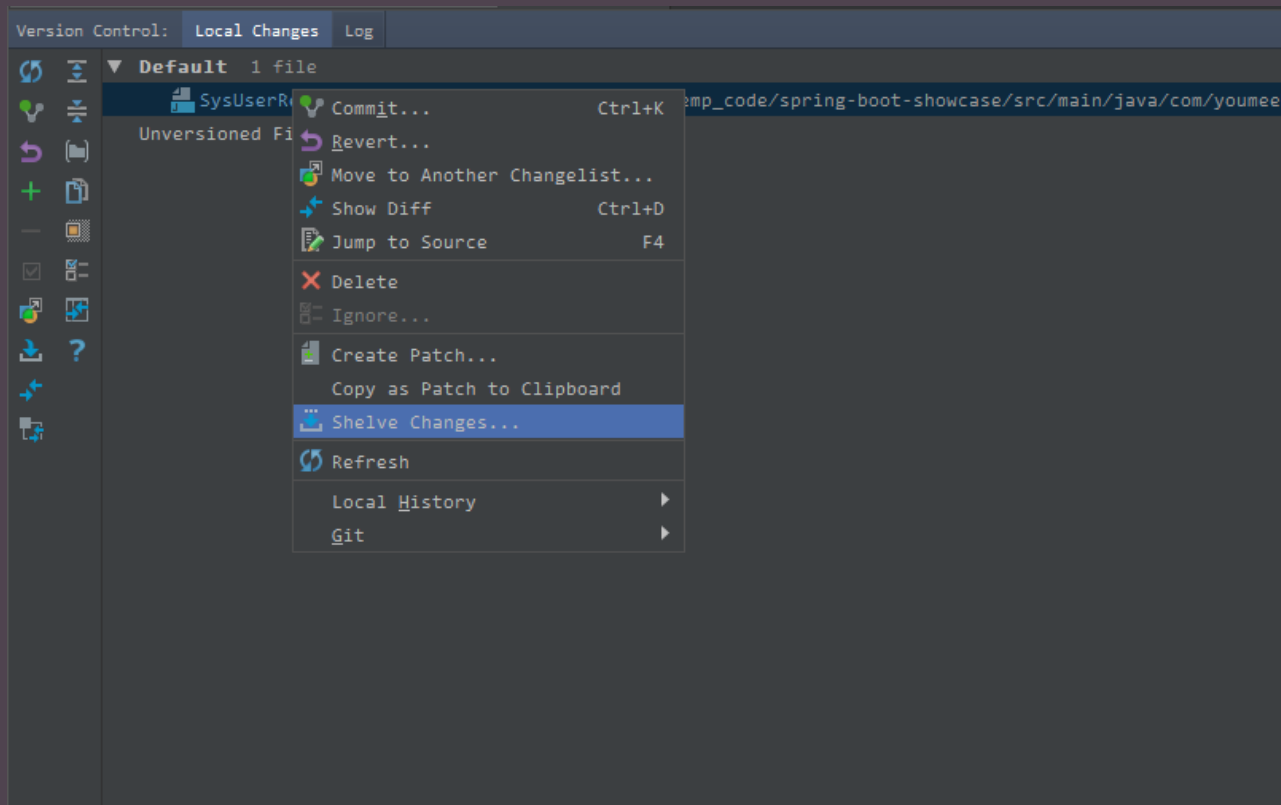


**git reset --soft**：回退到某个版本，只回退了commit的信息，不会恢复到index file一级。如果还要提交，直接commit即可，也就是说新增的文件还是在git版本控制中还是绿色状态。

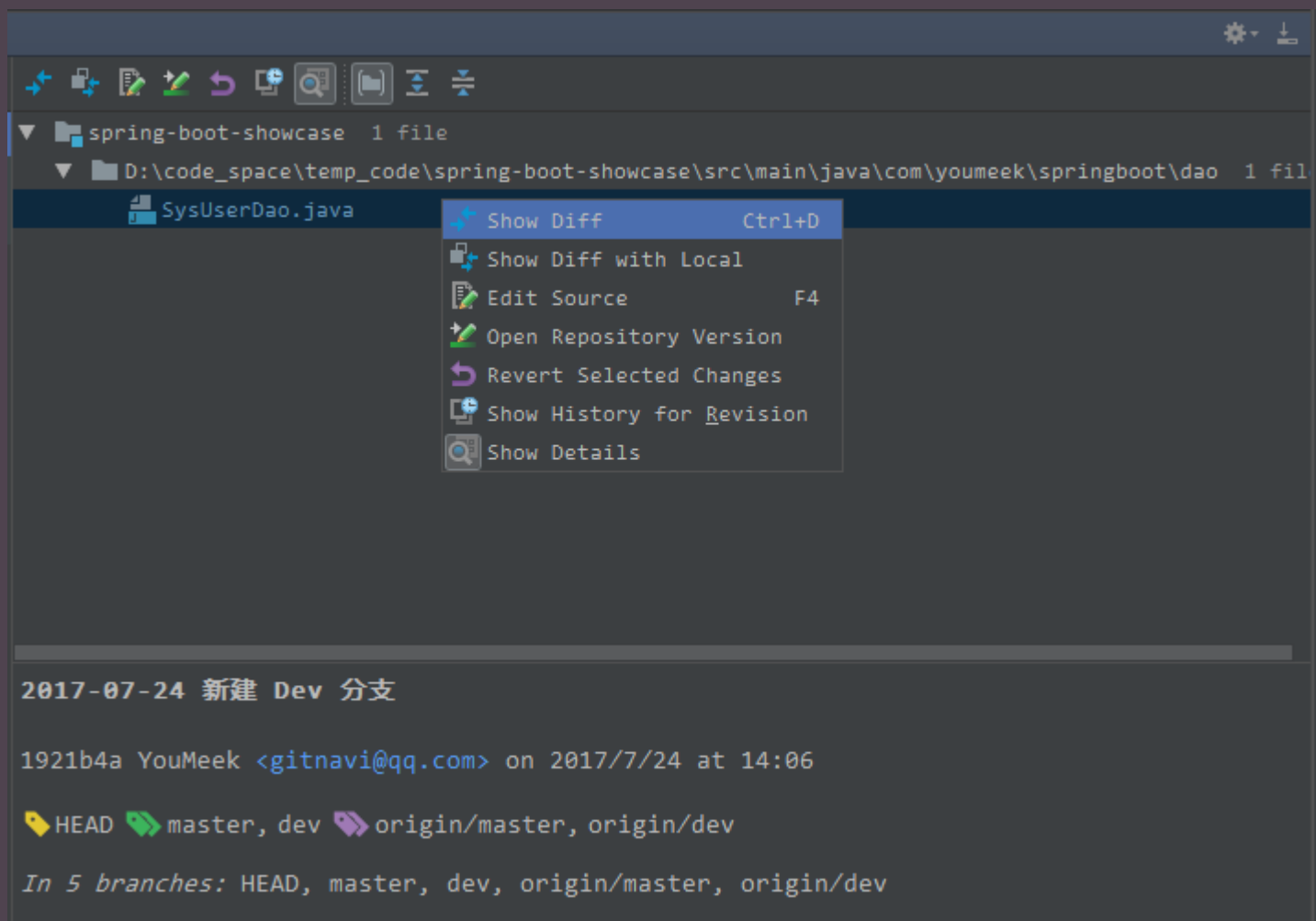
**git reset --mixed**：此为默认方式，不带任何参数的git reset，即时这种方式，它回退到某个版本，只保留源码，回退commit和index信息  
>>比如当前版本1101，我们要回退到1000，在相差101个版本之间，有新增5个文件，修改2个文件。  
那回退的效果是：5个新增文件还存在，但是已经没有被加入到版本控制，2个被修改的文件，还原到1000版本的状态下，你可以Revert

**git reset --hard**：彻底回退到某个版本，本地的源码也会变为上一个版本的内容，此命令 慎用！

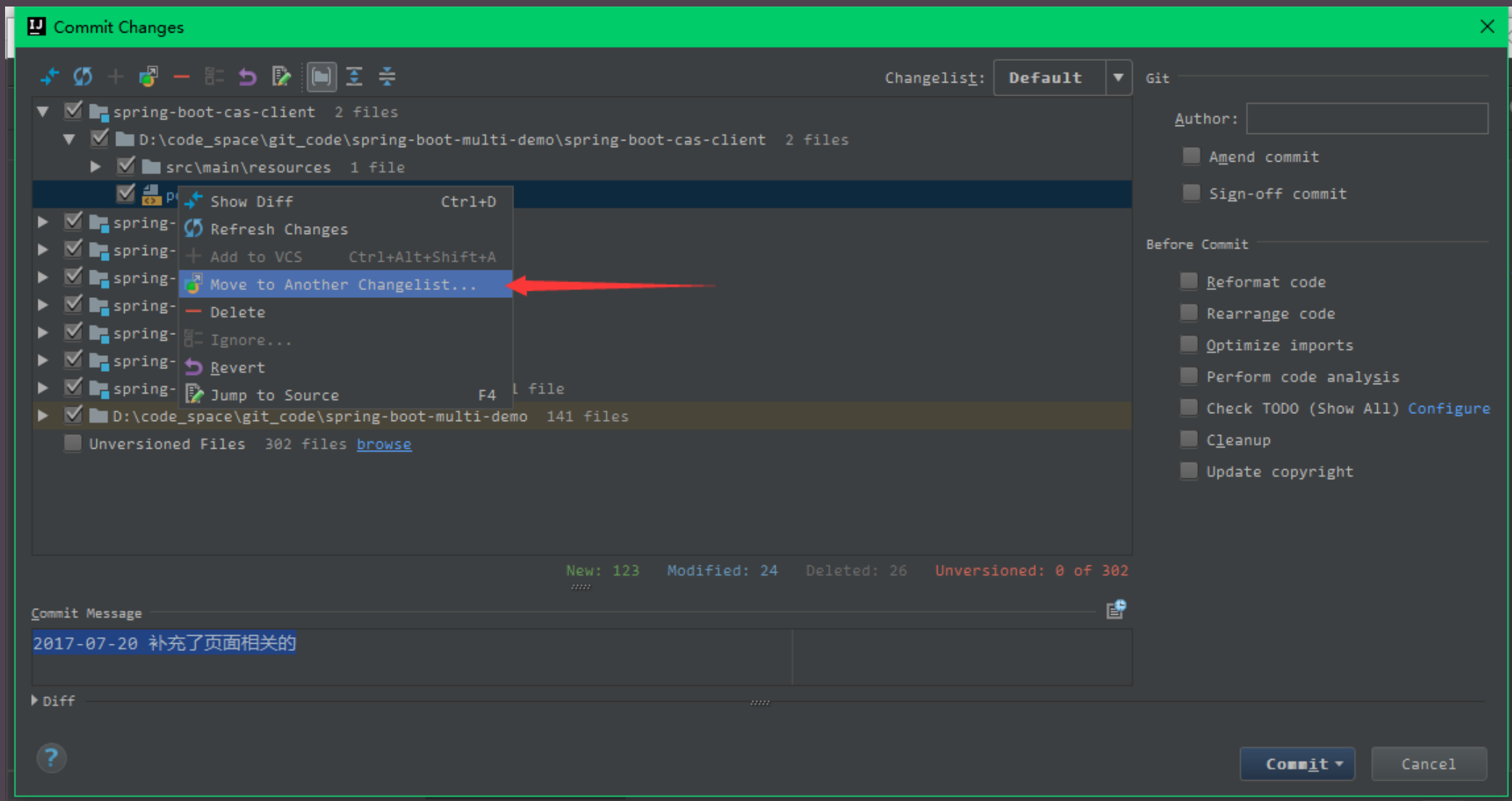
# Shelved Changes 与 Stash Changes 区别



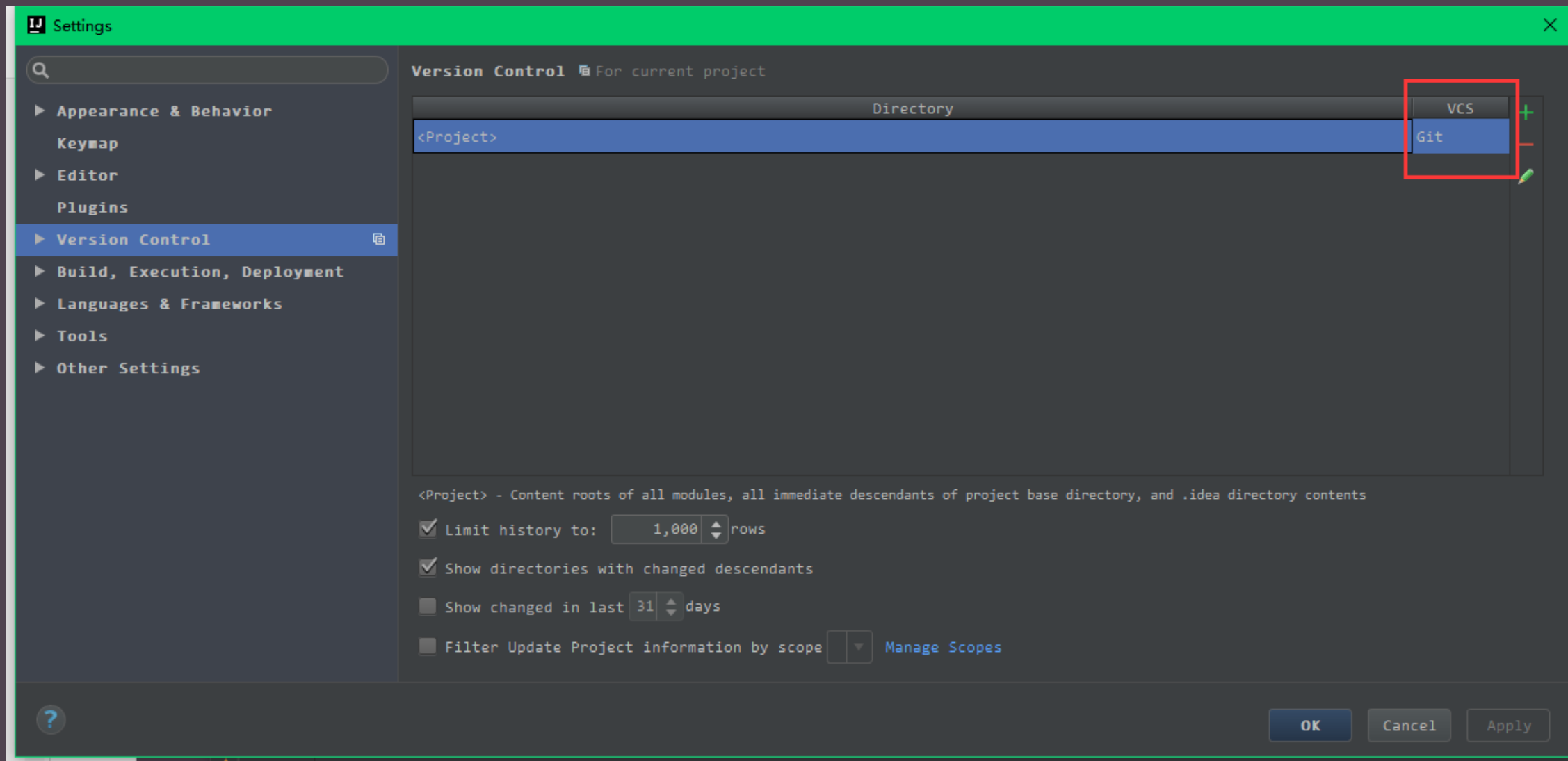
# 版本之间的差异对比



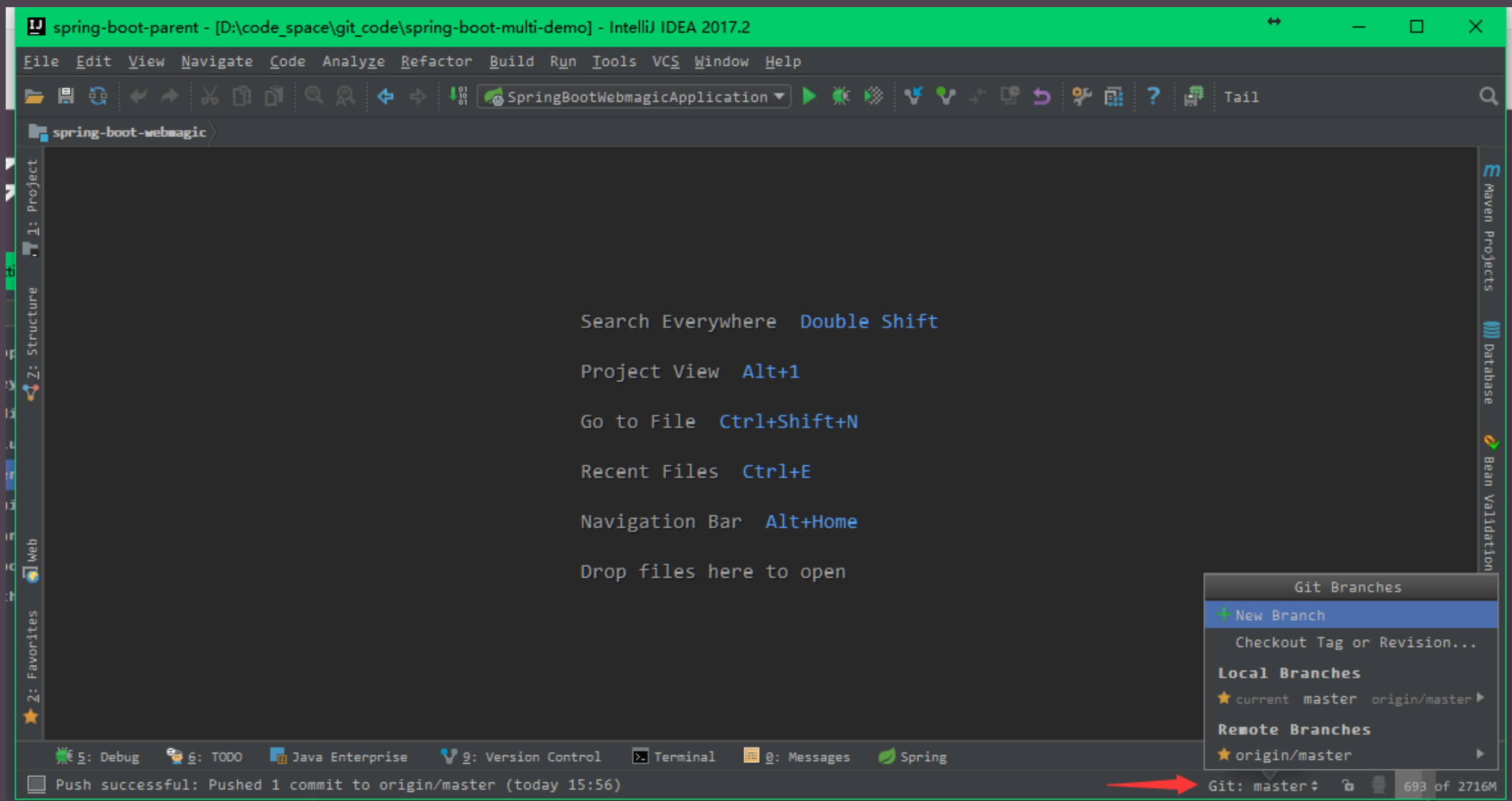
# .gitignore 和 Changelist 使用场景：



# 多个版本控制切换：

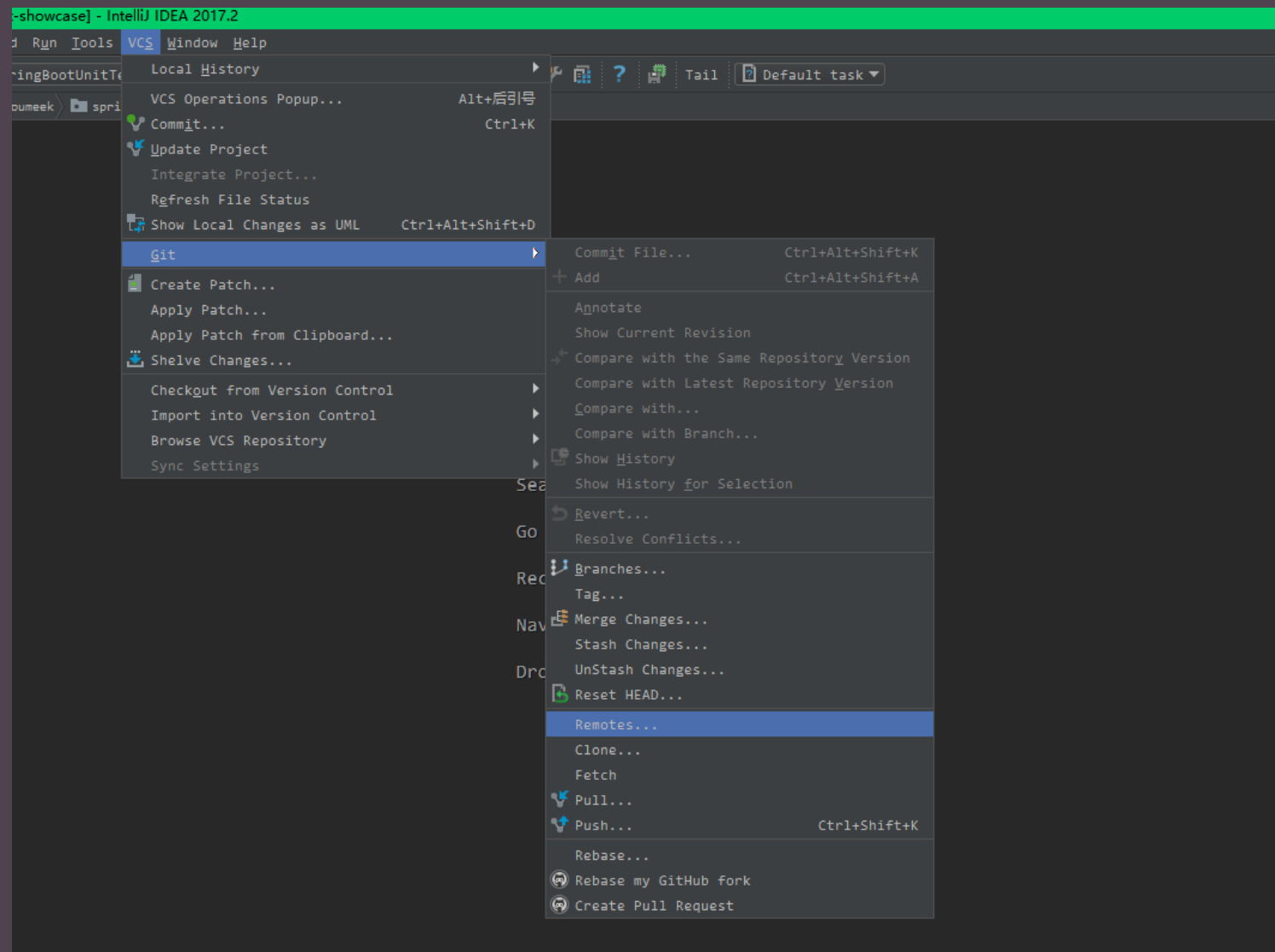


# 分支的管理：





# 多个托管管理：



# IntelliJ IDEA 的 Git Flow 流程

# Git Flow 概念：

Git Flow 是一个 git 扩展集，按 Vincent Driessen 的分支模型提供高层次的库操作。这里的重点是 Vincent Driessen 的分支模型思想，下面讲解的内容也是基于 Vincent Driessen 思想。

Vincent Driessen 的观点：<http://nvie.com/posts/a-successful-git-branching-model/>

Git Flow 是一个 git 扩展集，这句话你可以理解为 Git Flow 是一个基于 Git 的插件，这个插件简化了 Git 一些复杂的命令，比如 Git Flow 用一条命令，就可以代替 Git 原生 10 条命令。

Git Flow 对原生的 Git 不会有任何影响，你可以照旧用 Git 原生命令，也可以使用 Git Flow 命令。

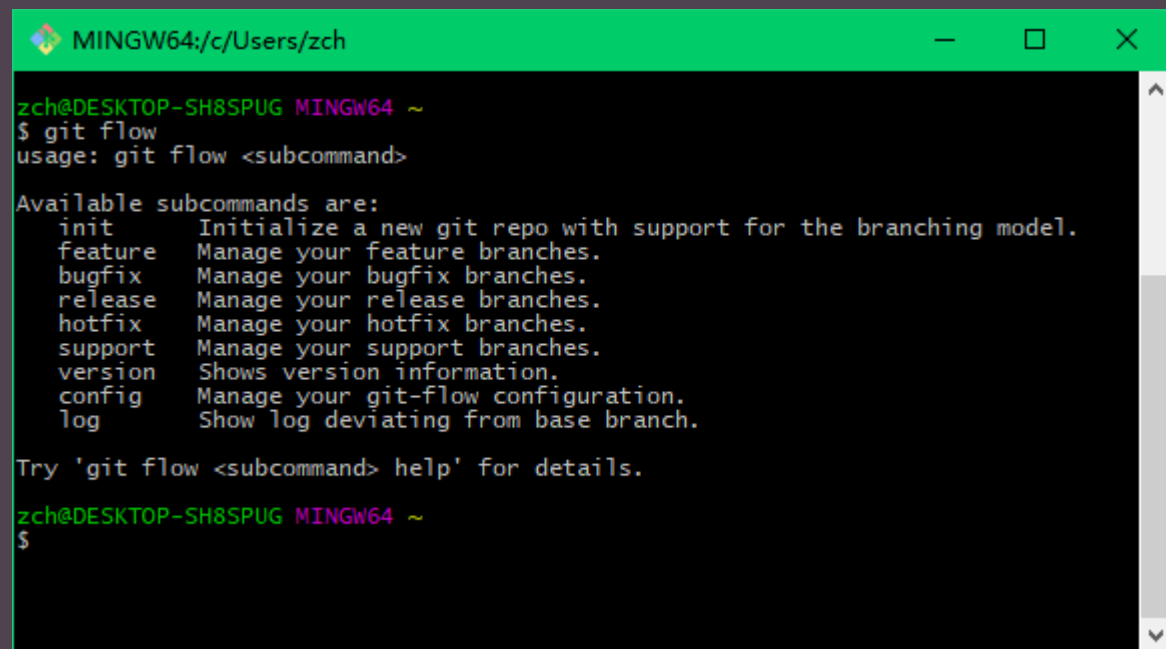
还有其他的一些分支管理模型思想，具体可以看：<http://www.ruanyifeng.com/blog/2015/12/git-workflow.html>

# Git Flow 安装：

**Windows**：如果你安装 Git 用的是 [Git for Windows](#)，那它已经内置了。

**Mac**：brew install git-flow-avh

更多系统安装版本：<https://github.com/petervanderdoes/gitflow-avh/wiki/Installation>

A screenshot of a Windows terminal window with a green title bar. The title bar text is 'MINGW64:/c/Users/zch'. The terminal content shows the command 'git flow' being executed, which results in a usage message and a list of available subcommands. The subcommands listed are: init (Initialize a new git repo with support for the branching model.), feature (Manage your feature branches.), bugfix (Manage your bugfix branches.), release (Manage your release branches.), hotfix (Manage your hotfix branches.), support (Manage your support branches.), version (Shows version information.), config (Manage your git-flow configuration.), and log (Show log deviating from base branch.). The terminal also prompts the user to try 'git flow <subcommand> help' for details. The prompt 'zch@DESKTOP-SH8SPUG MINGW64 ~' is visible at the top and bottom of the terminal window.

```
MINGW64:/c/Users/zch

zch@DESKTOP-SH8SPUG MINGW64 ~
$ git flow
usage: git flow <subcommand>

Available subcommands are:
  init      Initialize a new git repo with support for the branching model.
  feature   Manage your feature branches.
  bugfix    Manage your bugfix branches.
  release   Manage your release branches.
  hotfix    Manage your hotfix branches.
  support   Manage your support branches.
  version   Shows version information.
  config    Manage your git-flow configuration.
  log       Show log deviating from base branch.

Try 'git flow <subcommand> help' for details.

zch@DESKTOP-SH8SPUG MINGW64 ~
$
```

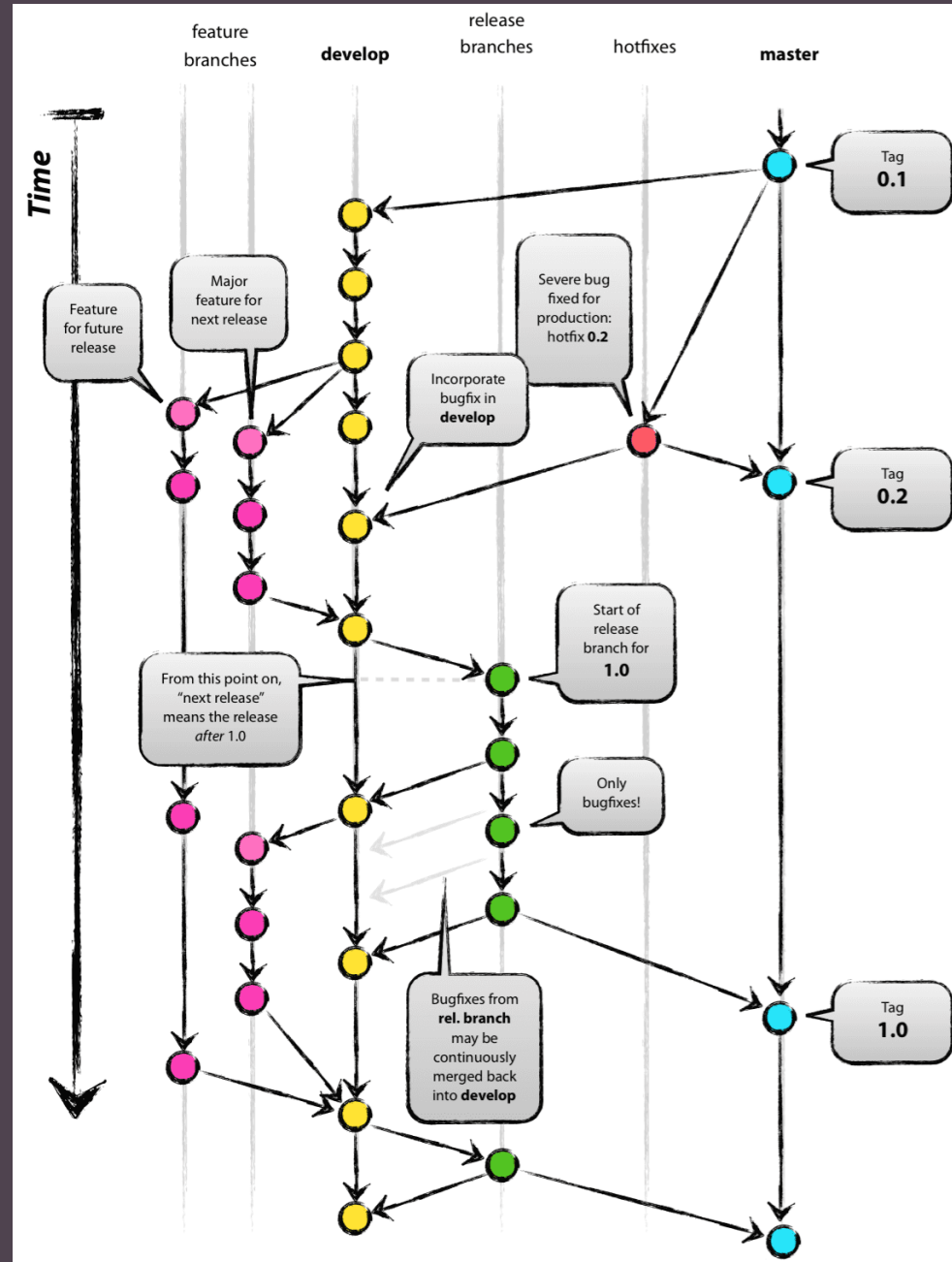
# Git Flow 核心流程：

## • 必须有的两个核心分支（长期分支）：

- master，Git 代码仓库中默认的一条主分支。这条分支上的代码一般都建议为是正式版本的代码，并且这条分支不能进行代码修改，只能用来合并其他分支。
- develop，一般用于存储开发过程的代码分支，并且这条分支也不能进行代码修改，只能用来合并其他辅助分支。

## • 根据情况创建的辅助分支（临时分支）

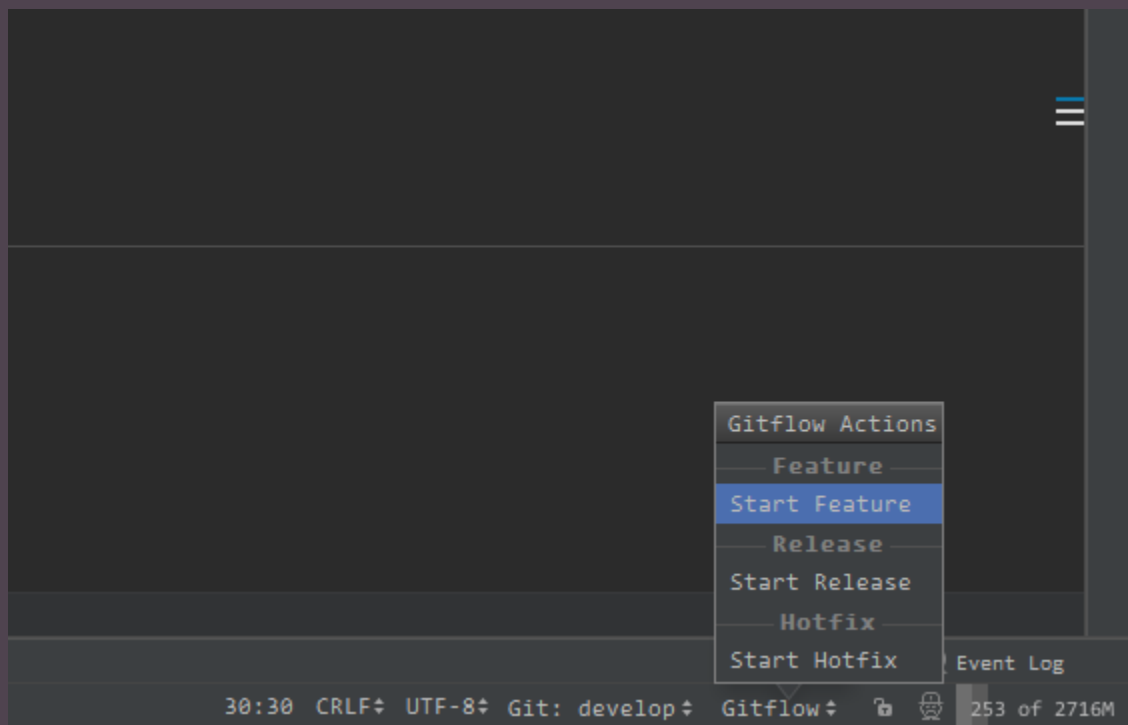
- feature branches（功能分支）
  - 基于 develop 分支上创建
  - 开发完成后合并到 develop 分支上
  - 当要开始一个新功能的开发时，我们可以创建一个 Feature branches。等待这个新功能开发完成并确定应用到新版本中就合并回 develop
  - 对于单人开发的 feature branches，start 之后，开发完成后可以直接 finish。
  - 对于多人开发的 feature branches，start 之后，开发完成后先 publish 给其他开发人员进行合并，最后大家都开发完成后再 finish。这个思路也同样适用下面几个辅助分支场景。
  - feature branches 开发过程有 bug，直接在 feature branches 上修改、提交。
- release branches（预发布分支）
  - 基于 develop 分支上创建
  - 测试确定新功能没有问题，合并到 develop 分支和 master 分支上
  - 用来做新版本发布前的准备工作，在上面可以做一些小的 bug 修复、准备发布版本号等等和发布有关的小改动，其实已经是一个比较成熟的版本了。另外这样我们既可以在预发布分支上做一些发布前准备，也不会影响 "develop" 分支上下一版本的新功能开发。
- hotfix branches（基于 master 基础上的生产环境 bug 的修复分支）
  - 基于 master 分支上创建
  - 修复测试无误后合并到 master 分支和 develop 分支上
  - 主要用于处理线上版本出现的一些需要立刻修复的 bug 情况



# Git Flow 实现：

IntelliJ IDEA 对 Git Flow 支持的插件：

<https://plugins.jetbrains.com/plugin/7315-git-flow-integration>



# IntelliJ IDEA 代码 Debug 技巧

# Debug 过程中的必备的快捷键：

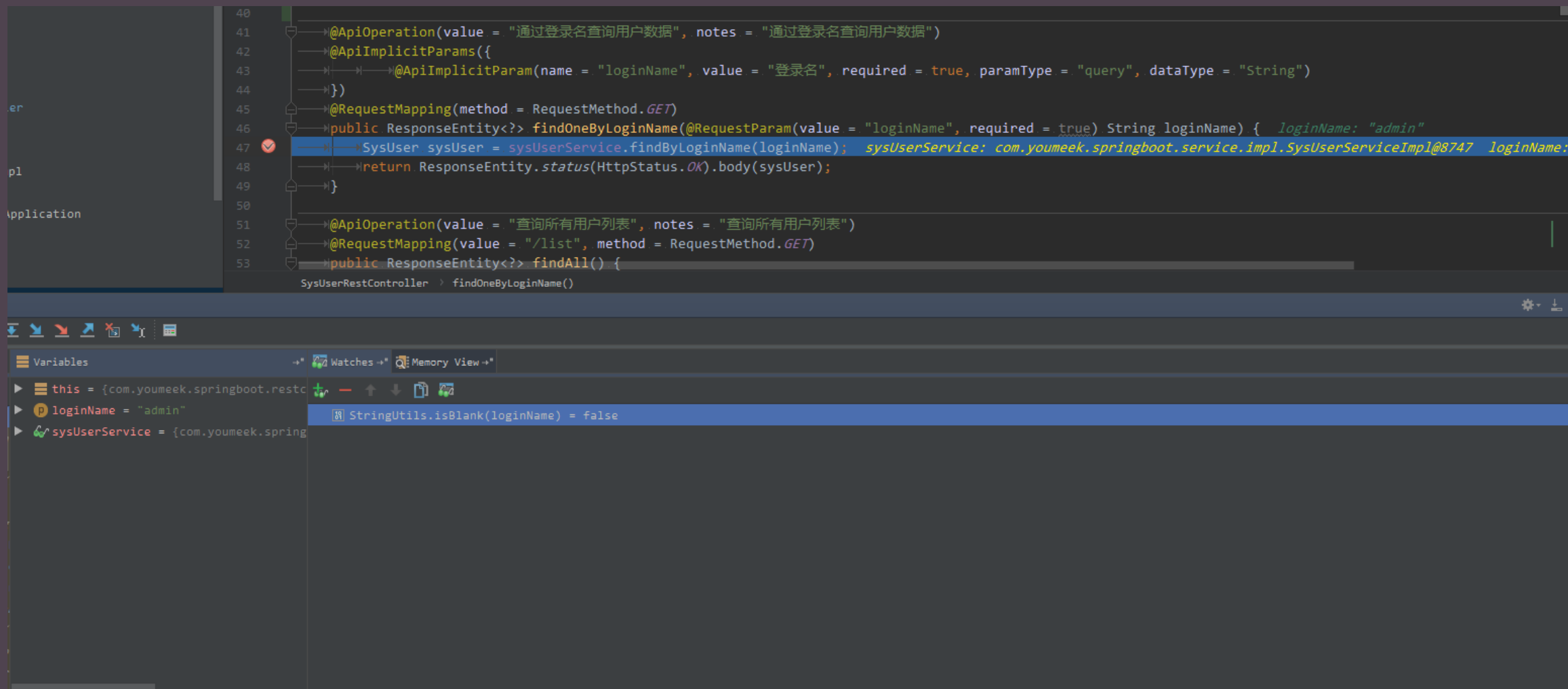
F7	在 Debug 模式下，进入下一步，如果当前行断点是一个方法，则进入当前方法体内，如果该方法体还有方法，则不会进入该内嵌的方法中
F8	在 Debug 模式下，进入下一步，如果当前行断点是一个方法，则不进入当前方法体内
F9	在 Debug 模式下，恢复程序运行，但是如果该断点下面代码还有断点则停在下一个断点上
Alt + F8	在 Debug 的状态下，选中对象，弹出可输入计算表达式调试框，查看该输入内容的调试结果
Ctrl + F8	在 Debug 模式下，设置光标当前行为断点，如果当前已经是断点则去掉断点
Shift + F7	在 Debug 模式下，智能步入。断点所在行上有多个方法调用，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
Drop Frame	这个不是一个快捷键，而是一个 Debug 面板上的按钮。该按钮可以用来退回到当前停住的断点的上一层方法上，可以让过掉的断点重新来过



# Debug 三种方式监控变量：

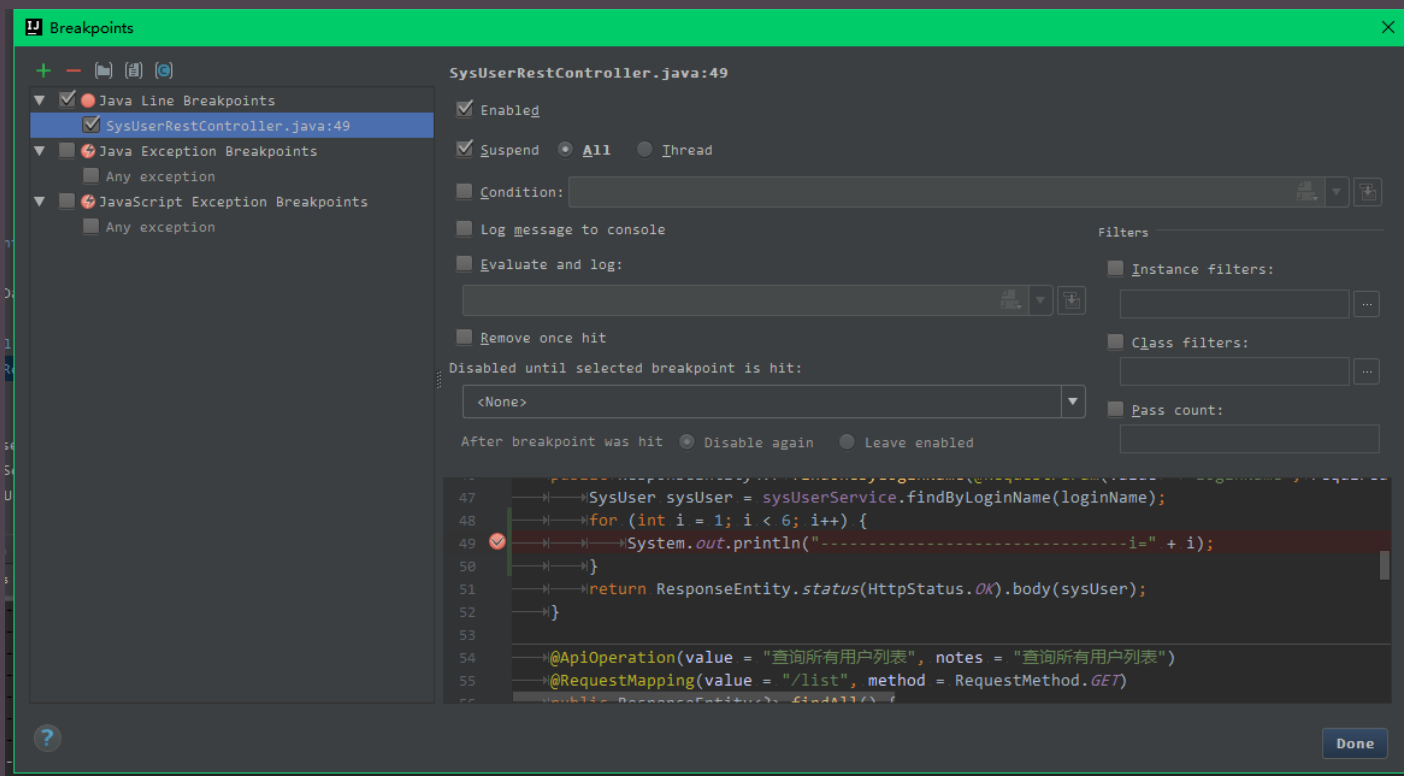
1. 选中对象后，使用快捷键 Alt + F8
2. 选中对象后，拖动对象到 Watches
3. 选中对象后，鼠标悬停在对象上 2 秒左右

# Debug 手动自己输入表达式：

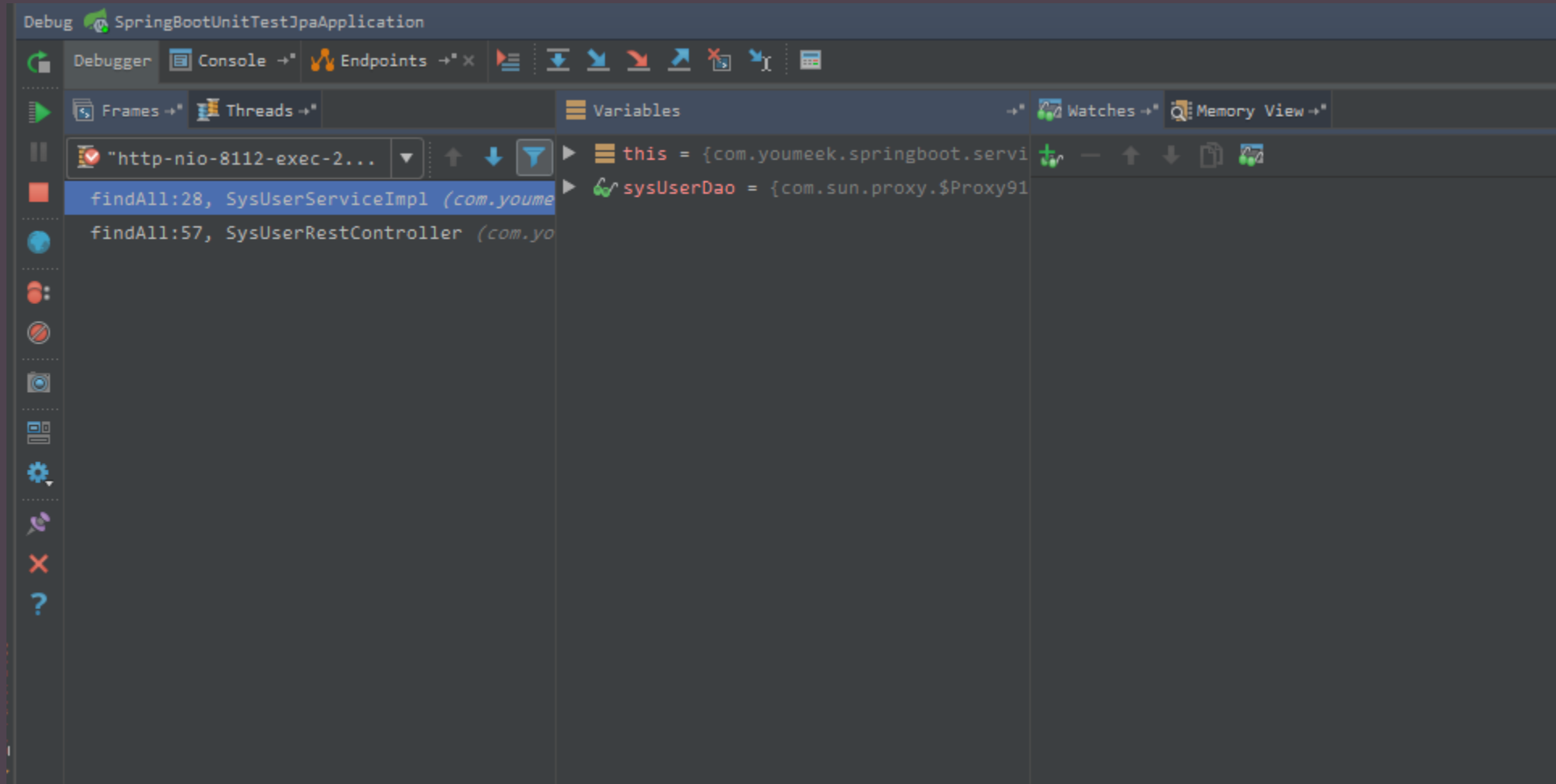


# Debug 条件断点的使用：

1. Condition 满足条件：loginName.equals("admin")
2. Pass count 满足条件：for 循环 i == 4 的时候触发
3. Remove once hit，只运行一次的断点
4. Disable until selected breakpoint is hit，该断点的触发取决于其他断点的触发



# Debug 堆栈的查看：



# IntelliJ IDEA 实时代码模板使用场景

# 实时代码模板的介绍：

New 一个对象演示

# 自带变量参数介绍：

\$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，比如：

```
$VAR1$ $VAR2$ = new $VAR1$();  
$END$
```

# 实时代码模板的设置：

新建 resp 模板（限定范围 Java）：

```
return ResponseEntity.status(HttpStatus.OK).body(sysUser);
```



# 变量参数和函数的介绍：

Item	Description
annotated("annotation qname")	Creates a symbol of type with an annotation that resides at the specified location. For an example, see Live Templates in the <b>iterations</b> group.
arrayVariable()	Suggests all array variables applicable in the current scope. For an example, see Live Templates in the <b>iterations</b> group.
anonymousSuper()	Suggests a supertype for a Kotlin object expression.
camelCase(String)	Returns the string passed as a parameter, converted to camel case. For example, my-text-file/my text file/my_text_file will be converted to myTextFile.
capitalize(String)	Capitalizes the first letter of the name passed as a parameter.
capitalizeAndUnderscore(sCamelCaseName)	Capitalizes the all letters of a CamelCase name passed as a parameter, and inserts an underscore between the parts. For example, if the string passed as a parameter is FooBar, then the function returns FOO_BAR.
castToLeftSideType()	Casts the right-side expression to the left-side expression type. It is used in the <b>iterations</b> group to have a single template for generating both raw-type and Generics Collections.
className(sClassName)	Returns the name of the current class (the class where the template is expanded).
classNameComplete()	This expression substitutes for the <a href="#">class name completion</a> at the variable position.
clipboard()	Returns the contents of the system clipboard.
snakeCase(String)	Returns CamelCase string out of snake_case string. For example, if the string passed as a parameter is foo_bar, then the function returns fooBar.
complete()	This expression substitutes for the code completion invocation at the variable position.
completeSmart()	This expression substitutes for the smart type completion invocation at the variable position.
componentTypeOf (<array variable or array type>)	Returns component type of an array. For example, see the <a href="#">Live Templates</a> in the <b>iterations</b> group in the <b>other</b> group.
currentPackage()	Returns the current package name.

官网更多说明：<https://www.jetbrains.com/help/idea/live-template-variables.html>

# 环绕功能介绍：

演示下面环绕效果：

```
System.out.println("-----" + $SELECTION$);$END$
```

# IntelliJ IDEA 文件代码模板使用场景

# 文件代码模板的介绍：

演示：新建 HTML 和 HTML4 文件差异的原因

# 文件代码模板的设置：

01. IntelliJ IDEA 的文件代码模板是可以使用 Velocity Template Language 进行书写的

02. 新建一个新的 Class 模板，里面带有 Main 方法：

```
public static void main(String[] args) {  
  
}
```

# 文件代码模板预设的变量介绍：

`${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.

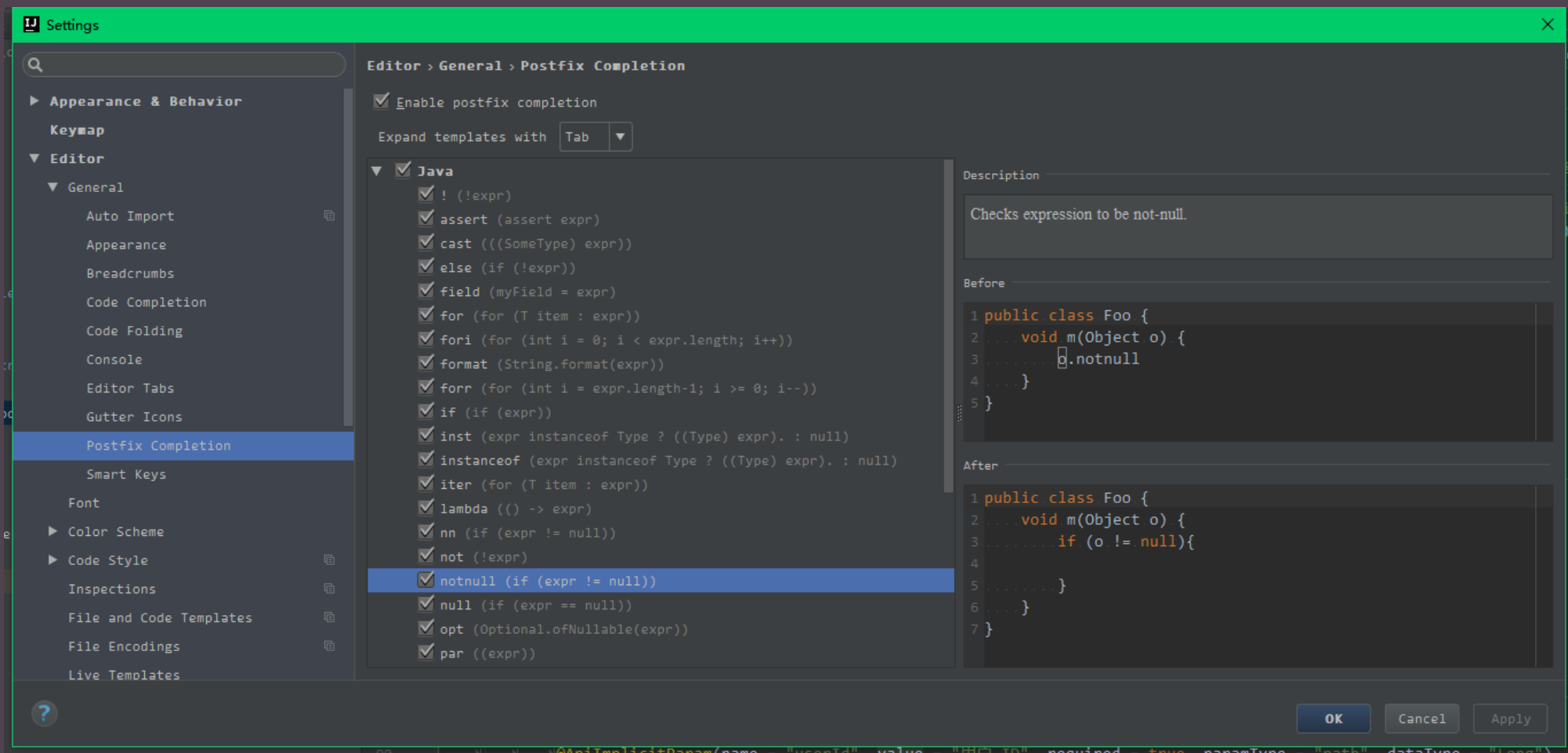
`${MONTH_NAME_FULL}` - full name of a month. Example: January, February, etc.

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

# IntelliJ IDEA Postfix Completion 代码模板介绍

# 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





# IntelliJ IDEA 辅助开发的各类插件推荐

# 介绍：

String Manipulation，常用于驼峰命令和下划线命名之间的切换

Maven Helper，更加直观查看 Maven 依赖关系

GsonFormat，通过 JSON 字符串，生成 POJO 类

Properties to YAML Converter，把 Properties 格式转换成 YAML

Grep Console，自定义控制台输出样式和声音

# IntelliJ IDEA 提高效率的个性化设置

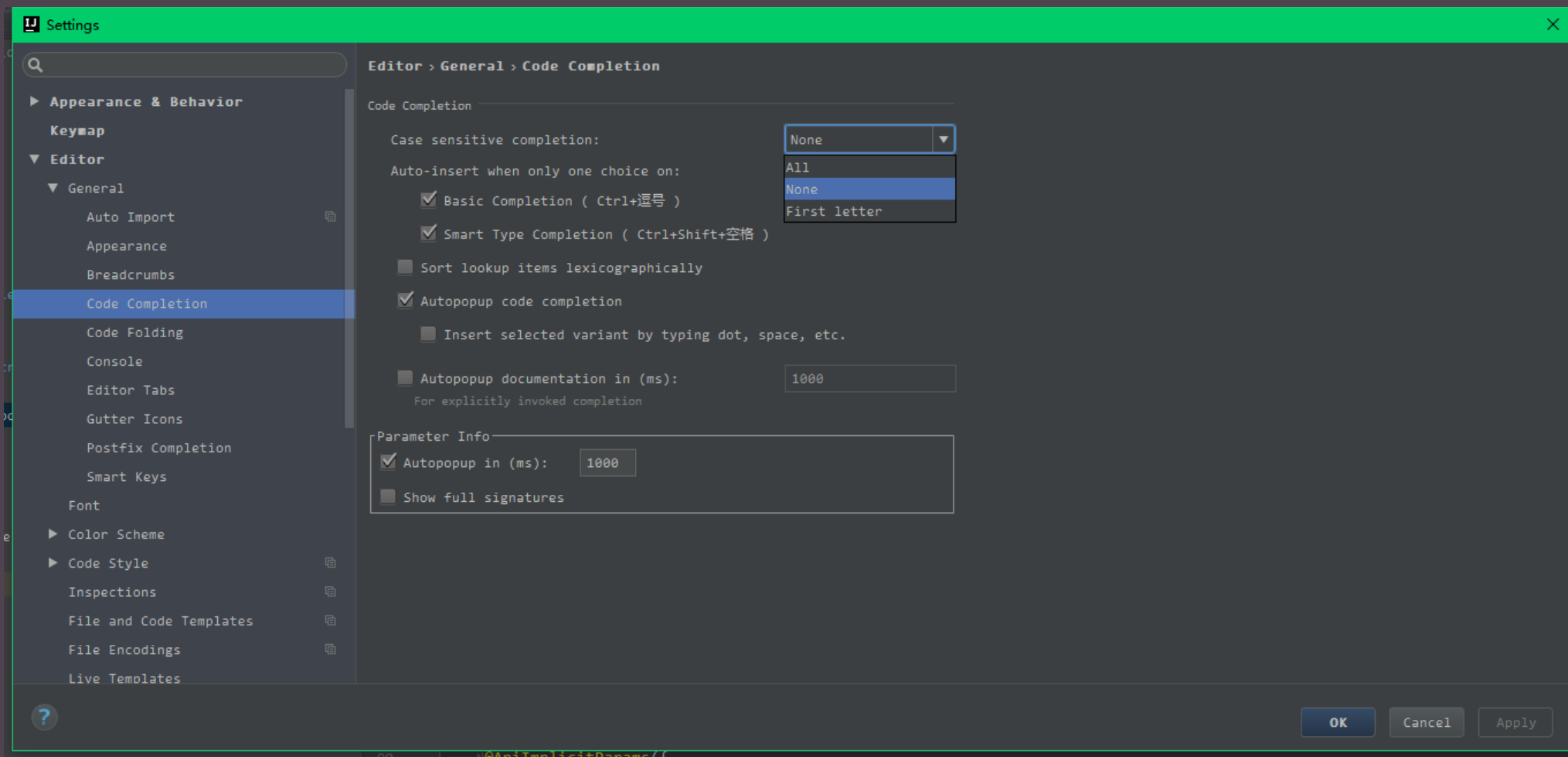
# 必改的快捷键：

代码提示：Main menu | Code | Completion | Basic

垂直/水平分组：Main menu | Window | Editor Tabs | Split Vertically

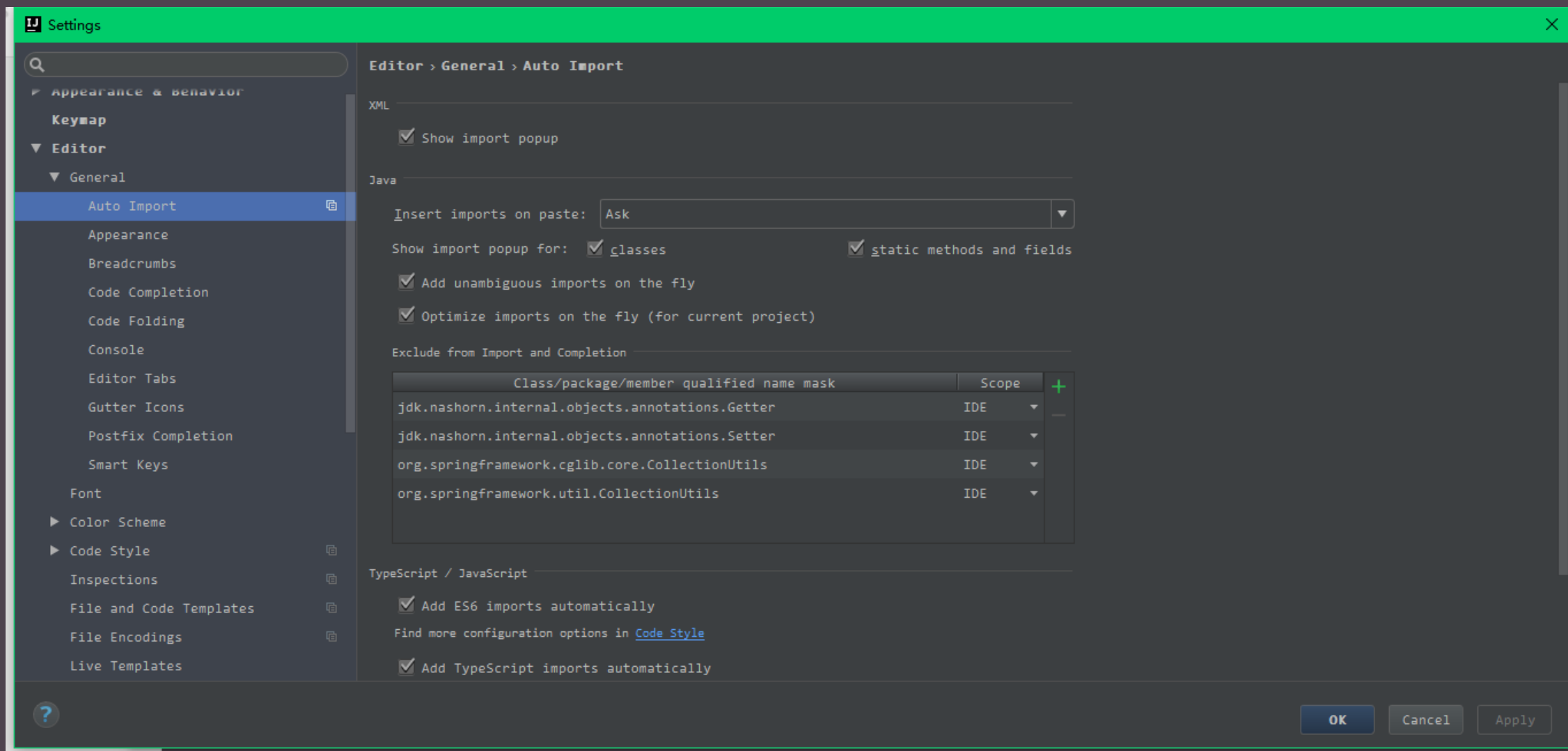
复制行或块：Editor Actions | Duplicate Entire Lines

# 代码补全设置：

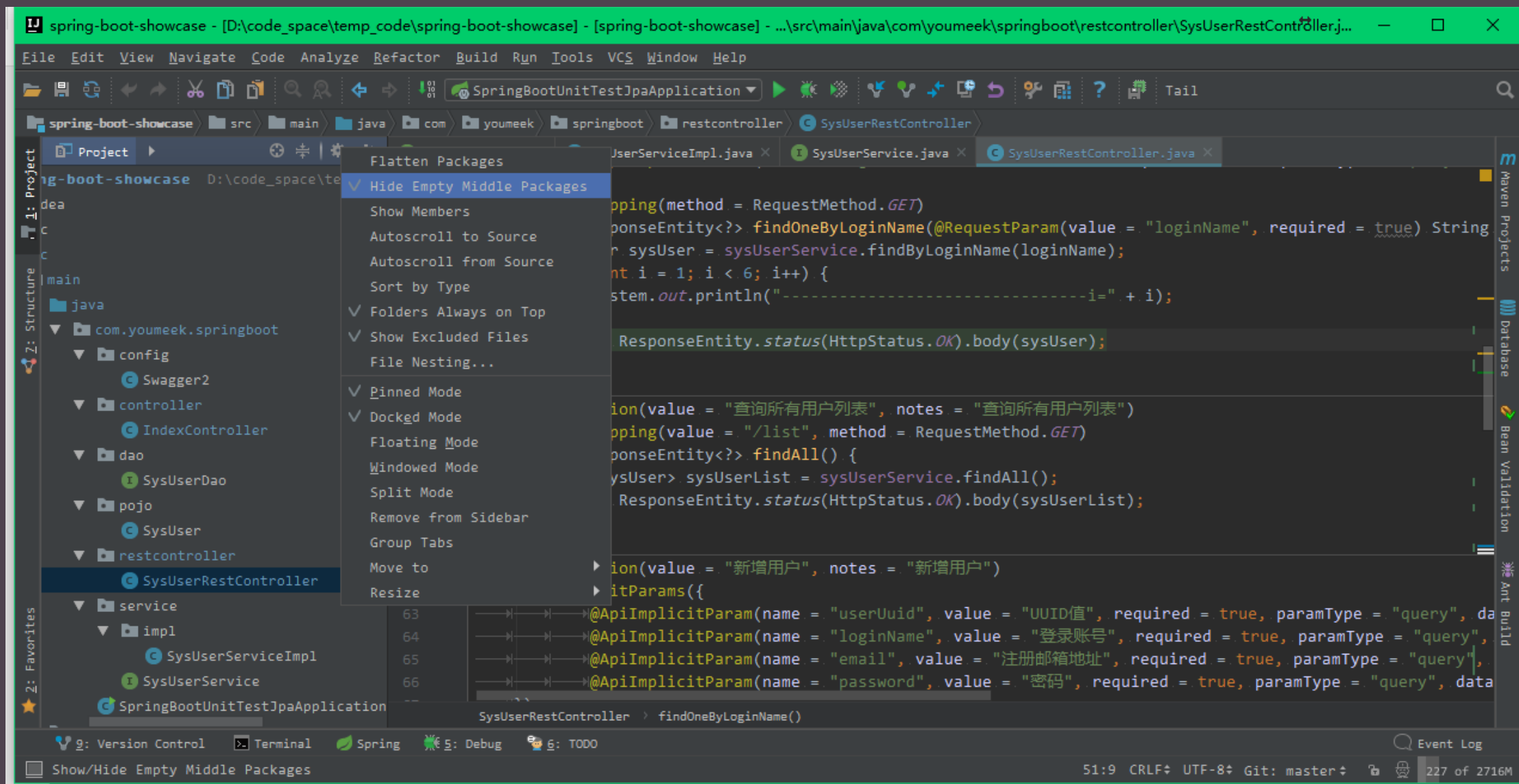


只有 None 才是不区分大小写的，推荐设置

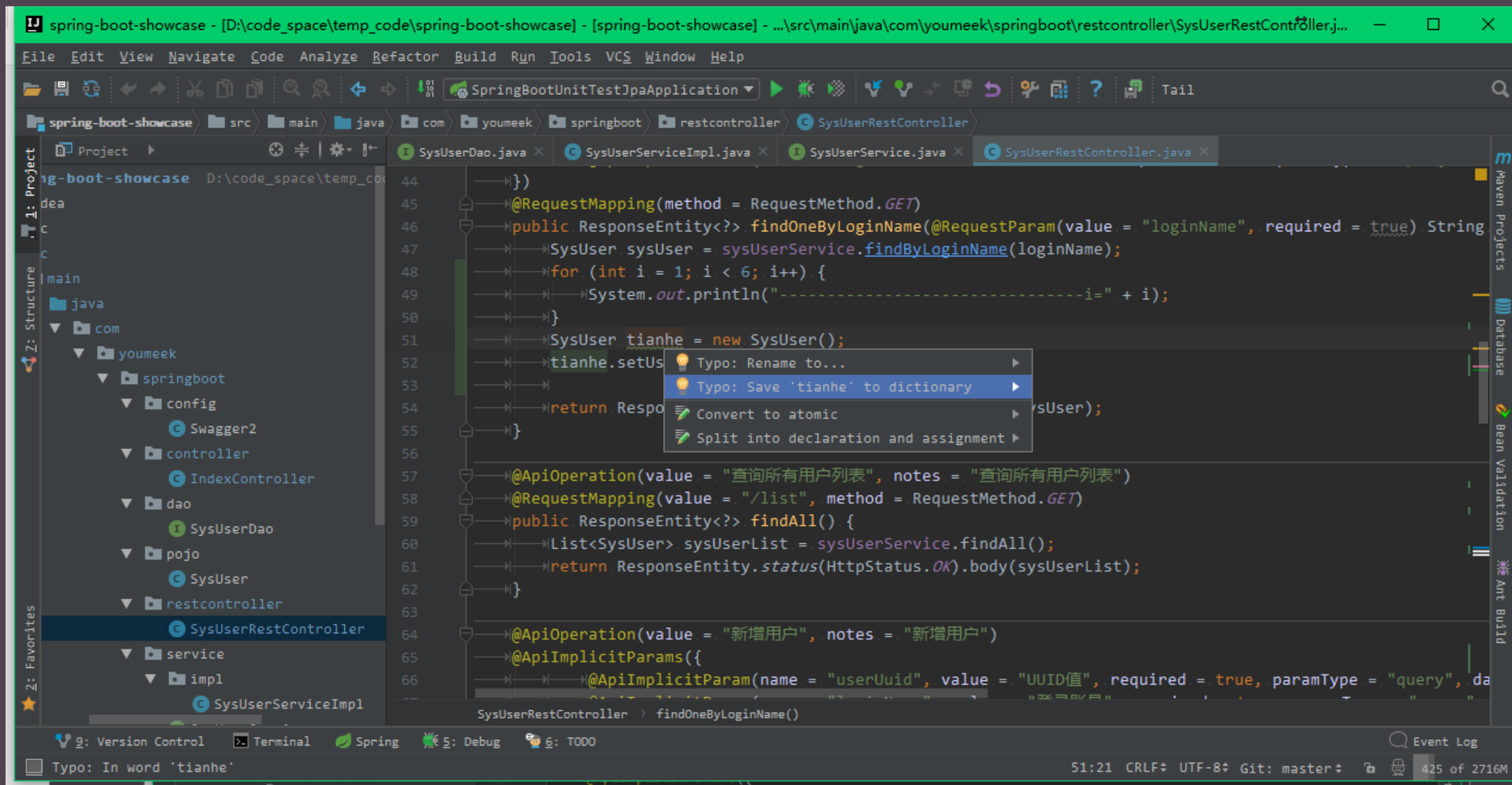
# 自动导包功能：



# 项目树显示结构设置：

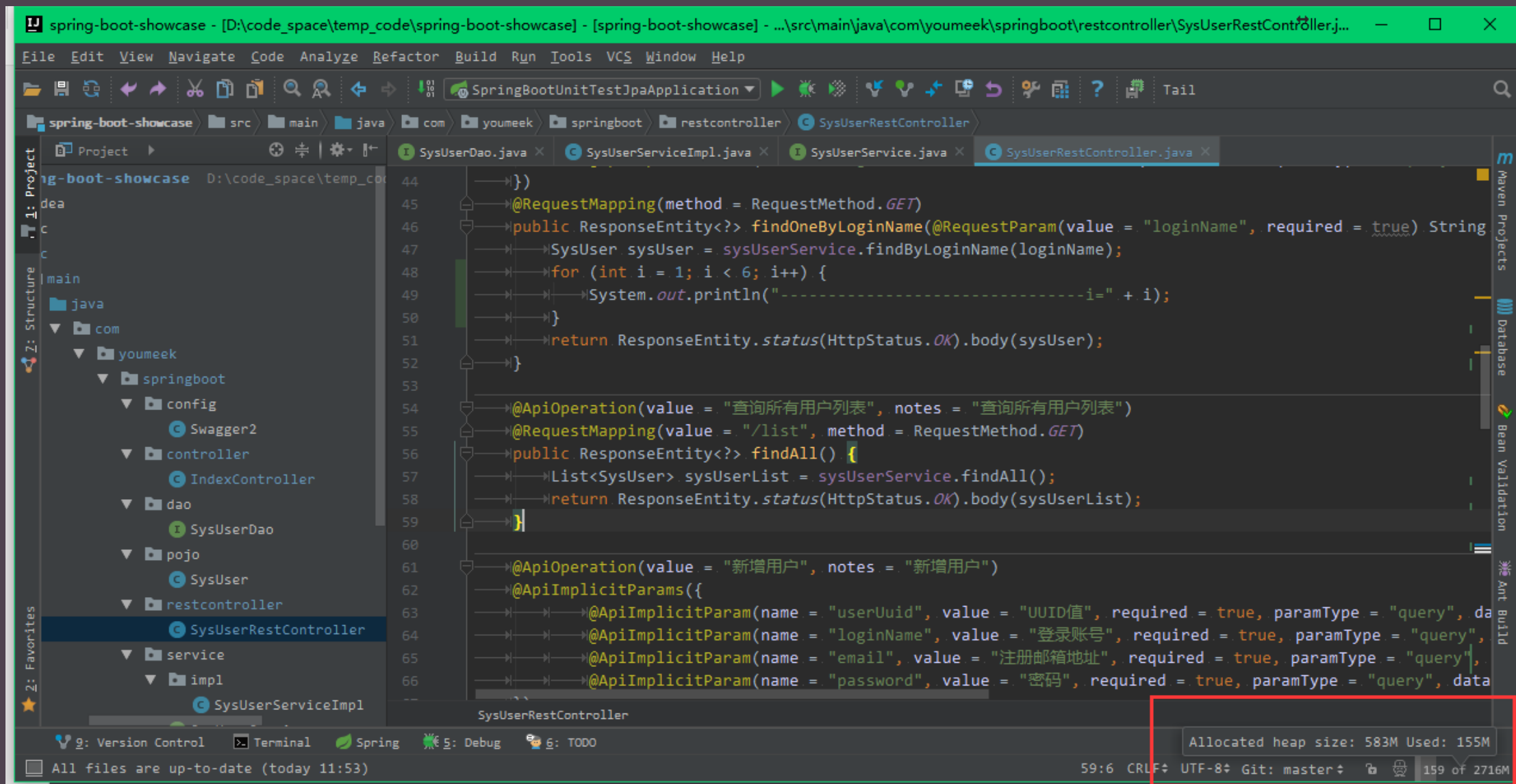


# 拼写检查设置：

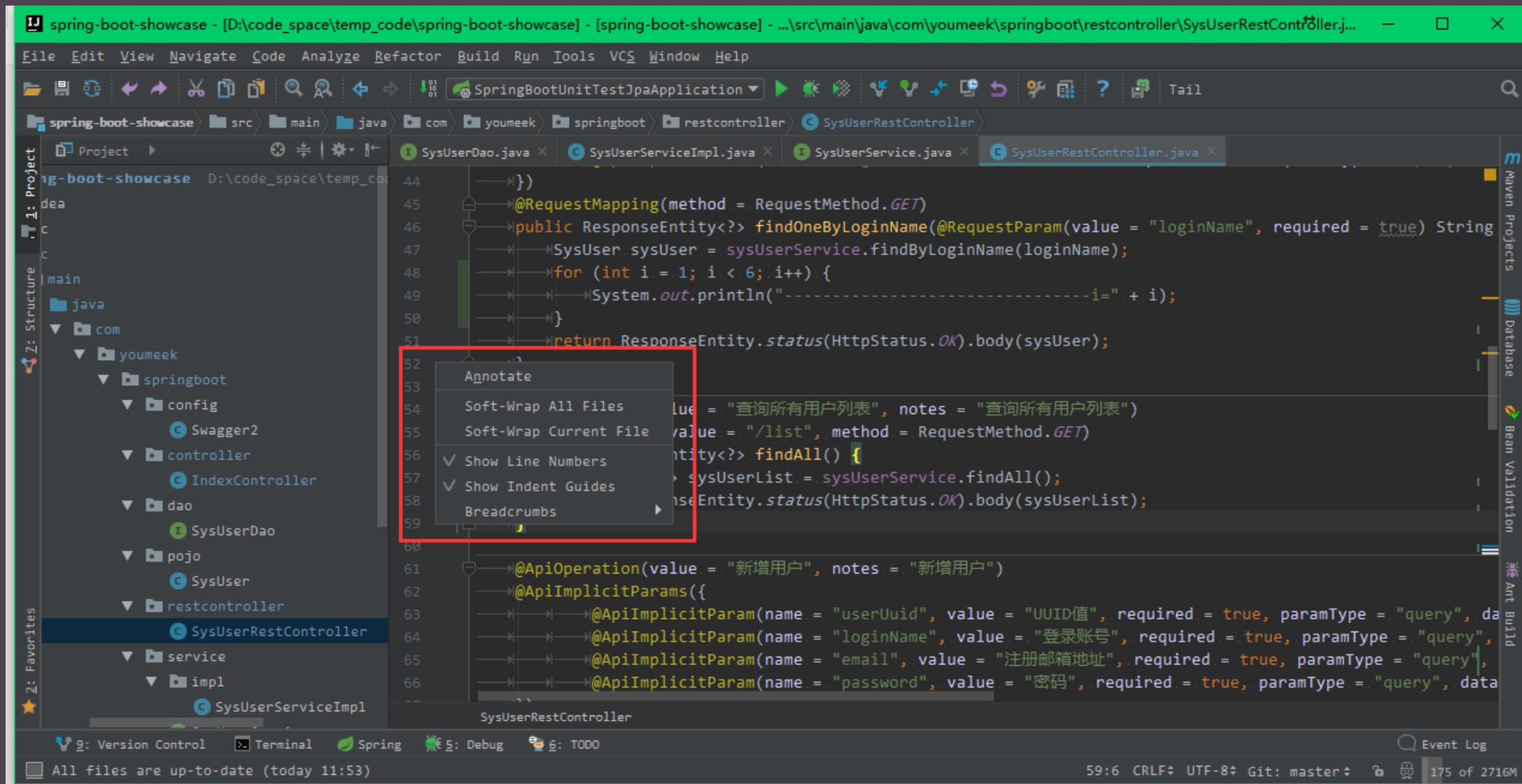




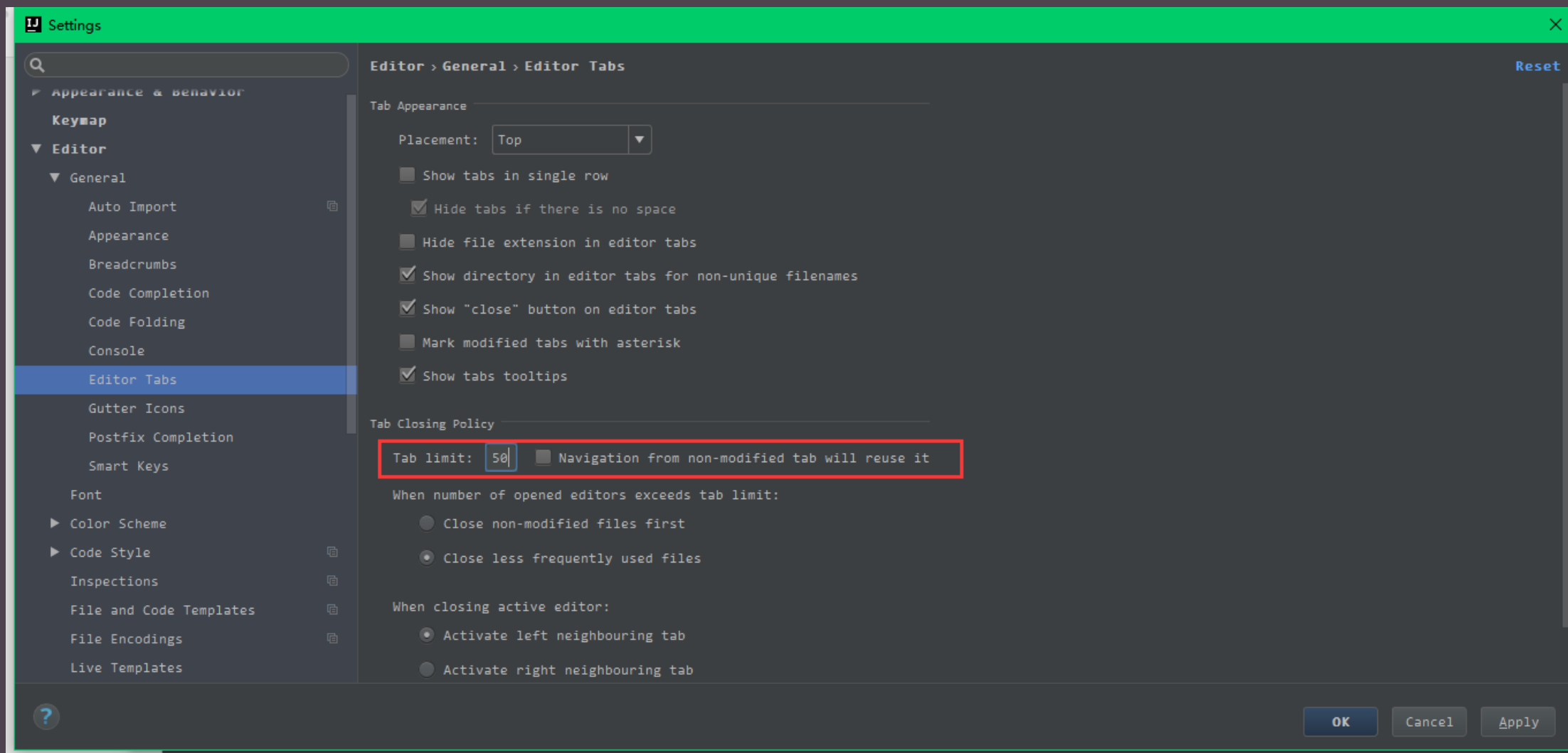
# 内存回收显示：



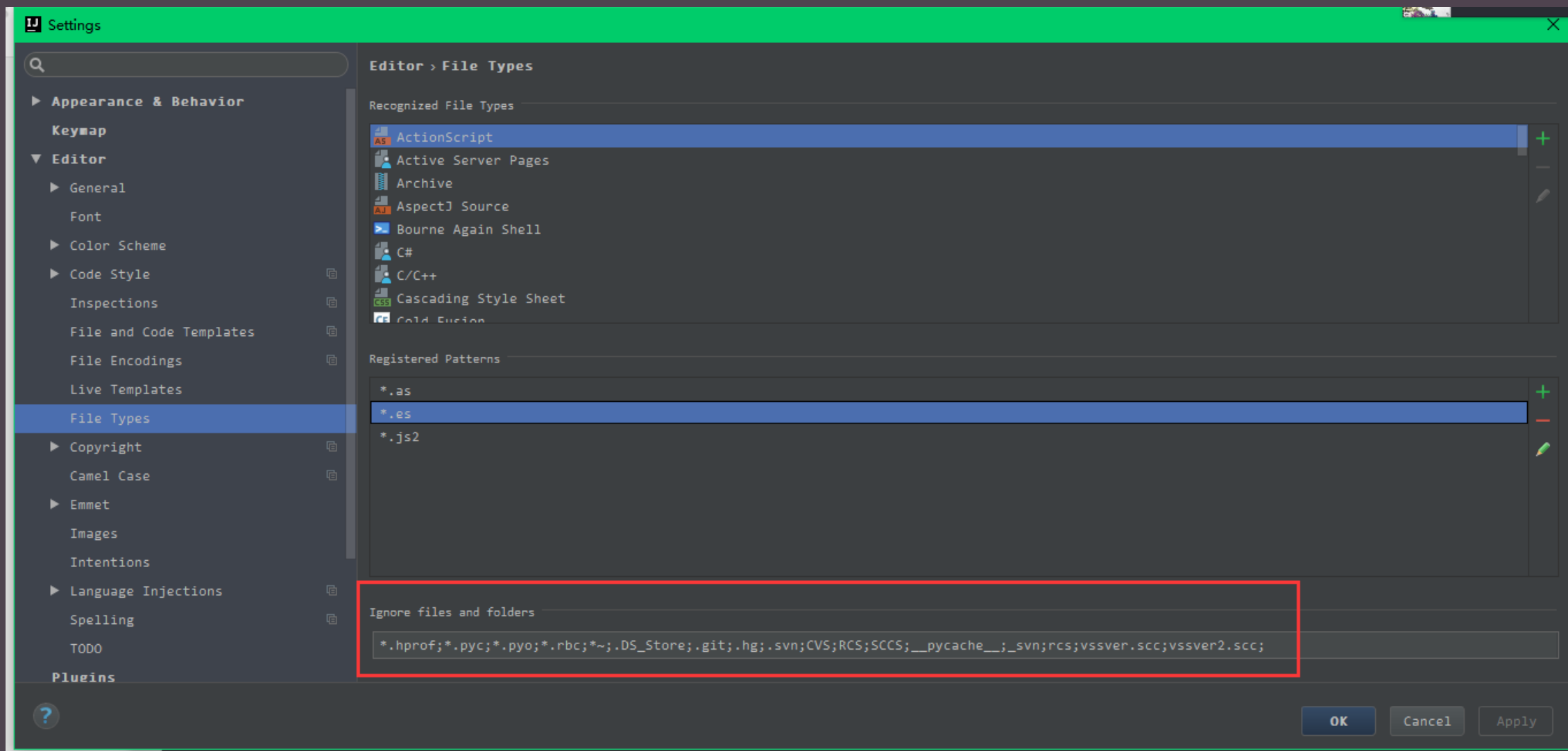
# 软分行使用场景：



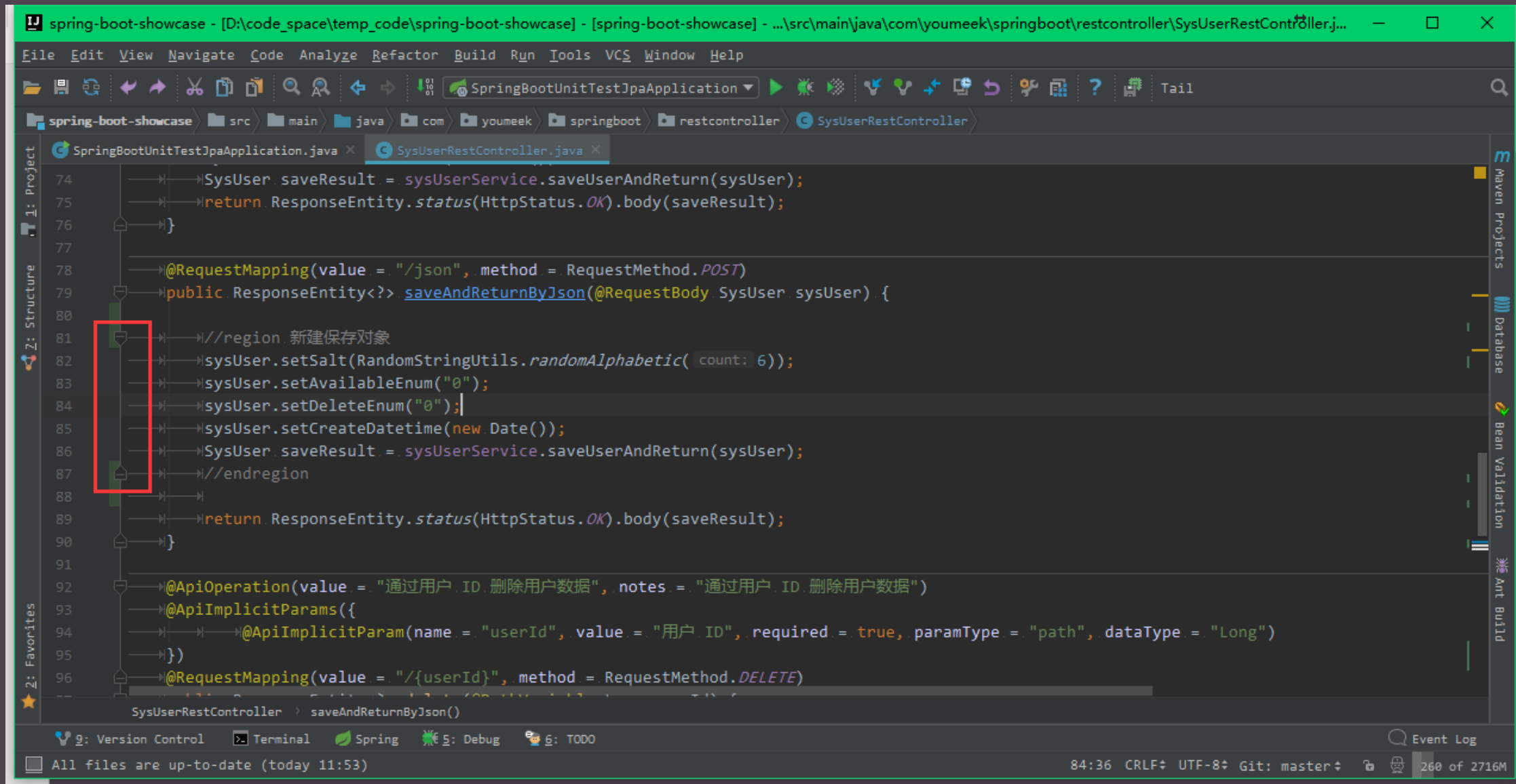
# 最大打开文件数设置：



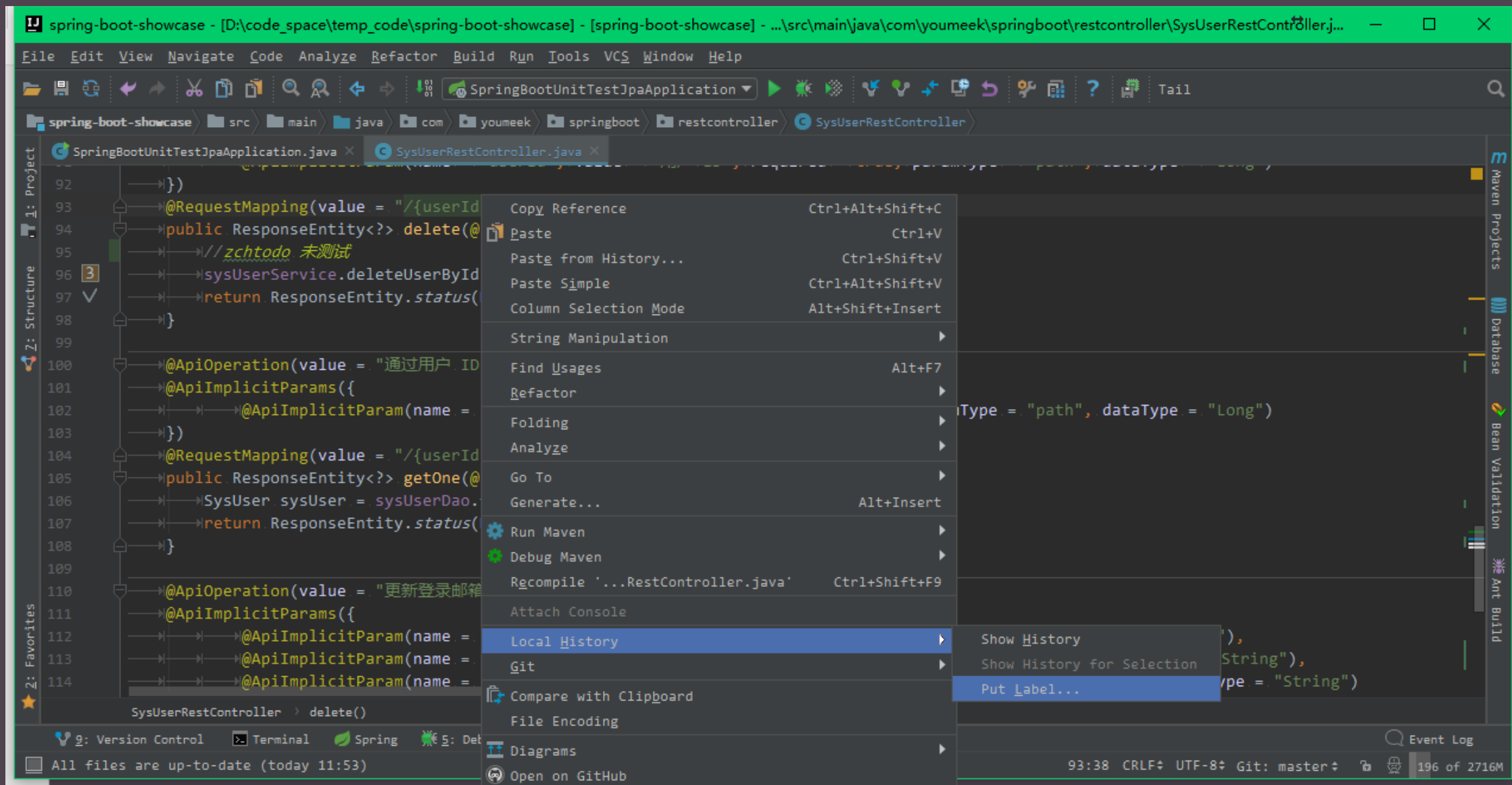
# 隐藏指定后缀文件或是文件夹：



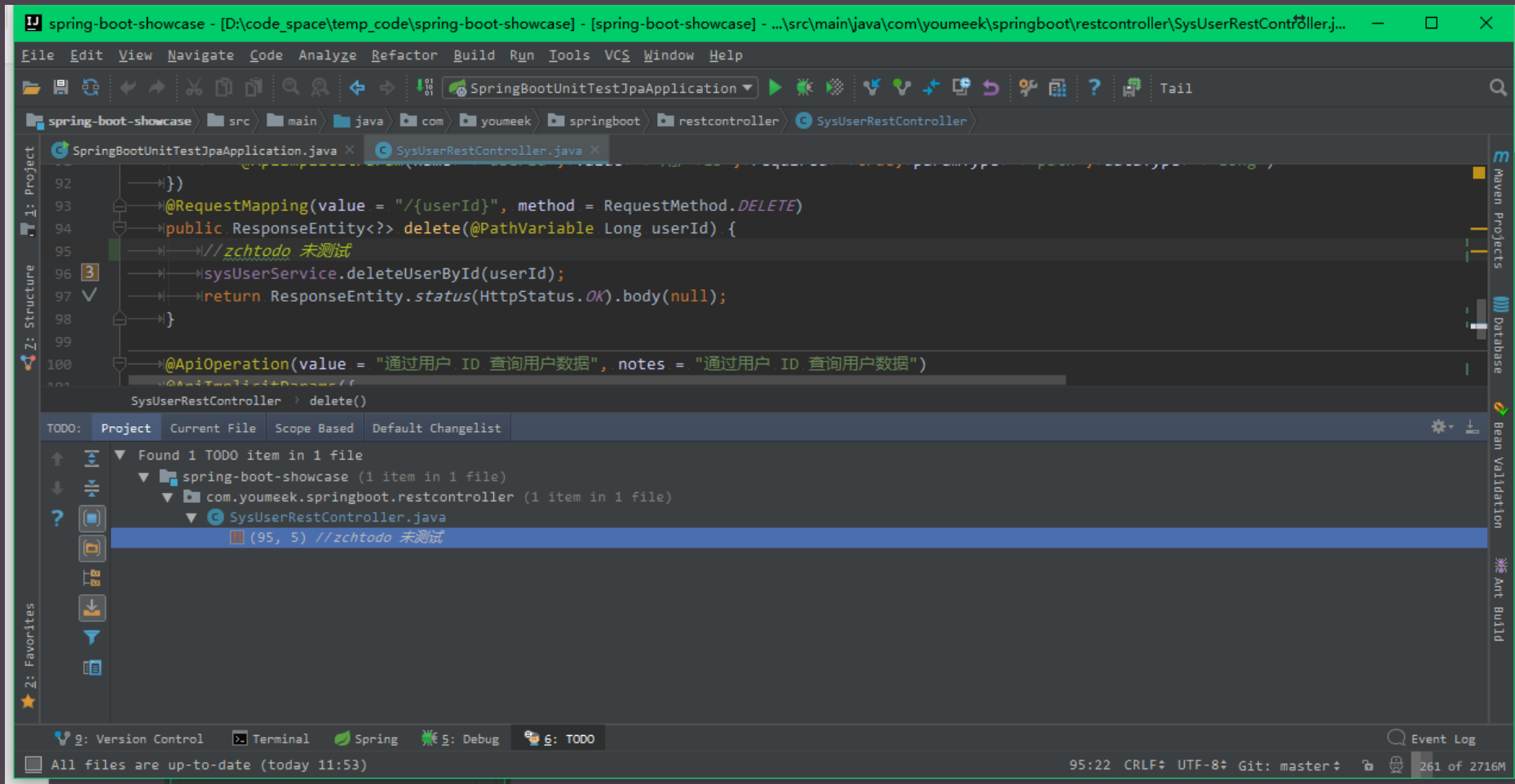
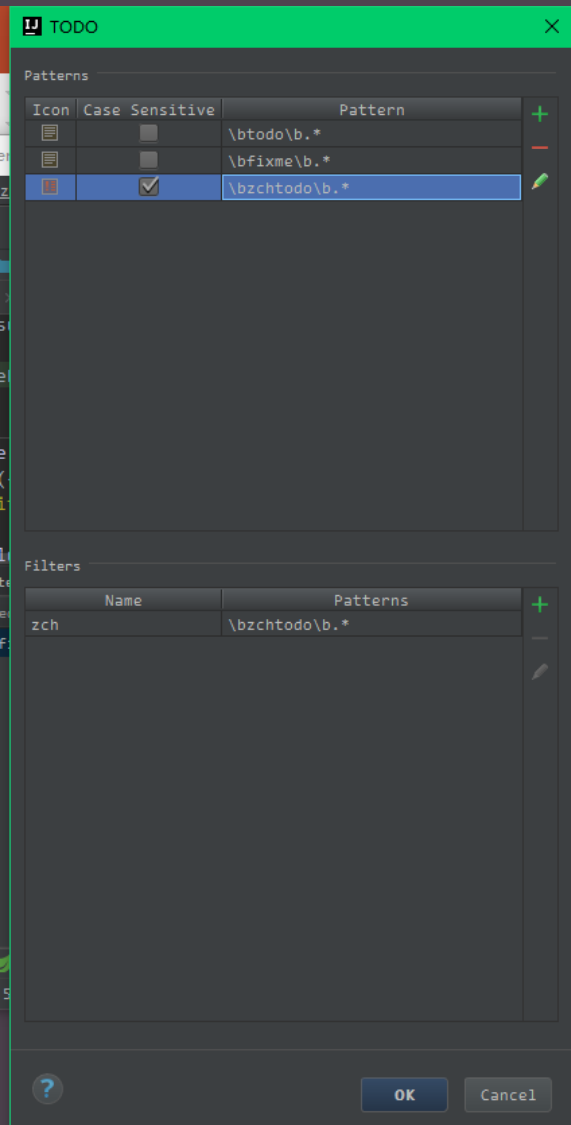
# 自定义折叠代码区域：



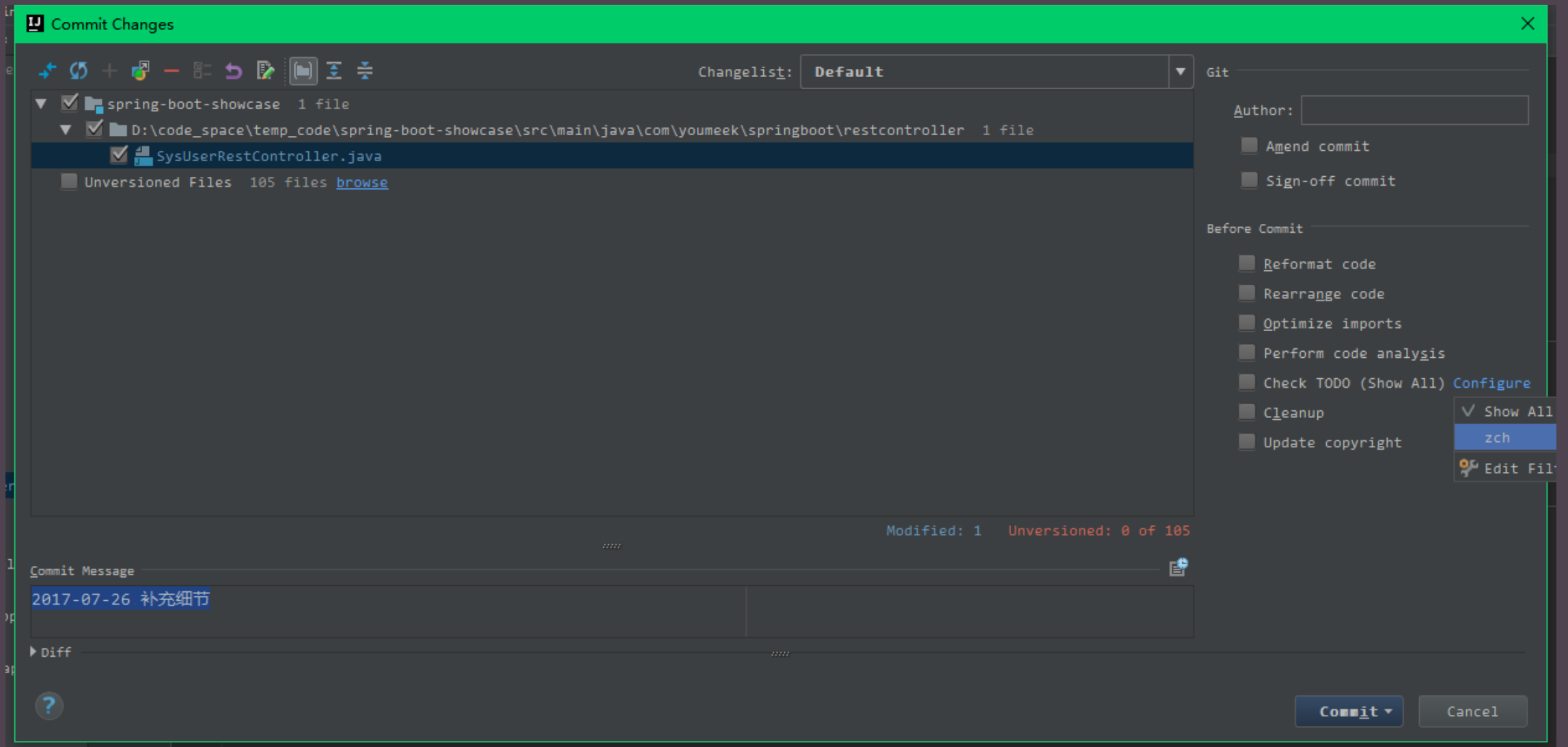
# 本地历史的 label 使用场景：



# 自定义 TODO 的设置（提交代码的 Check TODO）：

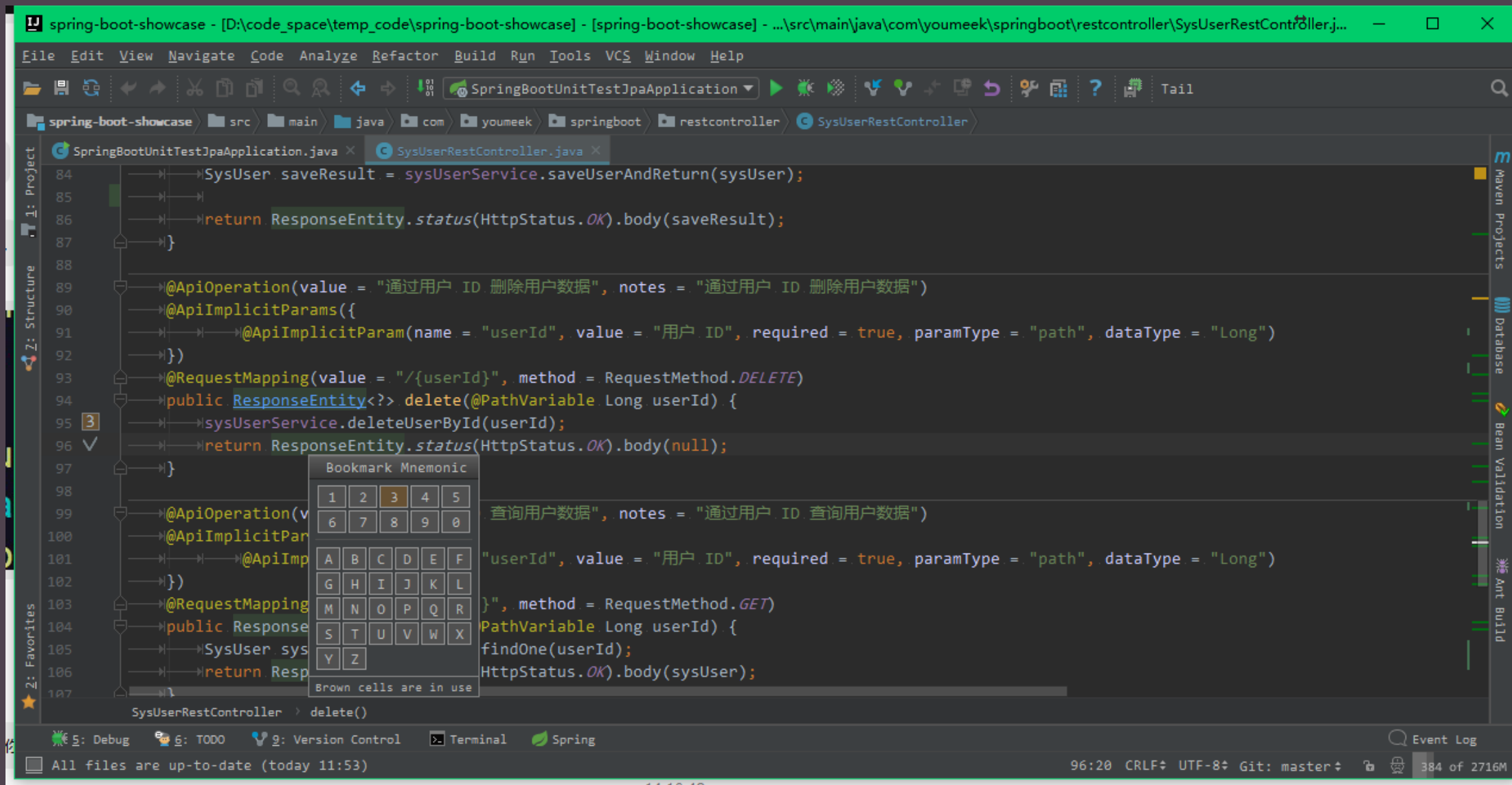


# 自定义 TODO 的设置（提交代码的 Check TODO）：

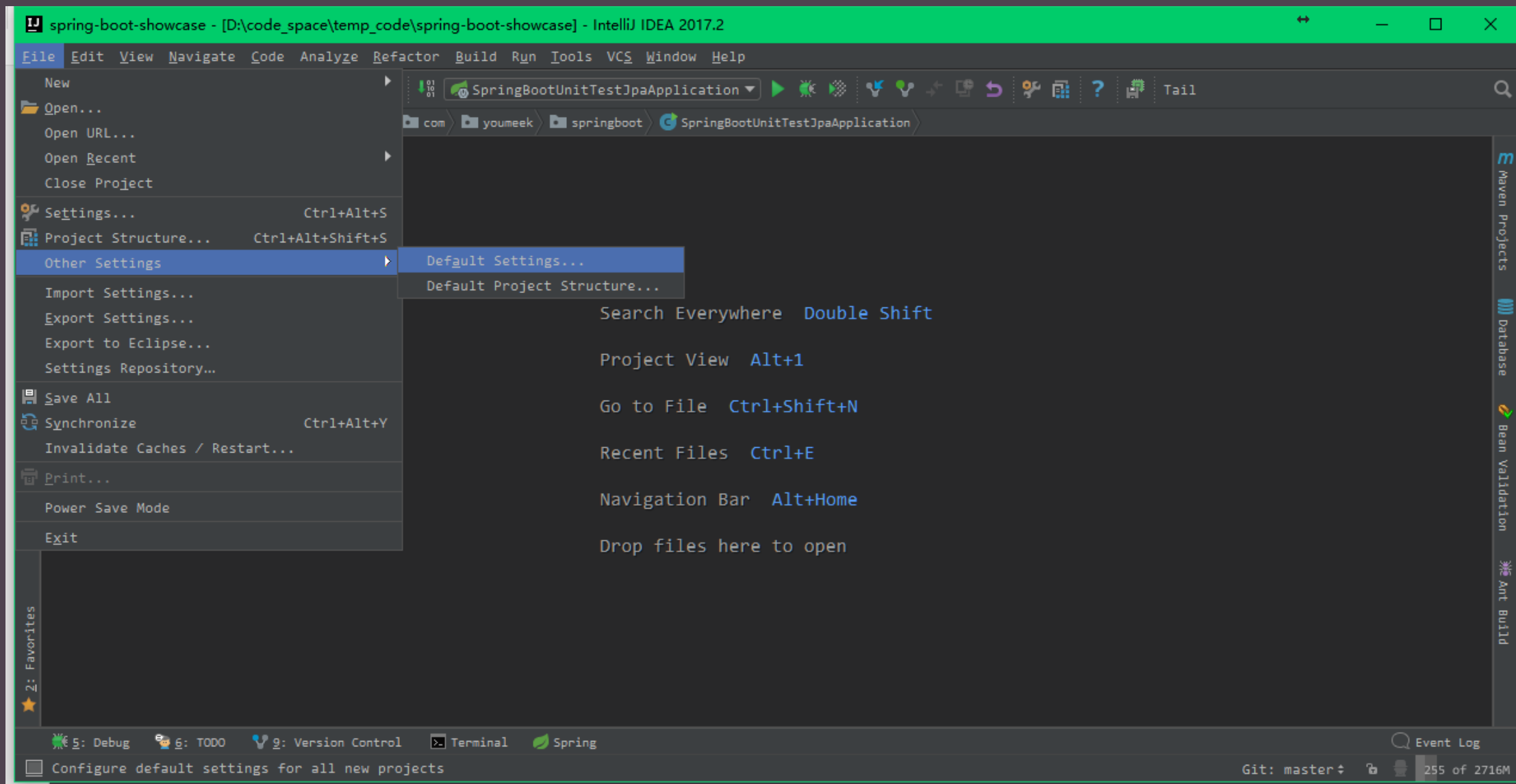




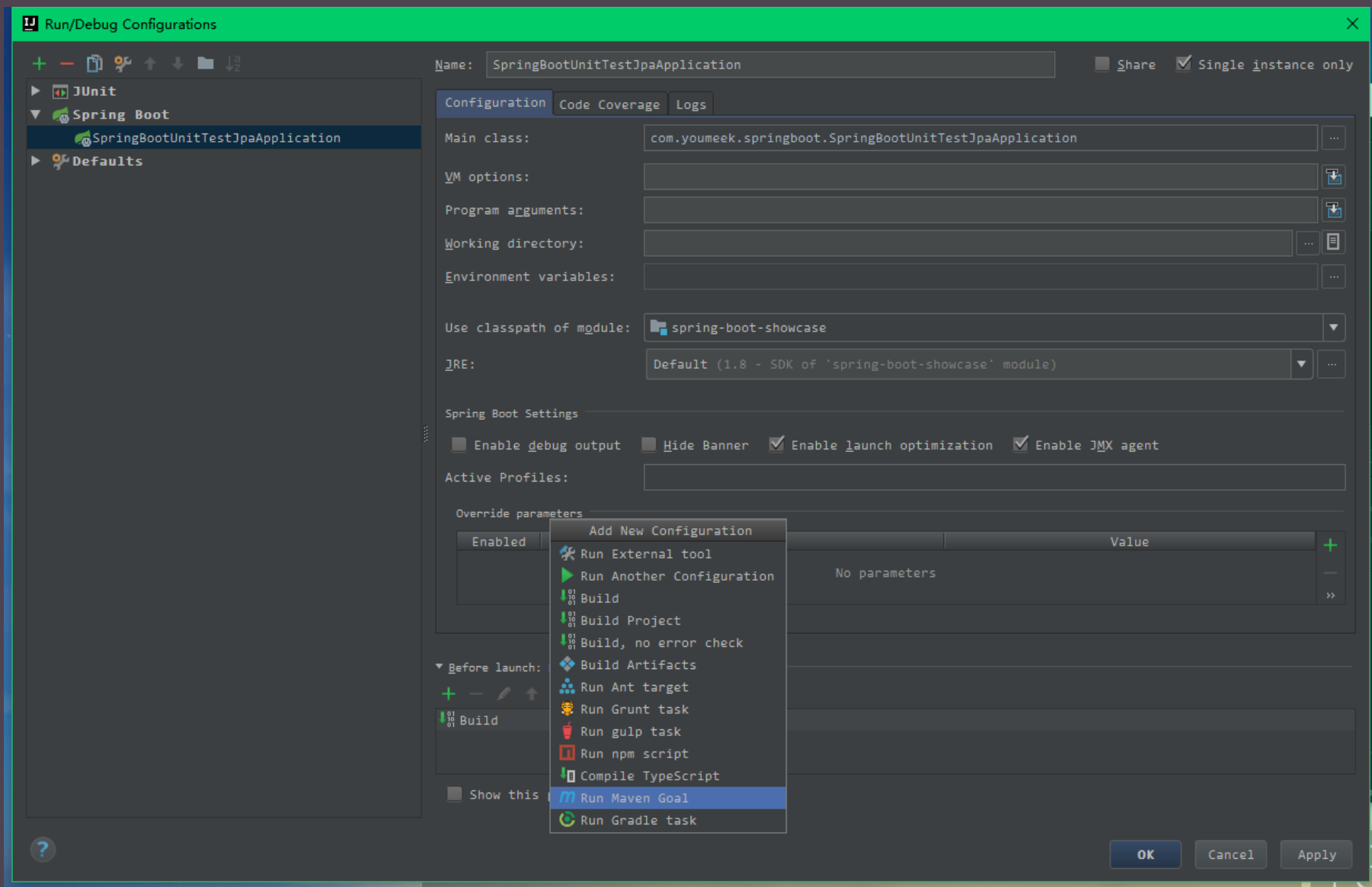
# 设置代码书签/书签定位：



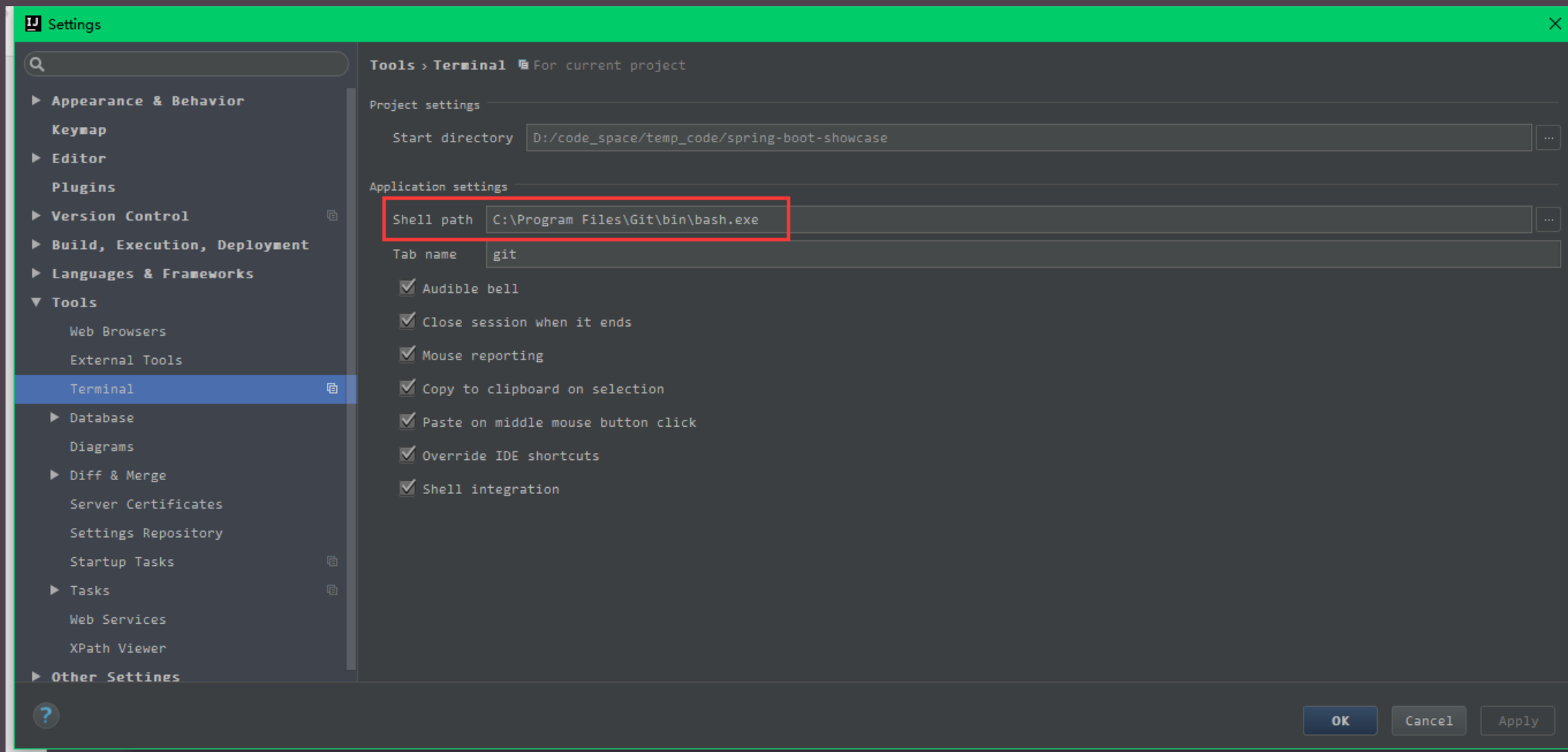
# IntelliJ IDEA 默认新建项目的设置模板修改：



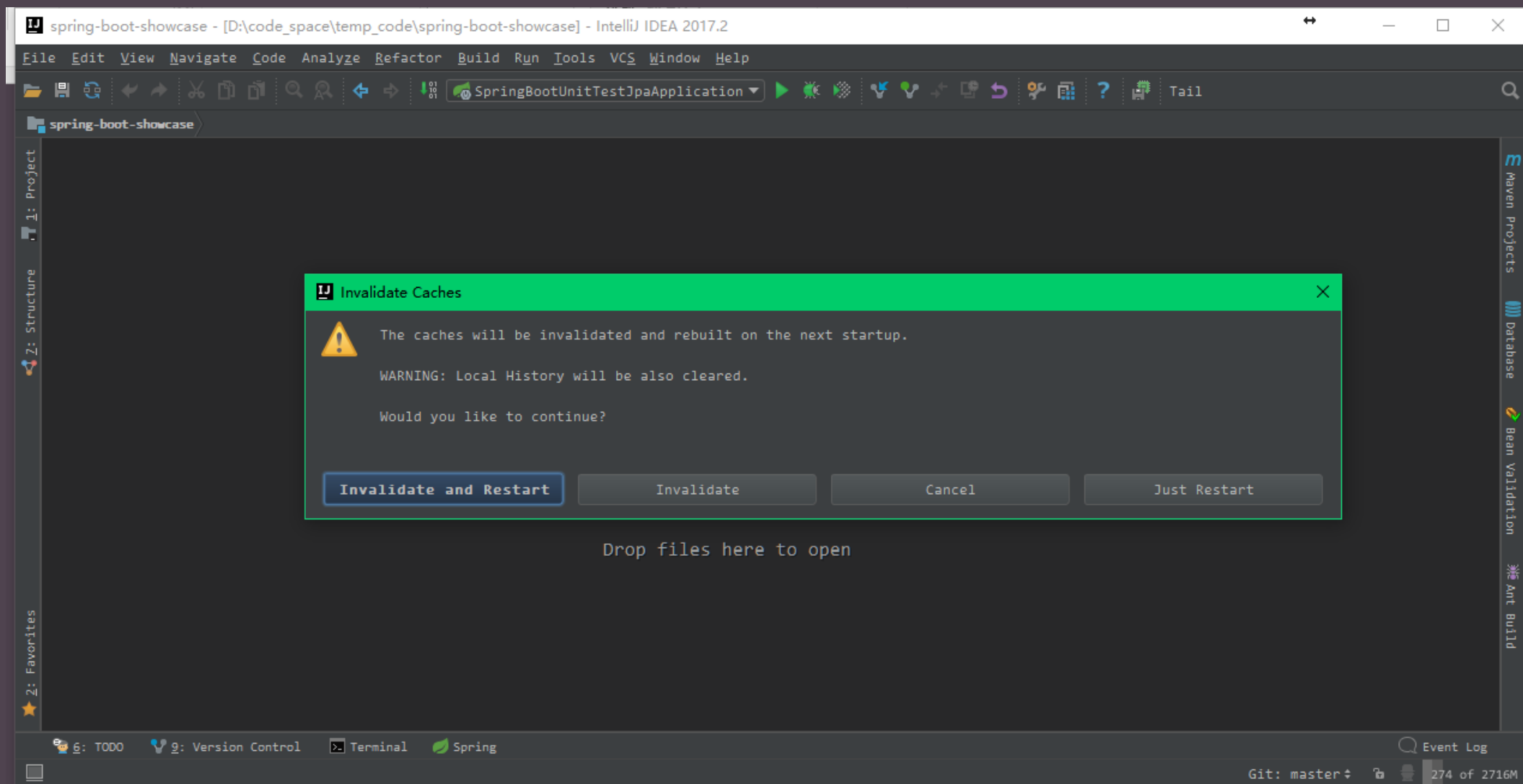
# Before launch 的使用场景：



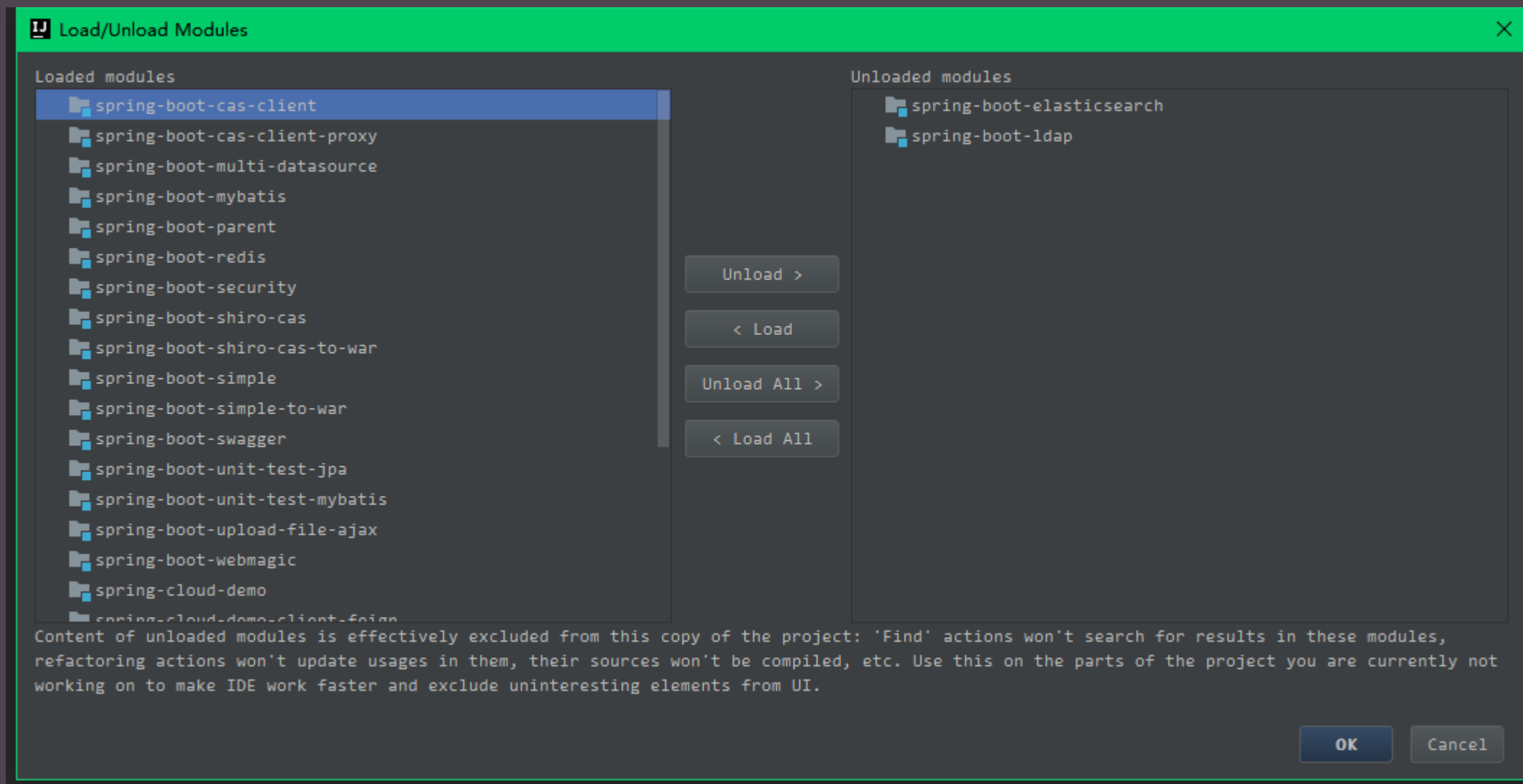
# 命令行终端的修改：



# 索引异常处理方式：

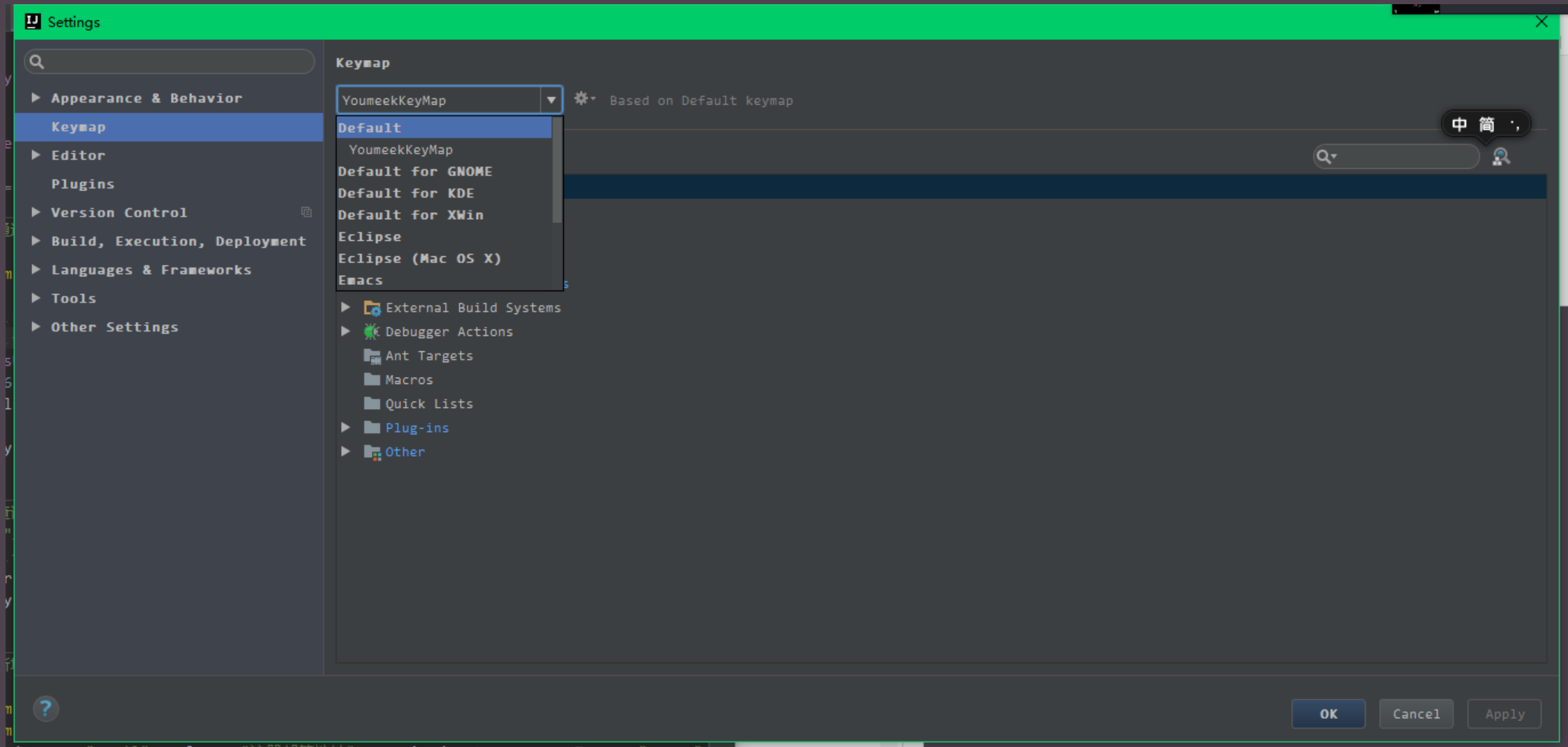


# Load/Unload Modules 新特性：



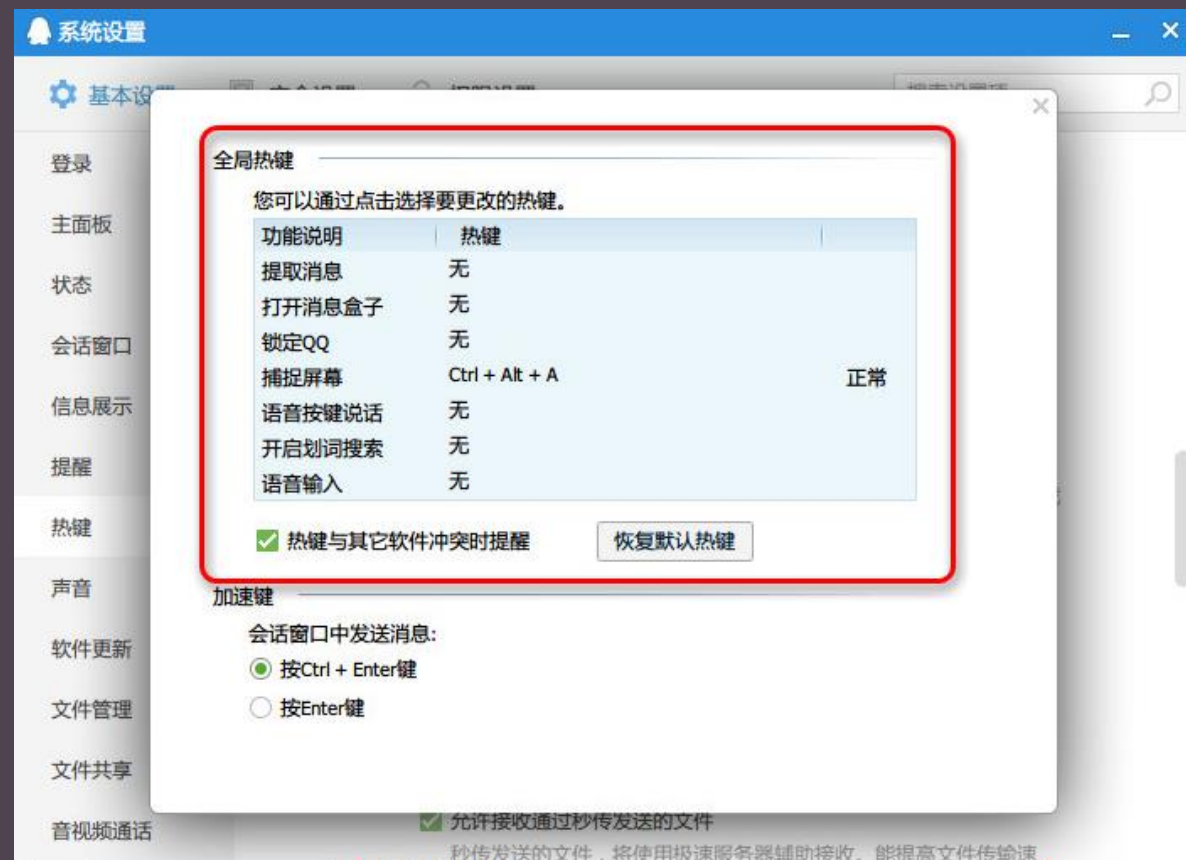
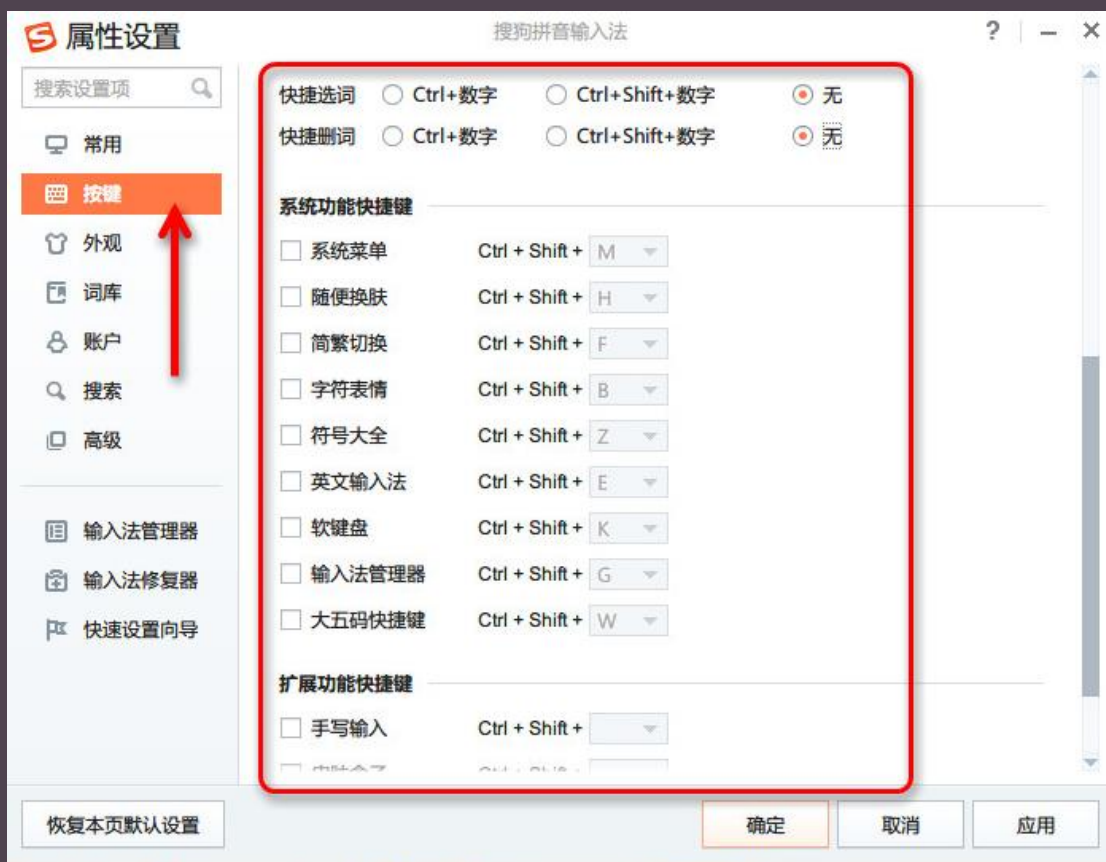
# IntelliJ IDEA 快捷键常见问题

# 不推荐使用：Eclipse 方案的快捷键：

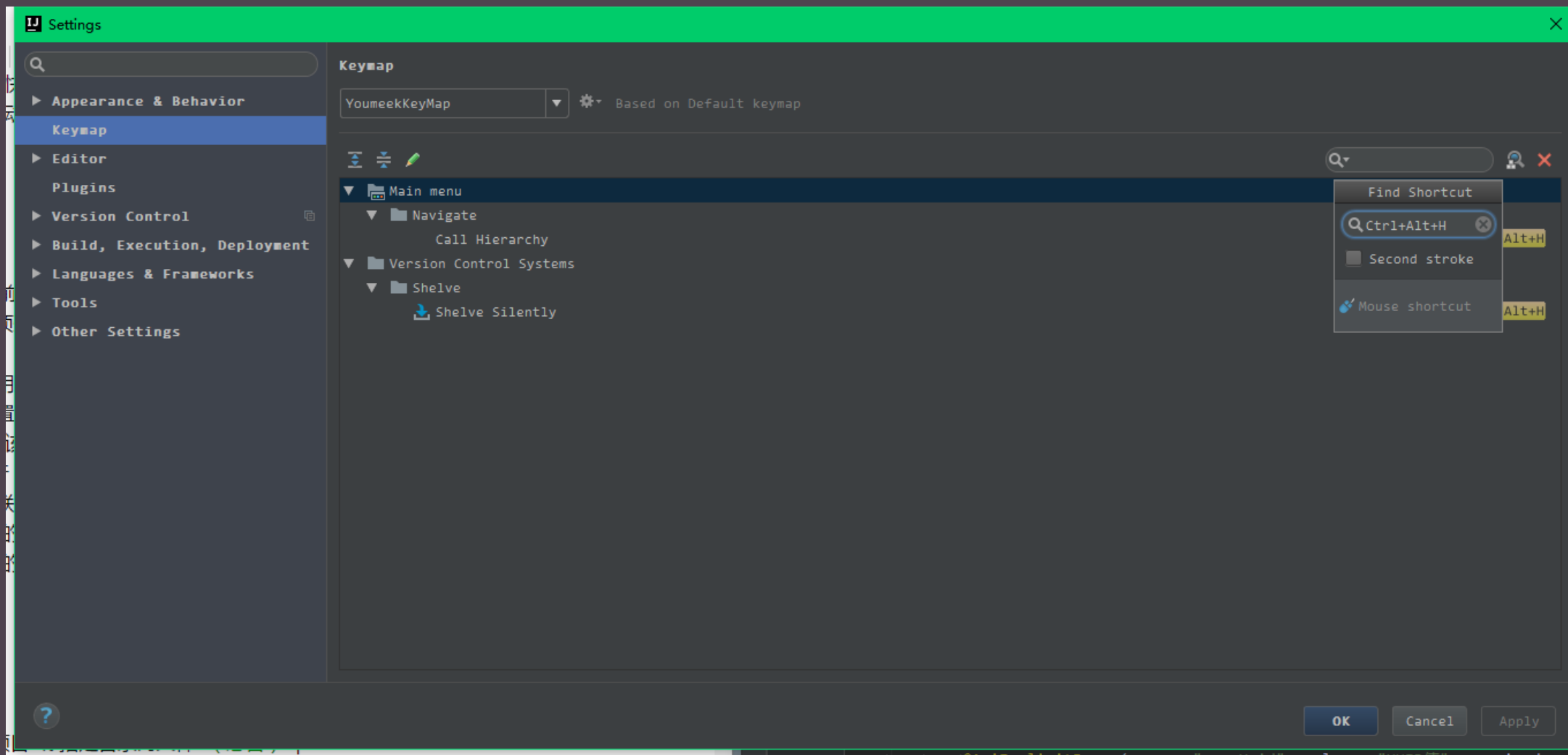




# 常见快捷键冲突场景：



# 验证快捷键是否被占用：



# IntelliJ IDEA 必备快捷键使用场景

# Ctrl :

快捷键	介绍
Ctrl + F	在当前文件进行文本查找（ 设置选项使用场景 ）
Ctrl + R	在当前文件进行文本替换（ 设置选项使用场景 ）
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
Ctrl + E	显示 <b>最近打开</b> 的文件记录列表
Ctrl + N	根据输入的 <b>类名</b> 查找类文件
Ctrl + J	插入自定义动态代码模板
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 + 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 + 1,2,3...9	定位到对应数值的书签位置

# Alt :

快捷键	介绍
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
Alt + F7	查找光标所在的方法 / 变量 / 类被调用的地方
Alt + Insert	代码自动生成，如生成对象的 set / get 方法，构造函数，toString() 等

# Shift :

快捷键	介绍
Shift + F2	跳转到上一个高亮错误 或 警告位置
Shift + F4 ( 或者拖动 Tab )	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 + F11	弹出书签显示层，查看所有书签标记情况
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	在打开的文件名 Tab 上按此快捷键，可以关闭当前打开文件
Shift + 滚轮前后滚动	当前文件的横向滚动轴滚动

# Ctrl + Alt :

快捷键	介绍
Ctrl + Alt + 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	对选中的代码弹出环绕选项弹出层
Ctrl + Alt + B ( 或者 Ctrl + Alt + 鼠标单击 )	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 + Enter	光标所在行上空出一行，光标定位到新行
Ctrl + Alt + Home	弹出跟当前文件有关联的文件弹出层
Ctrl + Alt + 左方向键	退回到上一个操作的地方
Ctrl + Alt + 右方向键	前进到上一个操作的地方

# Ctrl + Shift :

快捷键	介绍
Ctrl + Shift + J	自动将下一行合并到当前行末尾
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
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
Ctrl + Shift + U	对选中的代码进行大 / 小写轮流转换
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
Ctrl + Shift + C	复制当前文件磁盘路径到剪贴板
Ctrl + Shift + E	显示 <b>最近修改</b> 的文件列表的弹出层
Ctrl + Shift + B	跳转到类型声明处
Ctrl + Shift + Enter	自动结束代码，行末自动添加分号
Ctrl + Shift + Backspace	退回到上次修改的地方
Ctrl + Shift + 1,2,3...9	快速添加指定数值的书签
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
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



# Alt + Shift :

快捷键	介绍
Alt + Shift + N	选择 / 添加 task
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
Alt + Shift + 前方向键	移动光标所在行向上移动
Alt + Shift + 后方向键	移动光标所在行向下移动

# 其他：

快捷键	介绍
F2	跳转到下一个高亮错误 或 警告位置
F11	添加书签
ESC	从工具窗口进入代码文件窗口
连按两次Shift	弹出 Search Everywhere 弹出层

# 更多：

官网资料：

Windows / Linux：[https://www.jetbrains.com/idea/docs/IntelliJIDEA\\_ReferenceCard.pdf](https://www.jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard.pdf)

Mac OS X：[https://www.jetbrains.com/idea/docs/IntelliJIDEA\\_ReferenceCard\\_Mac.pdf](https://www.jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard_Mac.pdf)

个人整理：

Windows / Linux：<https://github.com/judasn/IntelliJ-IDEA-Tutorial/blob/master/keymap-introduce.md>

Mac OS X：<https://github.com/judasn/IntelliJ-IDEA-Tutorial/blob/master/keymap-mac-introduce.md>

# IntelliJ IDEA Alt + Enter 特殊性

# 说明：

我个人整理的配带 Gif 动态图资料（内容较多）：

<https://github.com/judasn/IntelliJ-IDEA-Tutorial/blob/master/hotkey-alt-enter-introduce.md>

# THANK YOU

