

IDEA DeBug 调试技巧

菩提树下的杨过 Java后端 2019-11-18

点击上方 Java后端, 选择 [设为星标](#)

优质文章, 及时送达

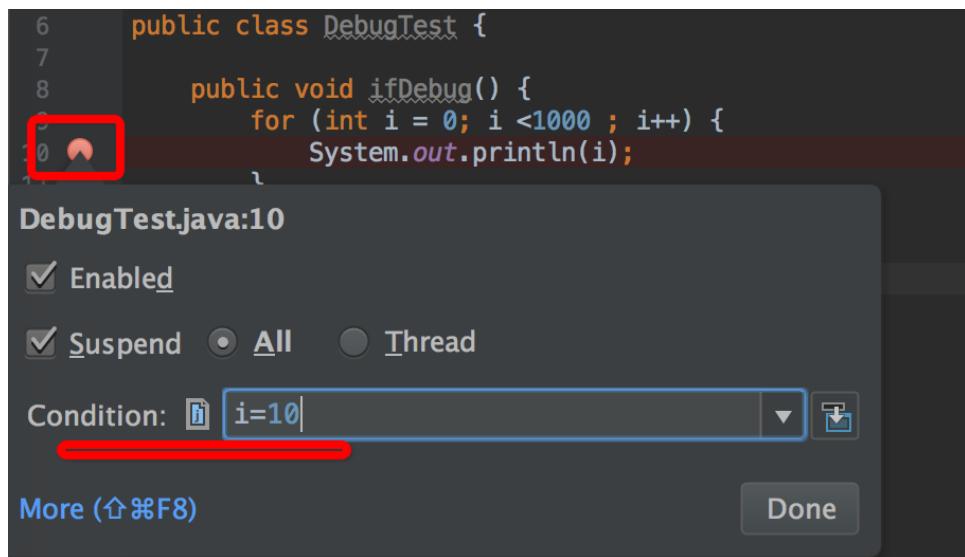
作者 | 菩提树下的杨过

来源 | www.cnblogs.com/yjmyzz

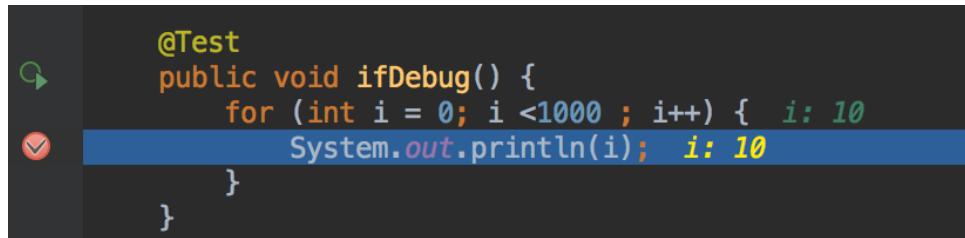
上篇 | 除了《颈椎康复指南》, 还有这 9 本书

一、条件断点

循环中经常用到这个技巧, 比如: 遍历1个大List的过程中, 想让断点停在某个特定值。



参考上图, 在断点的位置, 右击断点旁边的小红点, 会出来一个界面, 在Condition这里填入断点条件即可, 这样调试时, 就会自动停在i=10的位置



二、回到"上一步"

该技巧最适合特别复杂的方法套方法的场景, 好不容易跑起来, 一不小心手一抖, 断点过去了, 想回过头看看刚才的变量值, 如果不知道该技巧, 只能再跑一遍。

```

18     @Test
19     public void dropFrameDebug() {
20         int i = 99;
21         method1(i);
22     }
23
24     private void method1(int i) {
25         System.out.println("method1:" + i);
26         method2(i); // Breakpoint here
27     }
28
29     private void method2(int j) { j: 100
30         j++;
31         System.out.println("method2:" + j); j: 100
32     }
33
34 }
35

```

参考上图，method1方法调用method2，当前断点的位置j=100，点击上图红色箭头位置的Drop Frame图标后，时间穿越了

```

19     public void dropFrameDebug() {
20         int i = 99;
21         method1(i);
22     }
23
24     private void method1(int i) { i: 99
25         System.out.println("method1:" + i);
26         method2(i); i: 99 // Breakpoint here
27     }
28
29     private void method2(int j) {
30         j++;
31         System.out.println("method2:" + j);
32     }
33
34 }
35

```

回到了method1刚开始调用的时候，变量i变成了99，没毛病吧，老铁们，是不是很6:)

注：好奇心是人类进步的阶梯，如果想知道为啥这个功能叫Drop Frame，而不是类似Back To Previous 之类的，可以去翻翻JVM的书，JVM内部以栈帧为单位保存线程的运行状态，drop frame即扔掉当前运行的栈帧，这样当前“指针”的位置，就自然到了上一帧的位置。

三、多线程调试

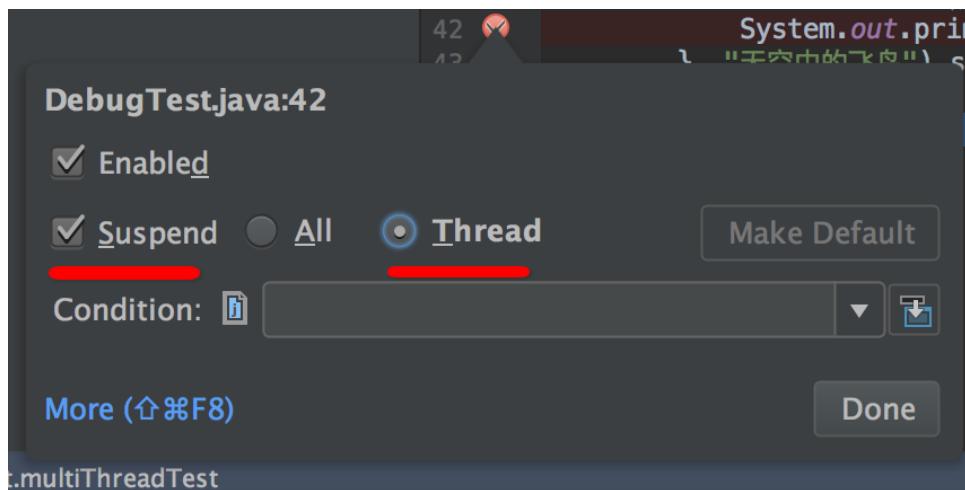
多线程同时运行时，谁先执行，谁后执行，完全是看CPU心情的，无法控制先后，运行时可能没什么问题，但是调试时就比较麻烦了，最明显的就是断点乱跳，一会儿停这个线程，一会儿停在另一个线程，比如下图：

```
@Test  
public void multiThreadTest() {  
    new Thread(() -> {  
        System.out.println("1.卧枝商恨低");  
    }, "菩提树下的杨过").start();  
  
    new Thread(() -> {  
        System.out.println("2.卧梅又闻花");  
    }, "天空中的飞鸟").start();  
  
    System.out.println("3.要问卧似水");  
    System.out.println("4.倚头答春绿");  
}  
}
```

如果想希望下一个断点位置是第2句诗句，可能要失望了：

```
@Test  
public void multiThreadTest() {  
    new Thread(() -> {  
        System.out.println("1.卧枝商恨低");  
    }, "菩提树下的杨过").start();  
  
    new Thread(() -> {  
        System.out.println("2.卧梅又闻花");  
    }, "天空中的飞鸟").start();  
  
    System.out.println("3.要问卧似水");  
    System.out.println("4.倚头答春绿");  
}  
}
```

如果想让线程在调试时，想按自己的愿意来，让它停在哪个线程就停在哪个线程，可以在图中3个断点的小红点上右击，



即：Suspend挂起的条件是按每个线程来，而非All。把这3个断点都这么设置后，再来一发试试

```
34
35     @Test
36     public void multiThreadTest() {
37         new Thread(() -> {
38             System.out.println("1.卧枝商恨低");
39             }, "菩提树下的杨过").start();
40
41         new Thread(() -> {
42             System.out.println("2.卧梅又闻花");
43             }, "天空中的飞鸟").start();
44
45         System.out.println("3.要问卧似水");
46         System.out.println("4.倚头答春绿");
47     }
48 }
49
```

Debug DebugTest.multiThreadTest

Console Debugger Variables Frames → Threads →

"菩提树下的杨过"@953 in group "main": RUNNING
"菩提树下的杨过"@953 in group "main": RUNNING
"main"@1 in group "main": RUNNING
"天空中的飞鸟"@955 in group "main": RUNNING
Finalizer@960: WAIT
"Reference Handler"@961: WAIT
"Signal Dispatcher"@959: RUNNING

注意上图中的红框位置，断点停下来时，这个下拉框可以看到各个线程（注：给线程起个容易识别的名字是个好习惯！），我们可以选择线程“天空中的飞鸟”

```
6     public void multiThreadTest() {
7         new Thread(() -> {
8             System.out.println("1.卧枝商恨低");
9             }, "菩提树下的杨过").start();
10
11         new Thread(() -> {
12             System.out.println("2.卧梅又闻花");
13             }, "天空中的飞鸟").start();
14
15         System.out.println("3.要问卧似水");
16         System.out.println("4.倚头答春绿");
17     }
18 }
```

Debug DebugTest.multiThreadTest

Console Debugger Variables Frames → Threads →

"天空中的飞鸟"@955 in group "main": RUNNING
lambda\$multiThreadTest\$1:42, DebugTest (com.example)
run:-1, 2619171 (com.example.DebugTest\$\$Lambda\$2)
run:745, Thread (java.lang)

断点如愿停在了第2句诗。

四、远程调试

这也是一个装B的利器，本机不用启动项目，只要有源代码，可以在本机直接远程调试服务器上的代码，打开姿势如下：

4.1 项目启动时，先允许远程调试

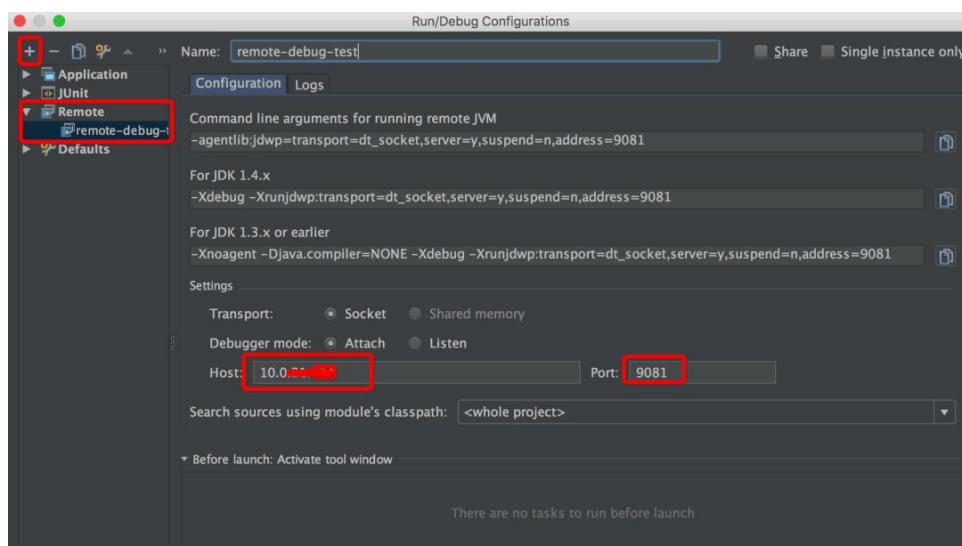
```
java -server -Xms512m -Xmx512m -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=9081
```

起作用的就是

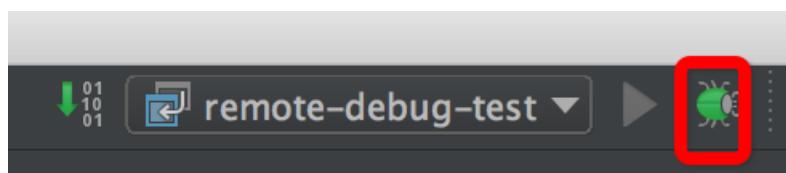
```
-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=9081
```

注意：远程调试从技术上讲，就是在本机与远程建立socket通讯，所以端口不要冲突，而且本机要允许访问远程端口，另外这一段参数，放要在-jar 或 \${main_class}的前面

4.2 idea中设置远程调试



然后就可以调试了

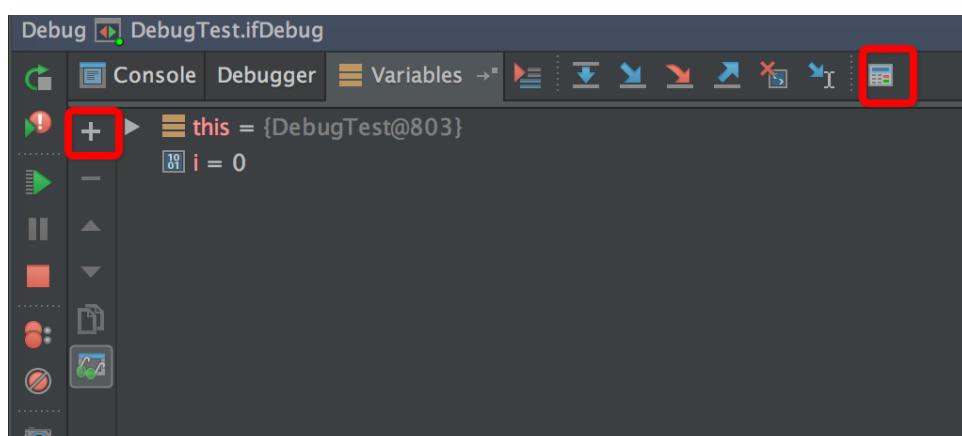


前提是本机有项目的源代码，在需要的地方打个断点，然后访问一个远程的url试试，断点就会停下来。

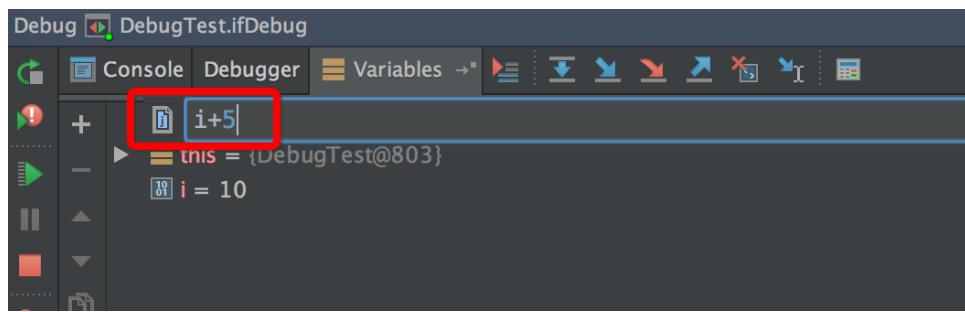
Tips：欢迎关注微信公众号：Java后端，每日技术博文推送。

五、临时执行表达式/修改变量的运行值

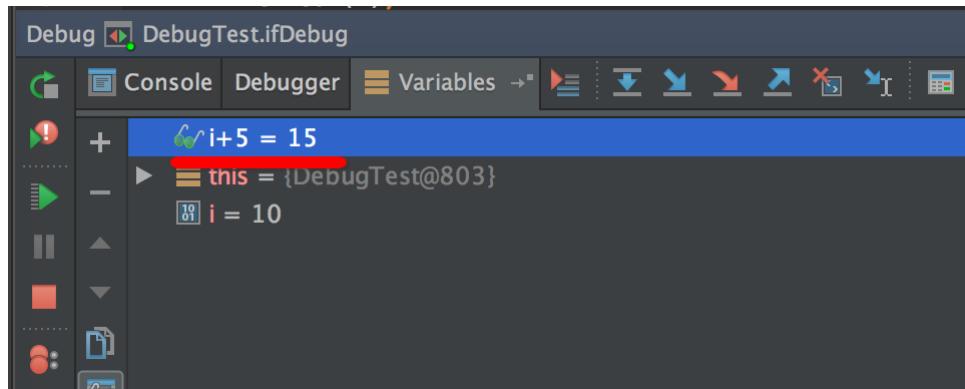
调试时，可以临时执行一些表达式，参考下图：点击这两个图标中的任何一个都可以



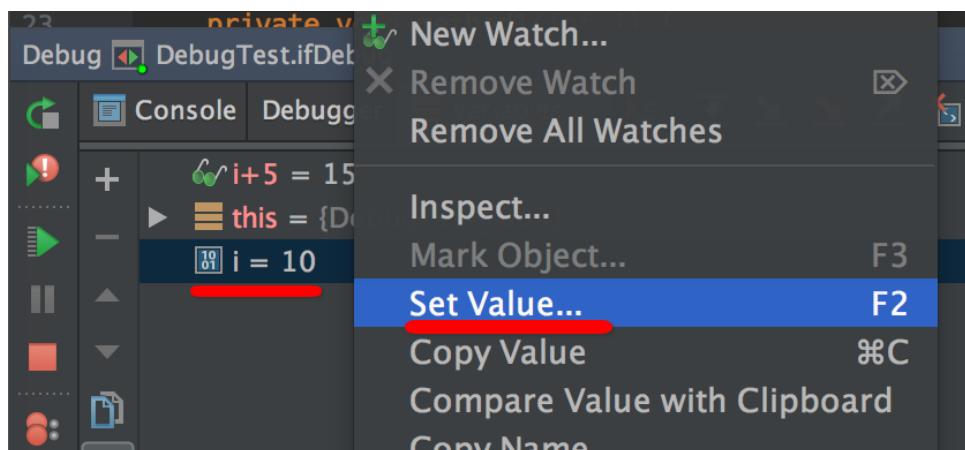
点击+号后，就可以在新出现的输入框里输入表达式，比如`i+5`



然后回车，马上就能看到结果



当然，如果调试时，想动态修改变量的值，也很容易，在变量上右击，然后选择Set Value，剩下的事，地球人都知道。



善用上述调试技巧，相信大家撸起代码来会更有感觉，祝大家周末愉快！

- END -

如果看到这里，说明你喜欢这篇文章，请[转发、点赞](#)。微信搜索「web_resource」，关注后回复「进群」或者扫描下方二维码即可进入无广告交流群。

↓ 扫描二维码进群 ↓



扫一扫上面的二维码图案，加我微信

推荐阅读

1. 你们心心念念的 GitHub 客户端终于来了！
2. Redis 实现「附近的人」这个功能
3. 一个秒杀系统的设计思考
4. 零基础认识 Spring Boot
5. 团队开发中 Git 最佳实践



喜欢文章，点个在看

[阅读原文](#)

IDEA 中使用 Git 图文教程

Java后端 2019-09-03

以下文章来源于macrozheng，作者梦想de星空



macrozheng

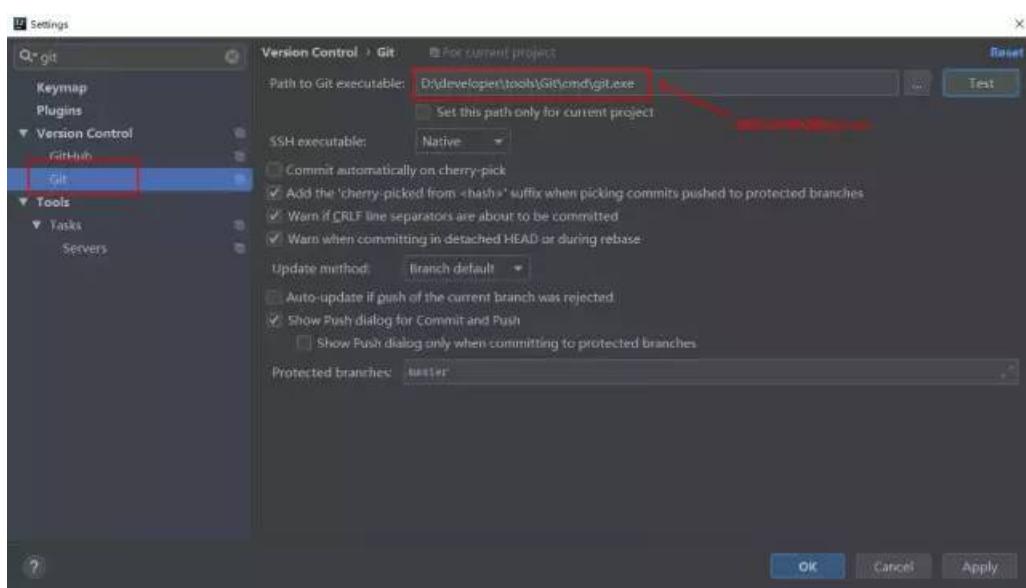
专注Java技术分享，涵盖SpringBoot、SpringCloud、Docker、中间件等实用技术，作者Github开源项目mall（25K+…

摘要

大家在使用Git时，都会选择一种Git客户端，在IDEA中内置了这种客户端，可以让你不需要使用Git命令就可以方便地进行操作，本文将讲述IDEA中的一些常用Git操作。

环境准备

- 使用前需要安装一个远程的Git仓库和本地的Git客户端，具体参考：[10分钟搭建自己的Git仓库](#)。
- 由于IDEA中的Git插件需要依赖本地Git客户端，所以需要进行如下配置：



操作流程

在Gitlab中创建一个项目并添加README文件

Blank project Create from template Import project

Project name

Project URL Project slug

Want to house several dependent projects under the same namespace? [Create a group](#).

Project description (optional)
mall-tiny project

Visibility Level [?](#)

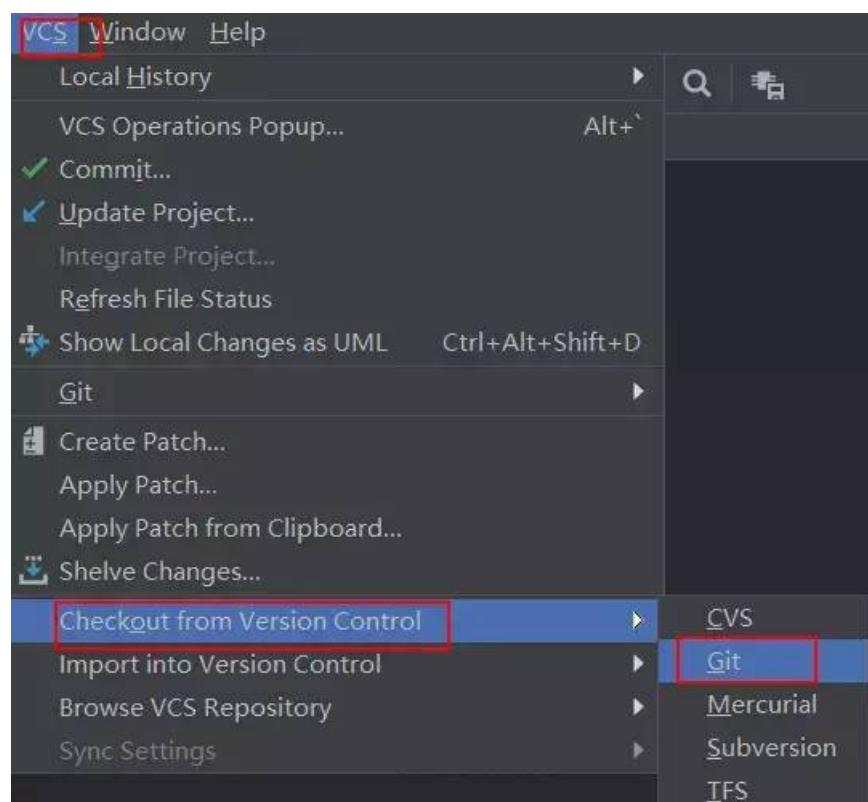
- Private Project access must be granted explicitly to each user.
- Internal
- Public

Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

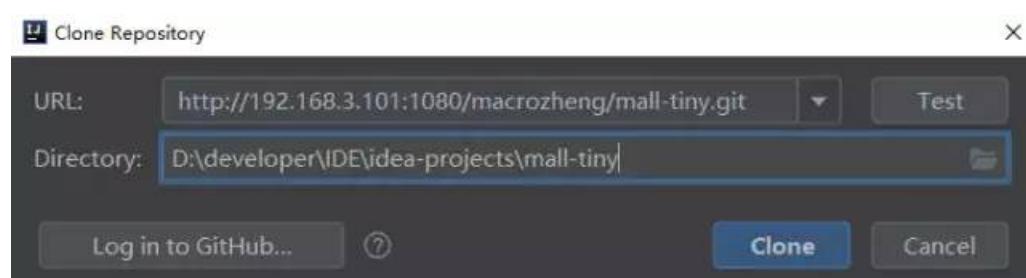
[Create project](#) [Cancel](#)

clone项目到本地

- 打开从Git检出项目的界面：



- 输入Git地址进行检出：



- 暂时不生成IDEA项目，因为项目还没初始化：



初始化项目并提交代码

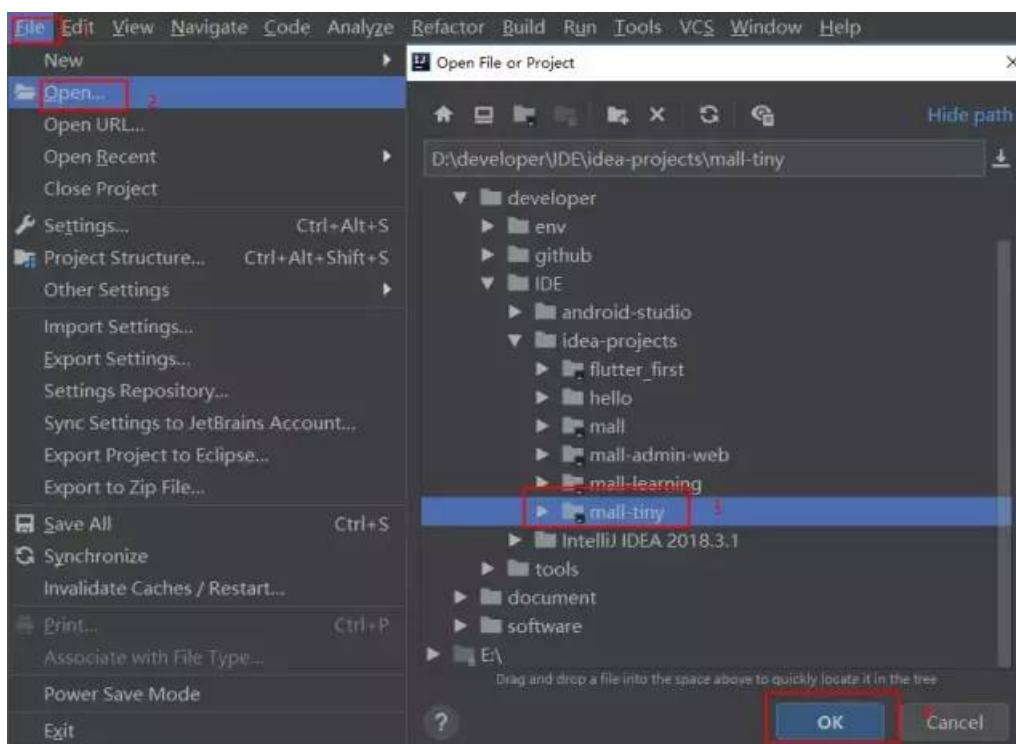
- 将mall-tiny的代码复制到该目录中：

developer > IDE > idea-projects > mall-tiny			
名称	修改日期	类型	大小
images	2019/8/12 20:24	文件夹	
sql	2019/8/10 10:08	文件夹	
src	2019/8/10 10:04	文件夹	
.gitignore	2019/4/12 21:34	文本文件	1 KB
LICENSE	2019/8/10 10:03	文件	12 KB
pom.xml	2019/8/4 14:36	XML 文档	7 KB
README.md	2019/8/12 20:27	MD 文件	14 KB

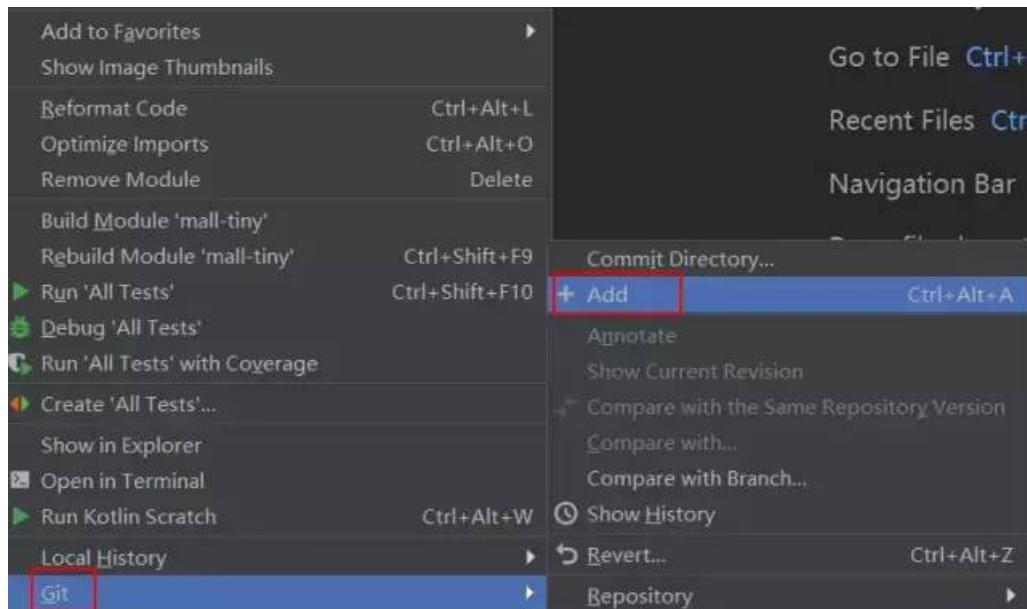
- 这里我们需要一个.gitignore文件来防止一些IDEA自动生成的代码被提交到Git仓库去：

```
1 # Maven
2 #
3 target/
4
5 # IDEA #
6 .idea/
7 *.iml
8
9 # Eclipse
10 #
11 .settings/
12 .classpath
13 .project
```

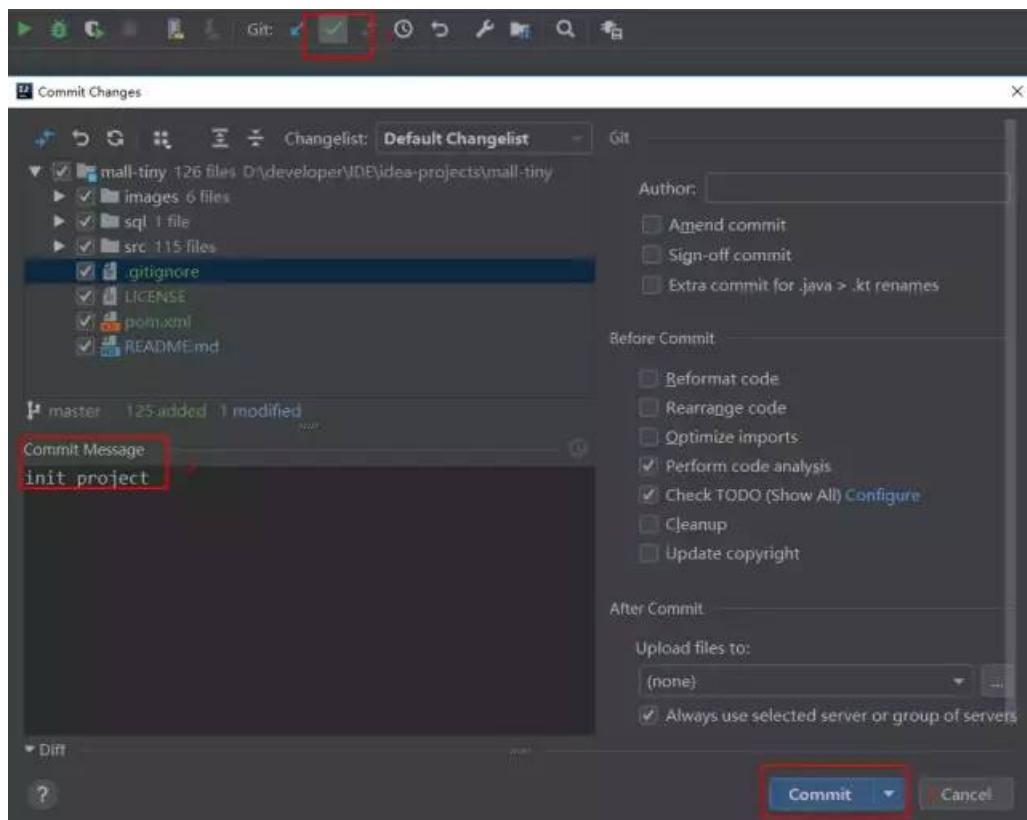
- 使用IDEA打开项目：



- 右键项目打开菜单，将所有文件添加到暂存区中：

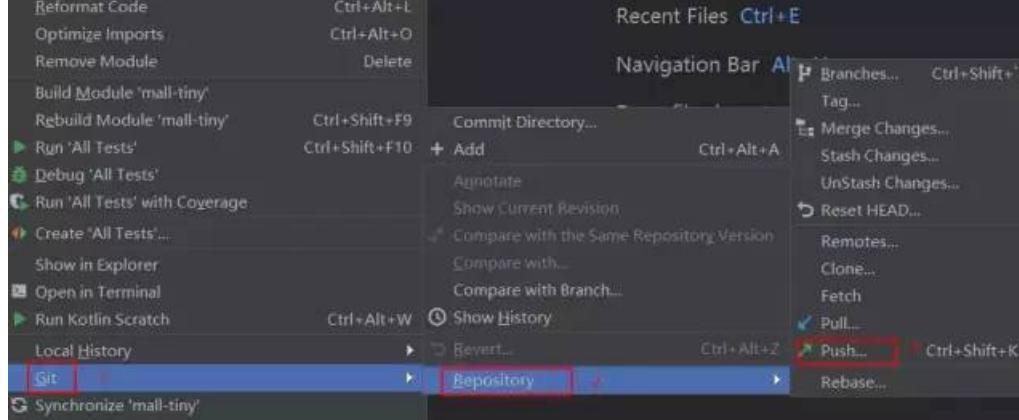


- 添加注释并提交代码：

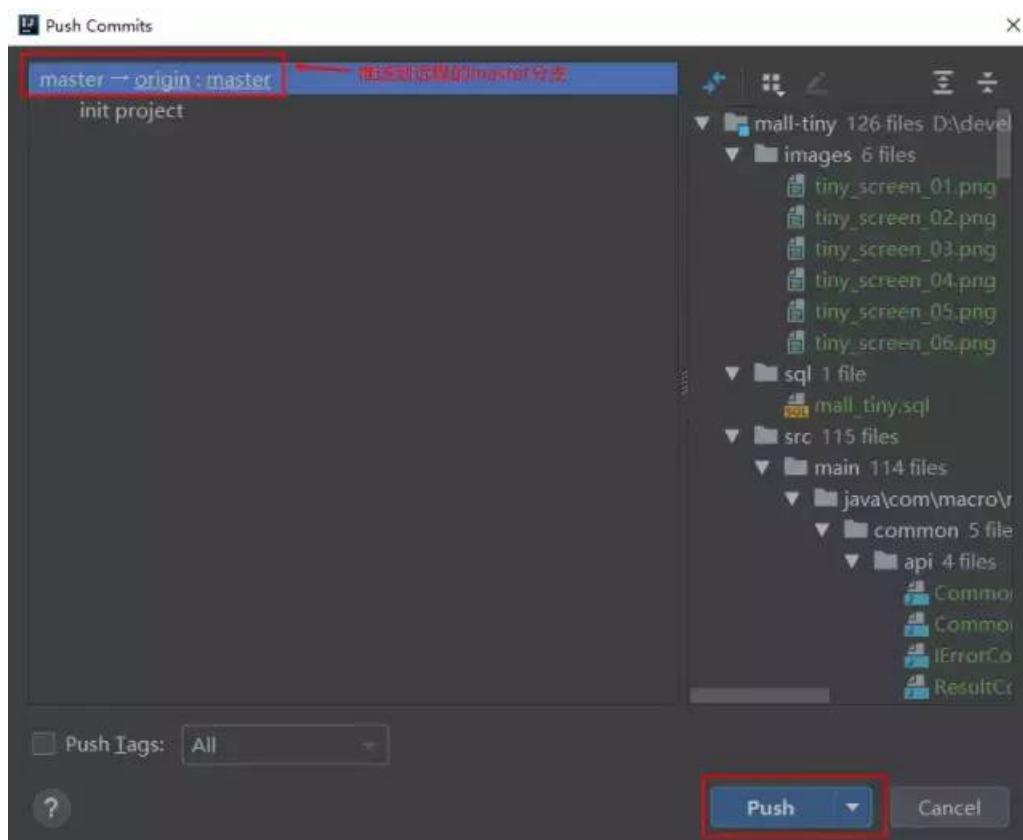


将代码推送到远程仓库

- 点击push按钮推送代码：



- 确认推送内容：



- 查看远程仓库发现已经提交完成：

mall-tiny 
Project ID: 2

LICENSE · 2 Commits · 1 Branch · 0 Tags · 369 KB Files
mall-tiny project

master · mall-tiny / + · History · Find file · Web IDE · ⚙

init project · macro authored 5 minutes ago · ec5197b3 · ⌂

 README ·  Auto DevOps enabled ·  Add CHANGELOG ·  Add CONTRIBUTING ·  Add Kubernetes cluster

Name	Last commit	Last update
images	init project	5 minutes ago
sql	init project	5 minutes ago
src	init project	5 minutes ago
.gitignore	init project	5 minutes ago
LICENSE	init project	5 minutes ago
README.md	init project	5 minutes ago
pom.xml	init project	5 minutes ago

从远程仓库拉取代码

- 在远程仓库添加一个README-TEST.md文件：

macrozheng > mall-tiny > Repository

New file · Template · Choose type

master / README-TEST.md

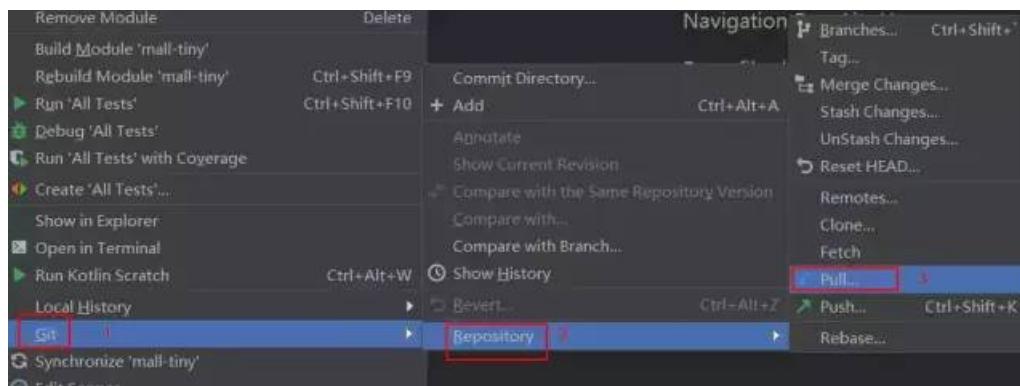
```
1 # README-TEST
```

Commit message · Add new file

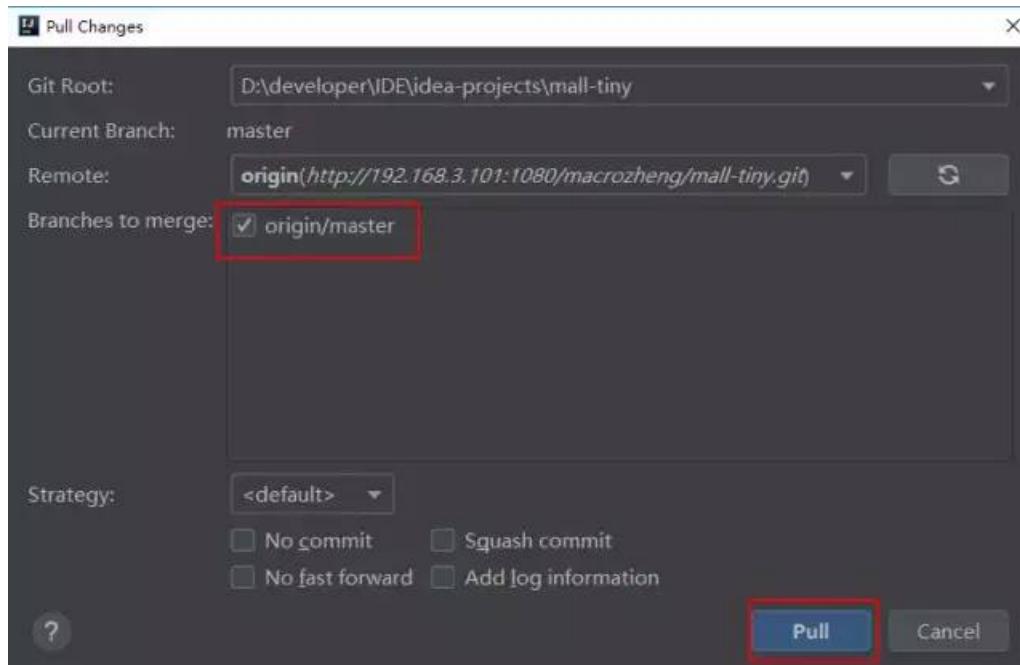
Target Branch · master

Commit changes

- 从远程仓库拉取代码：

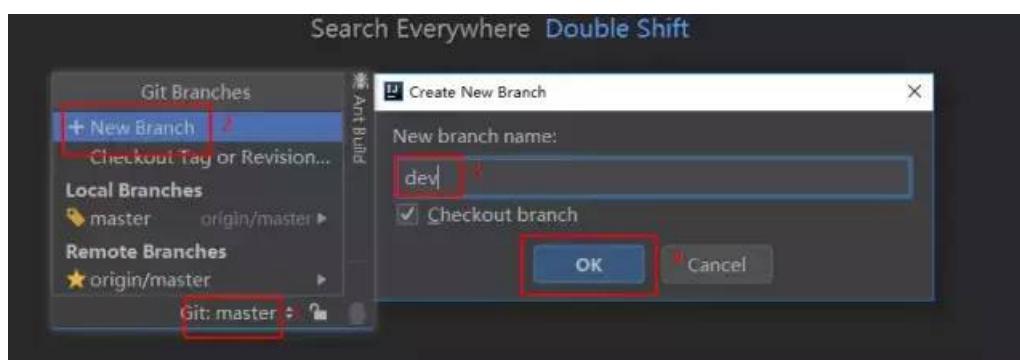


- 确认拉取分支信息：

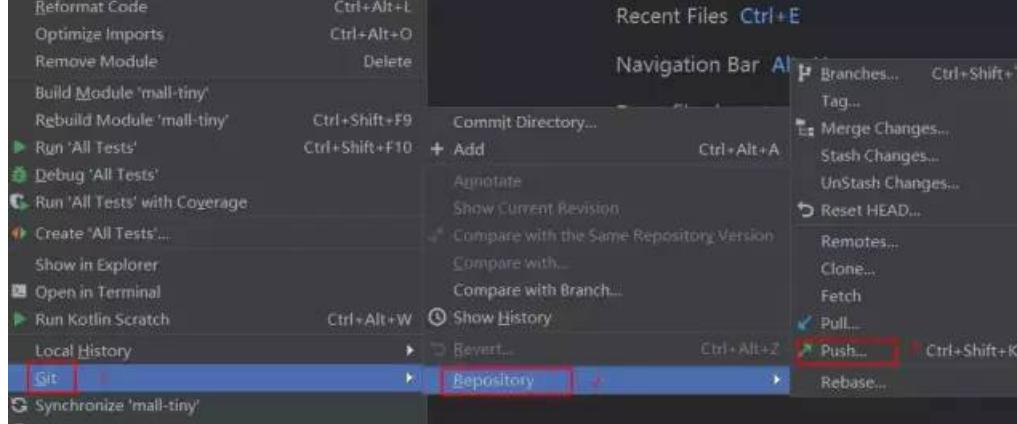


从本地创建分支并推送到远程

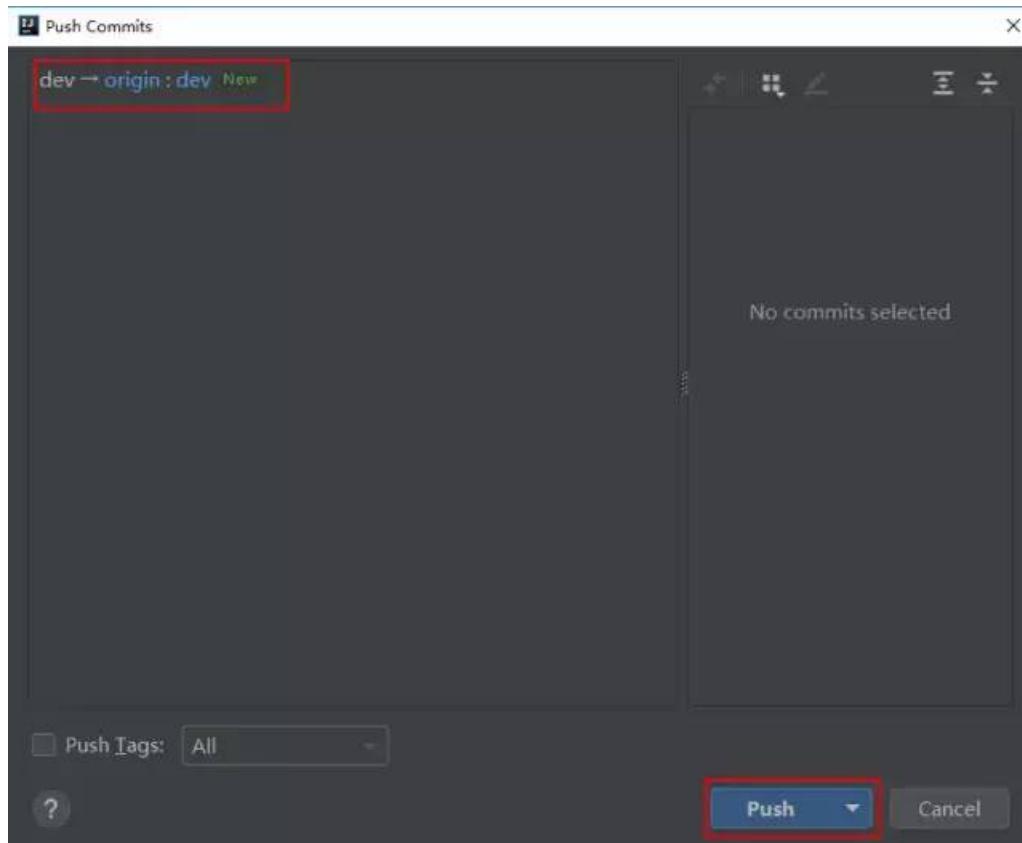
- 在本地创建dev分支，点击右下角的Git:master按钮：



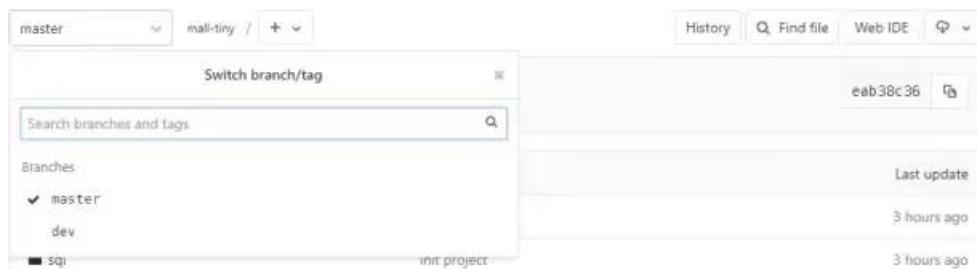
- 使用push将本地dev分支推送到远程：



- 确认推送内容：

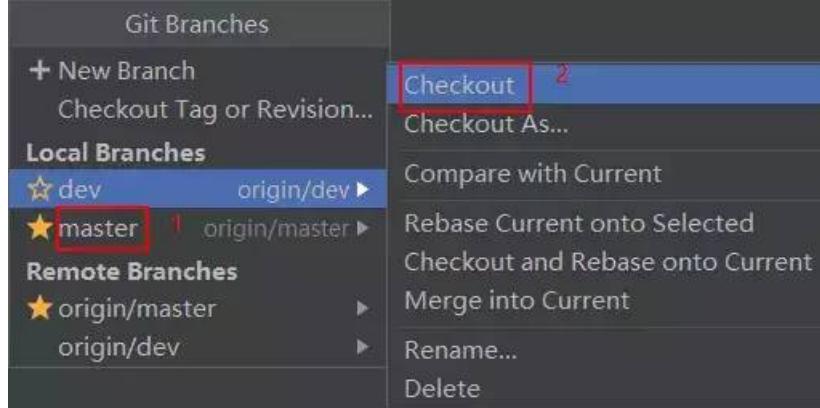


- 查看远程仓库发现已经创建了dev分支：



分支切换

- 从dev分支切换回master分支：

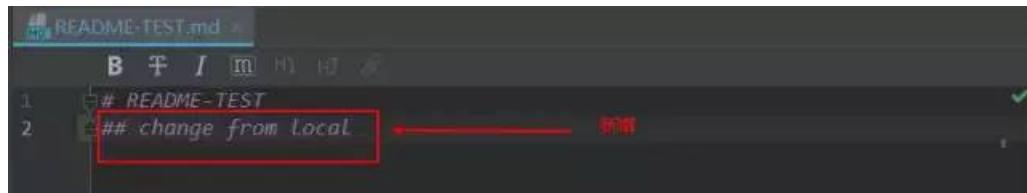


Git文件冲突问题解决

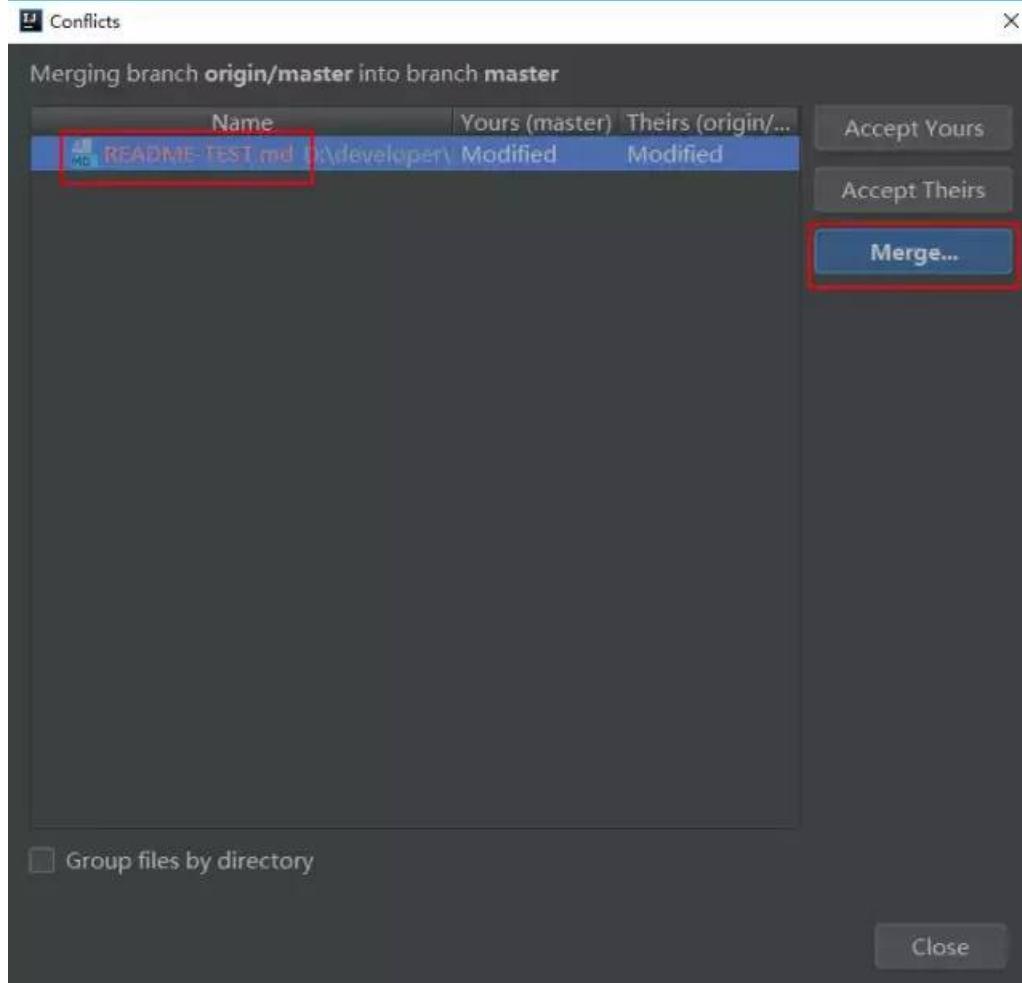
- 修改远程仓库代码：



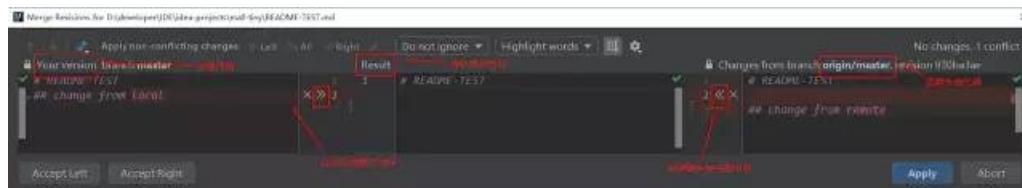
- 修改本地仓库代码：



- 提交本地仓库代码并拉取，发现代码产生冲突，点击Merge进行合并：



- 点击箭头将左右两侧代码合并到中间区域：



- 冲突合并完成后，点击Apply生效：



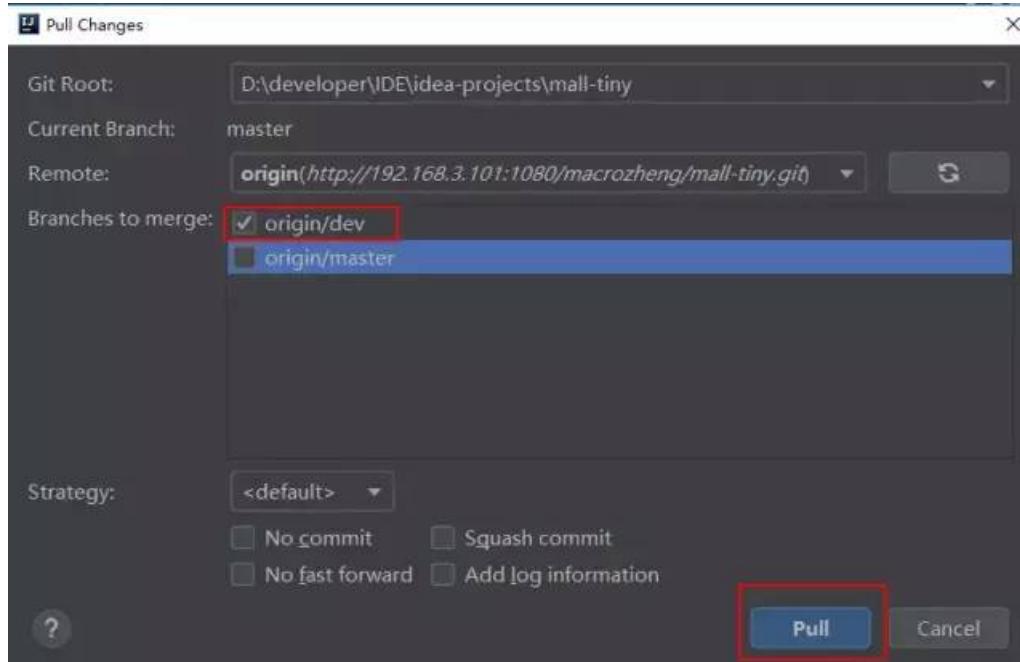
- 提交代码并推送到远程。

从dev分支合并代码到master

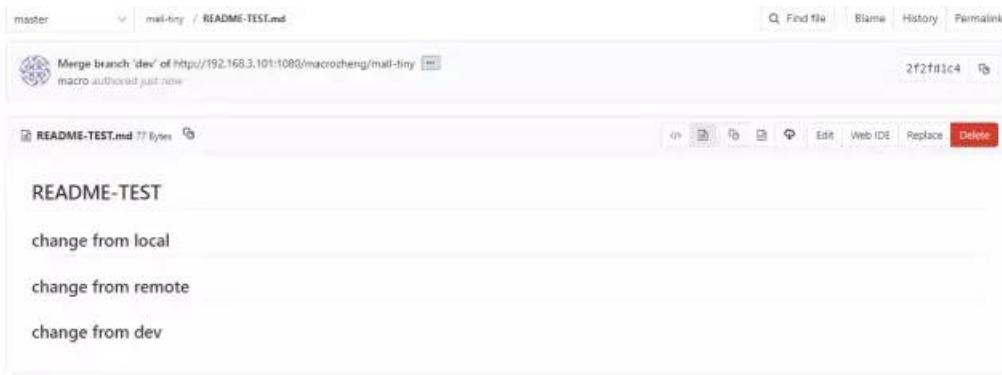
- 在远程仓库修改dev分支代码：



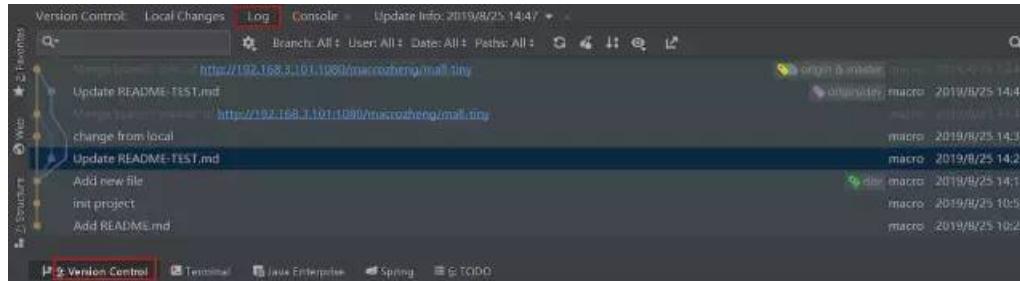
- 在本地仓库拉取代码，选择从dev分支拉取并进行合并：



- 发现产生冲突，解决后提交并推送到远程仓库即可。



查看Git仓库提交历史记录



作者：MacroZheng

链接：<https://juejin.im/post/5d667fc6e51d453b5d4d8da5>

如果喜欢本篇文章，欢迎[转发](#)、[点赞](#)。关注订阅号「Web项目聚集地」，回复「全栈」即可获取 2019 年最新 Java、Python、前端学习视频资源。

推荐阅读

- [1. 这代码写的，狗屎一样](#)
- [2. 这代码写的，狗屎一样（下）](#)

[3. 除了负载均衡, Nginx 还可以做很多](#)

[4. 快来薅当当的羊毛 !](#)

[5. 聊一聊 Java 泛型中的通配符](#)

[6. 数据库不使用外键的 9 个理由](#)



Web项目聚集地

微信扫描二维码, 关注我的公众号

喜欢文章, 点个在看 

[阅读原文](#)

声明: pdf仅供学习使用, 一切版权归原创公众号所有; 建议持续关注原创公众号获取最新文章, 学习愉快!

IDEA 插件推荐: EasyCode 一键生成所需代码

HelloWxl Java后端 2月18日

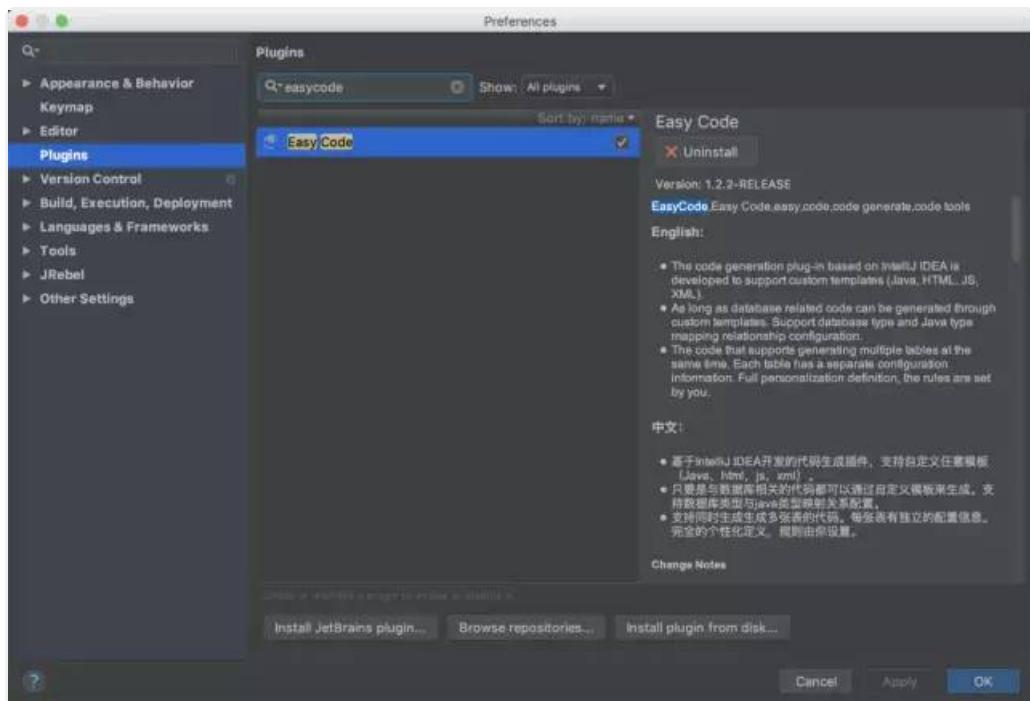


作者 | HelloWxl

来源 | <http://suo.im/6v9d64>

Easycode是idea的一个插件，可以直接对数据的表生成entity,controller,service,dao,mapper,无需任何编码，简单而强大。

1、安装(EasyCode)



我这里的话是已经那装好了。

- 建议大家在安装一个插件，叫做Lombok。Lombok能通过注解的方式，在编译时自动为属性生成构造器、getter/setter、equals、hashcode、toString方法。出现的神奇就是在源码中没有getter和setter方法，但是在编译生成的字节码文件中有getter和setter方法。

2、建立数据库



```
-- -----  
DROP TABLE IF EXISTS  
`user`  
;
```

```
CREATE TABLE  
`user`  
(
```

```
 `id`  
int  
(  
11  
) NOT NULL,
```

```
 `username`  
varchar(  
20  
) DEFAULT NULL,
```

```
 `sex`  
varchar(  
6  
) DEFAULT NULL,
```

```
 `birthday`  
date DEFAULT NULL,
```

```
 `address`  
varchar(  
20  
) DEFAULT NULL,
```

```
 `password`  
varchar(  
20  
) DEFAULT NULL,
```

```
 PRIMARY KEY (  
`id`  
)
```

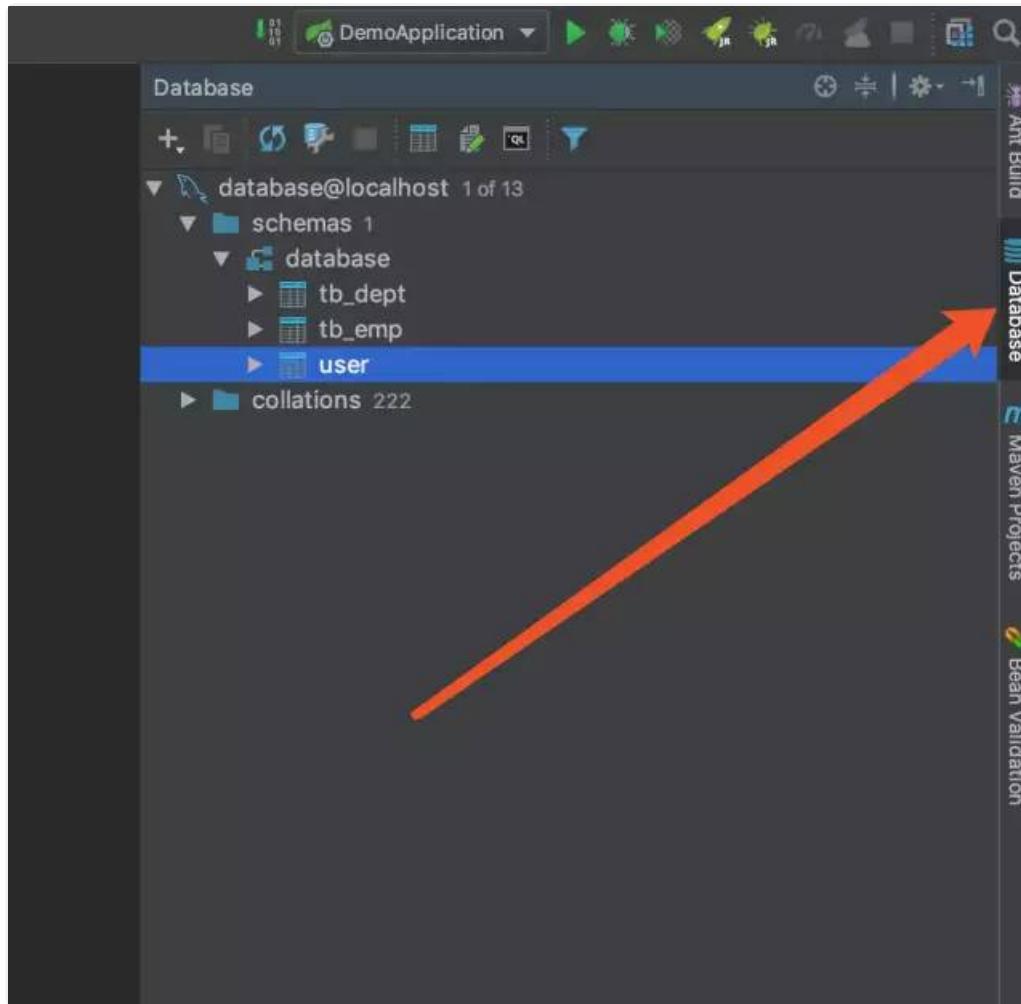
```
) ENGINE=InnoDB  
DEFAULT CHARSET=utf8;
```

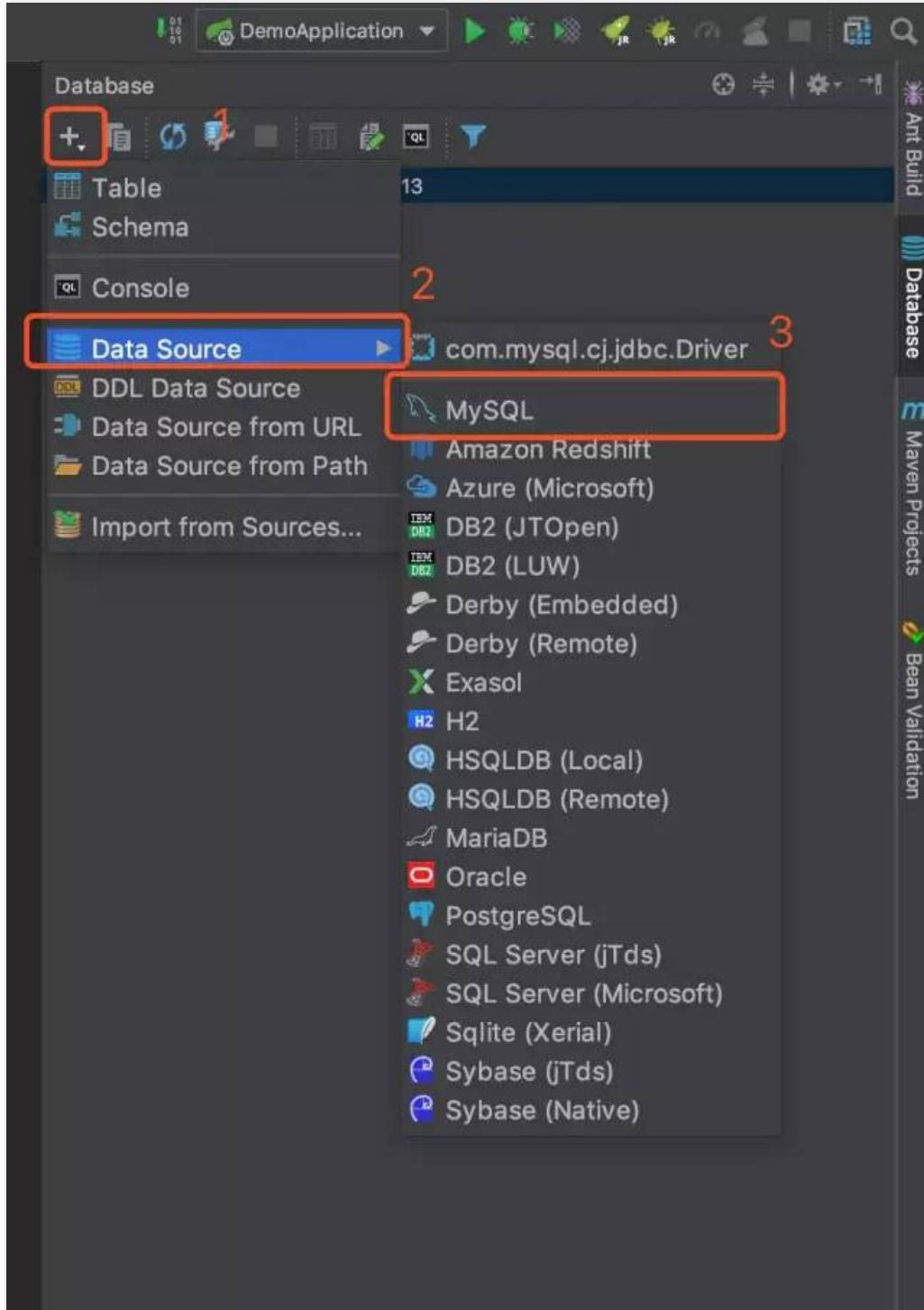
```
SET FOREIGN_KEY_CHECKS =  
1  
;
```

3、在IDEA配置连接数据库

在这个之前，新建一个Springboot项目，这个应该是比较简单的。

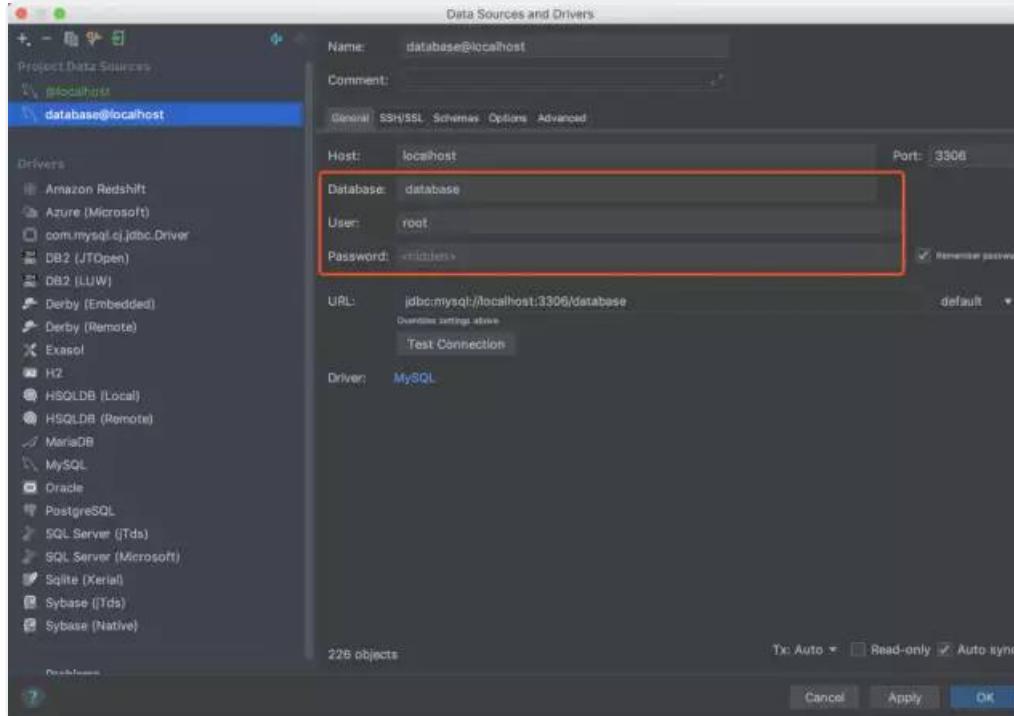
建好SpringBoot项目之后，如下图所示，找到这个Database





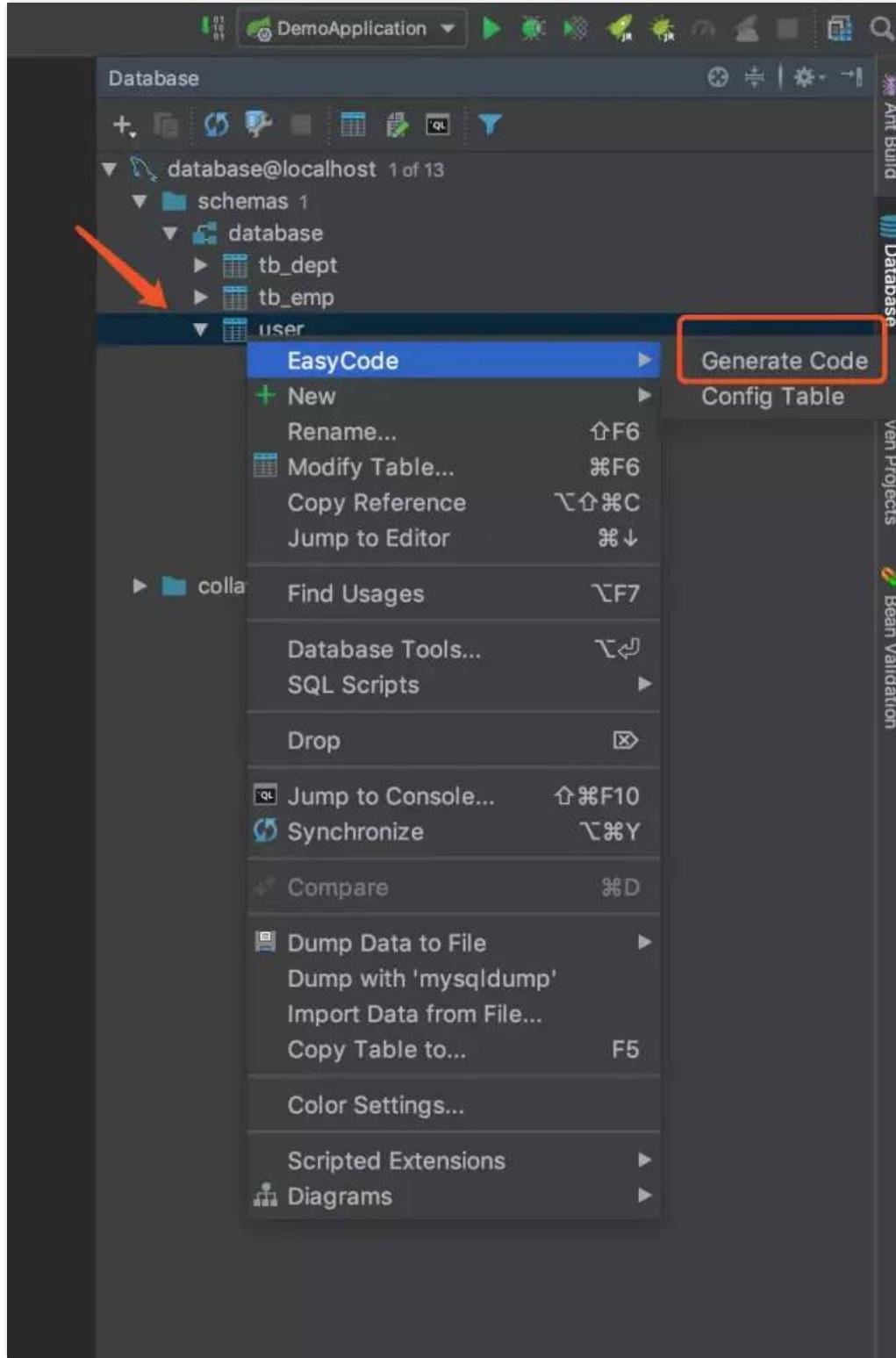
按照如下图所示进行操作：

然后填写数据库名字，用户名，密码。点击OK即可。这样的话，IDEA连接数据库就完事了。

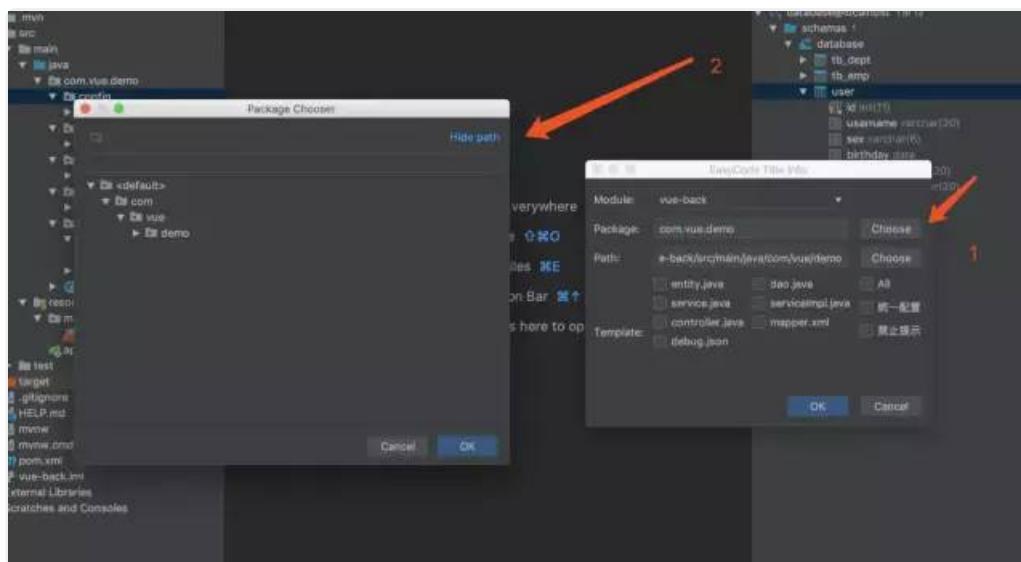


4、开始生成代码

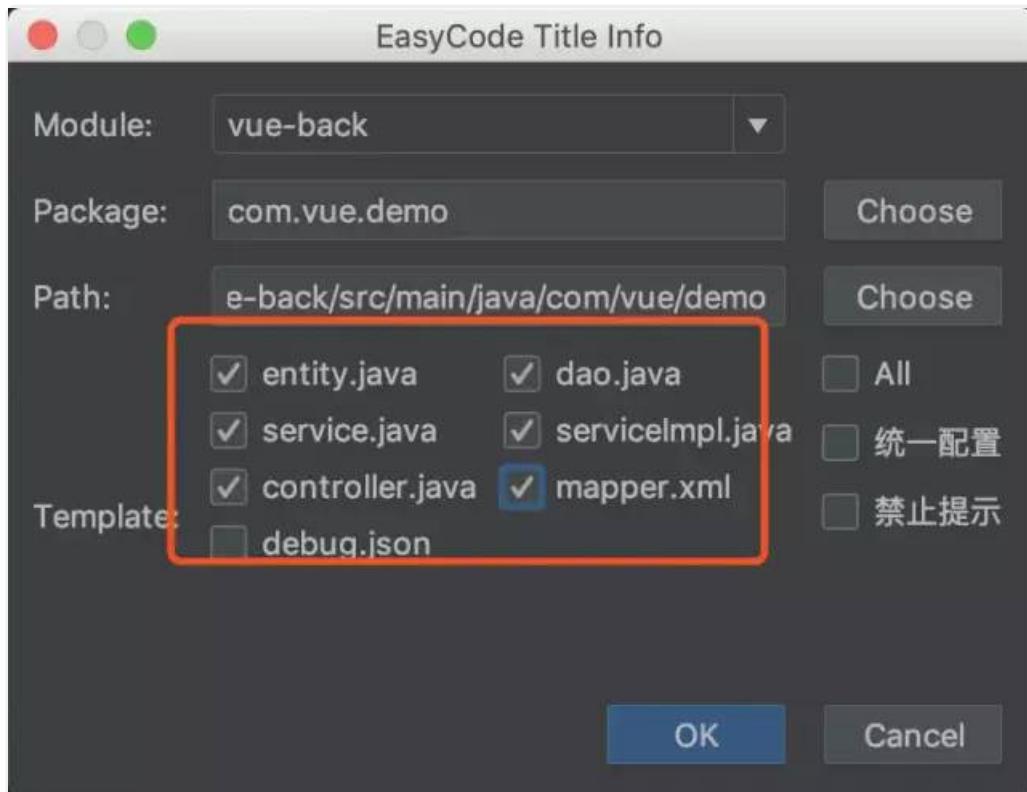
在这个里面找到你想生成的表，然后右键，就会出现如下所示的截面。



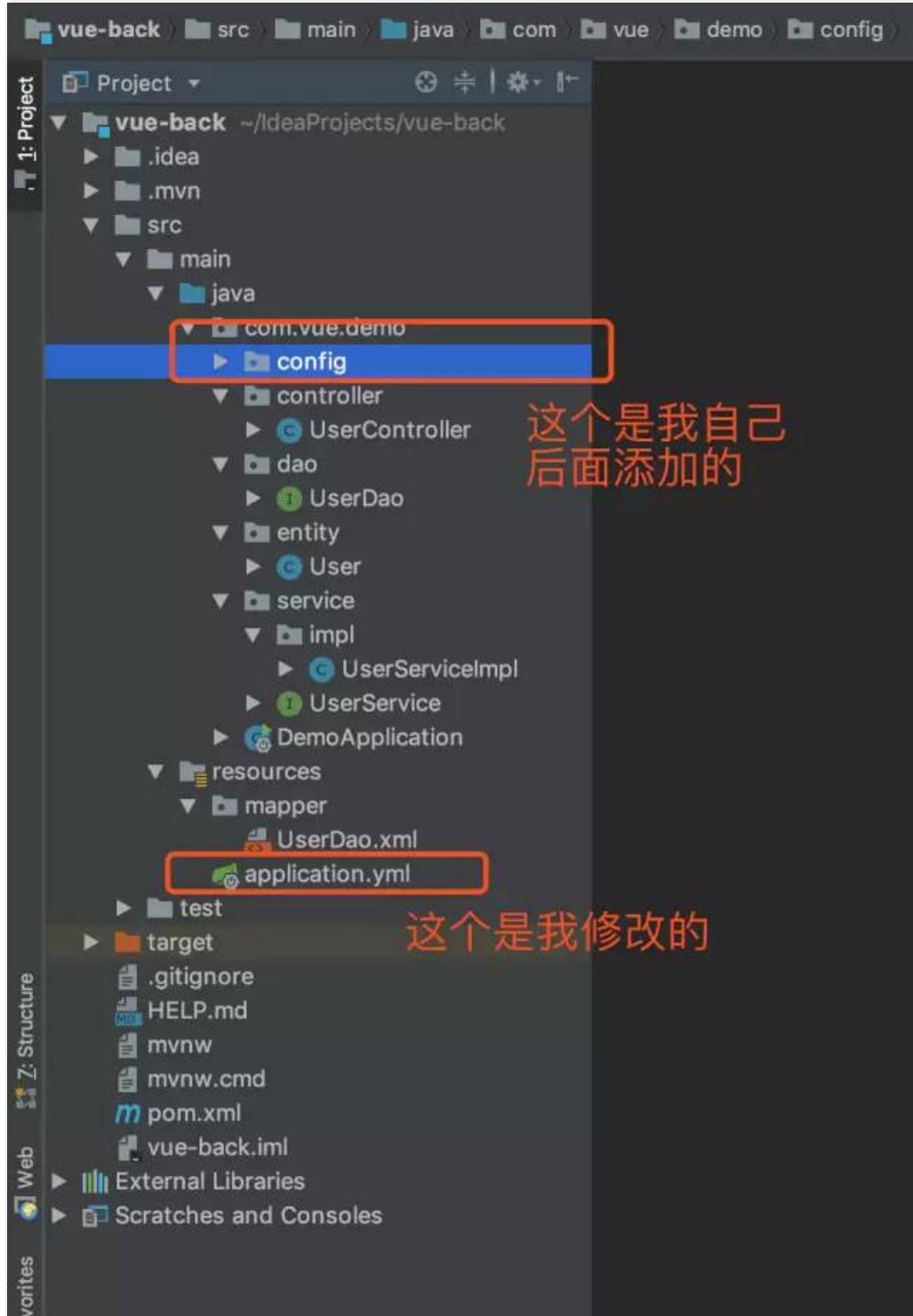
点击1所示的位置，选择你要将生成的代码放入哪个文件夹中，选择完以后点击OK即可。



勾选你需要生成的代码，点击OK。



这样的话就完成了代码的生成了，生成的代码如下图所示：



5、pom.xml

```
<dependency>

<groupId>
org.springframework.boot
</groupId>

<artifactId>
spring-boot-starter
</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
  <groupId>  
    org.springframework.boot  
  </groupId>
```

```
  <artifactId>  
    spring-boot-starter-web  
  </artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
  <groupId>  
    org.projectlombok  
  </groupId>
```

```
  <artifactId>  
    lombok  
  </artifactId>
```

```
  <optional>  
    true  
  </optional>
```

```
</dependency>
```

```
<!--热部署-->
```

```
<dependency>
```

```
  <groupId>  
    org.springframework.boot  
  </groupId>
```

```
  <artifactId>  
    spring-boot-devtools  
  </artifactId>
```

```
  <optional>  
    true  
  </optional>  
  <!-- 这个需要为 true 热部署才有效 -->
```

```
</dependency>
```

```
<!--mybatis-->
```

```
<dependency>
```

```
  <groupId>  
    org.mybatis.spring.boot  
  </groupId>
```

```
  <artifactId>  
    mybatis-spring-boot-starter  
  </artifactId>
```

```
  <version>  
    1.3.2  
  </version>
```

```
</dependency>
```

```
<!-- mysql -->
```

```
<dependency>
```

```
<groupId>
```

```
mysql
```

```
</groupId>
```

```
<artifactId>
```

```
mysql-connector-java
```

```
</artifactId>
```

```
<version>
```

```
5.1.47
```

```
</version>
```

```
</dependency>
```

```
<!--阿里巴巴连接池-->
```

```
<dependency>
```

```
<groupId>
```

```
com.alibaba
```

```
</groupId>
```

```
<artifactId>
```

```
druid
```

```
</artifactId>
```

```
<version>
```

```
1.0.9
```

```
</version>
```

```
</dependency>
```

6、Application.yml

```
server:  
  
port:  
8089  
  
spring:  
  
datasource:  
  
url: jdbc:mysql:  
//127.0.0.1:3306/database?useUnicode=true&characterEncoding=UTF-8  
  
username: root  
  
password:  
123456  
  
type: com.alibaba.druid.pool.  
DruidDataSource  
  
driver-  
class  
-name: com.mysql.jdbc.  
Driver
```

mybatis:

mapper-locations: classpath:

/mapper/

*

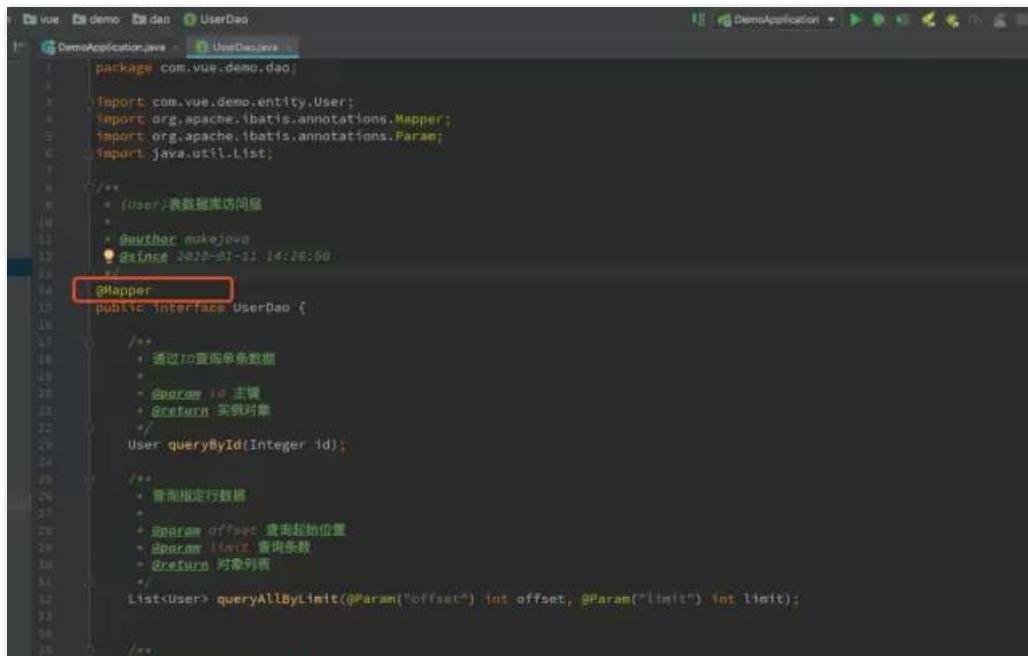
Dao

.xml

typeAliasesPackage: com.vue.demo.entity

7、启动项目

在启动项目之前，我们需要先修改两个地方。



```
package com.vue.demo.dao;

import com.vue.demo.entity.User;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Param;
import java.util.List;

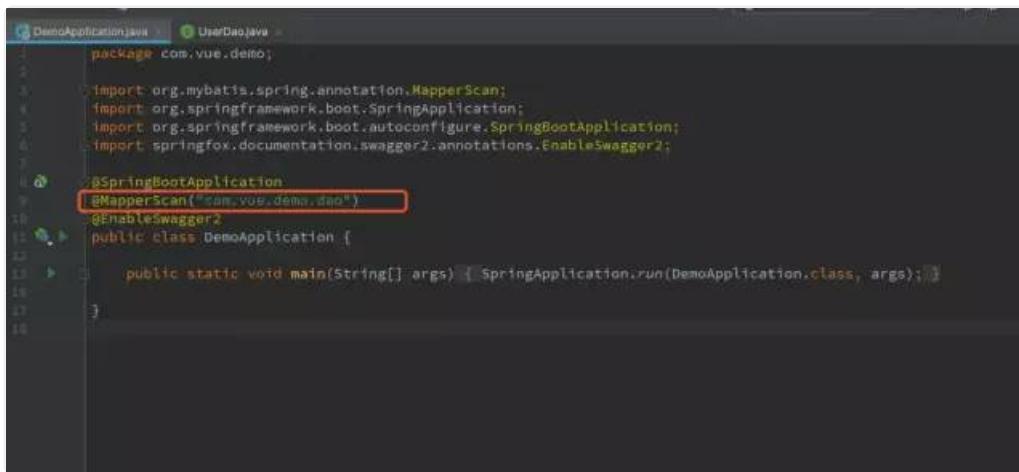
/**
 * (User)表数据库访问层
 */
@Mapper
public interface UserDAO {

    /**
     * 通过id查询单条数据
     *
     * @param id 主键
     * @return 实例对象
     */
    User queryById(Integer id);

    /**
     * 查询多条数据
     *
     * @param offset 查询起始位置
     * @param limit 查询参数
     * @return 对象列表
     */
    List<User> queryAllByLimit(@Param("offset") int offset, @Param("limit") int limit);
}
```

在dao层加上@Mapper注解

在启动类里面加上@MapperScan("com.vue.demo.dao")注解。



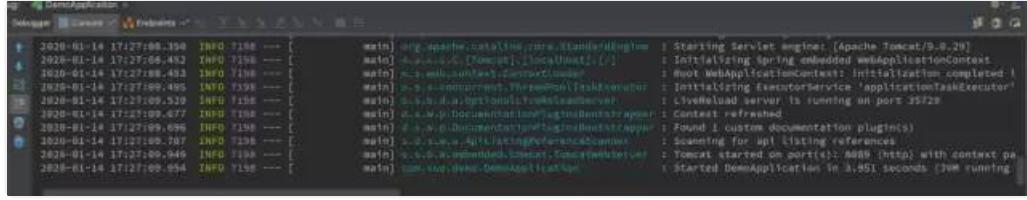
```
package com.vue.demo;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@SpringBootApplication
@MapperScan("com.vue.demo.dao")
@EnableSwagger2
public class DemoApplication {

    public static void main(String[] args) { SpringApplication.run(DemoApplication.class, args); }
}
```

启动项目



```
2020-01-14 17:27:08.194 INFO [main] org.apache.catalina.startup.Embedded: Starting Servlet engine: [Apache Tomcat/9.0.29]
2020-01-14 17:27:09.492 INFO [main] org.springframework.boot.web.embedded.tomcat.TomcatWebServer: Initializing Spring embedded webApplicationContext
2020-01-14 17:27:09.493 INFO [main] org.springframework.boot.web.embedded.tomcat.TomcatWebServer: Initialisation completed: 1 contexts
2020-01-14 17:27:09.495 INFO [main] org.springframework.boot.web.embedded.tomcat.TomcatWebServer: ContextLoaderListener: Context initialized
2020-01-14 17:27:09.520 INFO [main] org.springframework.boot.web.embedded.tomcat.TomcatWebServer: ContextLoaderListener: Context initialized
2020-01-14 17:27:09.677 INFO [main] org.springframework.boot.web.embedded.tomcat.TomcatWebServer: ContextLoaderListener: Context initialized
2020-01-14 17:27:09.677 INFO [main] org.springframework.boot.web.embedded.tomcat.TomcatWebServer: ContextLoaderListener: Context initialized
2020-01-14 17:27:09.696 INFO [main] org.springframework.boot.web.embedded.tomcat.TomcatWebServer: ContextLoaderListener: Context initialized
2020-01-14 17:27:09.787 INFO [main] org.springframework.boot.web.embedded.tomcat.TomcatWebServer: ContextLoaderListener: Context initialized
2020-01-14 17:27:09.946 INFO [main] org.springframework.boot.web.embedded.tomcat.TomcatWebServer: Tomcat started on port(s): 8080 (http) with context path
2020-01-14 17:27:09.954 INFO [main] com.vue.demo.DemoApplication: Started DemoApplication in 3.651 seconds (JVM running)
```

① 127.0.0.1:8089/api/user/selectOne?id=1
{"id":1,"username":"小明","sex":"男","birthday":"2019-12-13","address":"安徽合肥","password":"123"}

测试一下

② 127.0.0.1:8089/api/user/queryUser?offset=0&limit=10
[{"id":1,"username":"小明","sex":"男","birthday":"2019-12-13","address":"安徽合肥","password":"123"}, {"id":2,"username":"小龙","sex":"女","birthday":"2019-12-18","address":"上海浦东","password":"123"}]

-END-

如果看到这里，说明你喜欢这篇文章，请[转发、点赞](#)。微信搜索「web_resource」，欢迎添加小编微信「focusoncode」，每日朋友圈更新一篇高质量技术博文(无广告)。

↓ 扫描二维码添加小编 ↓



推荐阅读

1. Dubbo 爆出严重漏洞!
2. Spring Boot+Redis 分布式锁：模拟抢单
3. 安利一款 IDEA 中强大的代码生成利器
4. 如何获取靠谱的新型冠状病毒疫情



微信搜一搜

Java后端

IDEA 新特性：提前知道代码怎么走

简简单单Online Java后端 1周前

点击上方 Java后端, 选择 [设为星标](#)

优质文章, 及时送达

作者: 简简单单OnlineZuoZuo

链接:blog.csdn.net/qq_15071263/article/details/104186309

新特性

IDEA - 2020.1 版本针对调试器和代码分析器的改进，值得期待

1、对于调试器的加强：数据流分析辅助

2、调试加强：属性置顶功能

3、调试加强：IPV6 调试

4、性能分析的改进，剔除额外的东西

5、支持读取内存快照文件

6、IDEA 变更了代码提交的界面

7、LightEdit 用来作为简单的文本编辑器

8、可以预览变更意图了

9、禅定模式

1、对于调试器的加强：数据流分析辅助

IntelliJIDEA v2020.1向调试器添加数据流分析辅助，它根据程序执行的当前状态预测和显示可能的异常，并始终为真/始终为假条件。

调试Java代码并到达断点时，IDE将根据程序的当前状态运行数据流分析，并在代码执行达到此点之前向您展示下一步将发生什么

简单点说，就是在调试那些复杂的代码时，IDE可以预先显示不变的那些调试值，让你能够更好的调试代码，如图

The screenshot shows the IntelliJ IDEA code editor with the file 'LeapYear.java' open. The code implements a method to determine if a year is a leap year. A break point is set at the start of the 'isLeapYear' method. The code is as follows:

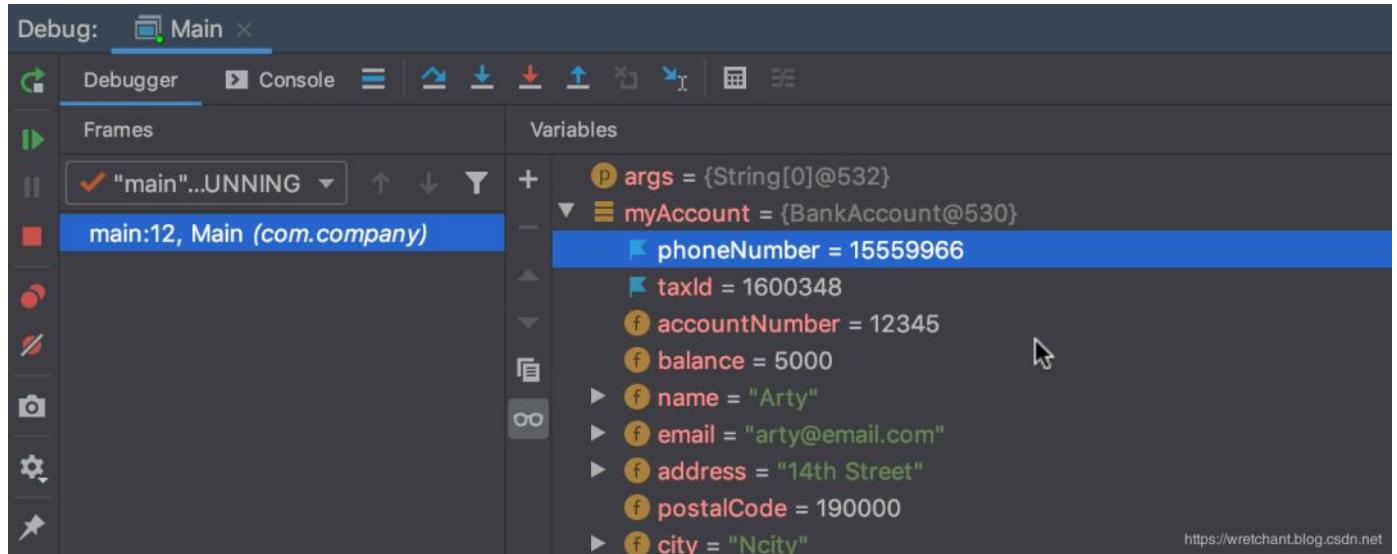
```
public static void main(String[] args) {
    System.out.println(getDaysInMonth(11, 2021));
}

public static boolean isLeapYear(int year) {    year: 2021
    if (year >= 1 = true && year <= 9999 = true ) {        year: 2021
        if (year % 4 == 0 = false && year % 100 != 0) {
            return true;
        } else return year % 4 == 0 = false && year % 100 == 0 && year % 400 == 0;
    }
    return false;
}
```

The line 'if (year >= 1 = true && year <= 9999 = true)' is highlighted with a blue selection bar. Two specific values are highlighted with red boxes: 'year: 2021' and 'year % 4 == 0 = false'. These represent the current state of the variables at the breakpoint, as predicted by the data flow analysis.

2、调试加强：属性置顶功能

这个改进不大，但是调试的时候很有用，就是说，你在调试的时候呢，有些对象的字段太多了，要去找他有时候还要翻页或者下拉很多，一般我们调试可能要走好多遍代码，你现在在第一次调试后，吧这个调试的字段置顶，以免老是要去找。



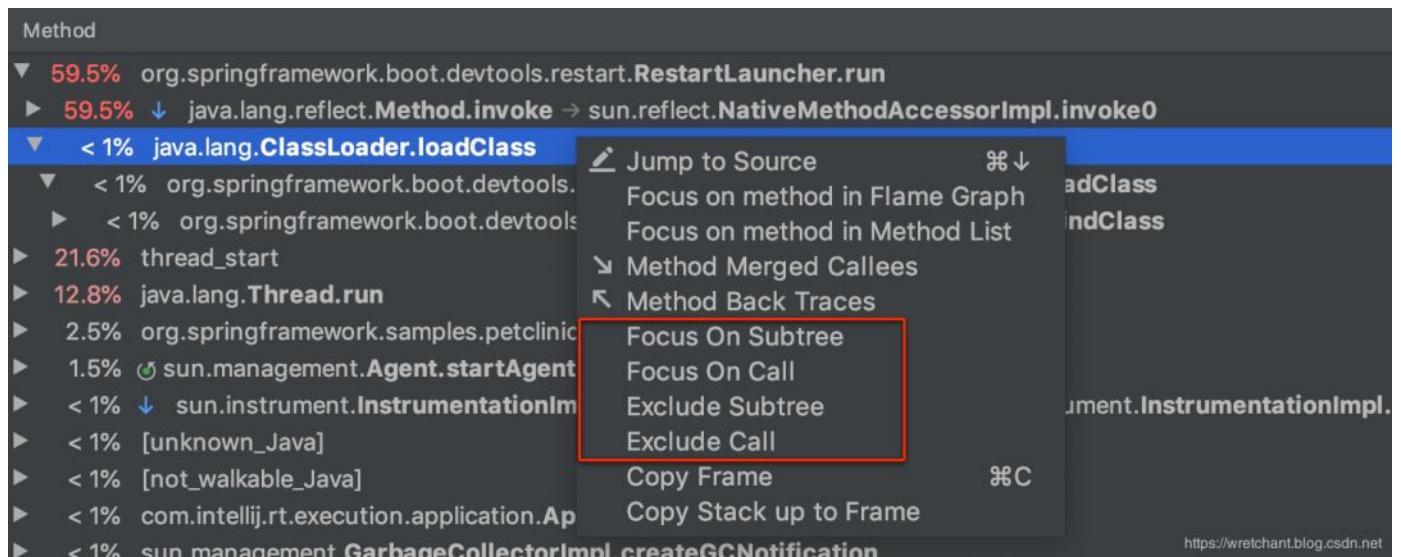
3、调试加强：IPV6 调试

现在IDEA 支持通过远程计算机 IPV6 进入到调试会话

4、性能分析的改进，剔除额外的东西

使用CPU 调试器进行性能分析的时候呢，可以通过隐藏一些方法啊什么的，或者只关注某个调用节点下的方法，来提供更高关注度的分析

提供了四个选项 1、只关注子集调用 2、只关注本调用 3、屏蔽子集调用 4、屏蔽本调用

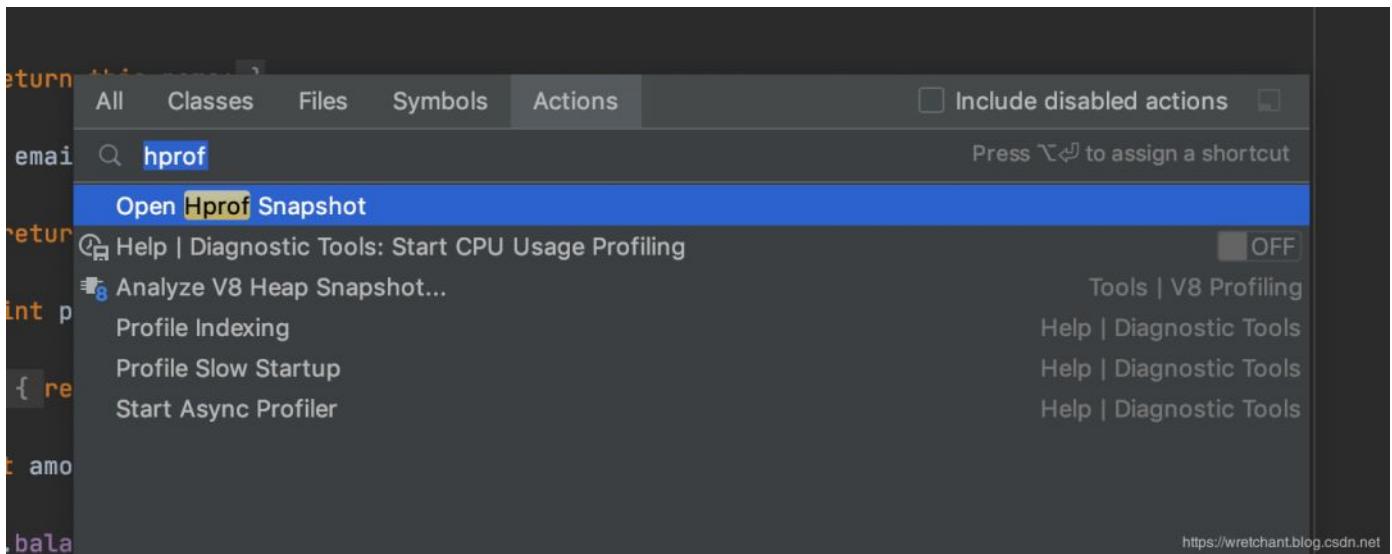


另外，IDEA 允许你绕过递归，让你能够进行更专注的性能分析

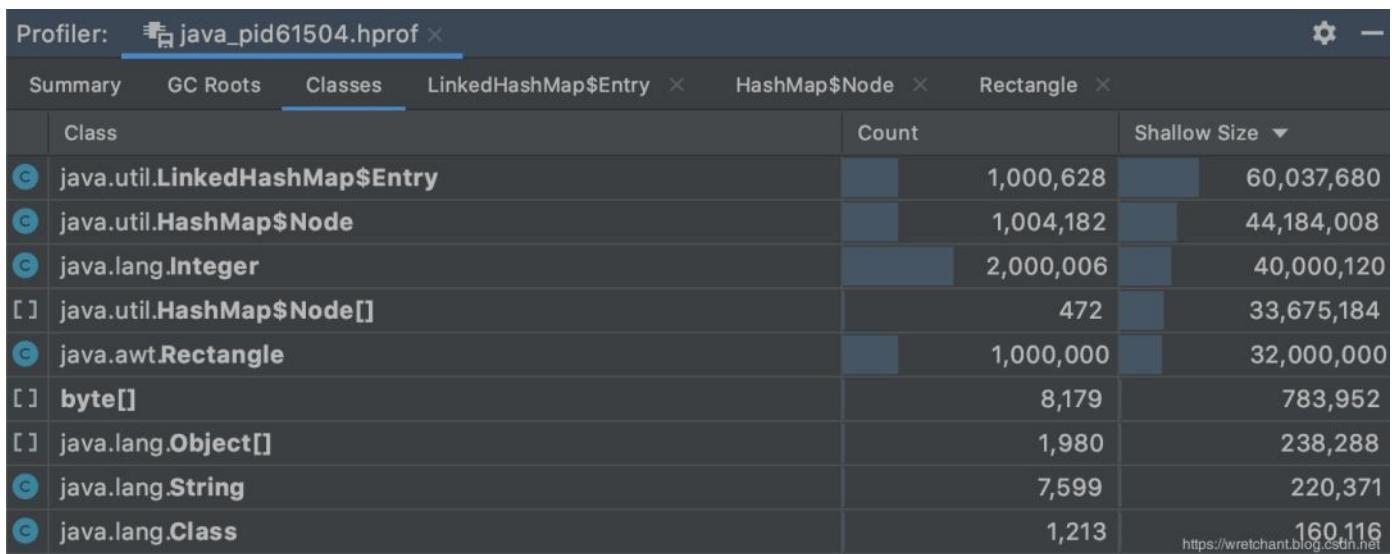
Method	Samples
▼ 100.0% nonapi.io.github.classgraph.concurrency.WorkQueue\$1.call	463
▼ 100.0% nonapi.io.github.classgraph.concurrency.WorkQueue.access\$000	463
▼ 100.0% nonapi.io.github.classgraph.concurrency.WorkQueue.runWorkLoop	463
▼ 94.4% io.github.classgraph.Scanner\$3.processWorkUnit	437
▼ 94.4% io.github.classgraph.Scanner\$3.processWorkUnit	437
► 90.7% io.github.classgraph.ClasspathElementZip.open	420
► 3.7% nonapi.io.github.classgraph.concurrency.SingletonMap.get	17
▼ 4.1% io.github.classgraph.Scanner\$5.processWorkUnit	19
▼ 4.1% io.github.classgraph.Scanner\$5.processWorkUnit	19
▼ 4.1% Collapse 1 recursive call ClasspathElementZip.scanPaths	19
► 1.3% io.github.classgraph.ClasspathElementZip.newResource	6
► < 1% io.github.classgraph.ClasspathElement.checkResourcePathWhiteBlackList	3
► < 1% nonapi.io.github.classgraph.scanspec.ScanSpec.dirWhitelistMatchStatus	2
< 1% java.util.concurrent.ConcurrentHashMap.putIfAbsent → java.lang.String.hashCode	1

5、支持读取内存快照文件

IDEA 现在支持打开 hprof 文件，也就是内存快照文件，并且打开内存快照文件不会占用你太多的内存，如果你要打开这种文件，你需要如图所示



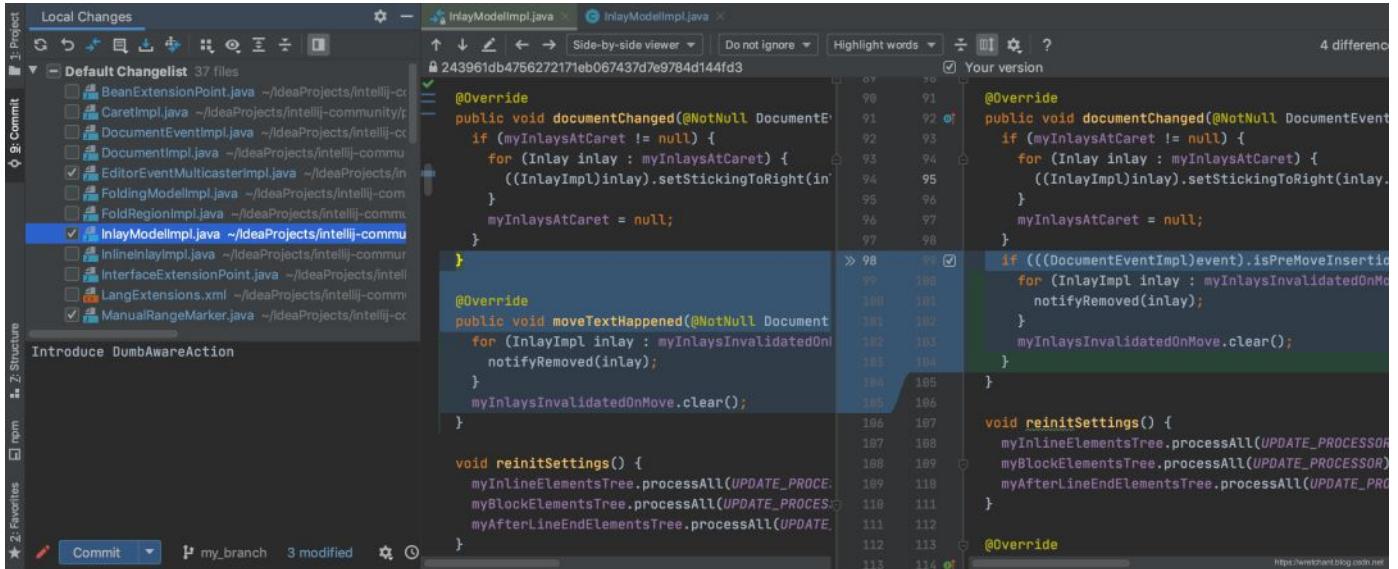
打开后呢，是这样的



就是暂时呢，只能进行简单的分析，后续的功能还在开发当中

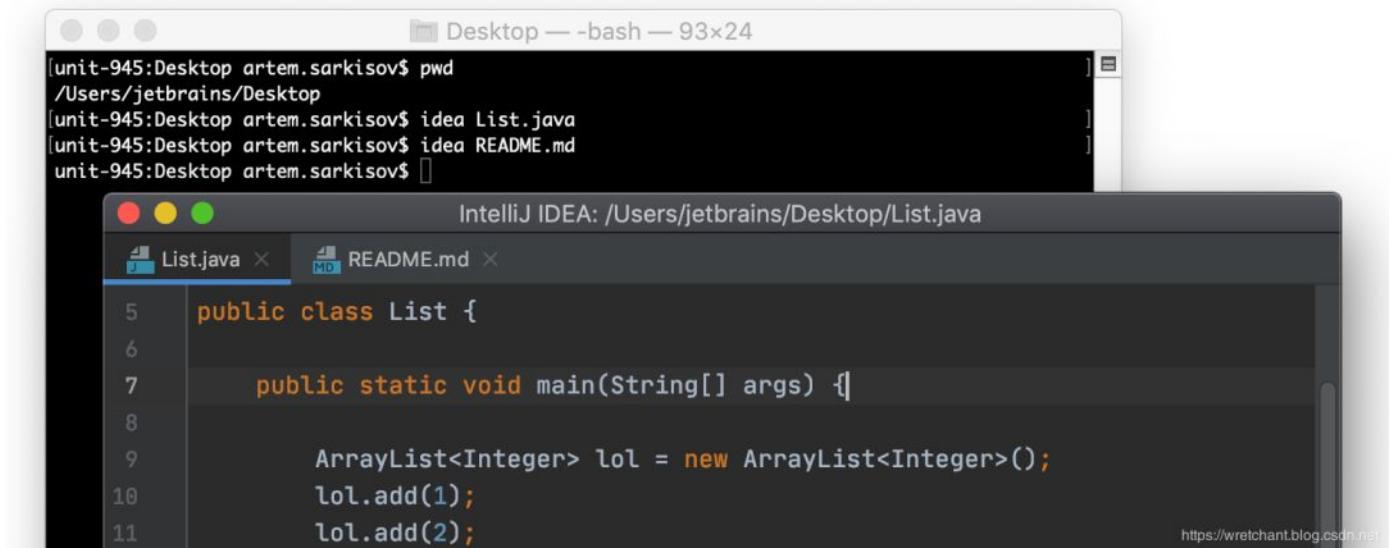
6、IDEA 变更了代码提交的界面

他大概是这个样子的



7、LightEdit 用来作为简单的文本编辑器

看着是个好功能，但是感觉还是有点鸡肋吧，因为他要通过命令行模式打开，并且有一定的功能阉割，然后就是打开快一点。



8、可以预览变更意图了

意思就是呢，我们通过快捷键可以打开一些IDEA提供的建议，比如这里有个警告，IDEA会提供一些建议的解决方案或者让你修改设置，现在呢，如果是需要更改代码，在改之前，IDEA可以让你预览一下改完了是个啥样子，不需要先改完，然后再取消了。

如果你不改快捷键呢，就是alt+空格了

9、禅定模式

用来消除分心的，解决之前的全屏模式的一些不足，让现在更好用了。

- END -

读到这里说明你喜欢本公众号的文章，欢迎 置顶（标星）本公众号 Java后端，这样就可以第一时间获取推送了~

推荐阅读

1. 如果面试官问 String，就把这篇文章丢给他！

[2. HashMap 在 JDK1.7 和 JDK1.8 中有哪些区别？](#)

[3. GitHub 移动端来了！](#)

[4. 一文读懂 HTTPS](#)



微信搜一搜

Q Java后端

[阅读原文](#)

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

IntelliJ IDEA 2019.3 这回真的要飞起来了，新特性抢先看！

局长 Java后端 2019-09-23



推荐阅读：面试官：说一说 Spring Boot 自动配置原理

IntelliJ IDEA 上周才公布下一个主要版本 2019.3 的 Roadmap，近日就发布了 IntelliJ IDEA 2019.3 的首个早期访问版本（即 EAP 版本），版本号为 2019.3 EAP build 193.2956.37。

EAP 版本的下载地址为：<https://www.jetbrains.com/idea/nextversion/>。如果是尝鲜的话，不妨下载来体验一下，生产环境建议别轻易尝试。

继续看看新版本带来了哪些有趣的改进。

新增平滑滚动（可选项）

团队表示下一个大版本的更新重点是提升性能，也将会解决各种可用性问题。为此他们推出了平滑滚动（Smooth scrolling）功能，让用户在使用鼠标滚轮查看内容时拥有更流畅的滚动体验。关于平滑滚动，最令人印象深刻的莫过于在微软在 Edge 浏览器上提供的平滑滚动体验，不知道 IDEA 实现的效果如何，欢迎有志之士发回使用反馈：）

启用“Smooth scrolling”选项后，相比于启用前，后者的滚动效果更佳顺滑和自然。 不过录制的 GIF 不能很好体现出“平滑滚动”的效果，可访问此链接进行观看动态效果（<https://youtu.be/MoVS6HOdeew>）

意图动作（Intention action）不会从建议列表中消失

这是对现有功能的继续打磨，此次更新引入了一项更有用的改进 —— 即使我们选择了某个意图动作然后取消对话框，IDE 也会在对话框中显示意图操作。

在此前的版本中，除非我们以某种方式修改文件，否则 IDE 不会二次显示同一个意图动作。

更方便查看字段的调用树

在解决各种大大小小的故障和不一致问题的同时，此版本还新增了查看字段调用树（Field Call Hierarchy）的功能。

在 IntelliJ IDEA 中，我们可以查看所选方法的调用者和被调用者的调用树（Ctrl+Alt+H），或查看所选类的父类和子类的调用树（Ctrl+H）。但当要查看一个字段的调用树时，这将变得十分不方便。为了理解所选字段的调用树，在此前的版本中，必须分析此字段的使用情况，并在该方法中导航至调用单个方法的调用树，这种做法十分不直观。因此，在即将发布的 2019.3 中，直接使用 Ctrl+Alt+H 快捷键即可查看字段的调用树。

```

    //from properties file
    private String accessTokenSecret;
    private String token;
    private String consumerKey;
    private String consumerSecret;
    private SecretKeySpec spec;

    //    private String _ = null;

    public TwitterOAuth() {
        this.timestampInSeconds = System.currentTimeMillis() / 1000;
        this.nonce = timestampInSeconds + (new Random().nextInt());
        loadProperties();
    }

```

全局搜索 (Find in path) 显示搜索结果的文件扩展名

IntelliJ IDEA 2019.3 将继续完善“全局搜索 (Find in path)”对话框功能。在此前的版本中，当我们在全局搜索中使用文件掩码 (file mask) 进行搜索时，搜索结果的文件扩展名会被隐藏。如果搜索结果中存在多个具有相同名称但文件扩展名不同的文件，则可能会造成混淆。

新版本修复了此问题，因此 IDE 会始终显示搜索结果的文件扩展名。

改进对 Maven 的支持

在 2019.3 版本中，IntelliJ IDEA 将能够自动配置打开、导入或创建 Maven 项目的设置，无需通过模态对话框 (Modal Dialog) 来手动配置设置。

此外，新版本还提升了 IDE 在编辑 POM.xml 文件时的性能，IDE 现在也已支持即时显示包含补全建议的对话框。

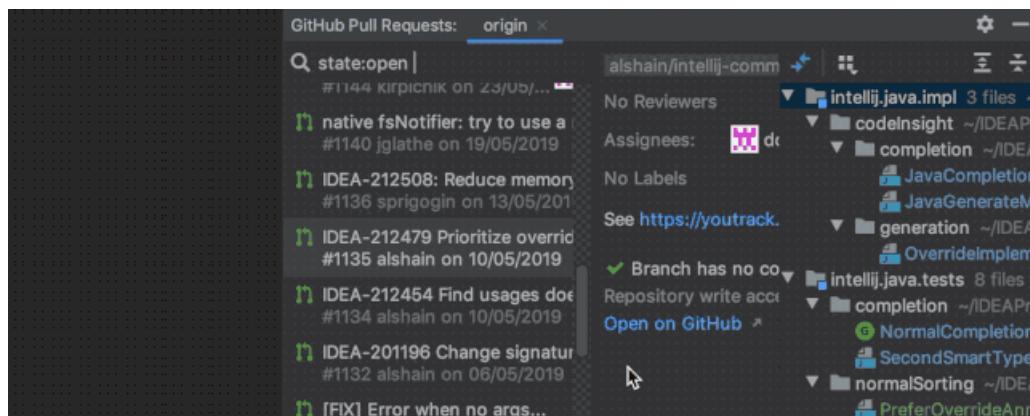
提升导入 Gradle 项目的性能

新版本还优化了导入 Gradle 项目的性能，以减少峰值内存消耗，这些改进对于大型项目来说意义重大。

更好地显示 GitHub PR 信息的时间轴

前段时间 IDEA 引入了对 GitHub PR 的初始支持，通过此功能我们可以查看项目所有 PR 的列表，并浏览它们的变化以及当前的状态。虽然此功能十分实用，但依然存在一些已知的限制，例如无法查看审查者提交的注释。

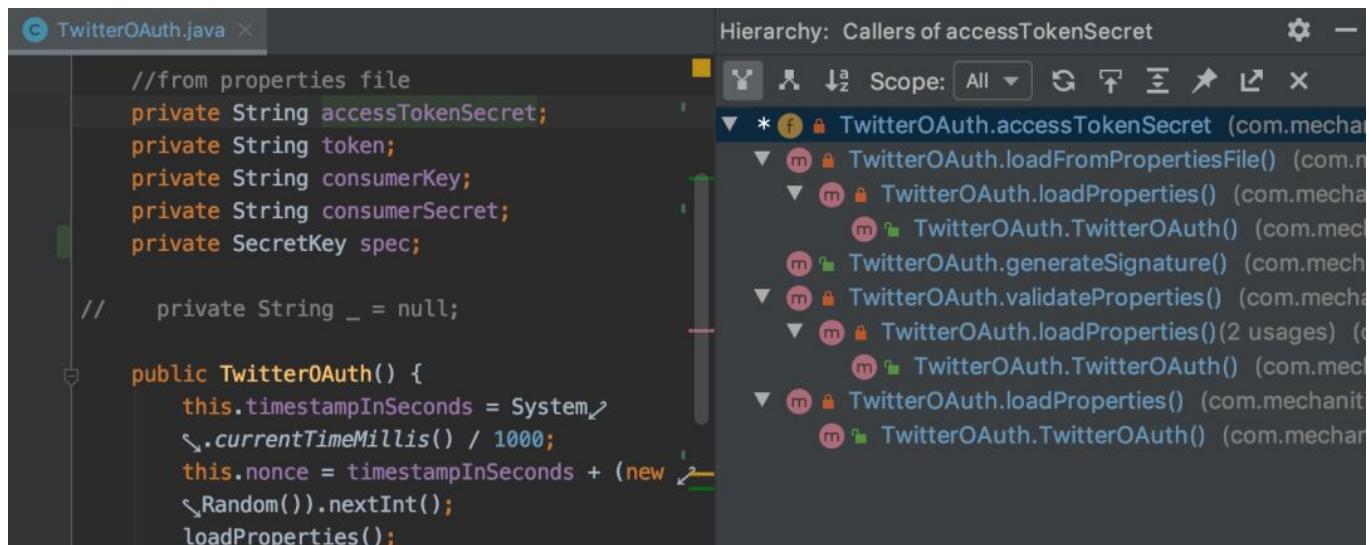
IntelliJ IDEA 2019.3 已将这个缺失的功能补充进来。具体来说就是，它将会在选定的 PR 中显示一个时间轴，其中包含有关 PR 的所有相关信息，例如注释、审查者和对 PR 所做的更新——基本上包含需要决定是否合并此 PR 的所有信息。



只需对 PR 进行双击，IDE 将在编辑器选项卡中显示注释。

重新修改过的 Clone 对话框

此版本改进了 Clone 对话框 (VCS | Get from Version control)。现在我们可以从对话框进行登录，或者如果已经处于登录状态，IDE 将立即预览按帐户或组织分组的所有 repo 的列表。



强制从忽略列表向 VCS 添加文件

在此前的版本中，如果文件位于忽略列表，则无法将文件添加到 VCS。IntelliJ IDEA 2019.3 对该限制进行了处理，现在即使文件位于 .gitignore（或 .hgignore）列表中，我们也可以将文件强制添加到 Git（或 Mercurial）。

JetBrains 运行时

默认情况下，IntelliJ IDEA 2019.3 将在 JetBrains Runtime 11（未经认证的 OpenJDK 11 分支）下运行。当然，JetBrains Runtime 8 也会继续提供（未经认证的 OpenJDK 8 分支）。

JBR 11（默认）已更新至 v11.0.4+12-b462.3：

- JetBrains Runtime 基于 OpenJDK 11.0.4
- 修复在 Windows 平台上的欢迎界面
- 检测到与 OS 的键盘布局冲突
- 修复编辑器中不正确的字体（斜体）

JBR 8（可选）已更新至 v1.8.0_222-release-1621-b1：

- JetBrains Runtime 基于 OpenJDK 8u222
- 修复出现在 macOS 10.15 Beta (19A501i) 上的崩溃问题

最后，EAP 版本每周都会发布更新，关于本次更新的详细内容请点此查看<http://suo.im/4qm7rk>

相关链接

[1] IDEA 的详细介绍：www.oschina.net/p/intellij-idea

[2] IDEA 的下载地址：<http://suo.im/5ovR3Z>

作者 | 局长

来源 | <https://www.oschina.net/news/109913/intellij-idea-starts-2019-3-early-access-program>

- END -

如果看到这里，说明你喜欢这篇文章，帮忙[转发](#)一下吧，感谢。微信搜索「web_resource」，关注后回复「进群」即可进入无广告交流群。



扫一扫上面的二维码图案，加我微信

推荐阅读

1. Java后端优质文章整理
2. 全网最牛掰的12306抢票神器
3. 面试官：说一说 Spring Boot 自动配置原理
4. 在浏览器输入 URL 回车之后发生了什么？
5. 接私活必备的 10 个开源项目



Java后端

长按识别二维码，关注我的公众号

喜欢文章，点个在看

[阅读原文](#)

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！



微信搜一搜



Java后端

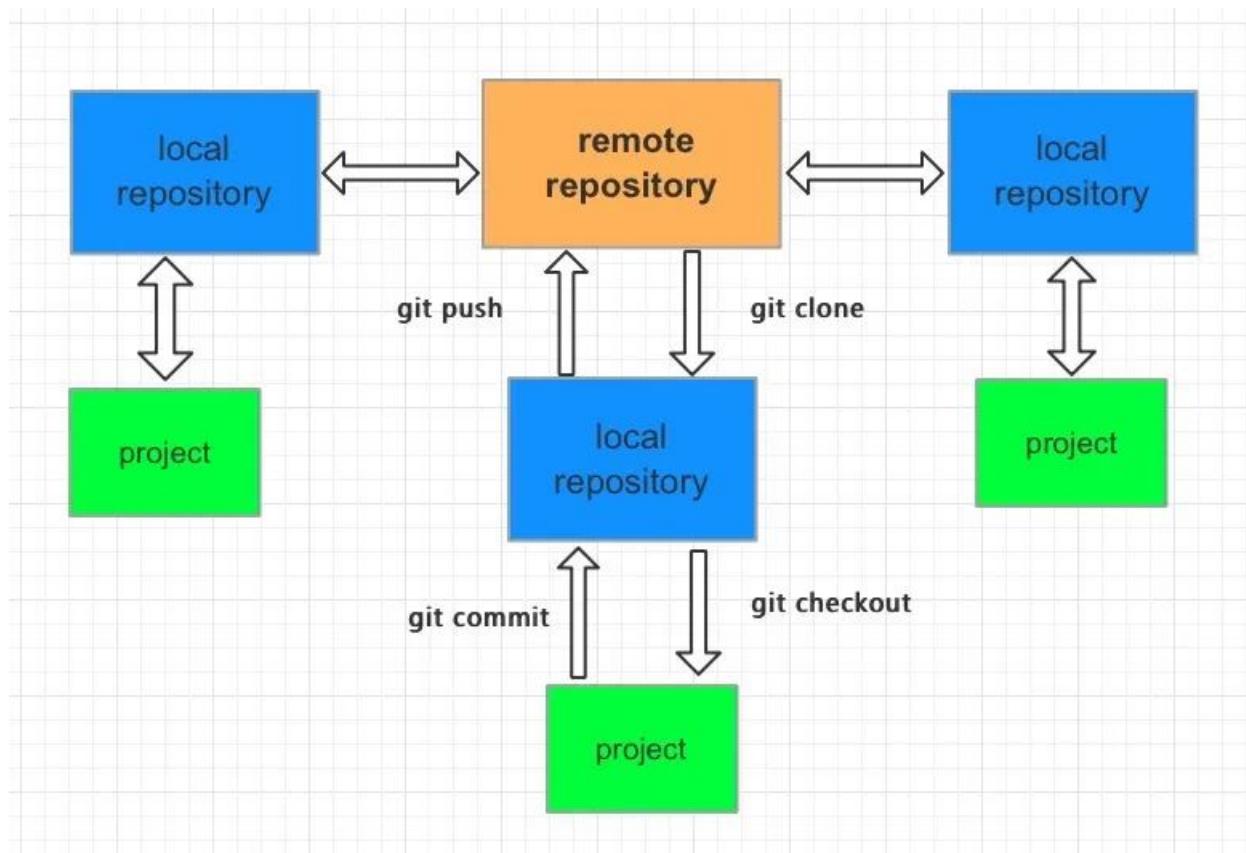
作者 | J'KYO

链接 | cnblogs.com/pejsidney/p/9199115.html

1、Git 简介

Git是目前流行的分布式版本管理系统。它拥有两套版本库，本地库和远程库，在不进行合并和删除之类的操作时这两套版本库互不影响。也因此其近乎所有的操作都是本地执行，所以在断网的情况下仍然可以提交代码，切换分支。git又使用了SHA-1哈希算法确保了在文件传输时变得不完整、磁盘损坏导致数据丢失时能立即察觉到。

git的基本工作流程：



- git clone：将远程的Master分支代码克隆到本地仓库
- git checkout：切出分支出来开发
- git add：将文件加入库跟踪区
- git commit：将库跟踪区改变的代码提交到本地代码库中
- git push：将本地仓库中的代码提交到远程仓库

Git 分支

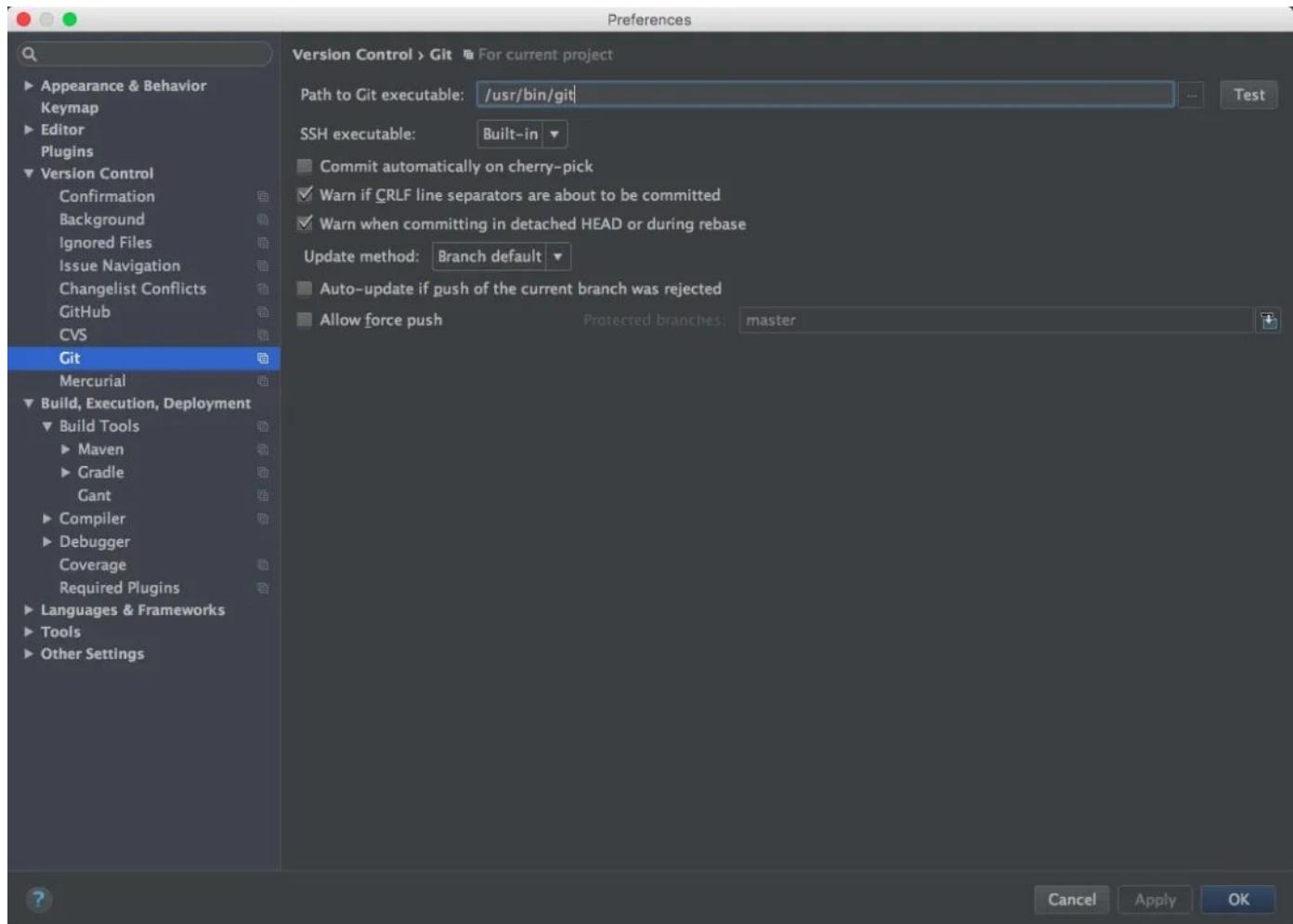
- 主分支
 - master分支：存放随时可供生产环境中的部署的代码

- develop分支：存放当前最新开发成果的分支，当代码足够稳定时可以合并到master分支上去。
- 辅助分支
 - feature分支：开发新功能使用，最终合并到develop分支或抛弃掉
 - release分支：做小的缺陷修正、准备发布版本所需的各项说明信息
 - hotfix分支：代码的紧急修复工作

2、Git在IntelliJ IDEA下的使用

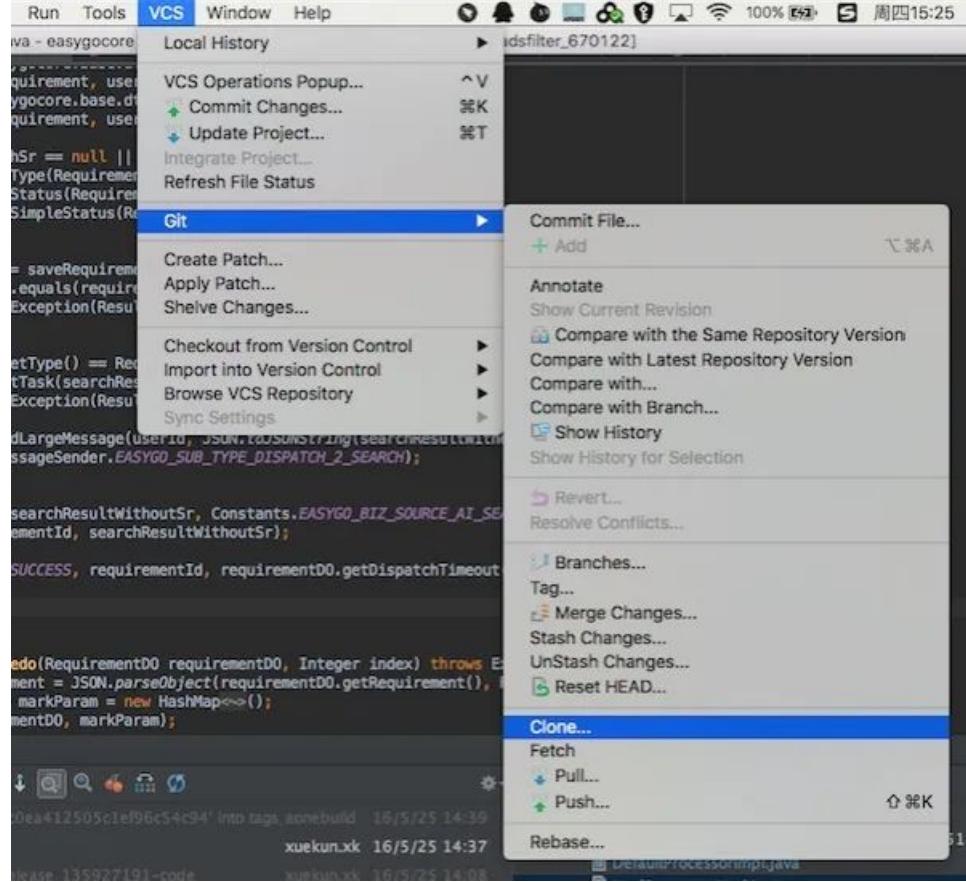
2.1、IntelliJ IDEA下配置 Git

- 本地安装好git，并配置合理的SSH key，具体看[这里](#)
- IntelliJ IDEA->Performance->Version Control->git 将自己安装git的可执行文件路径填入Path to Git executable，点击Test测试一下

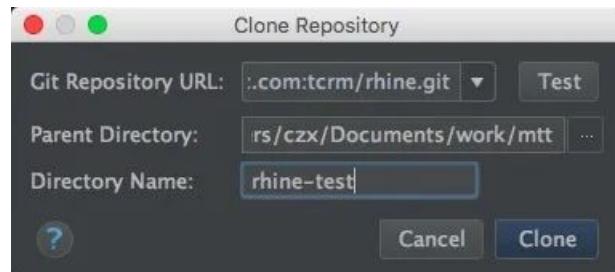


2.2、git clone

- VCS->Git->Clone

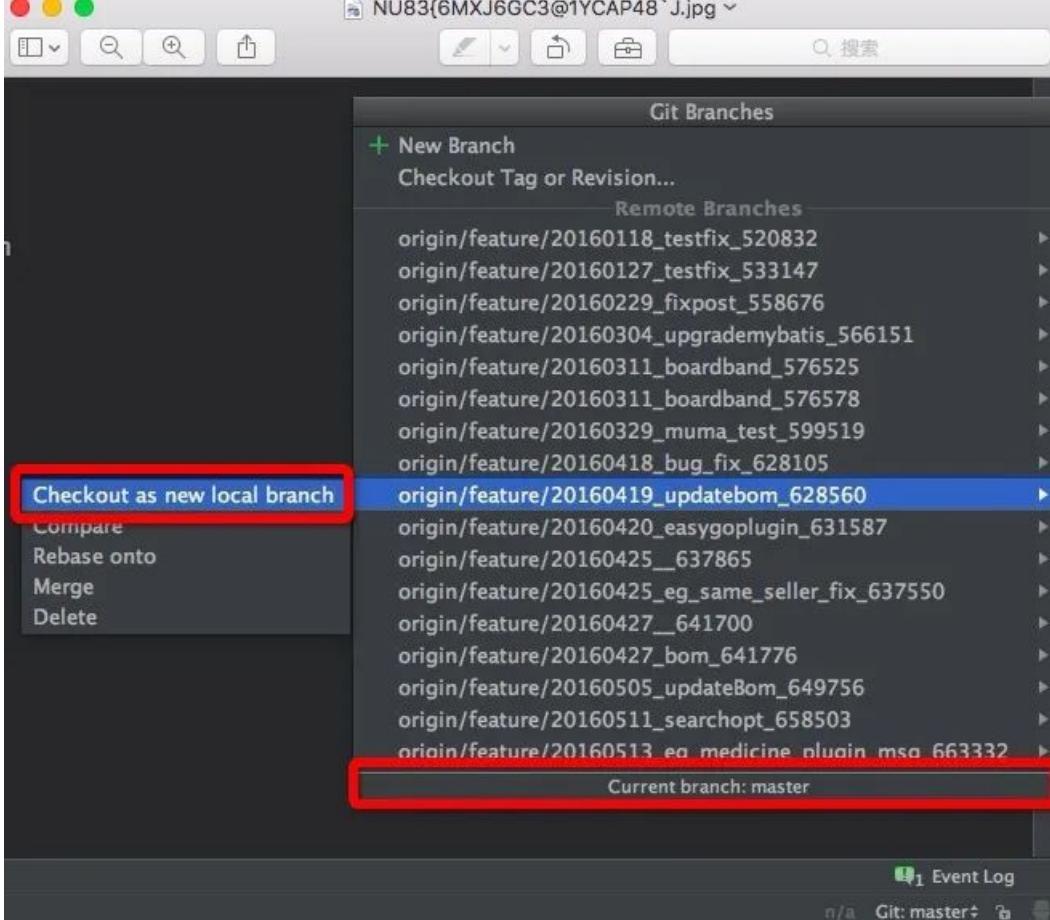


- 输入你的远程仓库地址,点击测试一下地址是否正确

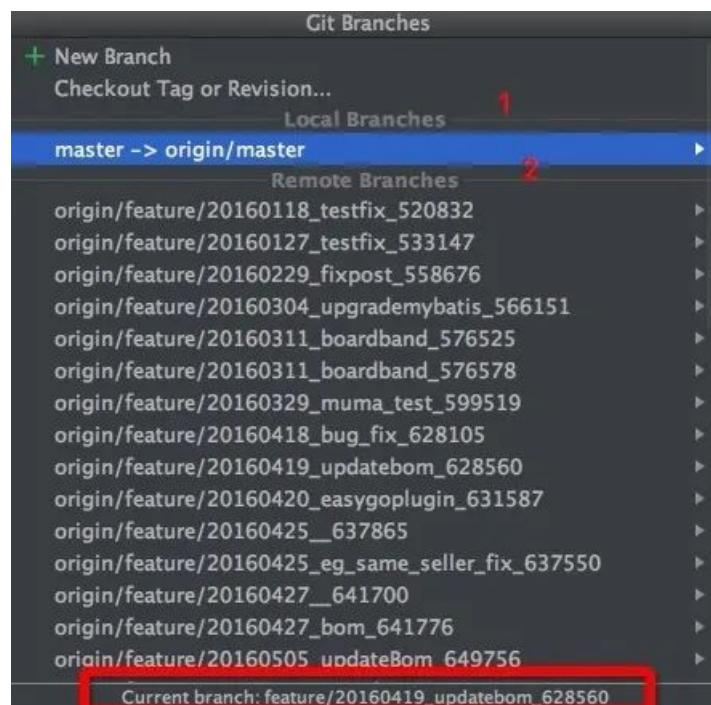


2.3、git checkout

- 在IntelliJ IDEA右下角有一个git的分支管理, 点击。选择自己需要的分支, checkout出来



- checkout出来，会在底端显示当前的分支。其中1显示的为本地仓库中的版本，2为远程仓库中的版本

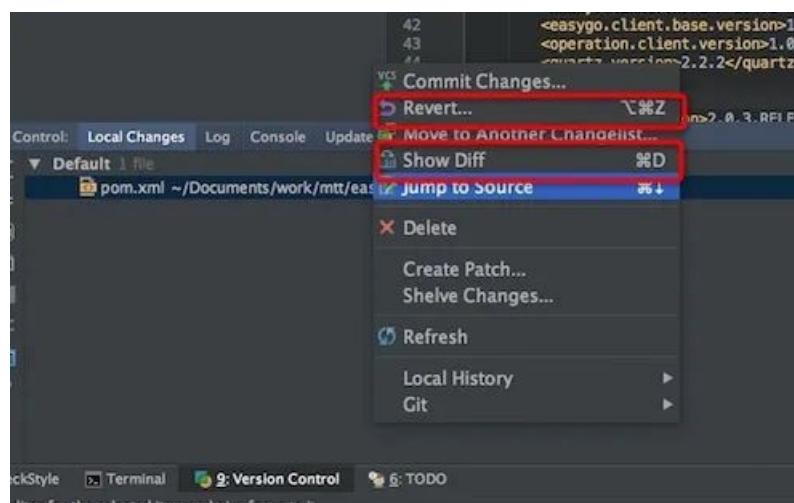


- 点击IDE的右上角的向下箭头的VCS，将分支的变更同步到本地



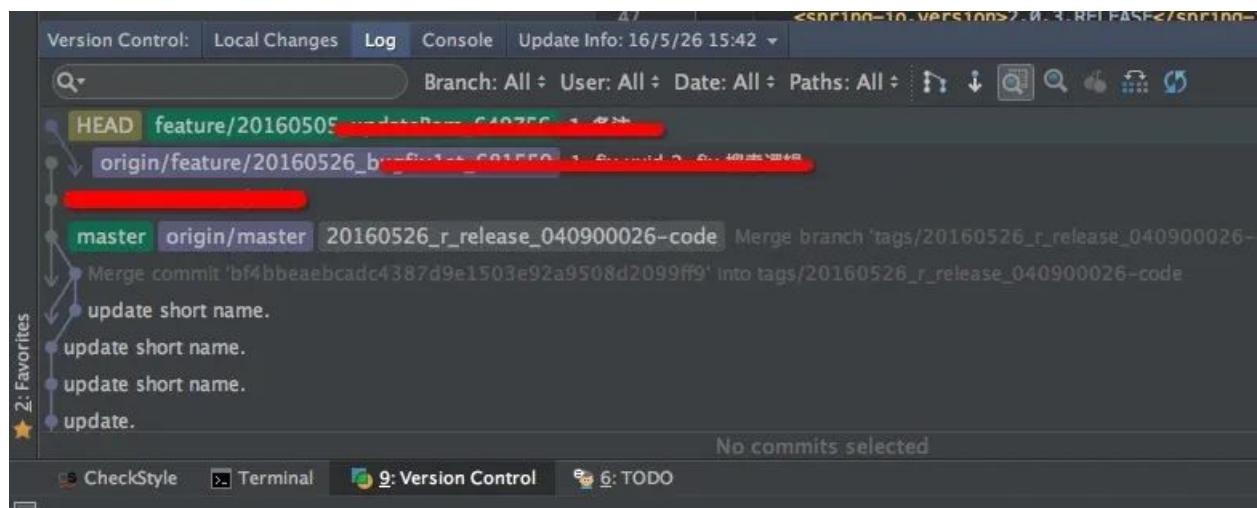
2.4、git diff

- 在local changes中选中要比对的文件，右键选择show diff便可以查看文件的变动。或者选择Revert放弃文件的改动



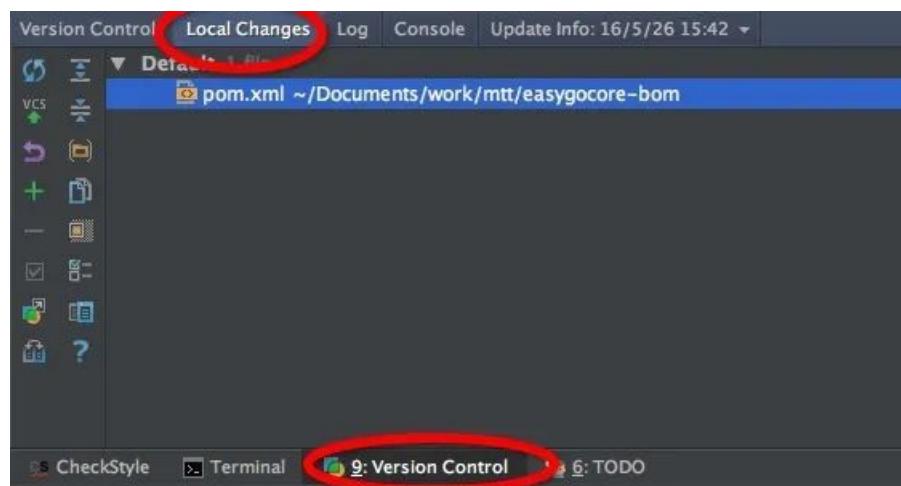
2.5、git log

- 在Version Control下选择Log，可以查看提交历史

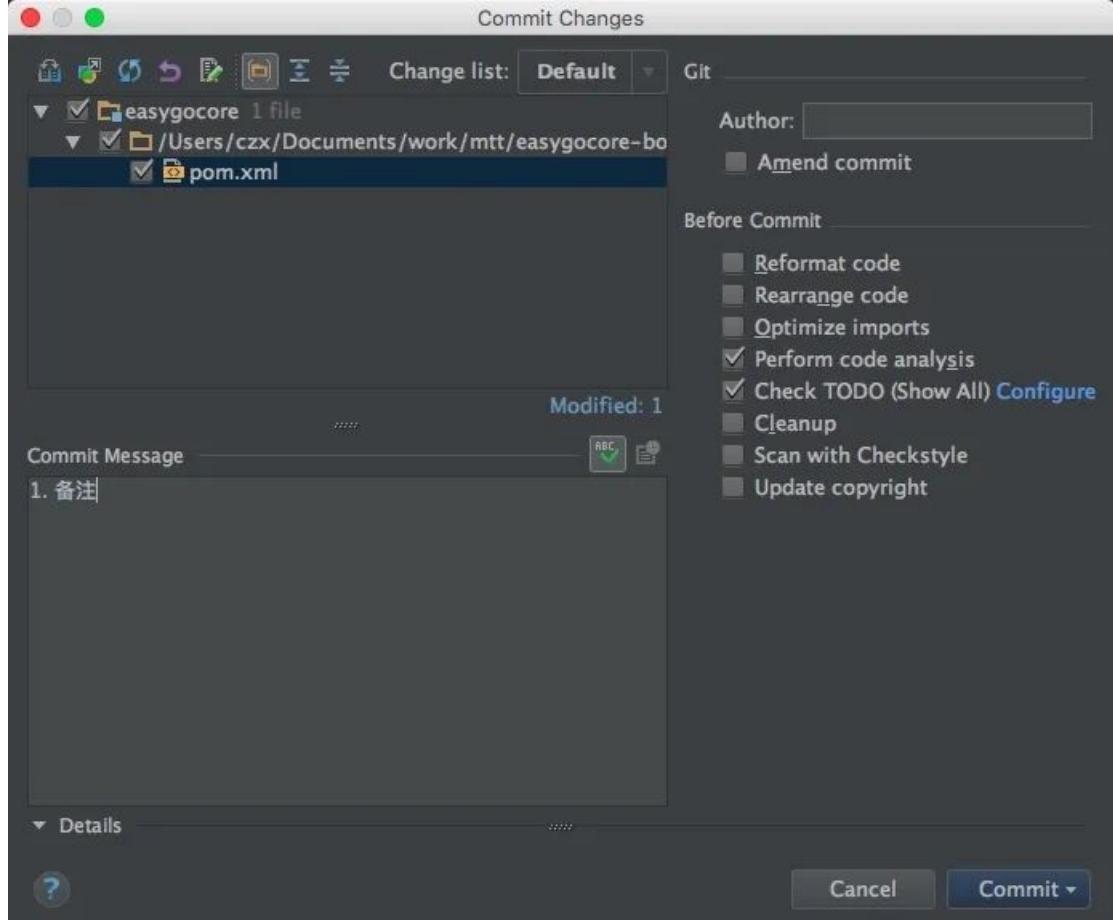


2.6、git commit

- 默认导入的工程已经git add加入库跟踪区了
- 随便修改一下pom.xml文件，其修改的文件会显示在Version Control中的local changes下

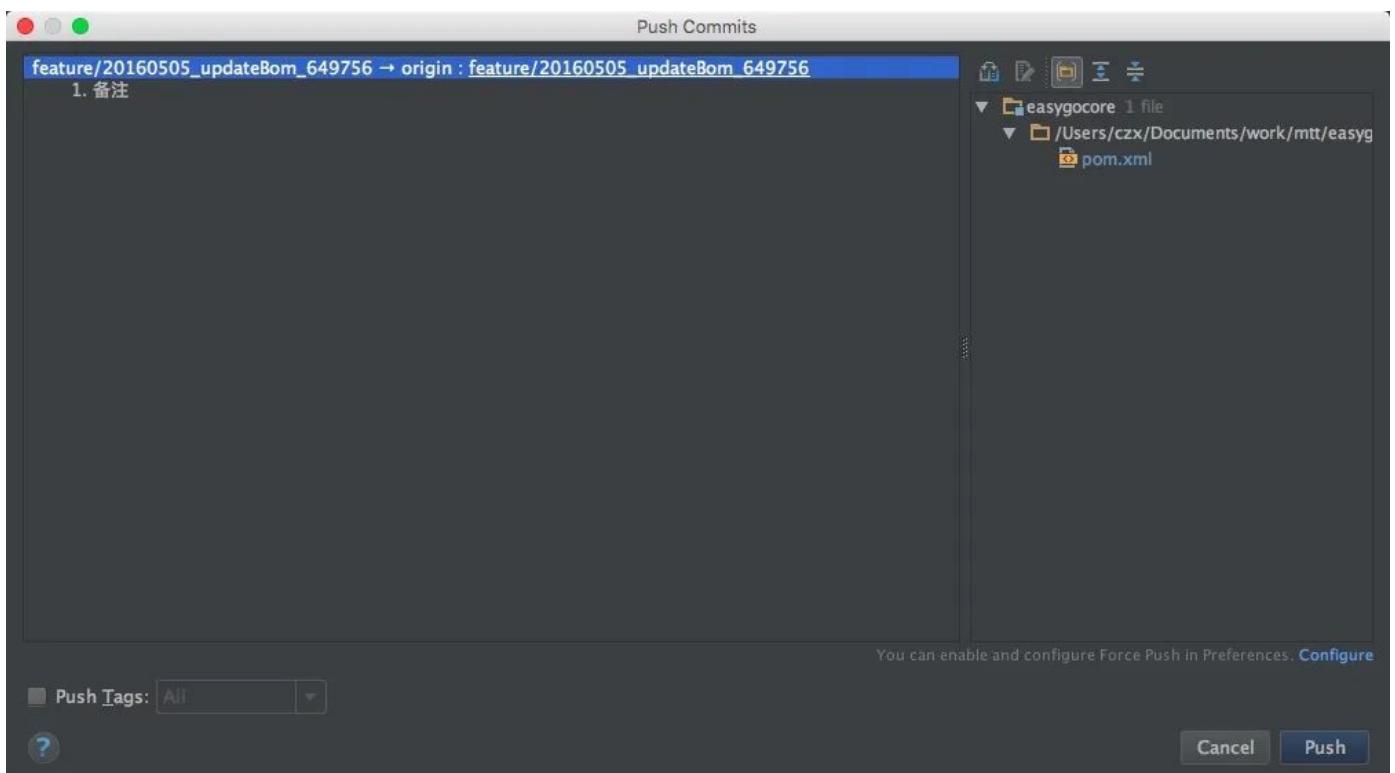


- 点击IDE右上角的向上箭头的VCS， git commit, 写上日志提交到本地代码库中



2.7、git push

- VCS->Git->Push 将本地代码提交到远程仓库



2.8、在Idea命令行使用git

常用命令请参考:

收藏了！IntelliJ IDEA 快捷键 Windows 版本

IntelliJ IDEA 常用快捷键 - Mac版本

推荐阅读

- 1.** 如何写出让同事无法维护的代码?
- 2.** 用好 Git 和 SVN, 轻松驾驭版本管理
- 3.** 如何使用 Java 灵活读取 Excel 内容 ?
- 4.** IntelliJ IDEA 快捷键 Windows 版本



[阅读原文](#)

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

IntelliJ IDEA 常用快捷键 - Mac版本

Simple_Zz Java后端 3月8日



作者 | Simple_Zz

博客 | blog.csdn.net/love284969214

前言

Windows版本的IDEA编辑器快捷键整理可以点击链接：[IntelliJ IDEA 常用快捷键总结](#)查看，本文中总结常用的Mac系统下的快捷键，首先阅读一下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 + ↓)
⇥ —> 右制表符(Tab键)
⇤ —> 左制表符(Shift + Tab)
⌫ —> Escape(Esc)
End —> Fn + →
Home —> Fn + ←

1. 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、hashCode、equals、toString、构造函数等)

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

2. 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)

3. 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

4. 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

5. 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

6. 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

7. 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

8. 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

9. 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

10 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)

如果喜欢本篇文章, 欢迎[转发](#)、[点赞](#)、[留言](#)。关注订阅号「Java后端」, 回复「技术博文」即可获取更多图文教程、技术博文。

推荐阅读

[1. 盘点那些改变过世界的代码](#)

[2. 基于token的多平台身份认证架构设计](#)

[3. 少侠! 如何写一手好 SQL?](#)

[4. 写给大忙人看的操作系统](#)



[阅读原文](#)

声明: pdf仅供学习使用, 一切版权归原创公众号所有; 建议持续关注原创公众号获取最新文章, 学习愉快!

IntelliJ IDEA新增禅模式和LightEdit模式

Java后端 2月13日

以下文章来源于开源中国，作者局长



开源中国
为开发者服务

>

IntelliJ IDEA 2020.1 的第二个早期访问版本已发布，新的 EAP 构建对调试器和事件探查器 (Profiler) 进行了改进，并引入了新的提交工具窗口 (Commit toolwindow) 以及禅模式 (Zen Mode)。

用于调试器的数据流分析协助功能

IntelliJ IDEA 2020.1 向调试器添加了数据流分析协助功能 (dataflow analysis assistance)，此功能根据程序执行的当前状态预测并显示可能的异常以及始终为真/假的条件。

当我们调试 Java 代码并到达断点时，IDE 将基于程序的当前状态运行数据流分析，并向我们显示在代码执行到该断点之前下一步将发生的情况：

```
LeapYear.java
4
5 public static void main(String[] args) {
6     System.out.println(getDaysInMonth(11, 2021));
7 }
8
9 public static boolean isLeapYear(int year) { year: 2021
10 if (year >= 1 = true && year <= 9999 = true) { year: 2021
11     if (year % 4 == 0 = false && year % 100 != 0) {
12         return true;
13     } else return year % 4 == 0 = false && year % 100 == 0 && year % 400 == 0;
14 }
15     return false;
16 }
```

折叠递归调用

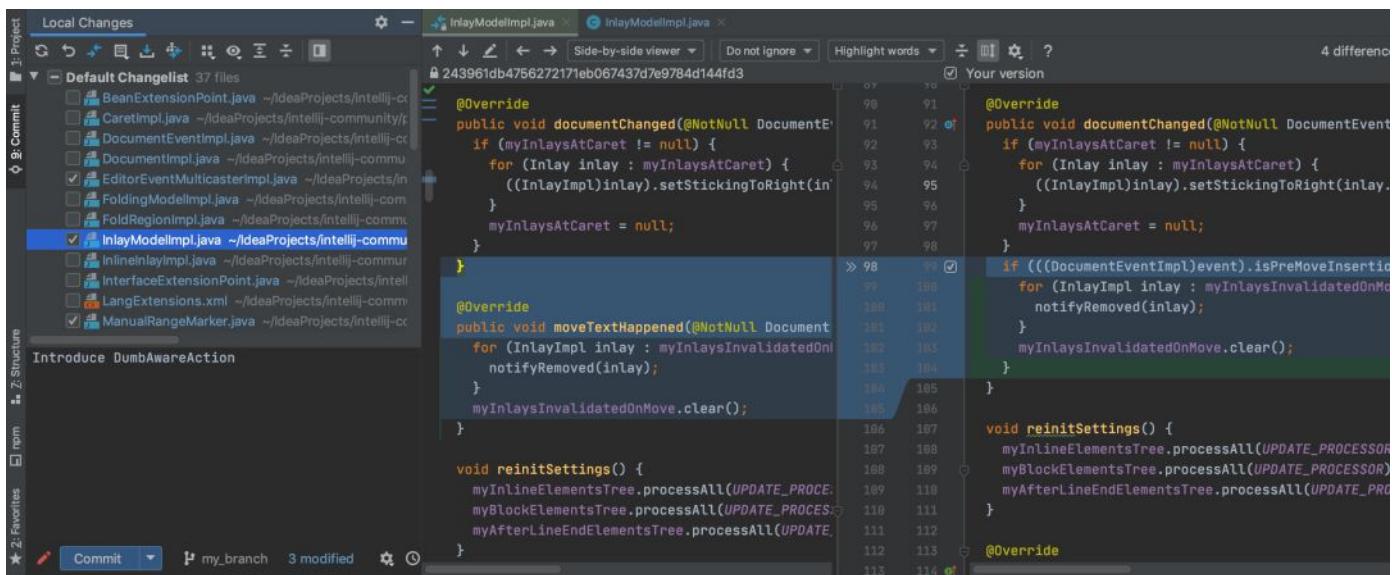
当在堆栈中的更高层调用同一方法时，IntelliJ IDEA 现在可以检测到递归调用。现在，IDE 会建议开发者将这些调用从子树中取出，从而可以绕过递归，并专注于消耗大部分资源的方法以及它们进行的调用。

递归调用在“调用树”选项卡中用以下新的特殊图标表示：

Method	Samples
▼ 100.0% nonapi.io.github.classgraph.concurrency.WorkQueue\$1.call	463
▼ 100.0% nonapi.io.github.classgraph.concurrency.WorkQueue.access\$000	463
▼ 100.0% nonapi.io.github.classgraph.concurrency.WorkQueue.runWorkLoop	463
▼ 94.4% io.github.classgraph.Scanner\$3.processWorkUnit	437
▼ 94.4% io.github.classgraph.Scanner\$3.processWorkUnit	437
► 90.7% io.github.classgraph.ClasspathElementZip.open	420
► 3.7% nonapi.io.github.classgraph.concurrency.SingletonMap.get	17
▼ 4.1% io.github.classgraph.Scanner\$5.processWorkUnit	19
▼ 4.1% io.github.classgraph.Scanner\$5.processWorkUnit	19
▼ 4.1% Collapse 1 recursive call ClasspathElementZip.scanPaths	19
► 1.3% io.github.classgraph.ClasspathElementZip.newResource	6
► < 1% io.github.classgraph.ClasspathElement.checkResourcePathWhiteBlackList	3
► < 1% nonapi.io.github.classgraph.scanspec.ScanSpec.dirWhitelistMatchStatus	2
< 1% java.util.concurrent.ConcurrentHashMap.putIfAbsent → java.lang.String.hashCode	1

新的 commit 工具窗口

在 2020.1 中为 Commit UI 提供了一个新的工具窗口。看起来如下：



新的 commit 工具窗口包含"Local Changes"和"Shelf"两个选项卡。该工具窗口涵盖了所有与提交有关的任务，例如检查差异、选择要提交的文件和代码块以及输入 commit 消息。

禅模式 (Zen Mode)

此版本添加了新的禅模式，以消除可能的干扰并帮助开发者完全专注于代码上。本质上，这种新模式结合了免打扰模式和全屏模式，因此不必每次想要进入或退出它们时都启用或禁用这两种模式。

要启用禅模式，请跳转至 View | Appearance | Enter Zen Mode，或从“快速切换方案”弹出窗口中选择它(Ctrl+` | View mode | Enter Zen Mode)

LightEdit 模式

顾名思义，这就是一个轻量的编辑模式，专用于打开和编辑文件。LightEdit 模式允许我们在简单的编辑器窗口中打开文件，而无需创建或加载项目。这也是对不少开发者希望将 IntelliJ IDEA 作为通用文本编辑器的要求的回应。

尝试 LightEdit 模式最简单的方式是通过命令行打开文件，如下所示：

```
[unit-945:Desktop artem.sarkisov$ pwd  
/Users/jetbrains/Desktop  
[unit-945:Desktop artem.sarkisov$ idea List.java  
[unit-945:Desktop artem.sarkisov$ idea README.md  
[unit-945:Desktop artem.sarkisov$ ]
```

IntelliJ IDEA: /Users/jetbrains/Desktop/List.java

List.java × README.md ×

```
5 public class List {  
6  
7     public static void main(String[] args) {  
8  
9         ArrayList<Integer> lol = new ArrayList<Integer>();  
10        lol.add(1);  
11        lol.add(2);
```

下载地址：<https://www.jetbrains.com/idea/nextversion>

推荐阅读

1. 远程办公的开始，也是进入 BAT 的开始 ...
2. 40 道 Java 多线程面试题及答案
3. 安利一款 IDEA 中强大的代码生成利器
4. 如何获取靠谱的新型冠状病毒疫情



[阅读原文](#)

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

IntelliJ IDEA 构建maven多模块工程项目

Java后端 2019-09-05

点击上方蓝色字体，选择“标星公众号”

优质文章，第一时间送达

链接 | blog.csdn.net/sinat_34344123

食用前须知

本文以a b c 三个模块为例来搭建项目,以达到通俗易懂的初衷

模块a --- 基模块,就是人们常说的parent

模块b --- 其他模块都需要使用的一些工具,比如时间工具,json工具等

模块c --- 项目主要的内容,一般为聚合工程

先简单讲一下maven的一些特点

1. 继承

这个可以理解为java中的继承类似,父类定义的东西,子类如果你想用就拿过来用就可以;

2. 依赖

依赖就相当于我们java中的导包,二者有着异曲同工之妙;

你想用的东西只需要告诉maven它在哪就可以,它会自动帮你找过来给你用

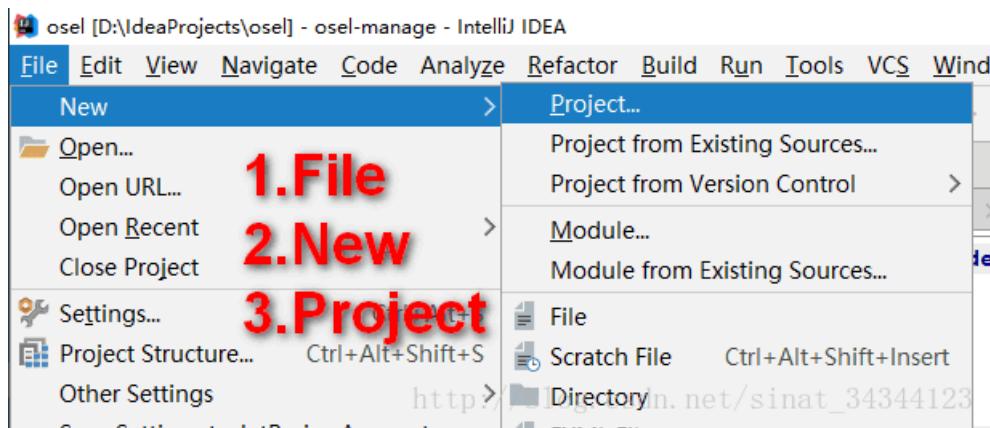
3. 聚合

这个暂时我还没找到java中能与之对应的原型;

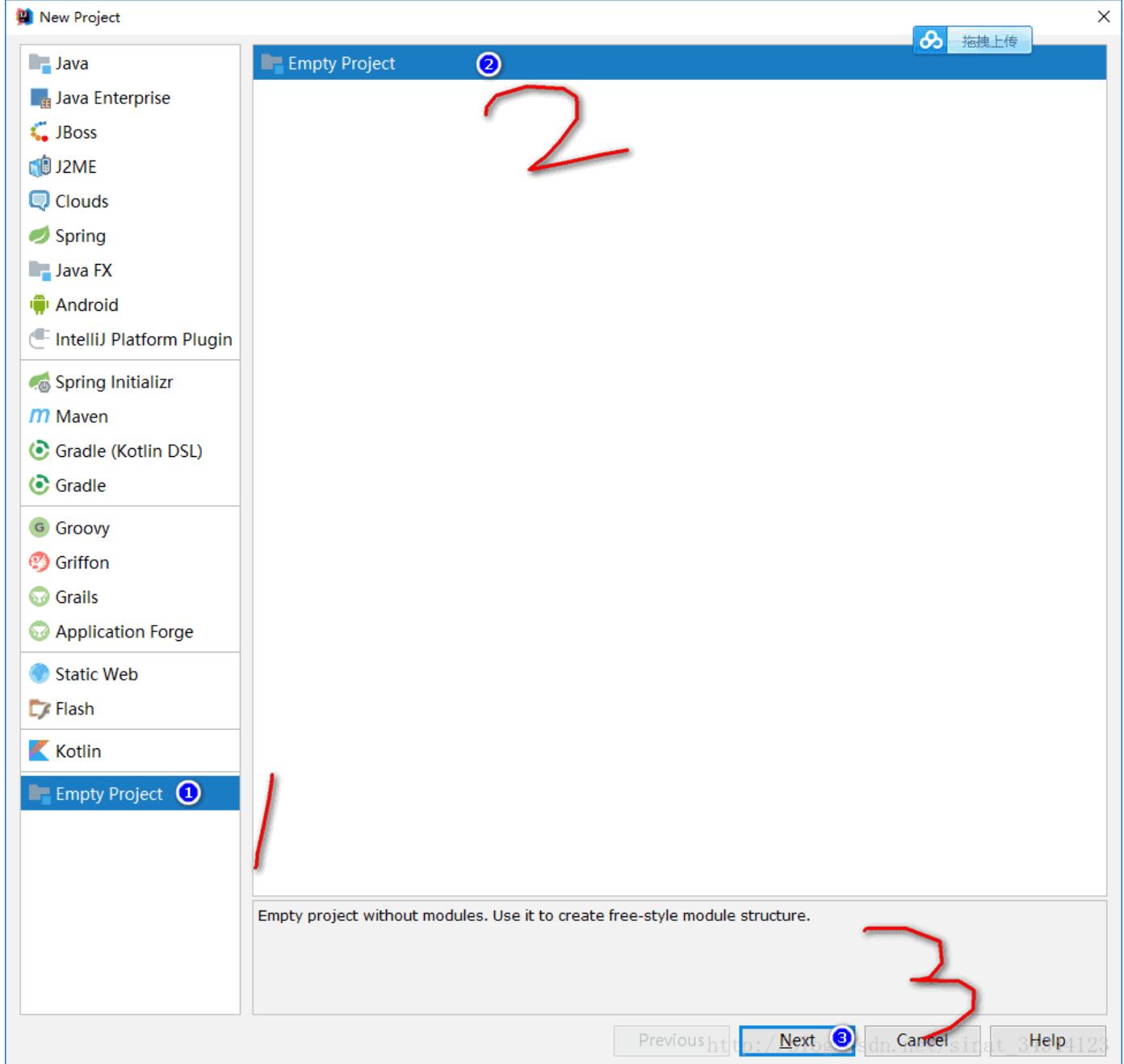
maven聚合是将多个模块组装在一起,相互协调依赖运行;

创建步骤 (详细多图)

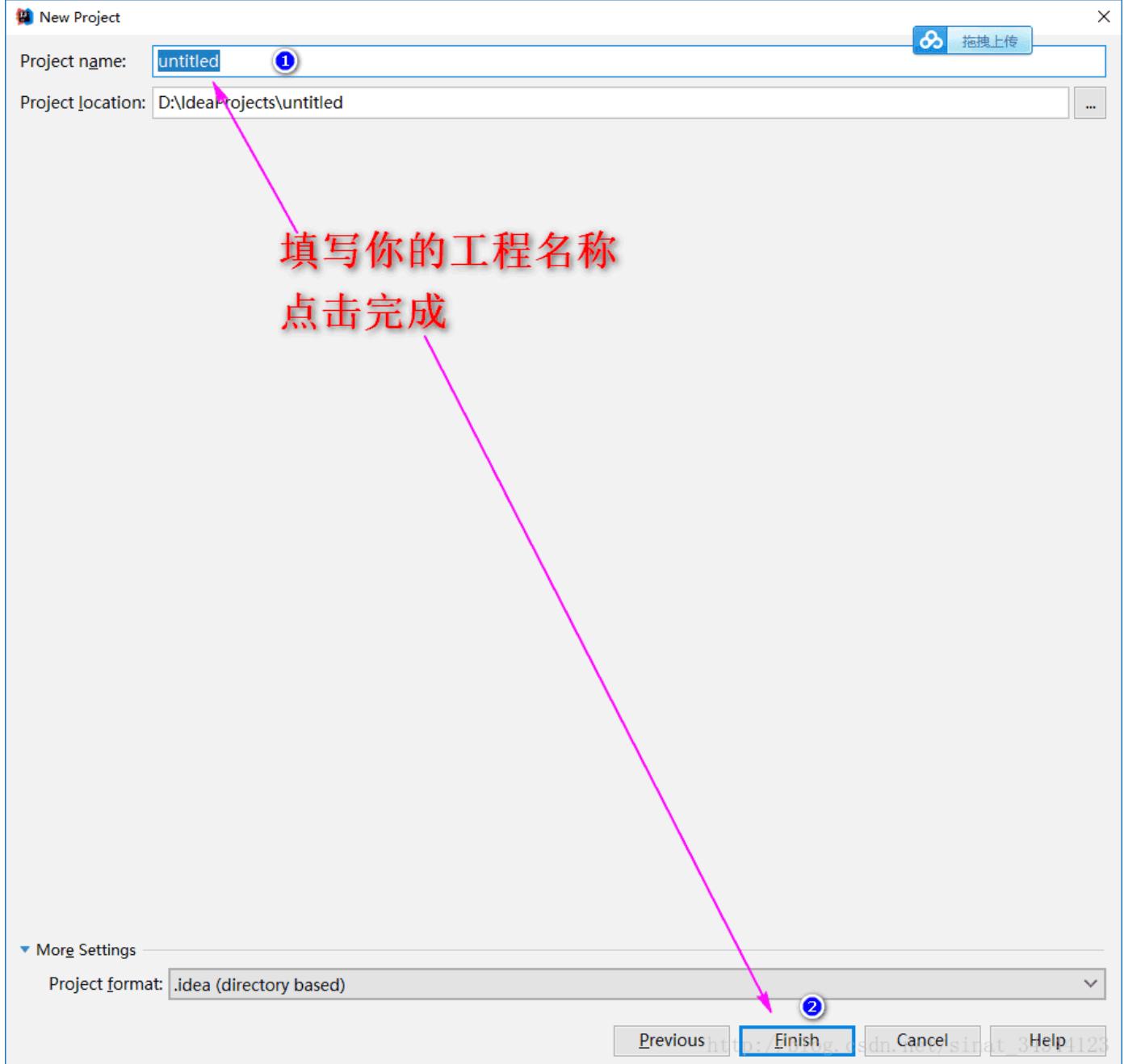
1. 创建一个空项目



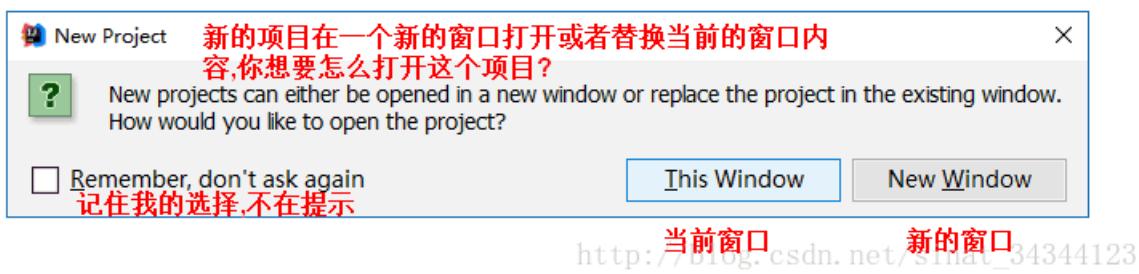
file - new - project 一个空的项目



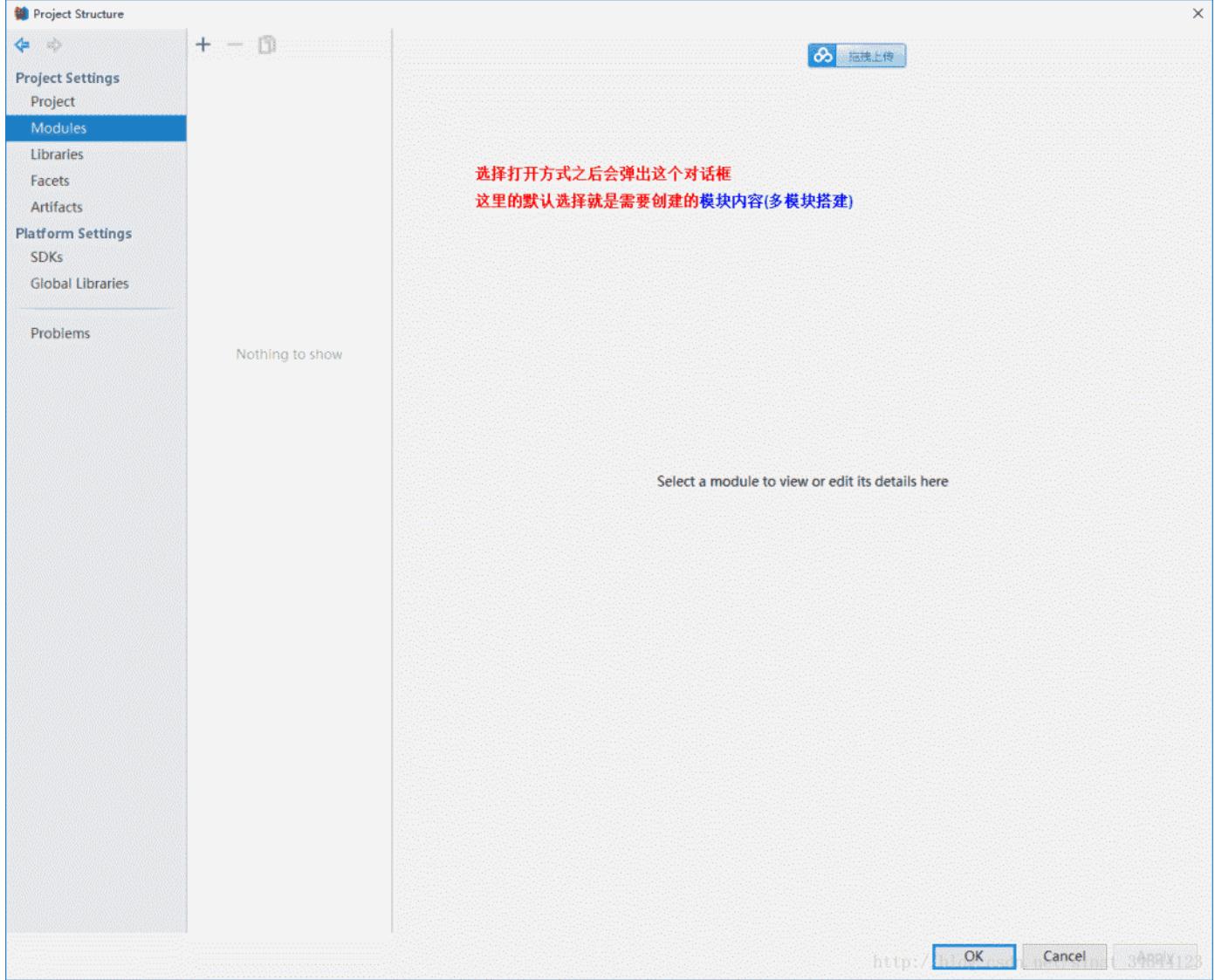
填写项目名称



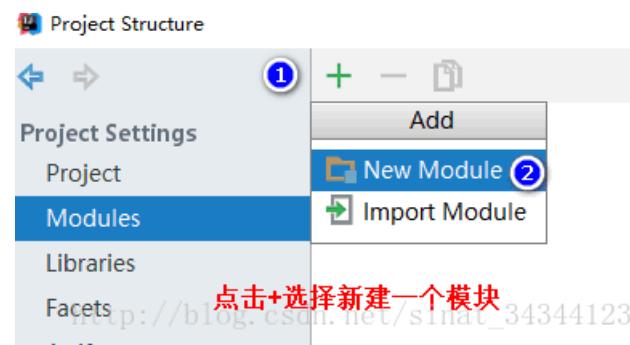
选择要打开项目的方式



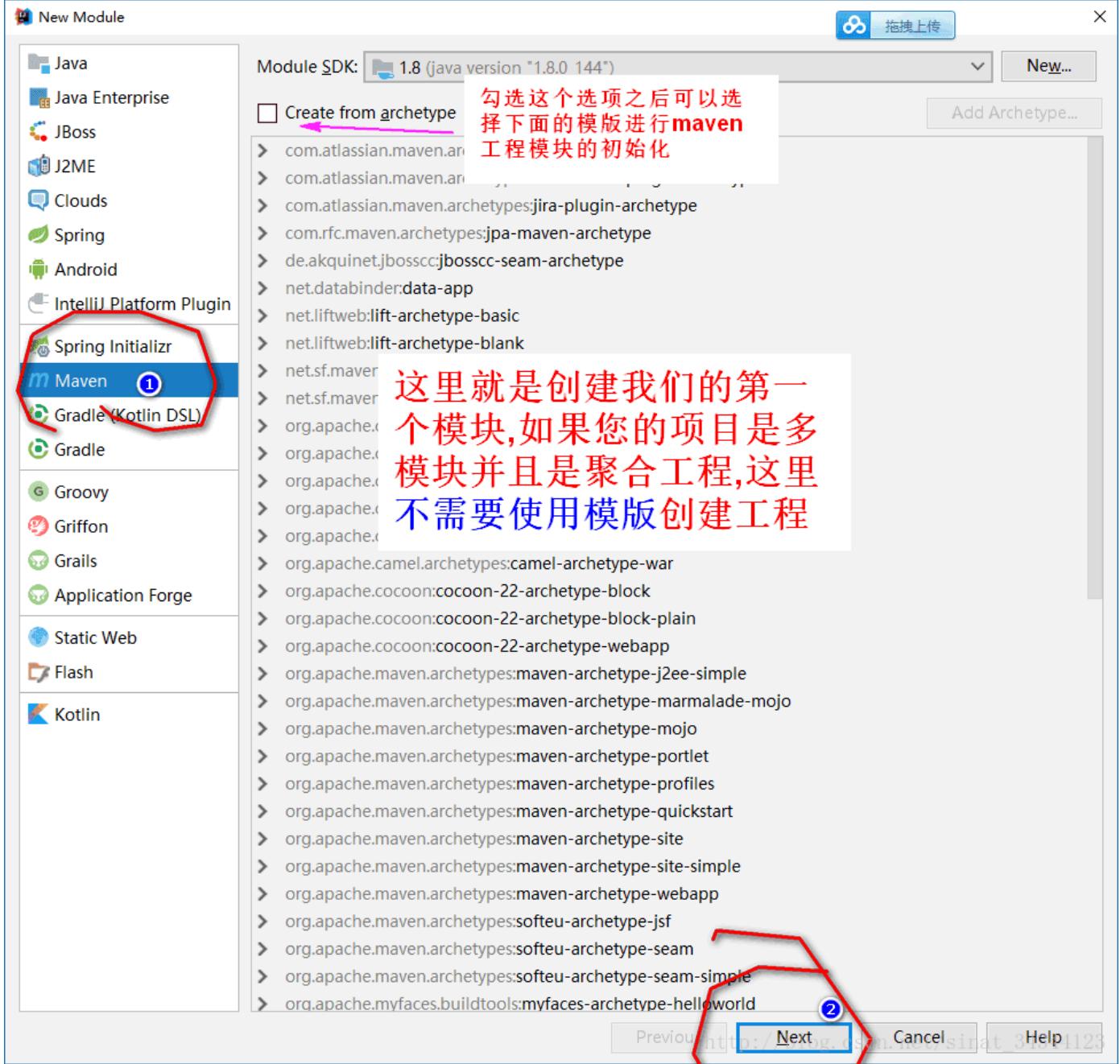
2. 创建第一个模块a



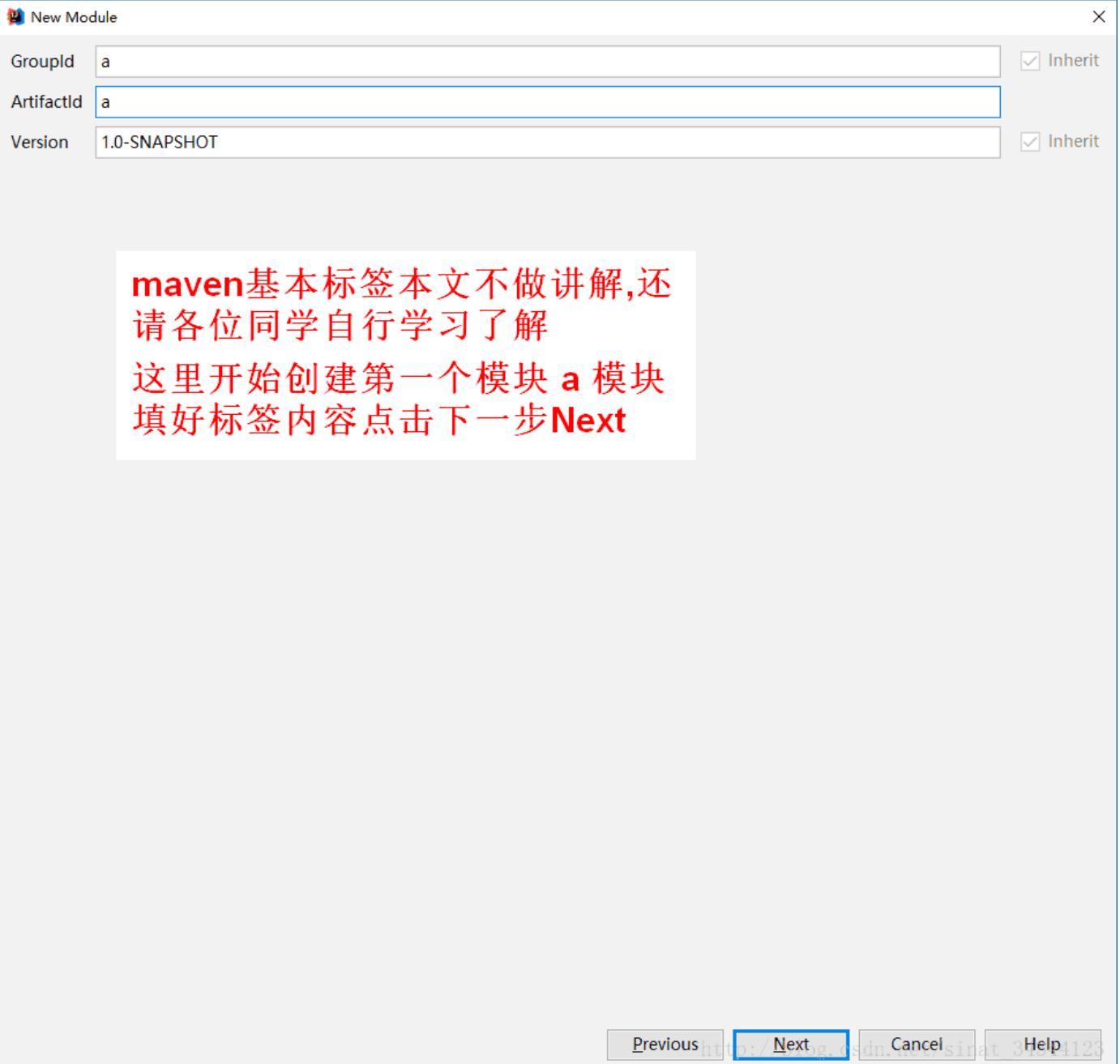
新建一个模块



选择创建一个maven模块



填写模块基本信息



解决创建速度慢

New Module

Maven home directory: Bundled (Maven 3)

(Version: 3.3.9)

User settings file: /home/lvgo/.m2/settings.xml Override

Local repository: /home/lvgo/.m2/repository Override

Properties

groupId	artifactId	version	archetypeGroupId	archetypeArtifactId	archetypeVersion
a	a				

Add Maven Property

Name: archetypeCatalog

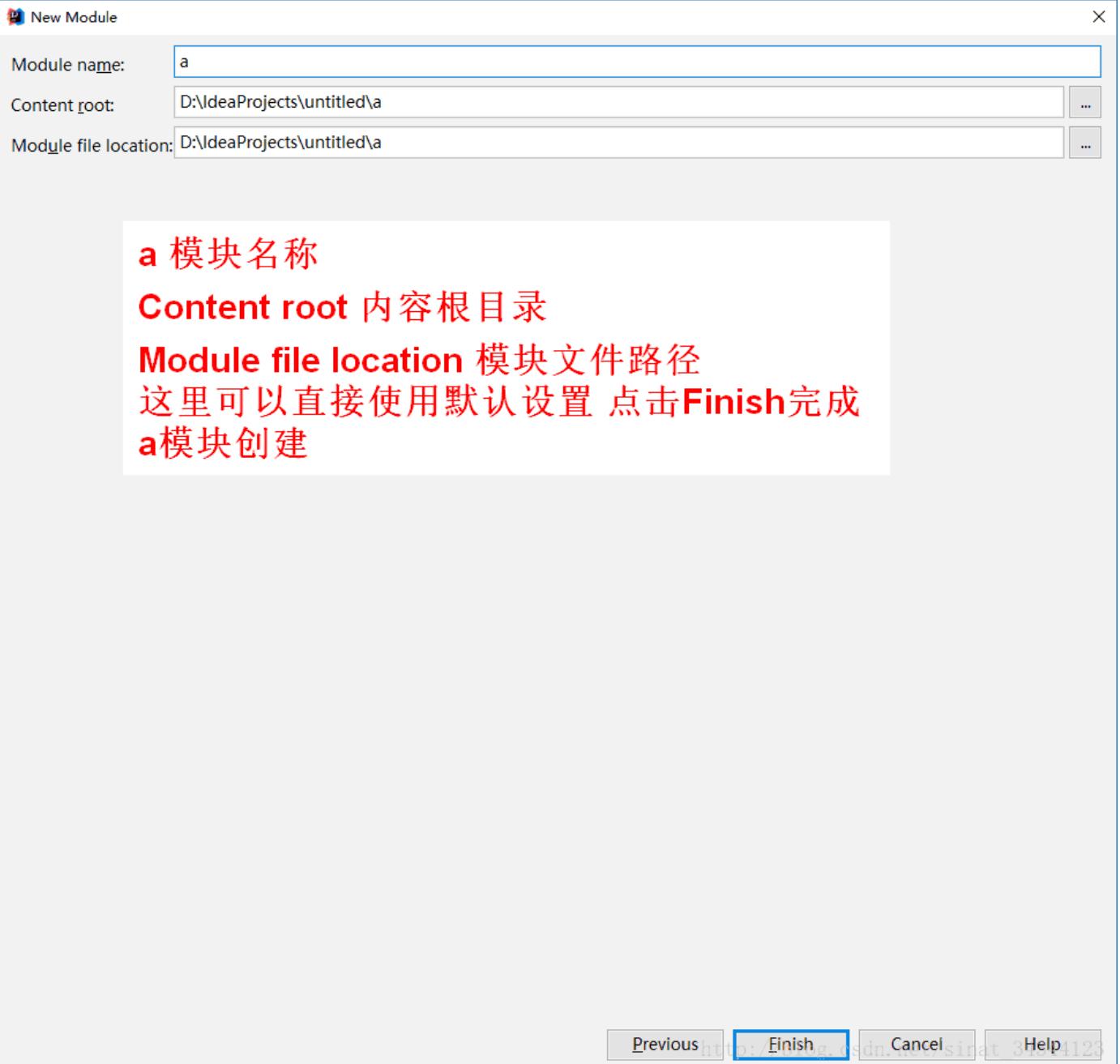
Value: local

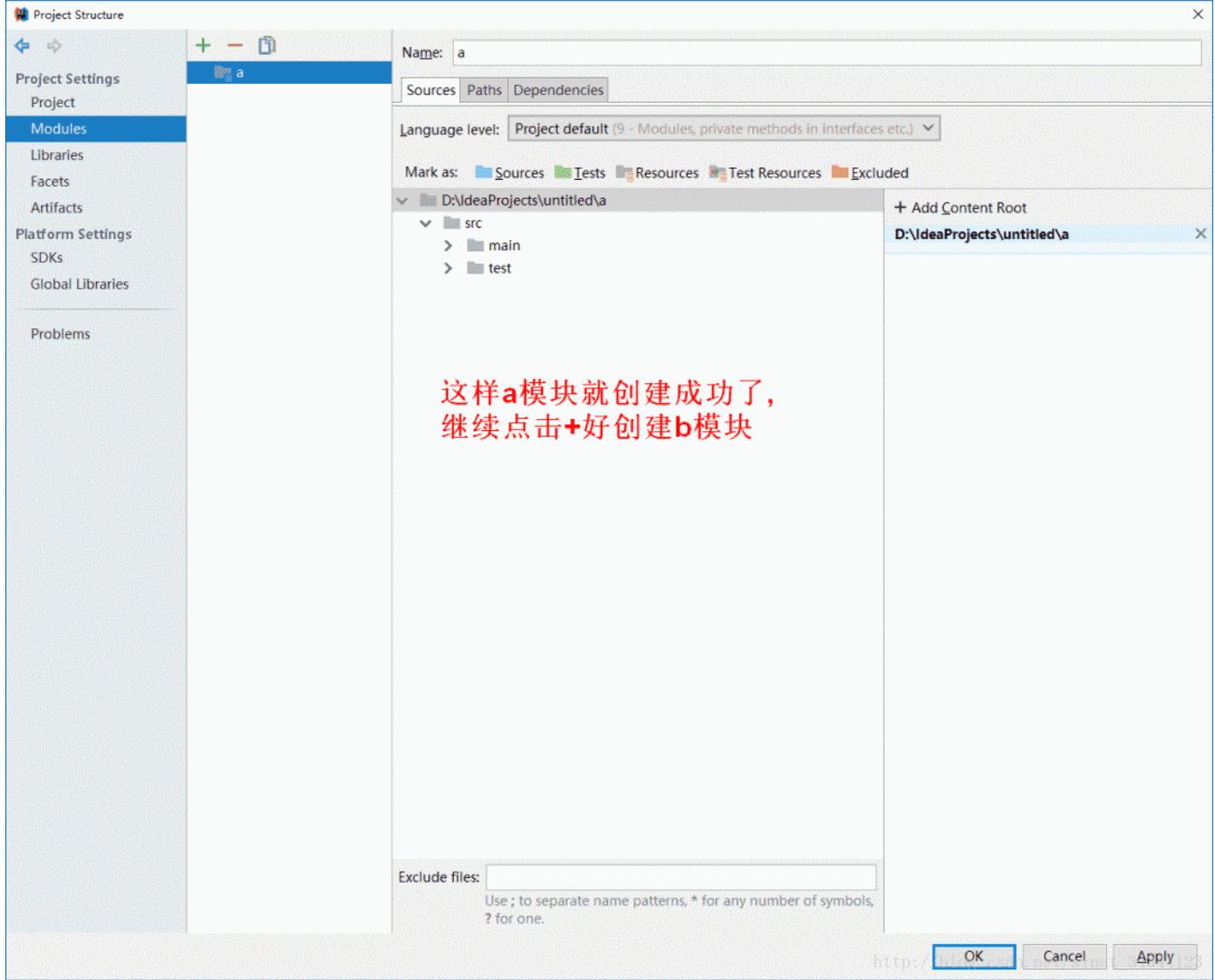
OK Cancel

http:// Previous **Next** / Cancelit 342 Help 23

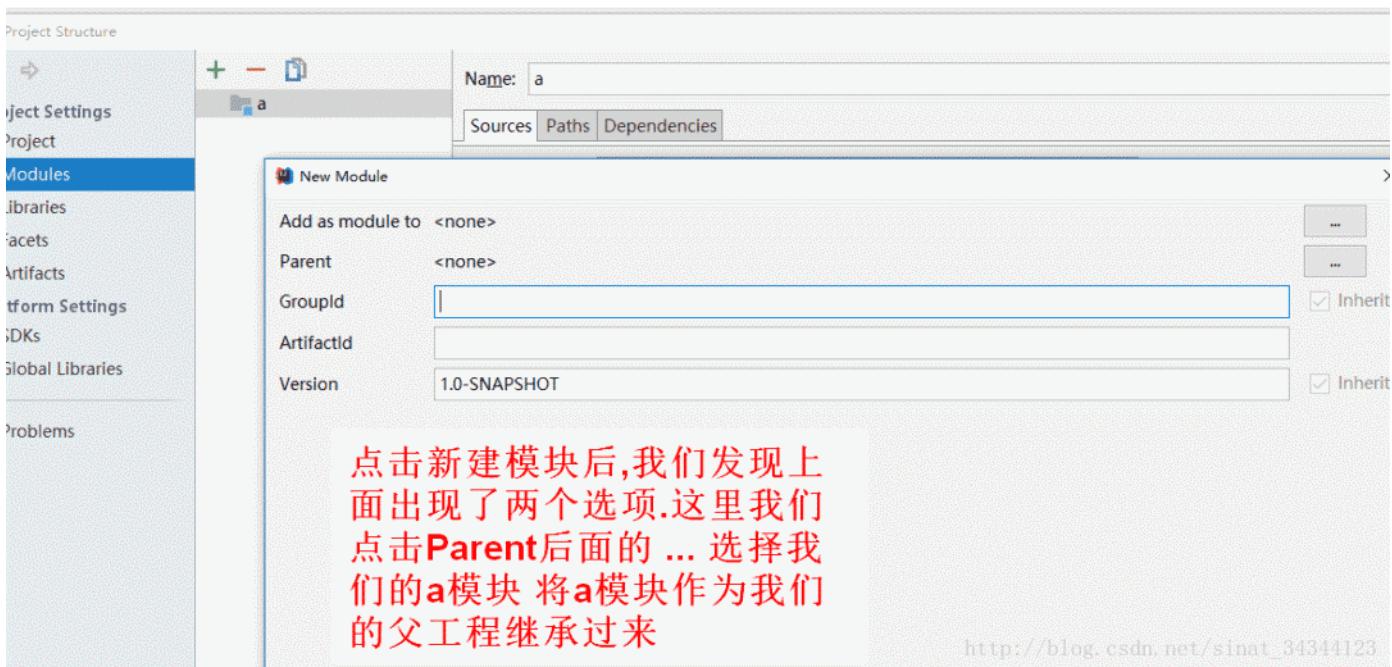
设置一个属性 archetypeCatalog , 具体原因感兴趣的自行了了姐吧

finish完成模块创建

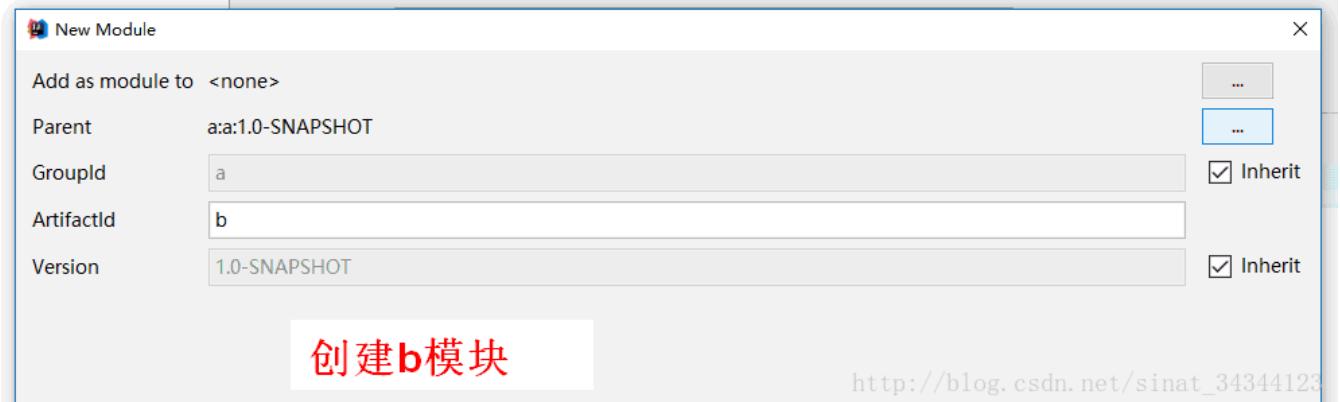




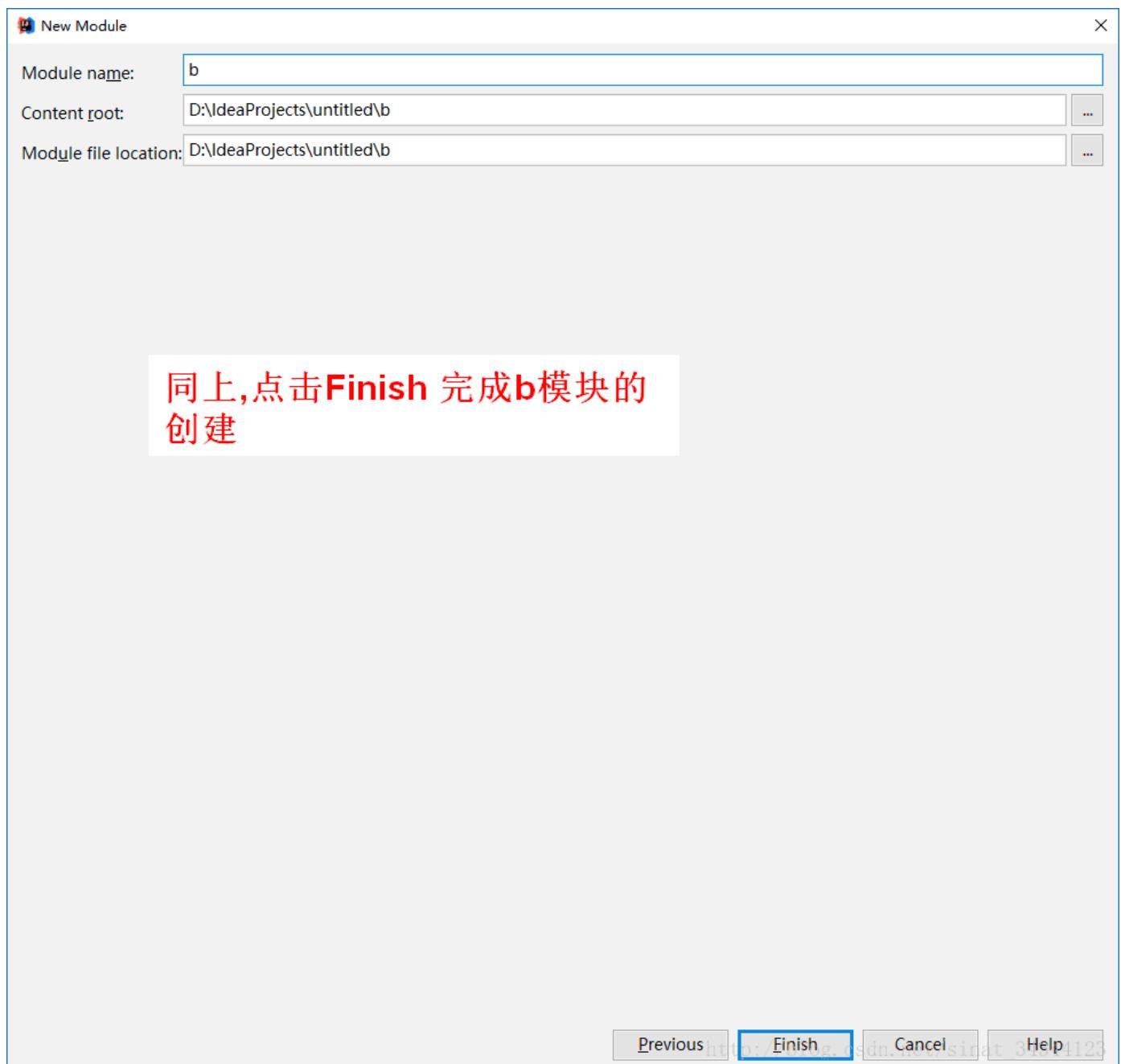
3. 创建第二个模块



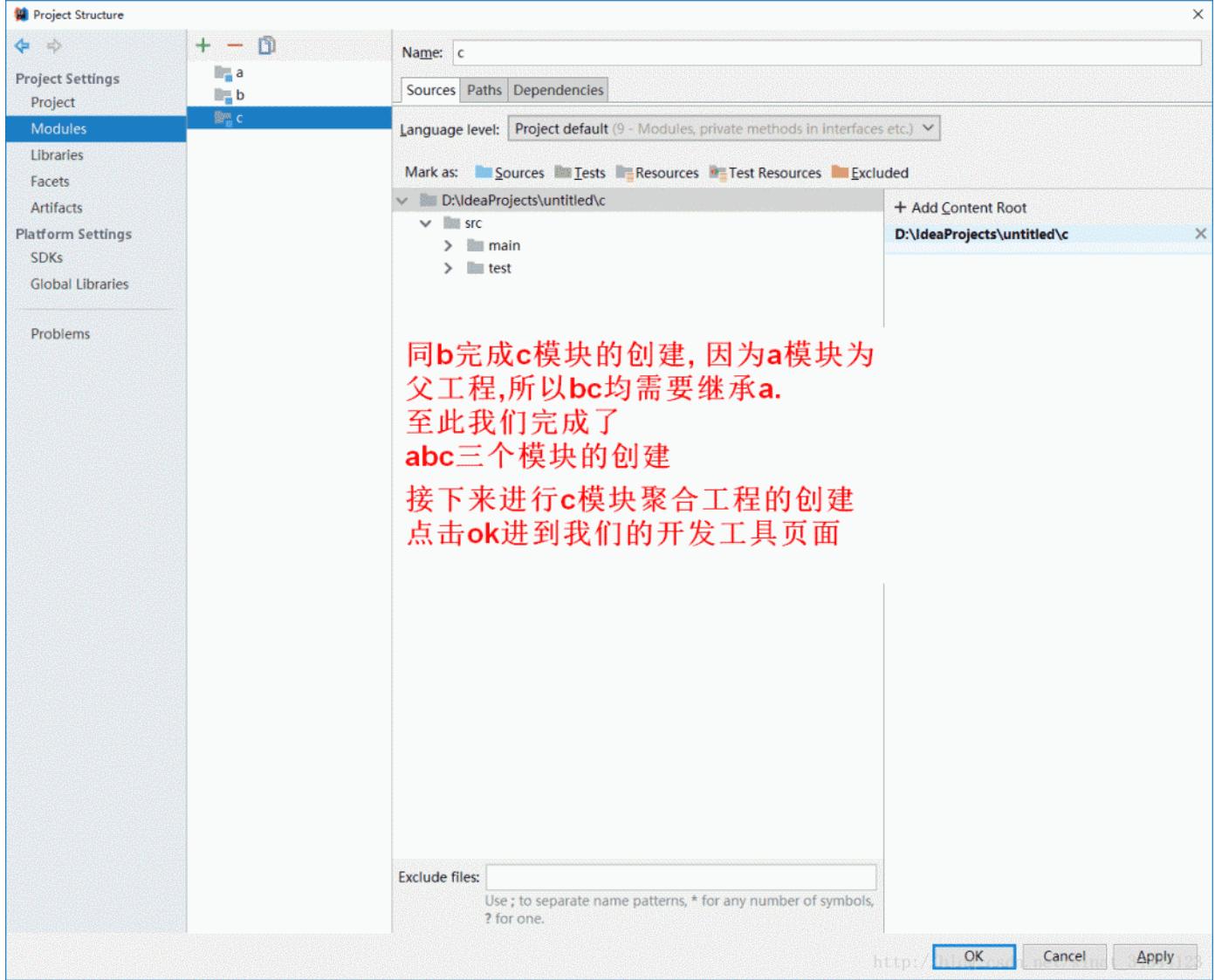
选择继承的父工程



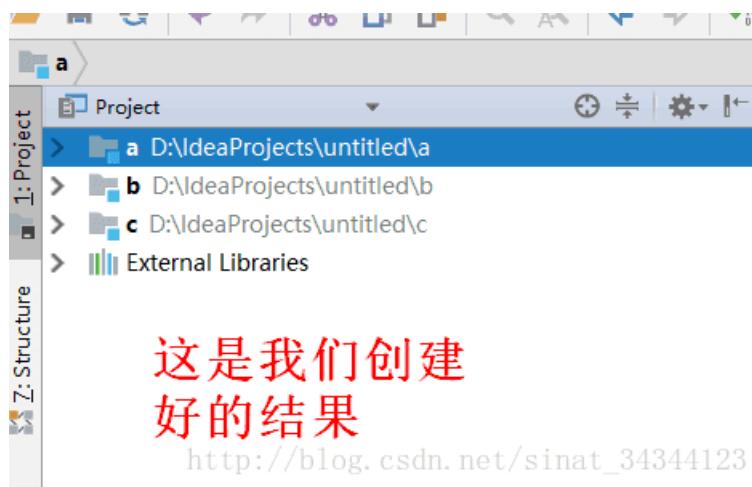
填好路径finish完成b模块创建



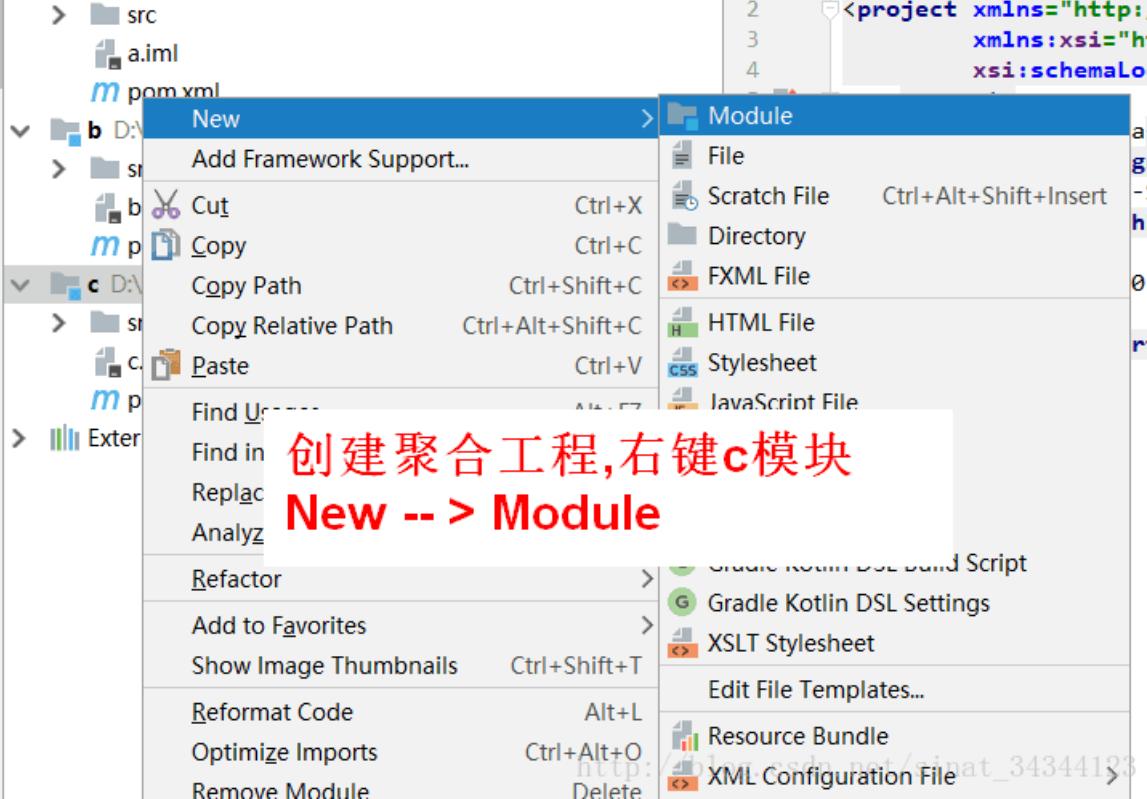
同 b 完成 c 模块创建



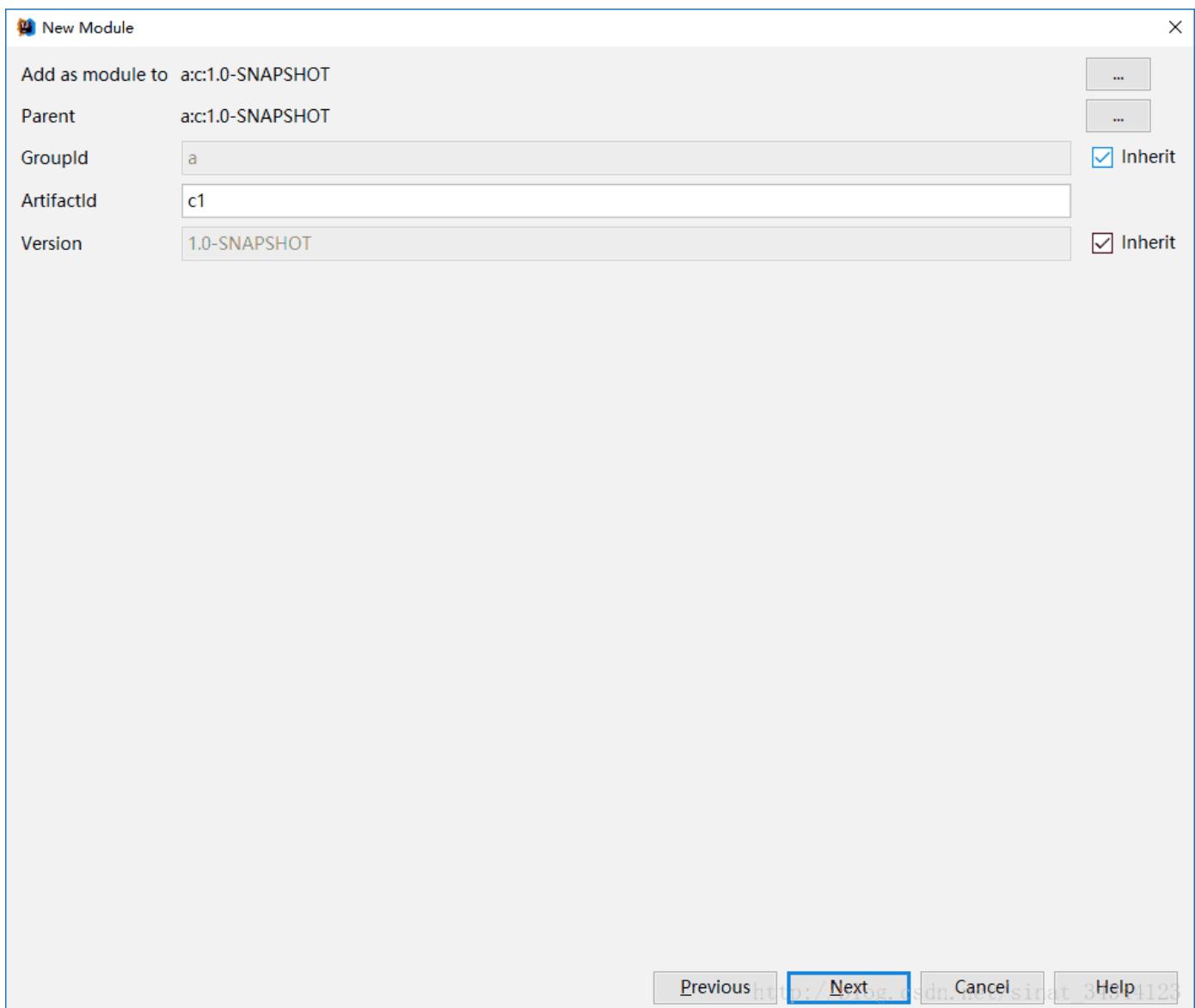
4. 创建聚合工程 c



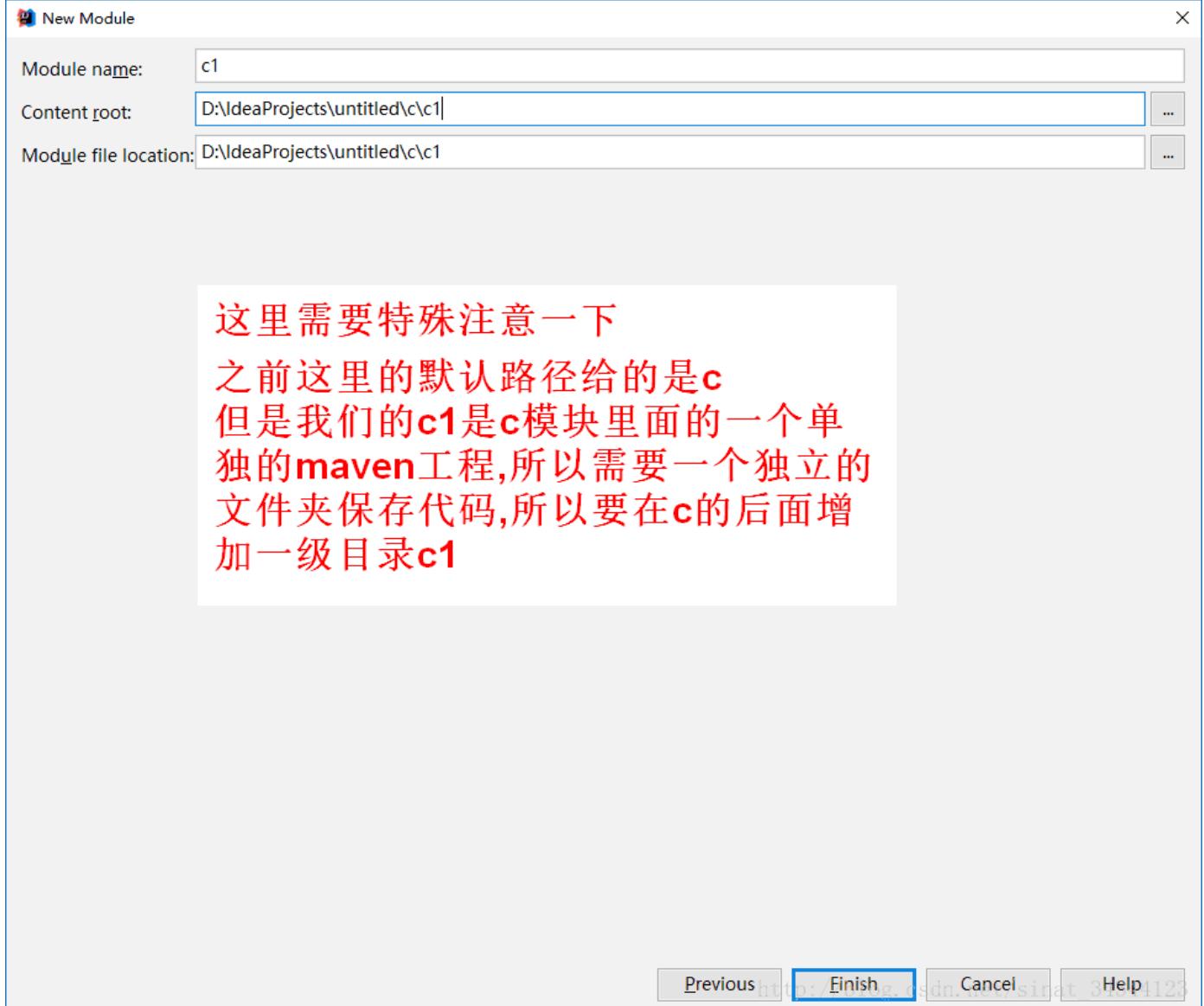
在 c 模块下新建一个maven工程模块 c1



继承 c 模块



配置 c1 工程路径



同 c1 完成 c2 工程模块创建

c2 模块 依赖 c1 模块

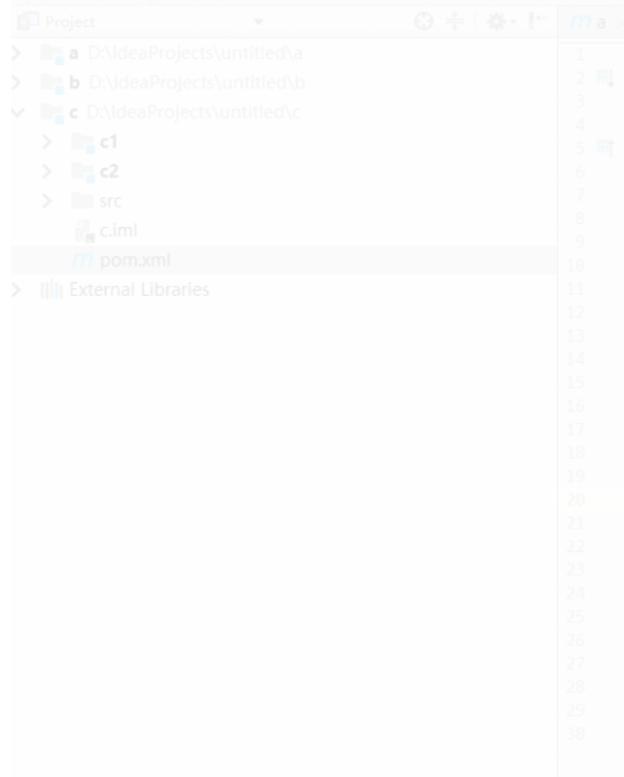
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                               http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>a</artifactId>
        <groupId>a</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>c2</artifactId>

    <!--依赖c1模块-->
    <dependencies>
        <dependency>
            <groupId>a</groupId>
            <artifactId>c1</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
</project>
```

http://blog.csdn.net/sinat_34344123

c 工程模块 依赖 b 工程模块



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>a</artifactId>
        <groupId>a</groupId>
        <version>1.0-SNAPSHOT</version>
        <relativePath>../a/pom.xml</relativePath>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>c</artifactId>
    <packaging>pom</packaging>
    <modules>
        <module>c1</module>
        <module>c2</module>
    </modules>

    <!--c模块依赖b模块-->
    <dependencies>
        <dependency>
            <groupId>a</groupId>
            <artifactId>b</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
    </dependencies>
</project>
```

https://www.jianshu.com/p/3a1f1d1030e0

最后再说两句

这种项目结构主要应用在大型项目中,多人协作开发,小型项目 或 人员配置不足情况下不建议使用;之前因为一直没有机会参与大型项目开发,或是这种分布式项目的搭建,网上也没有这方面系统全面的说明 ,可能真正用到这种架构的不需要自己动手去弄,等着别人搭好进去码代码就可以了.所以本人也是抱着学习的 ,态度发布这篇文章,希望能够帮助一些不清楚的同学一起学习,一起进步.

如果有不明白或者不清楚的,或者错误的地方,还希望大家能够在底下评论出来,好让我及时的去改正。

如果喜欢本篇文章,欢迎[转发](#)、[点赞](#)。关注订阅号「Web项目聚集地」,回复「进群」即可进入无广告技术交流。

推荐阅读

- [1. 什么时候进行分库分表 ?](#)
- [2. 从 Java 程序员的角度理解加密](#)
- [3. IDEA 中使用 Git 图文教程](#)
- [4. 一文梳理 Redis 基础](#)
- [5. 优化你的 Spring Boot](#)
- [6. 数据库不使用外键的 9 个理由](#)



Web项目聚集地

微信扫描二维码，关注我的公众号

喜欢文章，点个在看

[阅读原文](#)

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

IntelliJ IDEA 详细图解最常用的配置，新人收藏

李学凯 Java后端 2019-11-07

点击上方 Java后端, 选择 **设为星标**

优质文章, 及时送达

作者 | 李学凯

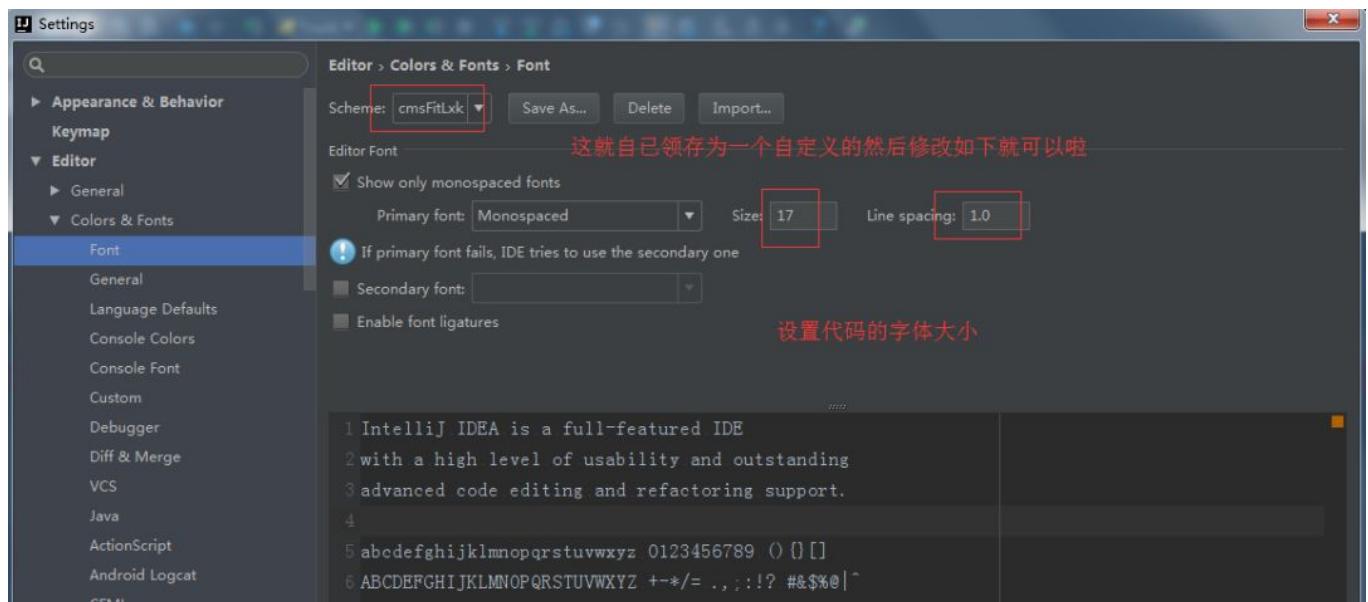
链接 | https://blog.csdn.net/qq_27093465

刚刚使用IntelliJ IDEA 编辑器的时候，会有很多设置，会方便以后的开发，磨刀不误砍柴工。

比如：设置文件字体大小，代码自动完成提示，版本管理，本地代码历史，自动导入包，修改注释，修改tab的显示的数量和行数，打开项目方式，等等一大堆东西。

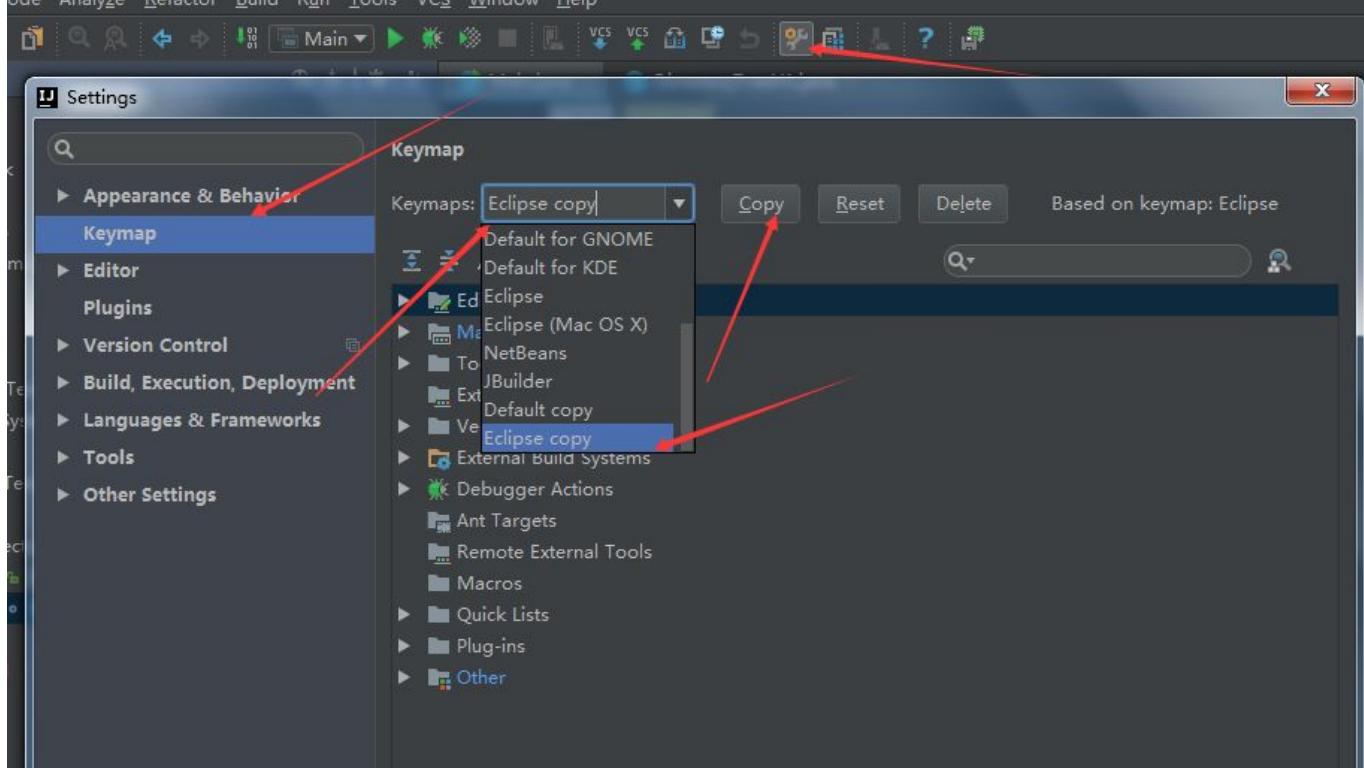
总结一下，免得下次换了系统，还得再找一遍配置。

设置外观和字体大小



这个呢是设置一下外观。和字体大小。放在第一个没问题。

设置编辑器的快捷键，也就是keymap

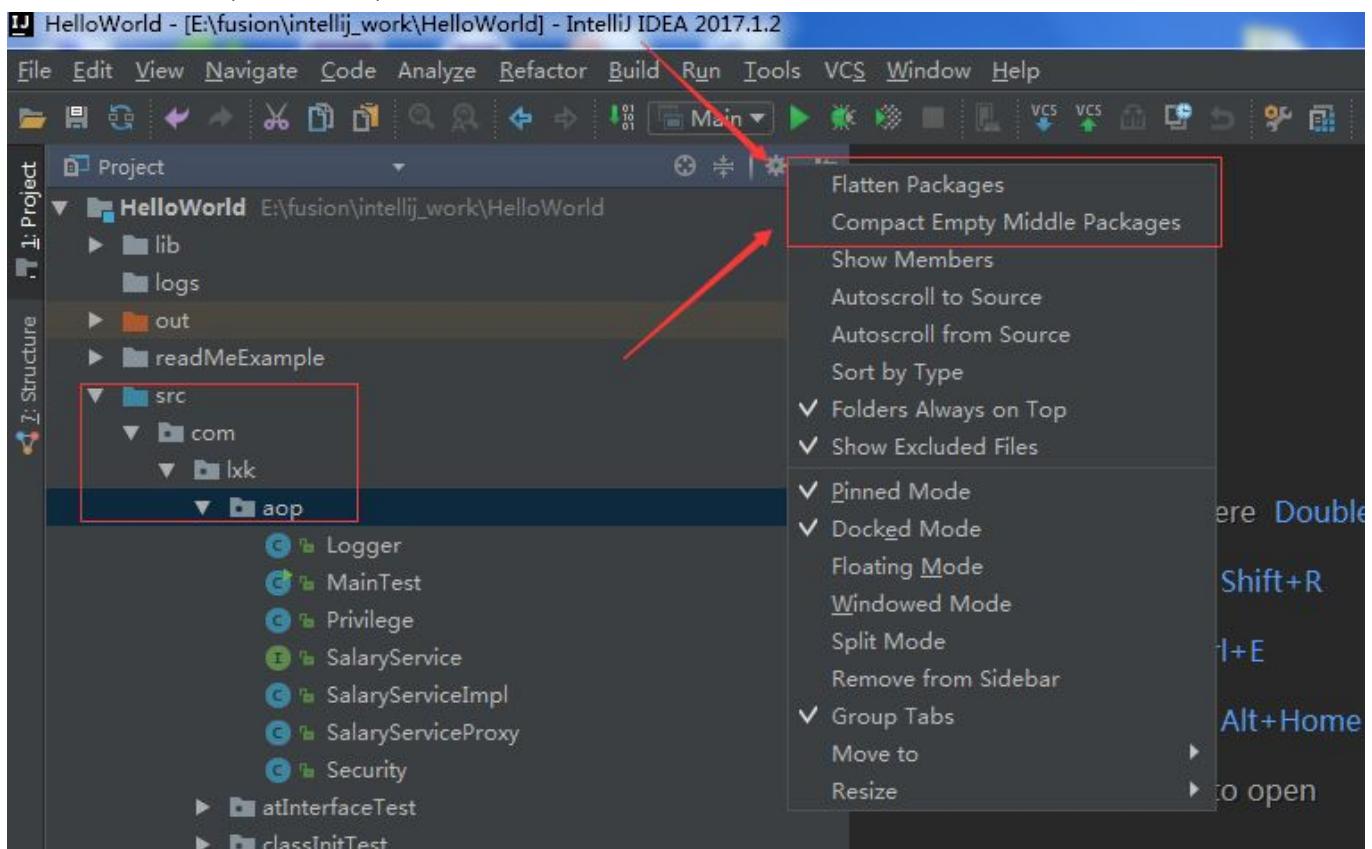


这个是修改咱习惯的快捷键映射表，因为我是从eclipse转来的，估计大部分都和我差不多啦，那就可以在这配置成eclipse的快捷键映射表，那么就没有必要再去记一套快捷键映射了，比如我们常用的删除一行 Ctrl d，复制一行Ctrl + alt + 下方向。注释一行，Ctrl + / 这都是我们常用的。也是我们习惯的，这个也是极好的设置啊。

我图上就是把eclipse的键盘映射复制一下，然后重命名一下，因为还是有些快捷键修改下，用着比较好。

关于整个项目的文件目录的说明

快捷键简单搞定之后，再熟悉一下，下面这个图。



这个也是通用结构，我箭头所指的地方有三个按钮，

第一个，点击之后，就会在左侧的文件一栏里，定位到你当前打开的文件的位置，找文件，定位文件位置用的非常多。

第二个，合并所有目录，这个在你打开太多目录的时候，一点击之后，就会把目录全部折叠起来。

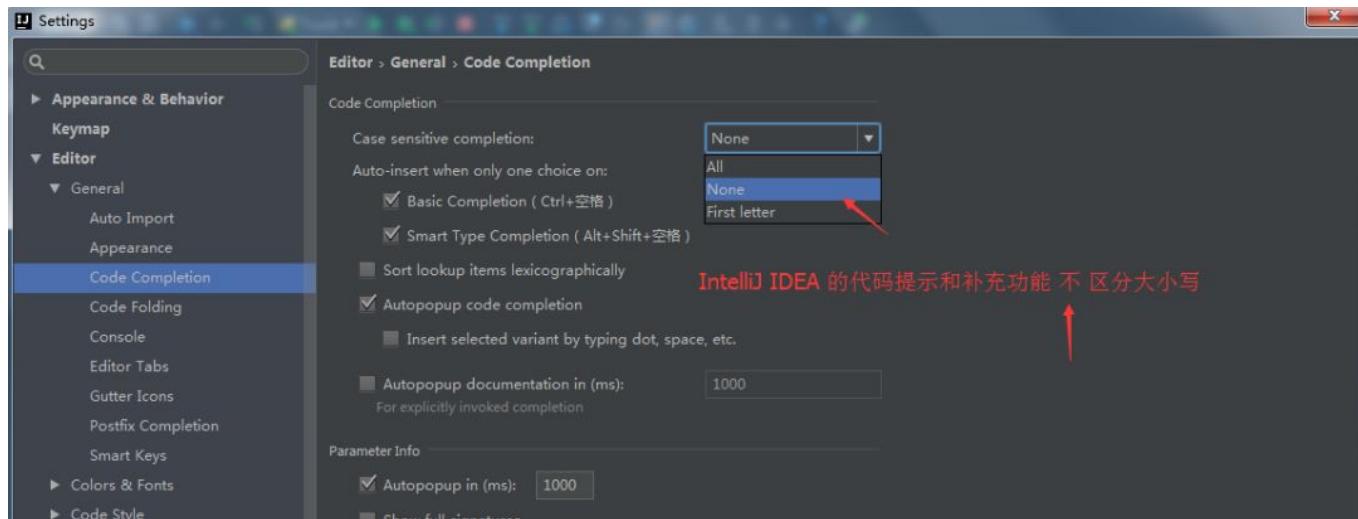
第三个，点了之后，就看到上面的那个弹出框。这个有点特殊，建议，红框里面的设置的跟我一样，比较好，这样的话你在左侧，查看项目目录结构的时候，就不会觉得奇怪。至于为什么会觉得奇怪，你可以先把这2个都点上之后，看看你的目录是什么情况，就知道我说的是啥啦，

这样子配置，可以很清楚的看到目录的层级结构。但是你要是点了，他就会把空的包直接连在一起，就是com.lxk.aop。。。等等吧，试一下就知道啦。

这个也是很有必要说明一下的。

Tips：关注微信公众号：Java后端，每日获取技术博文推送。

自动提示

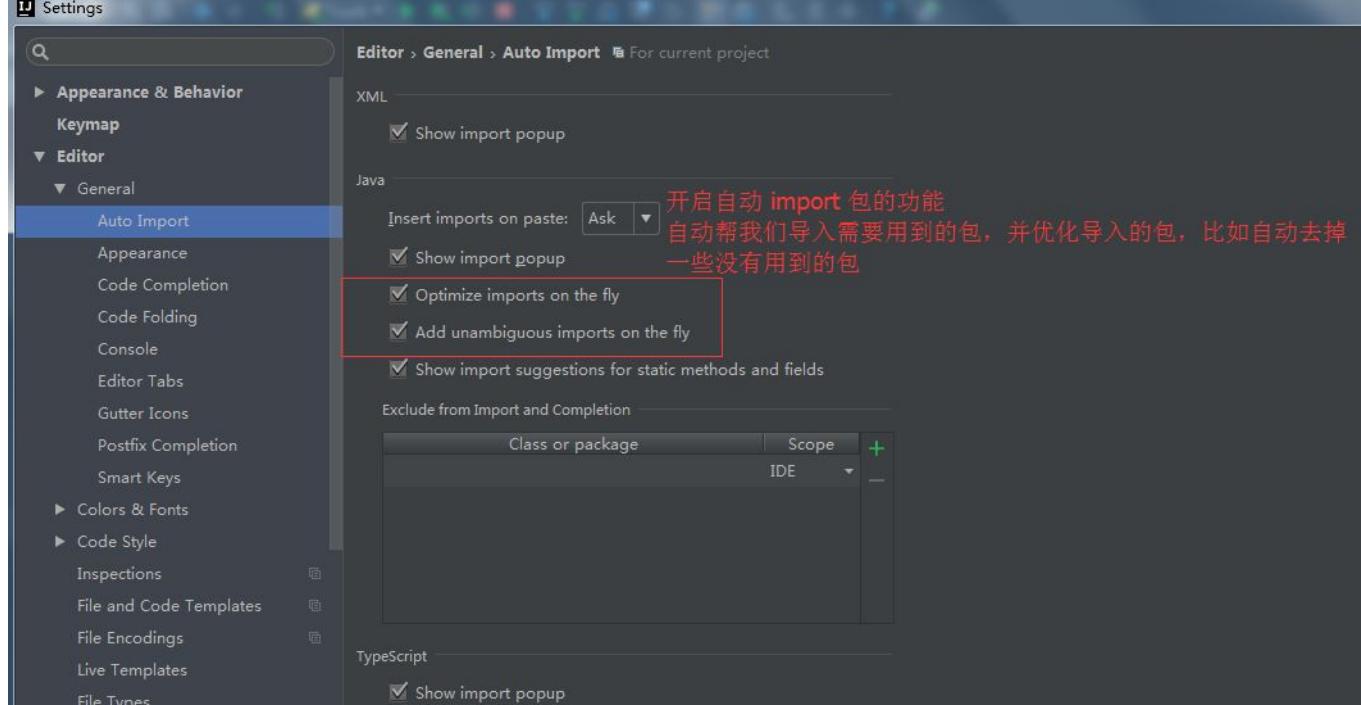


代码检测警告提示等级设置

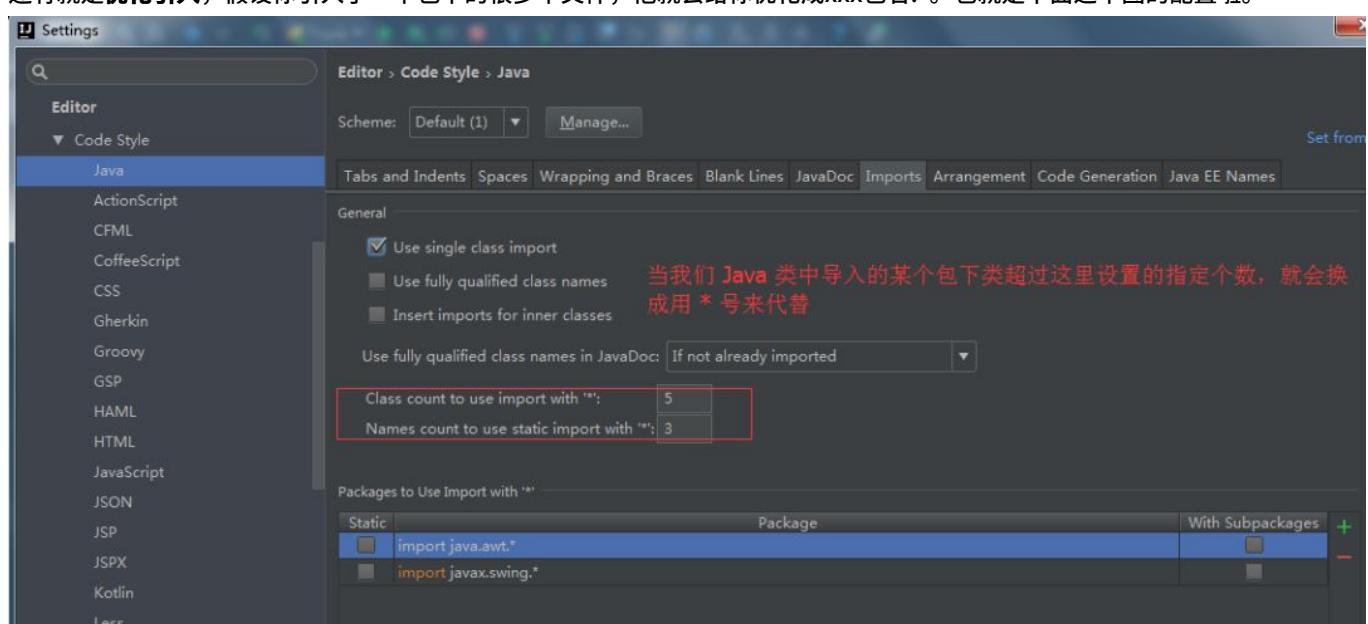


强烈建议，不要给关掉，不要嫌弃麻烦，他的提示都是对你好，帮助你提高你的代码质量，很有帮助的。

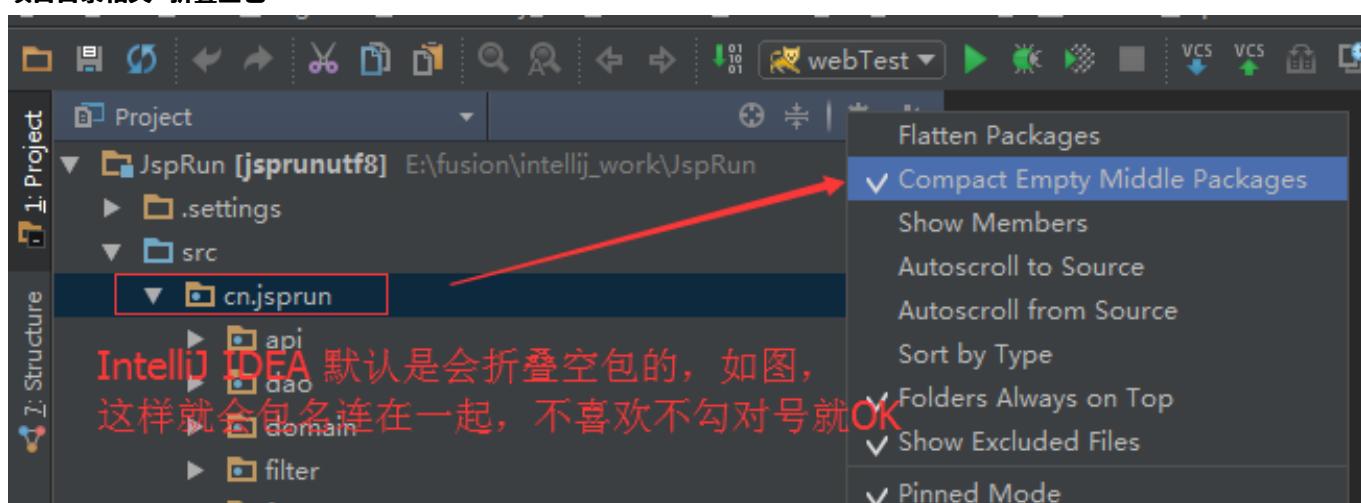
自动导入包和导入包优化的设置



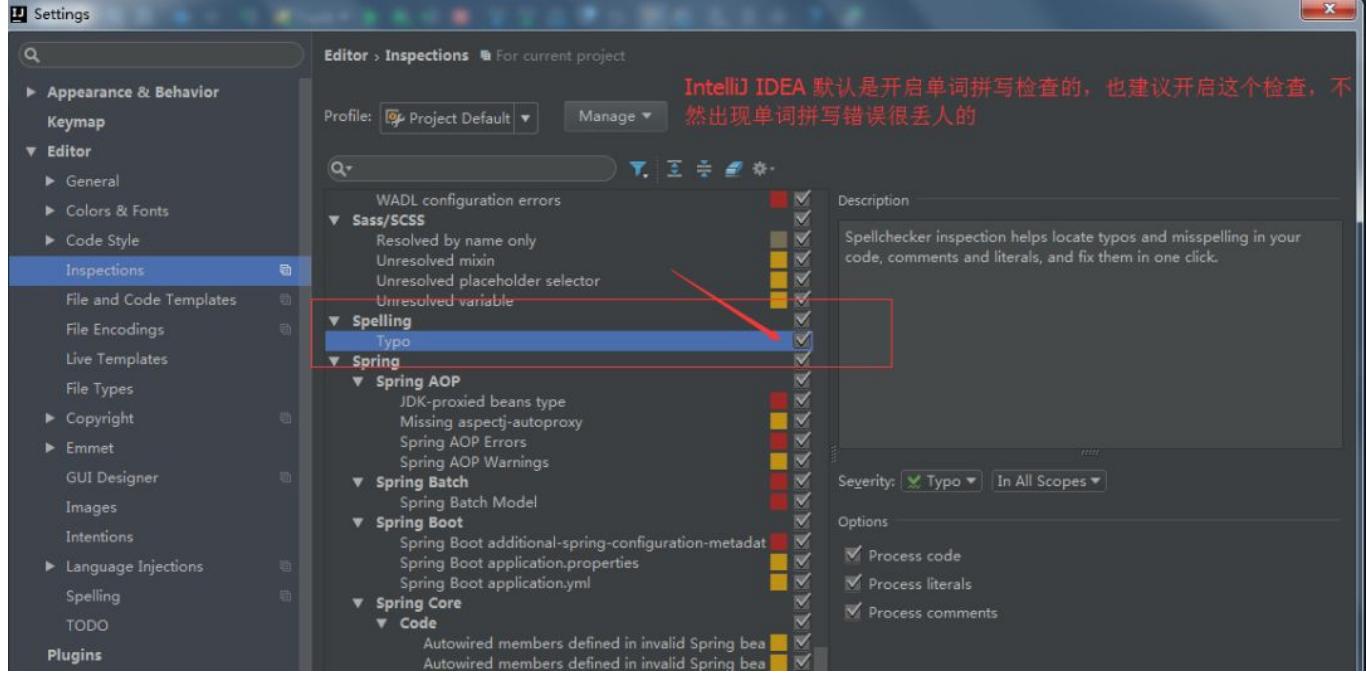
还有就是优化引入，假设你引入了一个包下的很多个文件，他就会给你优化成xxx包名.*。也就是下面这个图的配置啦。



项目目录相关--折叠空包

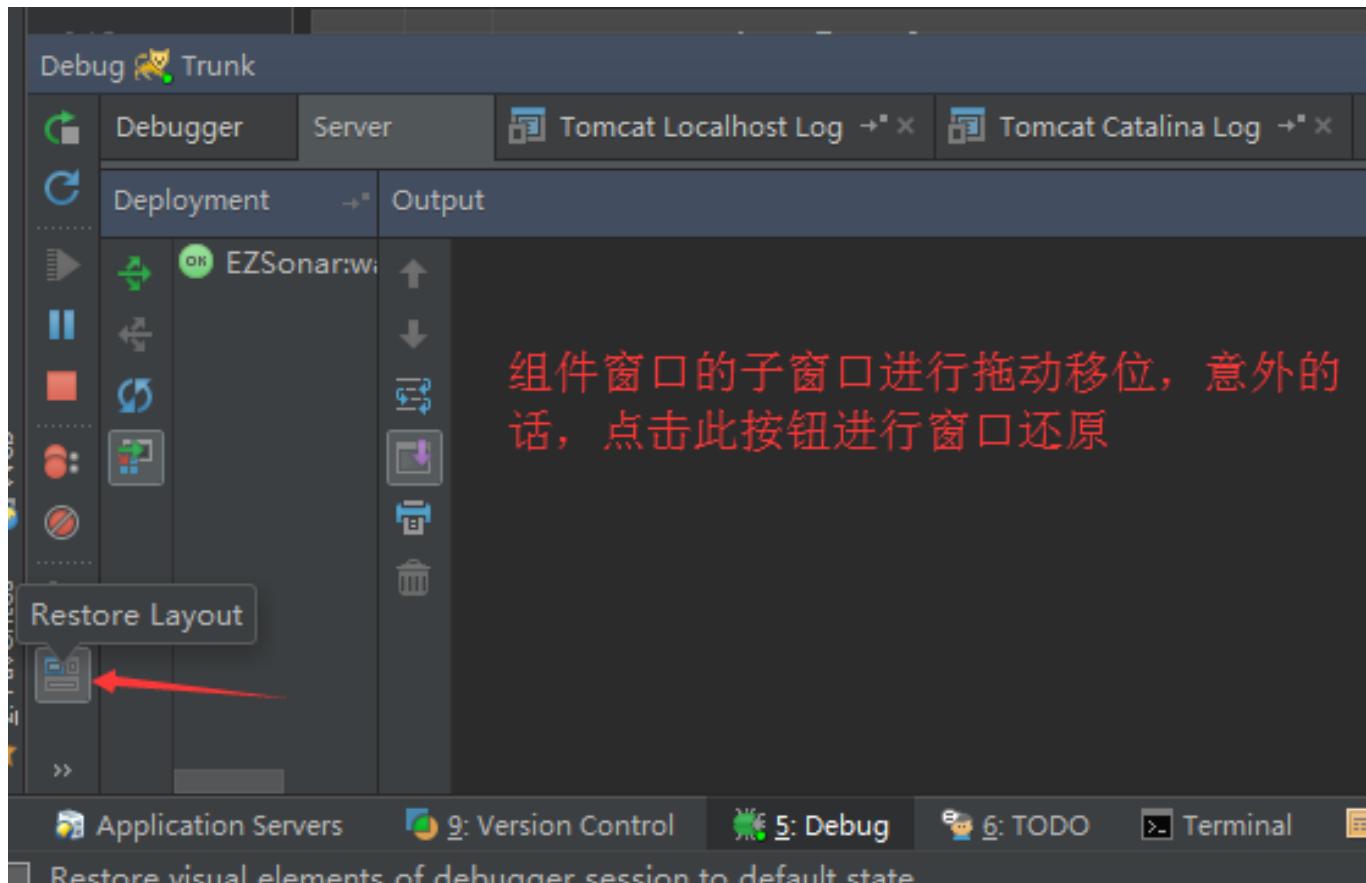


单词拼写提示--建议打开



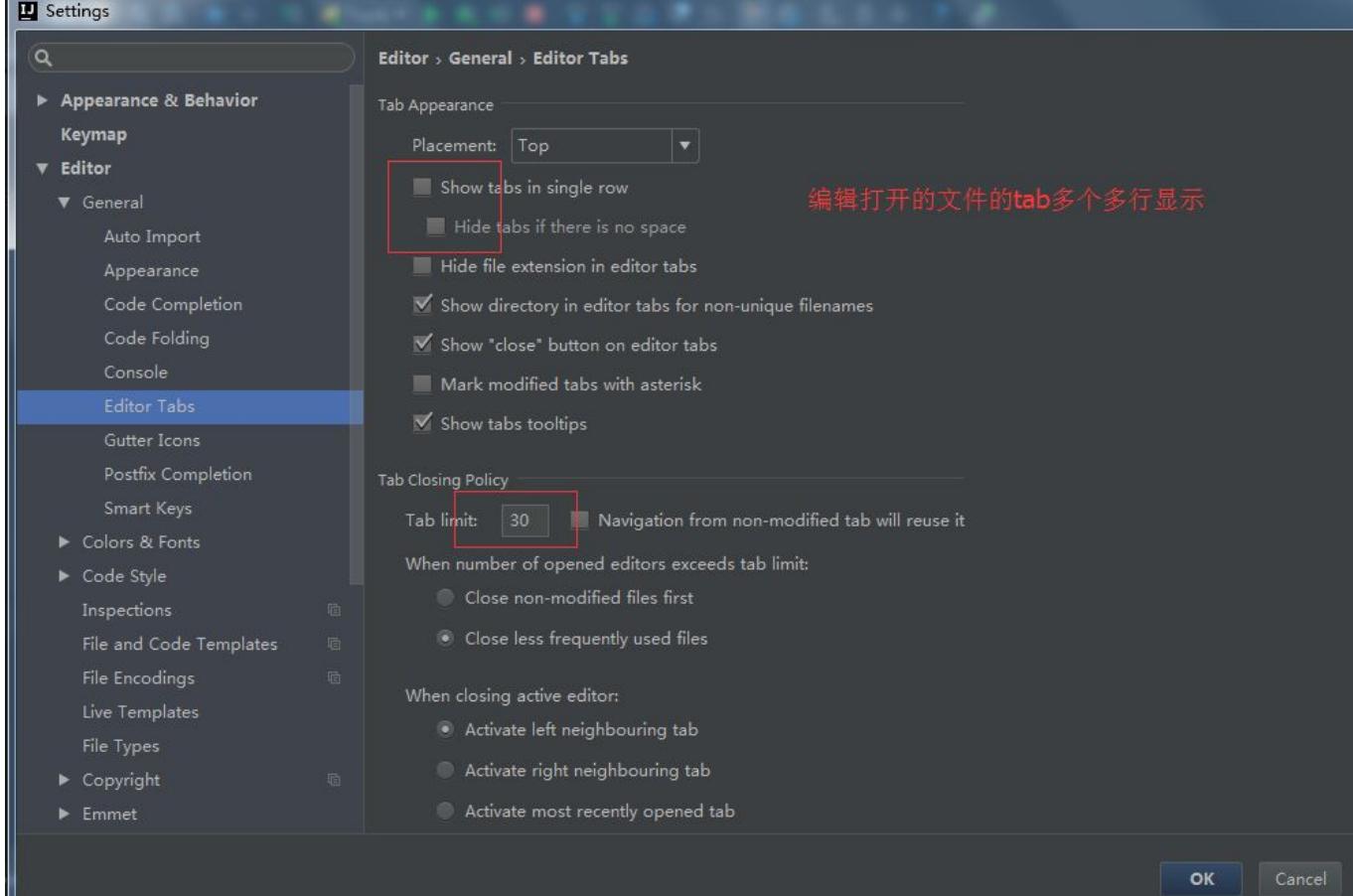
强烈建议，不要嫌弃他这个单词拼写检查，这也是为你好啊，免得你写一个简单的单词，但是你却写错了，还提示你使用驼峰命名法。也是很好的。

窗口复位的简单说明

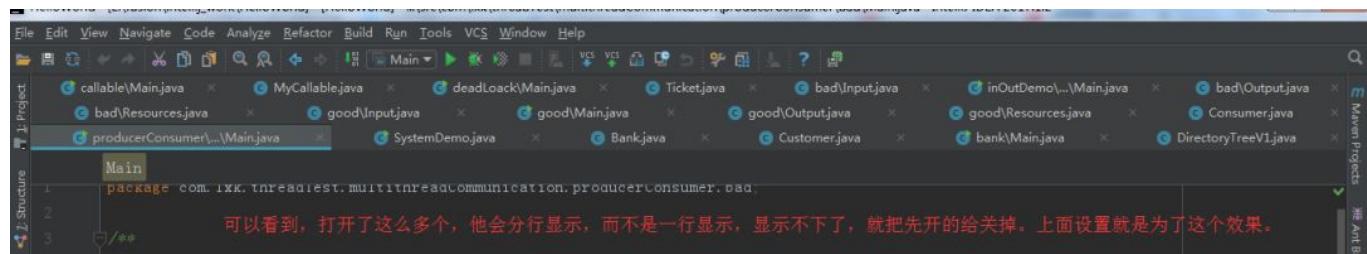


这个就是当你把窗口忽然间搞得乱七八糟的时候，还可以挽回，就是直接restore一下，就好啦。

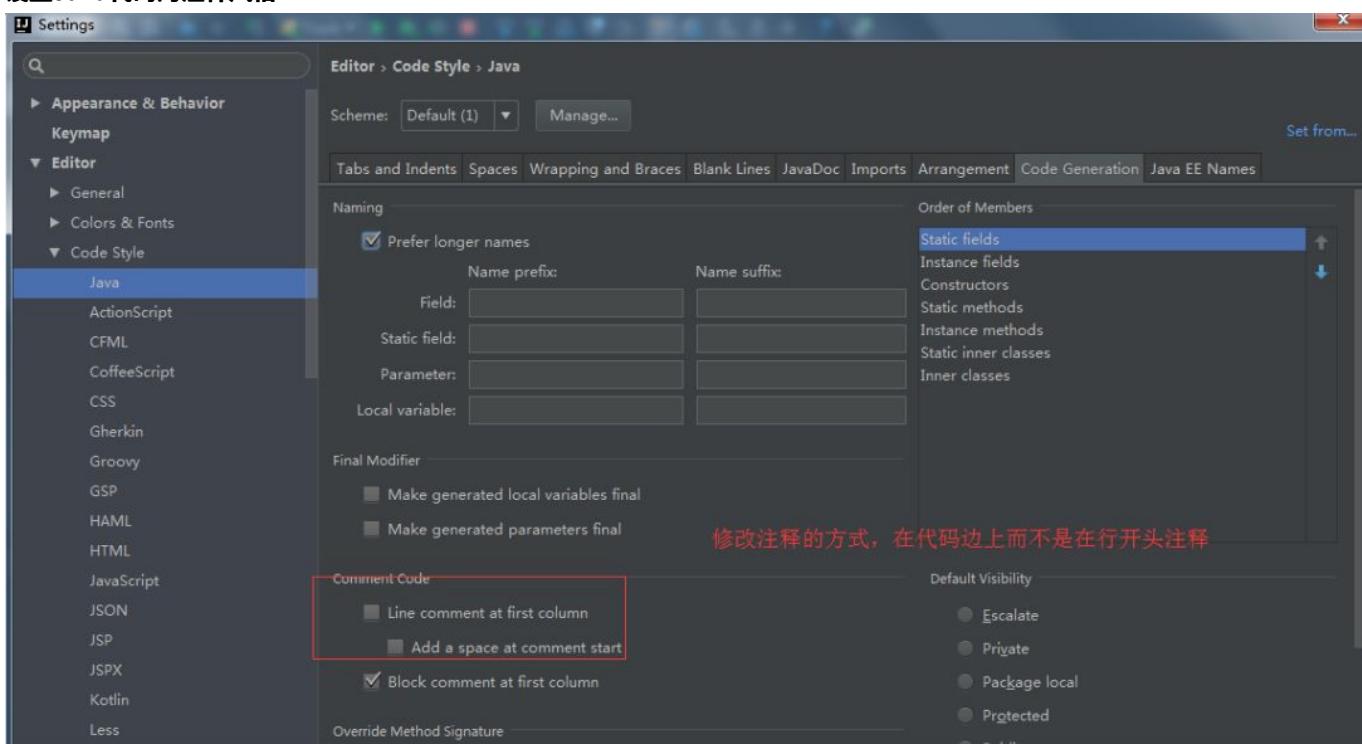
tab页面多行显示的设置



这个是在使用很多的tab页面的时候用的到，而不是要是打开多个页面的话，一些就会被关掉。那就不好啦。具体看下图，就知道我在说啥了。



设置Java代码的注释风格

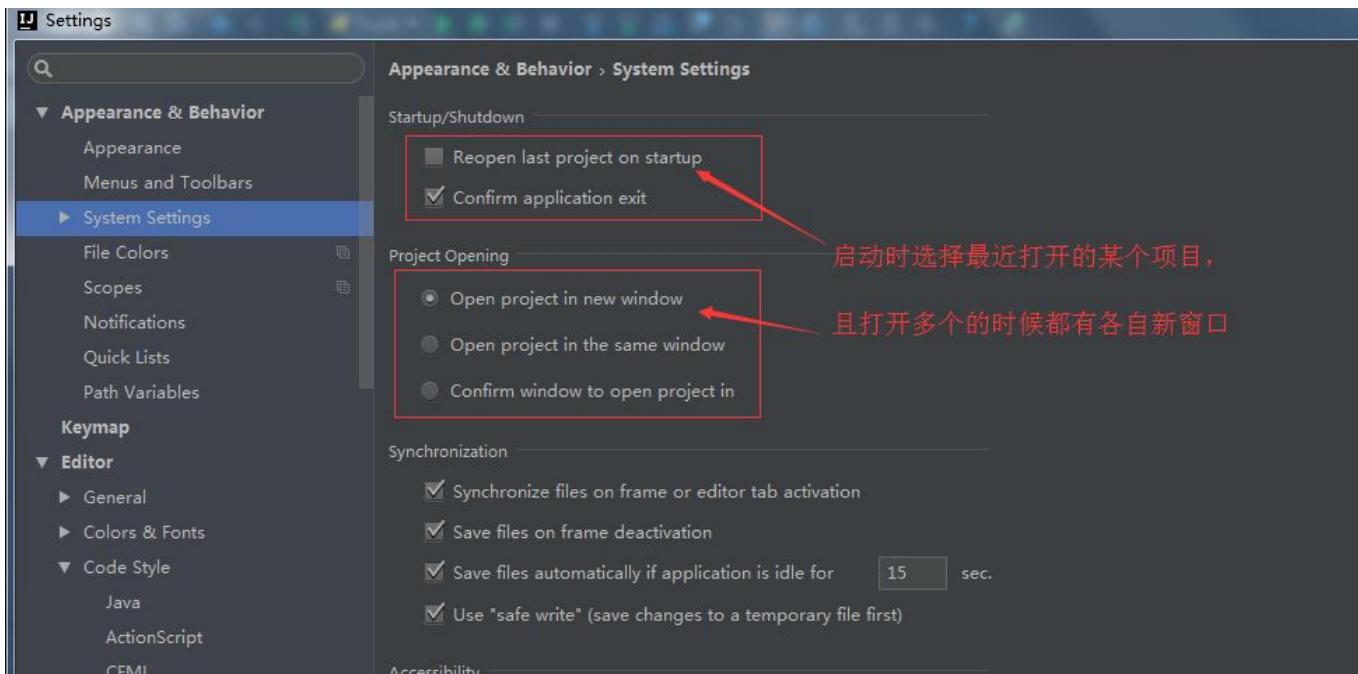


这个就是你在使用快捷键注释代码的时候，就比如我的快捷键是Ctrl + d就是注释所选中的代码，但是你要是不设置，这个//就是在代码行的开头。这个看着就不是很习惯。

具体就看下面这个图的三种情况下的注释的显示情况。看你喜欢哪个注释风格，就怎么设置。

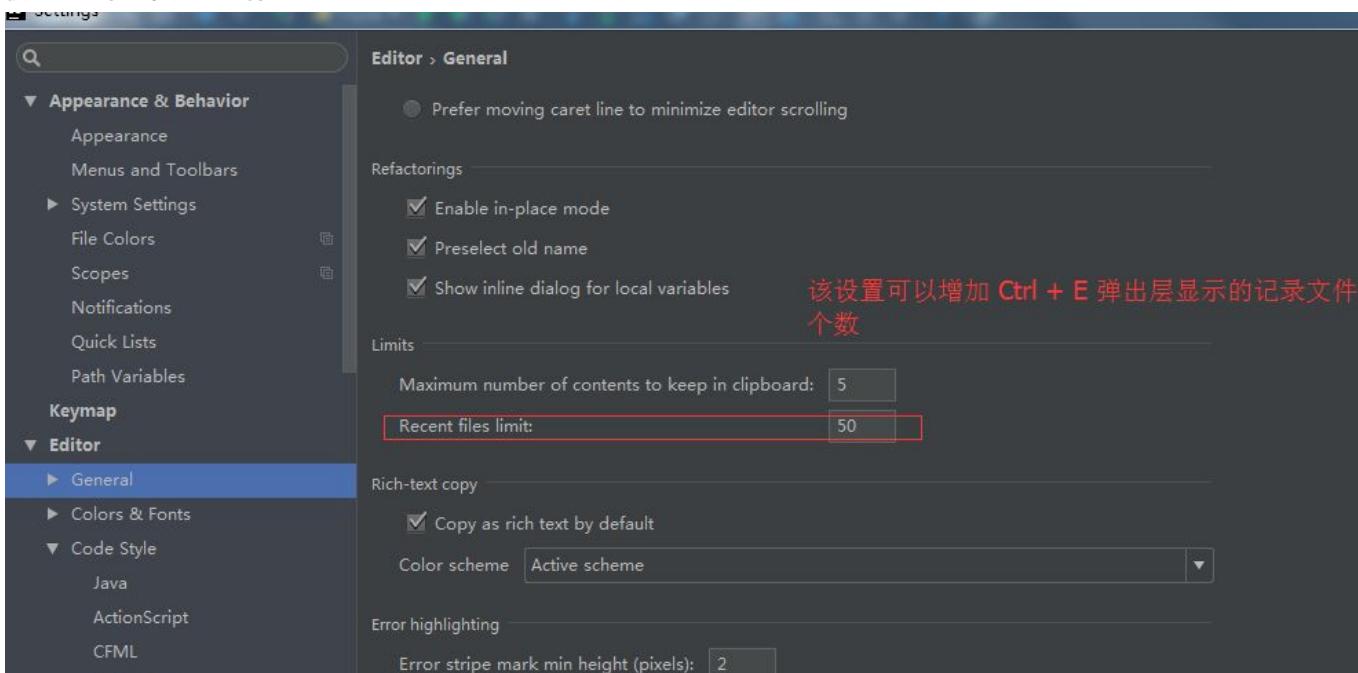
```
+ "d:test" + "\n" | grep '\"' + "d:test" + "\n" | grep -v '\"grep\"' | awk '{ print $2}'");  
System.out.println(Arrays.toString(command)); //结果如下: //第一个勾不打,就是这么注释效果。  
// System.out.println("测试注释代码的不同情况"); //这个就是只选第二个勾的情况,相比上面,多了一个空格。  
//[sh, -c, ps -ef | grep "d:test" | grep "d:test" | grep -v "grep" | awk '{ print $2}']
```

编辑器每次打开项目时候的设置



这个也是极其必要的配置，因为，你要不配置，一点击桌面的图标，那就直接打开项目了，这个就不能很好的选择你要打开哪个了。这个对新手来说，估计是个问题。

快速找到最近使用的文件的设置



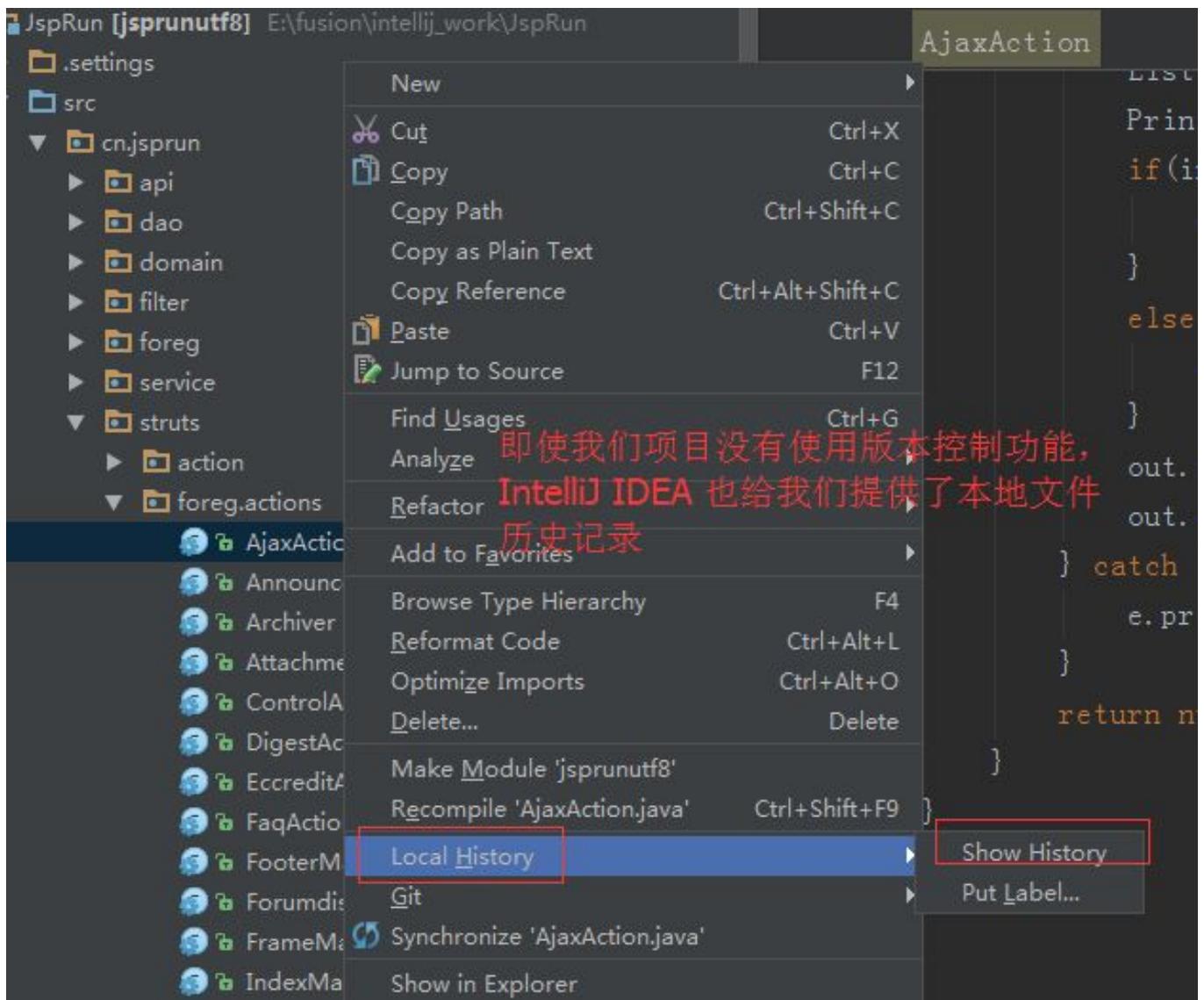
Java代码左面的边边栏的神奇地方使用

下面2张图，算是这个编辑器比较牛逼的地方，简单展现吧。



上面这个图呢，在svn一文中详细解释了，可以参观一下。

本地代码也是有历史的



这个也是这个编辑器比较牛逼的地方，他可以有自己的本地历史，也在其他文章中有详细描述。

- END -

如果看到这里，说明你喜欢这篇文章，请[转发、点赞](#)。微信搜索「web_resource」，关注后回复「进群」或者扫描下方二维码即可进入无广告交流群。

↓ 扫描二维码进群 ↓



扫一扫上面的二维码图案，加我微信

推荐阅读

1. Spring Boot 全局异常处理整理
2. 细说 Java 主流日志工具库
3. 9 个爱不释手的 JSON 工具
4. 12306 的架构到底有多牛逼？
5. 团队开发中 Git 最佳实践



学 Java，请关注公众号：Java后端

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

IntelliJ IDEA 高级用法之：集成 JIRA、SSH、FTP、Database管理、UML类图插件

菩提树下的杨过 Java后端 2019-11-18

点击上方 Java后端，选择 [设为星标](#)

优质文章，及时送达

作者 | 菩提树下的杨过

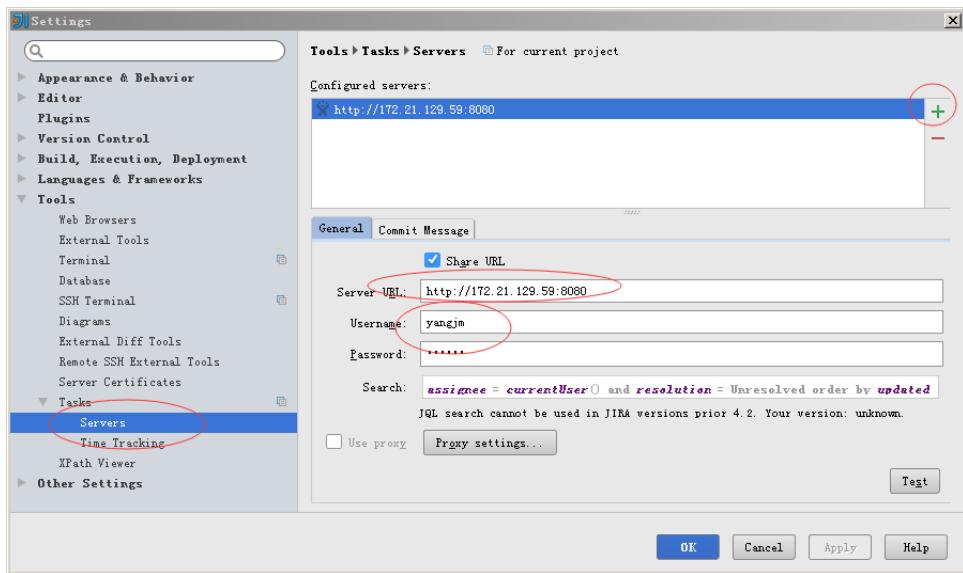
来源 | www.cnblogs.com/yjmmyzz

上篇 | 除了《颈椎康复指南》，还有这 9 本书

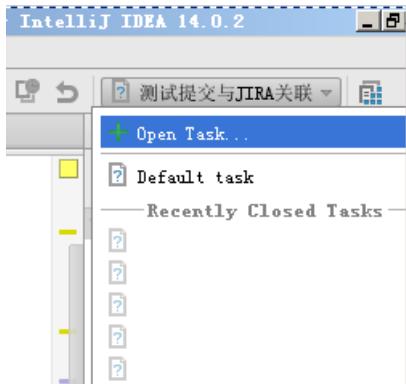
一、与JIRA集成

jira是一个广泛使用的项目与事务跟踪工具，被广泛应用于缺陷跟踪、客户服务、需求收集、流程审批、任务跟踪、项目跟踪和敏捷管理等工作领域。idea可以很好的跟它集成，参考下图：

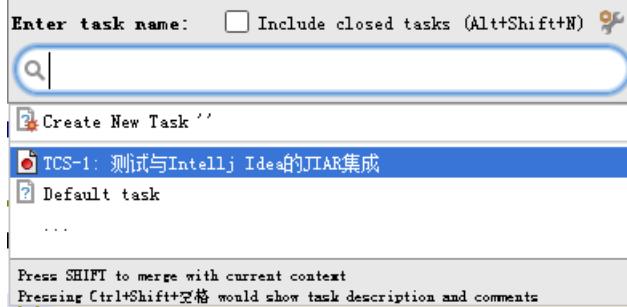
File -> Settings ->Task -> Servers 点击右侧上面的+号，选择JIRA，然后输入JIRA的Server地址，用户名、密码即可



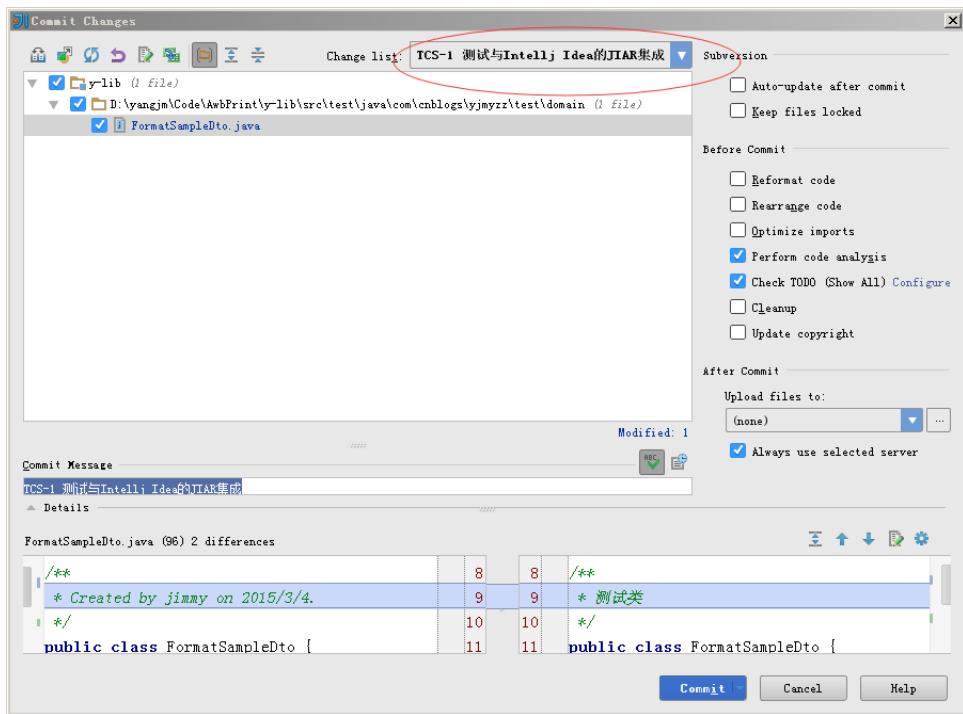
然后打开Open Task界面



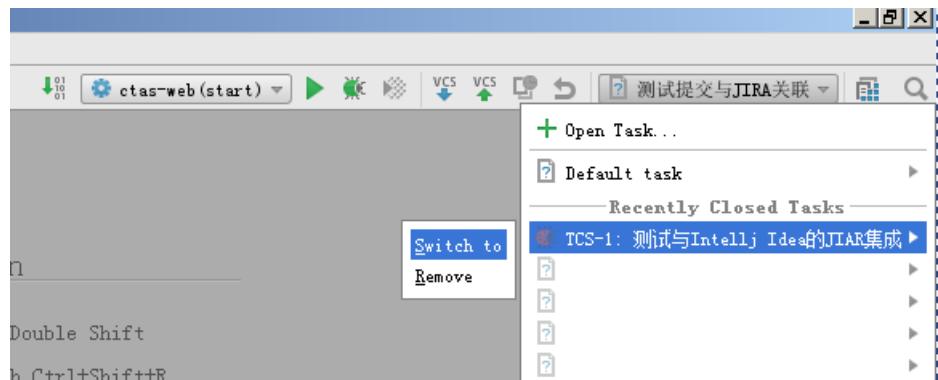
如果JIRA中有分配给你的Task，idea能自动列出来



代码修改后，向svn提交时，会自动与该任务关联



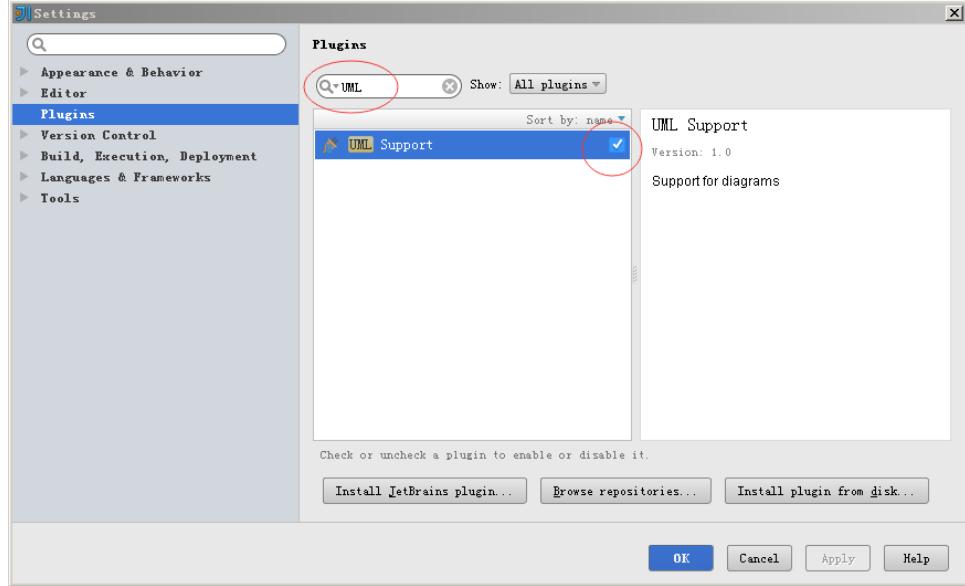
将每次提交的代码修改与JIRA上的TASK关联后，有什么好处呢？我们每天可能要写很多代码，修复若干bug，日子久了以后，谁也不记得当初为了修复某个bug做了哪些修改，不要紧张，只要你按上面的操作正确提交，idea都会帮你记着这些细节



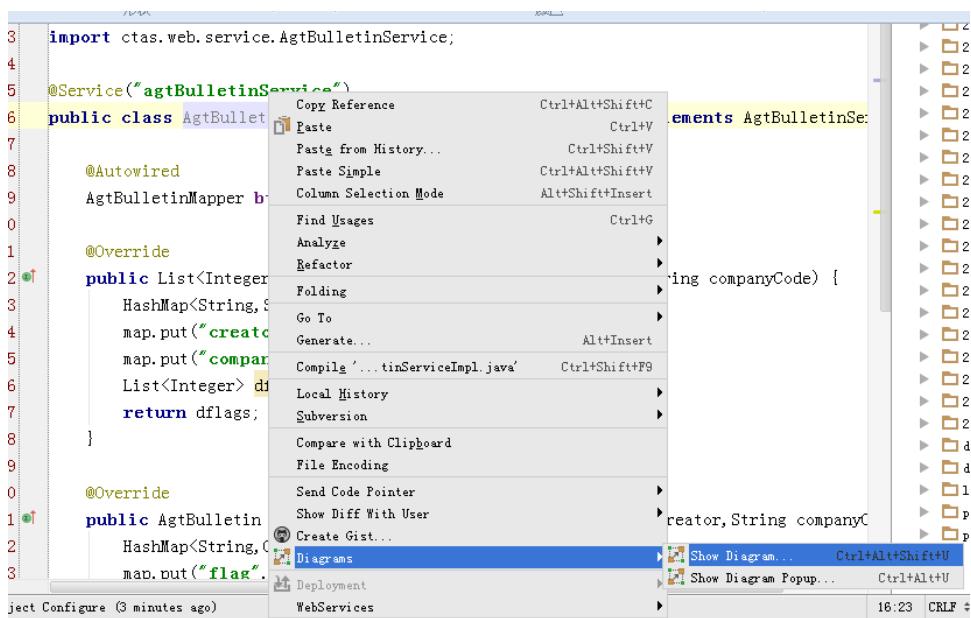
如上图，选择最近提交的TASK列表，选择Switch to，idea就会自动打开该TASK关联的源代码，并定位到修改过的代码行。当然如果该TASK已经Close了，也可以选择Remove将其清空。

二、UML类图插件

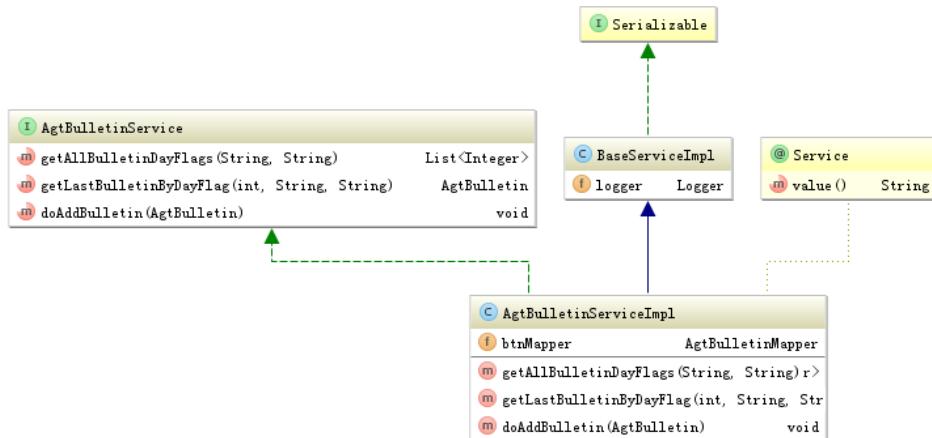
idea已经集成了该功能，只是默认没打开，仍然打开Settings界面，定位到Plugins，输入UML，参考下图：



确认UML这个勾已经勾上了，然后点击Apply，重启idea，然后仍然找一个java类文件，右击Diagram

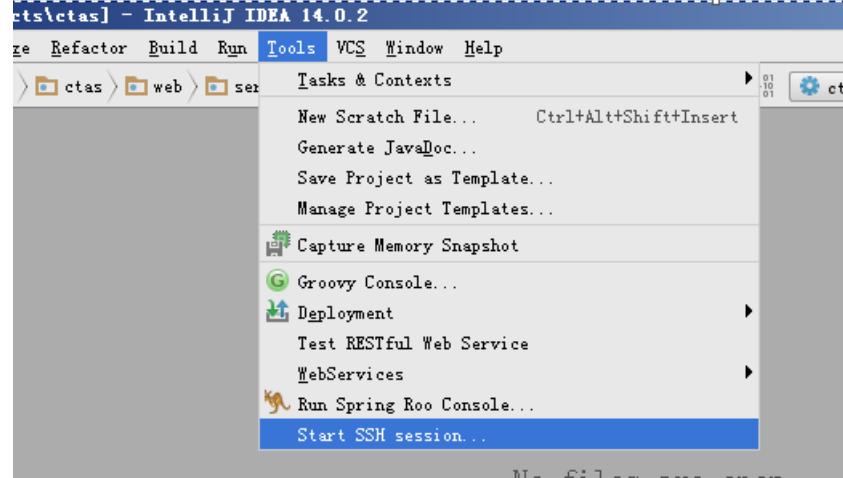


然后，就自个儿爽去吧

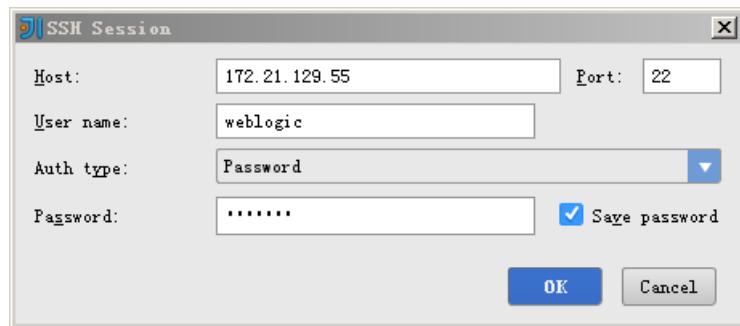


三、SSH集成

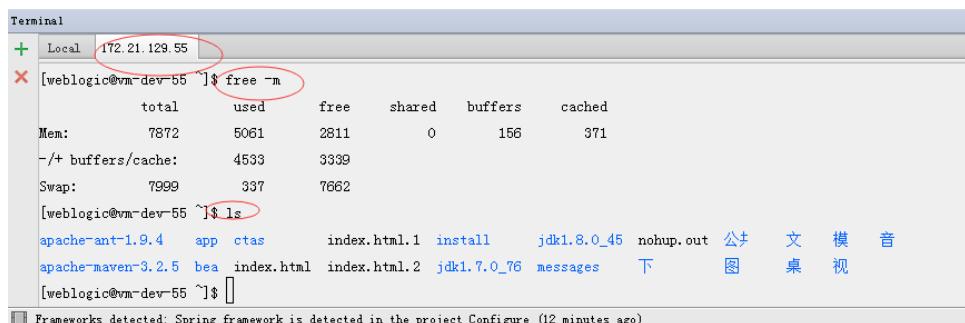
java项目经常会在linux上部署，每次要切换到SecureCRT这类终端工具未免太麻烦，idea也想到了这一点：



然后填入IP、用户名、密码啥的

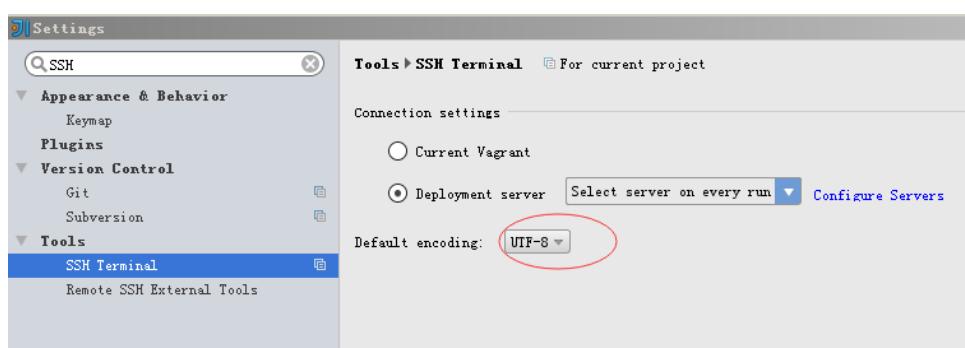


点击OK，就能连接上linux了

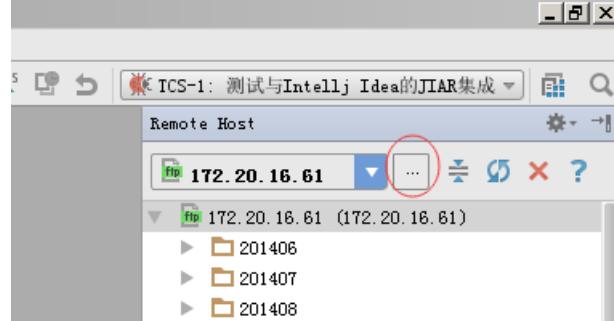


注：如果有中文乱码问题，可以在Settings里调整编码为utf-8

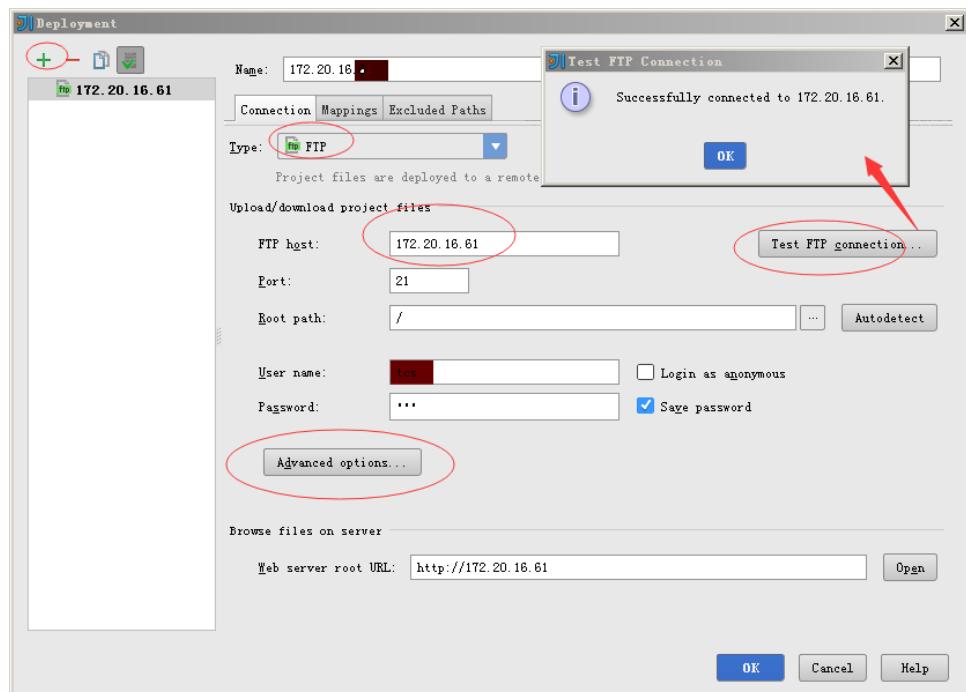
Tips：欢迎关注微信公众号：Java后端，每日技术博文推送。



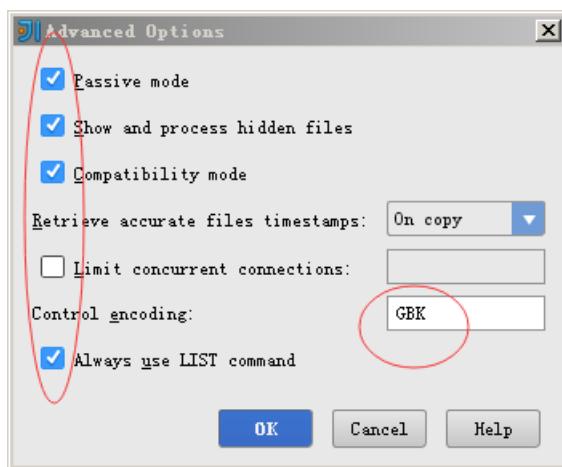
四、集成FTP



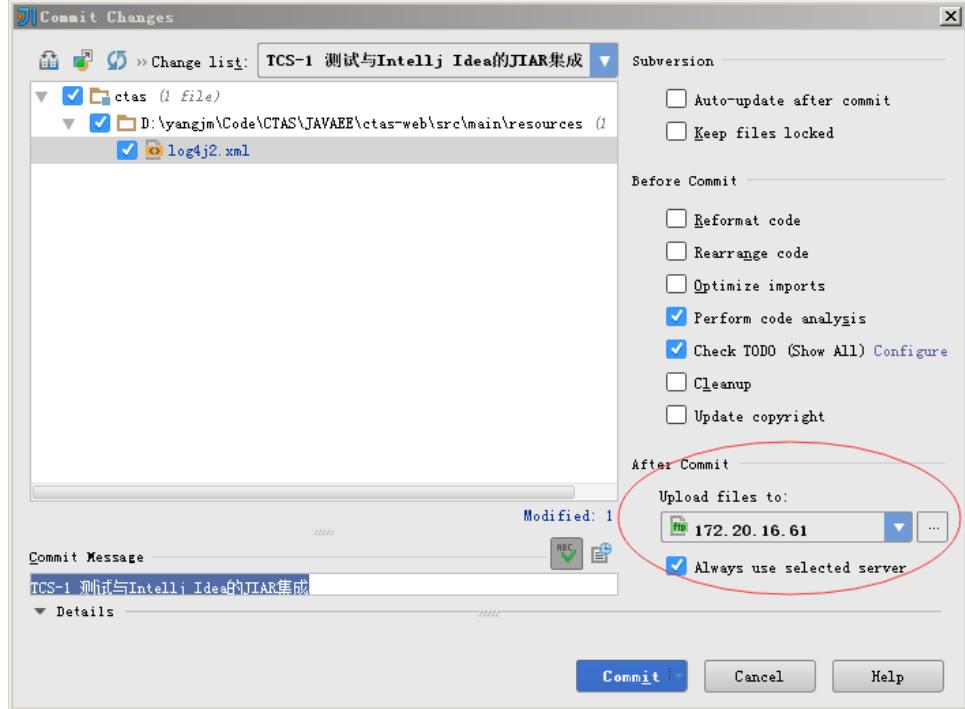
点击上图中的..., 添加一个Remote Host



填写ftp的IP、用户名、密码，根路径啥的，然后点击Test FTP Connection，正常的话，应该能连接，如果连接不通，点击Advanced Options，参考下图调整下连接选项



配置了FTP连接后，在提交代码时，可以选择提交完成后将代码自动上传到ftp服务器



五、Database管理工具

先看效果吧：

有了这个，再也不羡慕vs.net的db管理功能了。配置也很简单，就是点击+号，增加一个Data Source即可

唯一要注意的是，intellij idea不带数据库驱动，所以在上图中，要手动指定db driver的jar包路径。

如果看到这里，说明你喜欢这篇文章，请[转发、点赞](#)。微信搜索「web_resource」，关注后回复「进群」或者扫描下方二维码即可进入无广告交流群。

↓ 扫描二维码进群 ↓



推荐阅读

1. 你们心心念念的 GitHub 客户端终于来了！
2. Redis 实现「附近的人」这个功能
3. 一个秒杀系统的设计思考
4. 零基础认识 Spring Boot
5. 团队开发中 Git 最佳实践



喜欢文章, 点个在看 

[阅读原文](#)

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

MyBatis 代码生成器配置详解（IDEA）

阿进的写字台 Java后端 1月18日

点击上方 Java后端, 选择 **设为星标**

优质文章, 及时送达

作者 | 阿进的写字台

链接 | www.cnblogs.com/homejim/p/9782403.html

在使用 **mybatis** 过程中, 当手写 **JavaBean**和**XML** 写的越来越多的时候, 就越来越容易出错。这种重复性的工作, 我们当然不希望做那么多。

还好, **mybatis** 为我们提供了强大的代码生成--**MybatisGenerator**。

通过简单的配置, 我们就可以生成各种类型的实体类, Mapper接口, MapperXML文件, Example对象等。通过这些生成的文件, 我们就可以方便的进行单表进行增删改查的操作。

Tips: 关注微信公众号: Java后端, 获取每日推送。

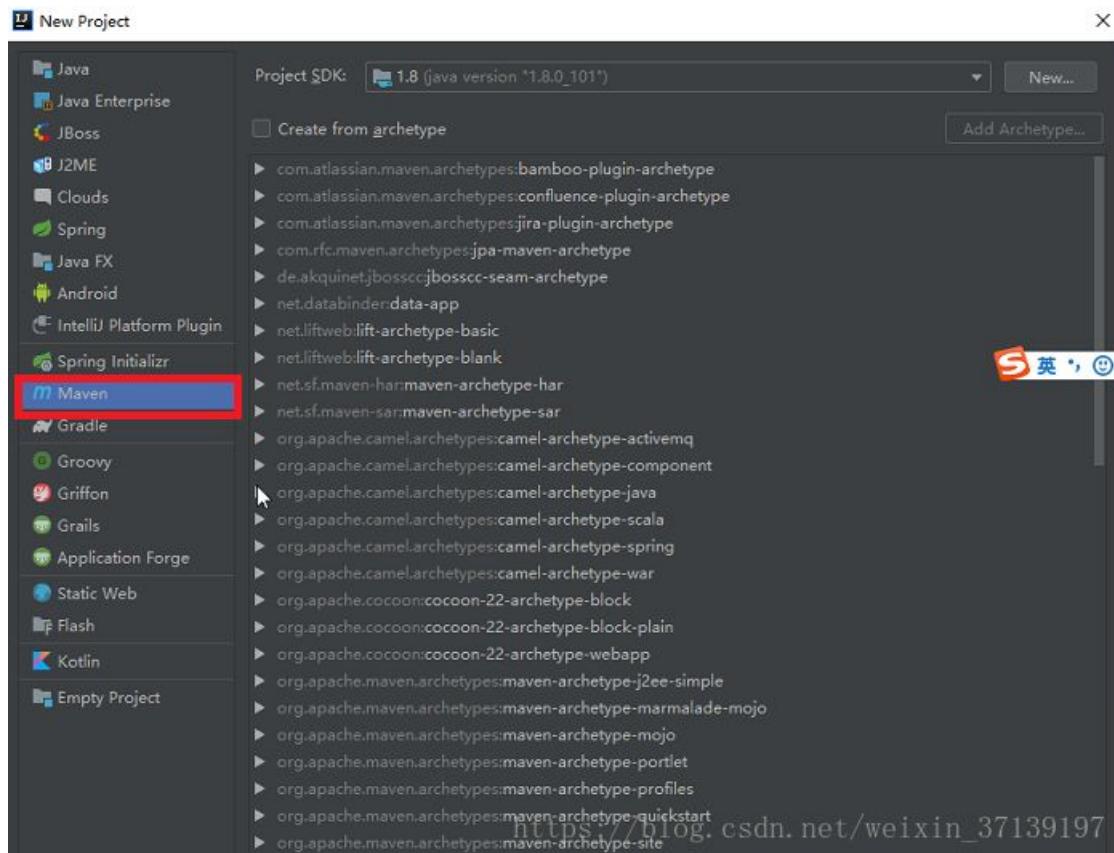
以下的工具使用的都是 **IDEA**

1.1 创建Maven项目

1.1.1 菜单上选择新建项目

File | New | Project

1.1.2 选择左侧的Maven



由于我们只是创建一个普通的项目， 此处点击 Next即可。

1.1.3 输入GroupId和ArtifactId

- 在我的项目中，

GroupId 填 com.homejim.mybatisplus

ArtifactId 填 mybatis-generator

点击 Next。

1.1.4 Finish

通过以上步骤， 一个普通的Maven项目就创建好了。

1.2 配置 generator.xml

其实名字无所谓， 只要跟下面的 **pom.xml** 文件中的对应上就好了。

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE generatorConfiguration PUBLIC
"-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
"http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd" >
<generatorConfiguration>

<classPathEntry location="C:\Users\Administrator\.m2\repository\mysql\mysql-connector-java\8.0.12\mysql-connector-java-8.0.12.jar" />
<context id="context" targetRuntime="MyBatis3">
    <commentGenerator>
        <property name="suppressAllComments" value="false"/>
        <property name="suppressDate" value="true"/>
    </commentGenerator>

    <jdbcConnection
        driverClass="com.mysql.jdbc.Driver"
        connectionURL="jdbc:mysql://localhost:3306/mybatis"
        userId="root"
        password="jim777"/>

    <javaTypeResolver>
        <property name="forceBigDecimals" value="false"/>
    </javaTypeResolver>

    <javaModelGenerator
        targetPackage="com.homejim.mybatis.entity"
        targetProject=".src\main\java">
        <property name="enableSubPackages" value="false"/>
        <property name="trimStrings" value="true"/>
    </javaModelGenerator>

    <sqlMapGenerator
        targetPackage="mybatis/mapper"
        targetProject=".src\main\resources">
        <property name="enableSubPackages" value="false"/>
    </sqlMapGenerator>

    <javaClientGenerator type="XMLMAPPER"
        targetPackage="com.homejim.mybatis.mapper"
        targetProject=".src\main\java">
        <property name="enableSubPackages" value="false"/>
    </javaClientGenerator>

    <table tableName="blog" />
</context>
</generatorConfiguration>

```

需要改一些内容：

1. 本地数据库驱动程序jar包的全路径（**必须要改**）。
2. 数据库的相关配置（**必须要改**）
3. 相关表的配置（**必须要改**）
4. 实体类生成存放的位置。
5. MapperXML 生成文件存放的位置。
6. Mapper 接口存放的位置。

如果不知道怎么改，请看后面的[配置详解](#)。

1.3 配置 pom.xml

在原基础上添加一些内容。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>com.homejim.mybatisplus</groupId>
<artifactId>mybatis-generator</artifactId>
<version>1.0-SNAPSHOT</version>

<build>
  <finalName>mybatis-generator</finalName>
  <plugins>
    <plugin>
      <groupId>org.mybatis.generator</groupId>
      <artifactId>mybatis-generator-maven-plugin</artifactId>
      <version>1.3.7</version>
      <configuration>

        <configurationFile>src/main/resources/generator.xml</configurationFile>
        <verbose>true</verbose>
        <overwrite>true</overwrite>
      </configuration>
    <executions>
      <execution>
        <id>Generate MyBatis Artifacts</id>
        <goals>
          <goal>generate</goal>
        </goals>
      </execution>
    </executions>
    <dependencies>
      <dependency>
        <groupId>org.mybatis.generator</groupId>
        <artifactId>mybatis-generator-core</artifactId>
        <version>1.3.7</version>
      </dependency>
    </dependencies>
  </plugin>
</plugins>
</build>

</project>
```

需要注意的是 **configurationFile** 中的文件指的是 **generator.xml**。因此路径写的是该文件的相对路径，名称也跟该文件相同。

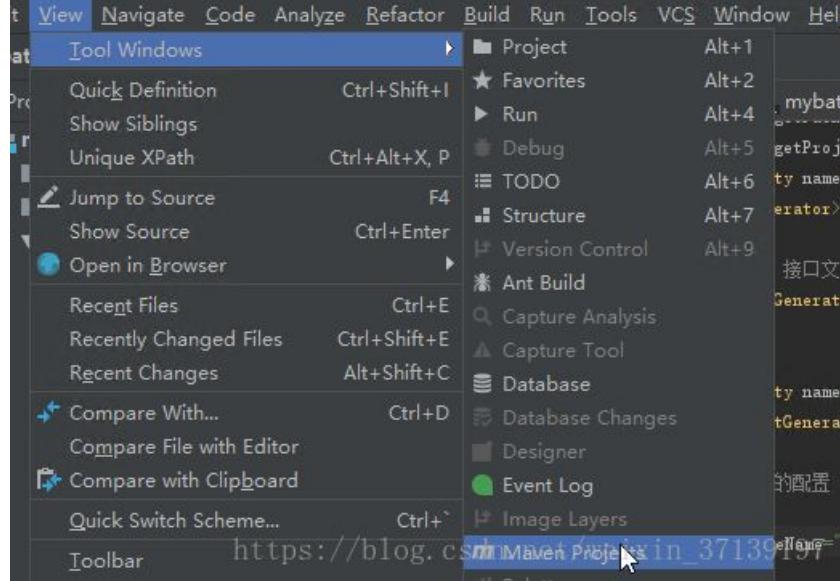
到此， **mybatis-generator** 就可以使用啦。

1.4 使用及测试

1.4.1 打开 Maven Projects 视图

在 IDEA 上，打开：

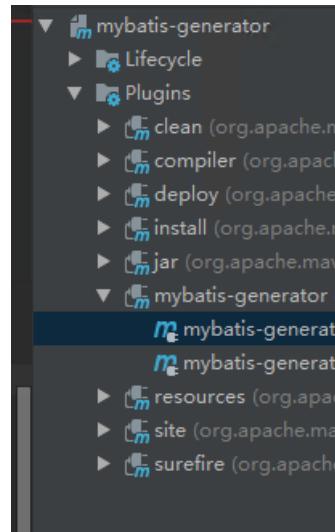
View | Tools | Windwos | Maven Projects



1.4.2 Maven Projects 中双击 mybatis-generator

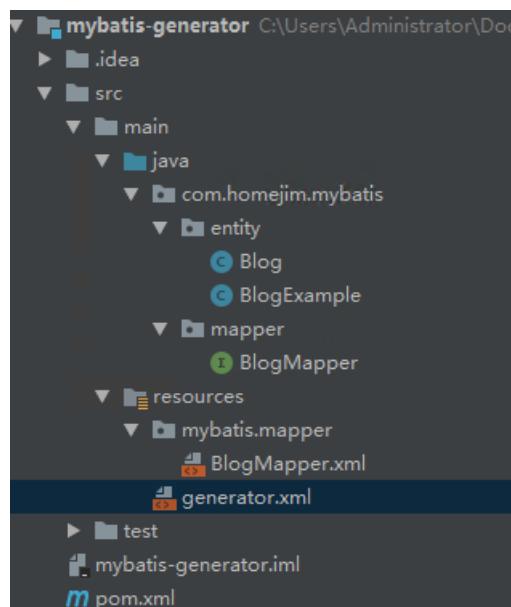
在右侧此时可以看到 Maven Projects 了。找到 mybatis-generator 插件。

mybatis-generator | Plugins | mybatis-generator | mybatis-generator



1.4.3 双击运行

运行正确后，生成代码，得到如下的结构



仅仅是上面那么简单的使用还不够爽。那么我们就可以通过更改 generator.xml 配置文件的方式进行生成的配置。

2.1 文档

推荐查看官方的文档。

英文不错的：<http://www.mybatis.org/generator/configreference/xmlconfig.html>

中文翻译版：<http://mbg.cndocs.ml/index.html>

2.2 官网没有的

2.2.1 property 标签

该标签在官网中只是说用来指定元素的属性，至于怎么用没有详细的讲解。

2.2.1.1 分隔符相关

```
<property name="autoDelimitKeywords" value="true"/>
<property name="beginningDelimiter" value="`"/>
<property name="endingDelimiter" value="`"/>
```

以上的配置对应的是 **mysql**，当数据库中的字段和数据库的关键字一样时，就会使用分隔符。

比如我们的数据列是 delete，按以上的配置后，在它出现的地方，就变成 `delete`。

2.2.1.2 编码

默认是使用当前的系统环境的编码，可以配置为 GBK 或 UTF-8。

```
<property name="javaFileEncoding" value="UTF-8"/>
```

我想项目为 UTF-8，如果指定生成 GBK，则自动生成的中文就是乱码。

2.2.1.3 格式化

```
<property name="javaFormatter" value="org.mybatis.generator.api.dom.DefaultJavaFormatter"/>

<property name="xmlFormatter" value="org.mybatis.generator.api.dom.DefaultXmlFormatter"/>
```

这些显然都是可以自定义实现的。

2.2.2 plugins 标签

plugins 标签用来扩展或修改代码生成器生成的代码。

在生成的 XML 中，是没有 **<cache>** 这个标签的。该标签是配置缓存的。

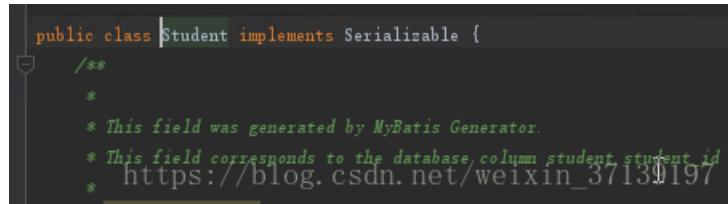
如果我们想生成这个标签，那么可以在 **plugins** 中进行配置。

```
<plugin type="org.mybatis.generator.plugins.CachePlugin">
  <property name="cache_eviction" value="LRU"/>
</plugin>
```

```
<cache eviction="LRU">
<!--
  WARNING - @mbg generated
  This element is automatically generated by MyBatis Generator, do not modify.
  -->
  https://blog.csdn.net/weixin_37139197
</cache>
```

比如你想生成的 **JavaBean** 中自行实现 **Serializable** 接口。

```
<plugin type="org.mybatis.generator.plugins.SerializablePlugin" />
```



还能自定义插件。

这些插件都蛮有用的，感觉后续可以专门开一篇文章来讲解。

看名称，就知道是用来生成注释用的。

默认配置：

```
<commentGenerator>
<property name="suppressAllComments" value="false"/>
<property name="suppressDate" value="false"/>
<property name="addRemarkComments" value="false"/>
</commentGenerator>
```

suppressAllComments：阻止生成注释， 默认值是false。

suppressDate：阻止生成的注释包含时间戳， 默认为false。

addRemarkComments：注释中添加数据库的注释， 默认为 false。

还有一个就是我们可以通过 **type** 属性指定我们自定义的注解实现类，生成我们自己想要的注解。自定义的实现类需要实现 org.mybatis.generator.api.CommentGenerator。

2.2.4 源码

<https://github.com/homejim/mybatis-cn>

homejim / mybatis-cn

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

mybatis源码的中文注释以及mybatis的使用和源码解析

mybatis mybatis-sources mybatis-chinese Manage topics

83 commits 1 branch 0 releases 1 contributor View license

Branch: master New pull request Create new file Upload files Find file Clone or download



微信扫描二维码，关注我的公众号

[阅读原文](#)

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

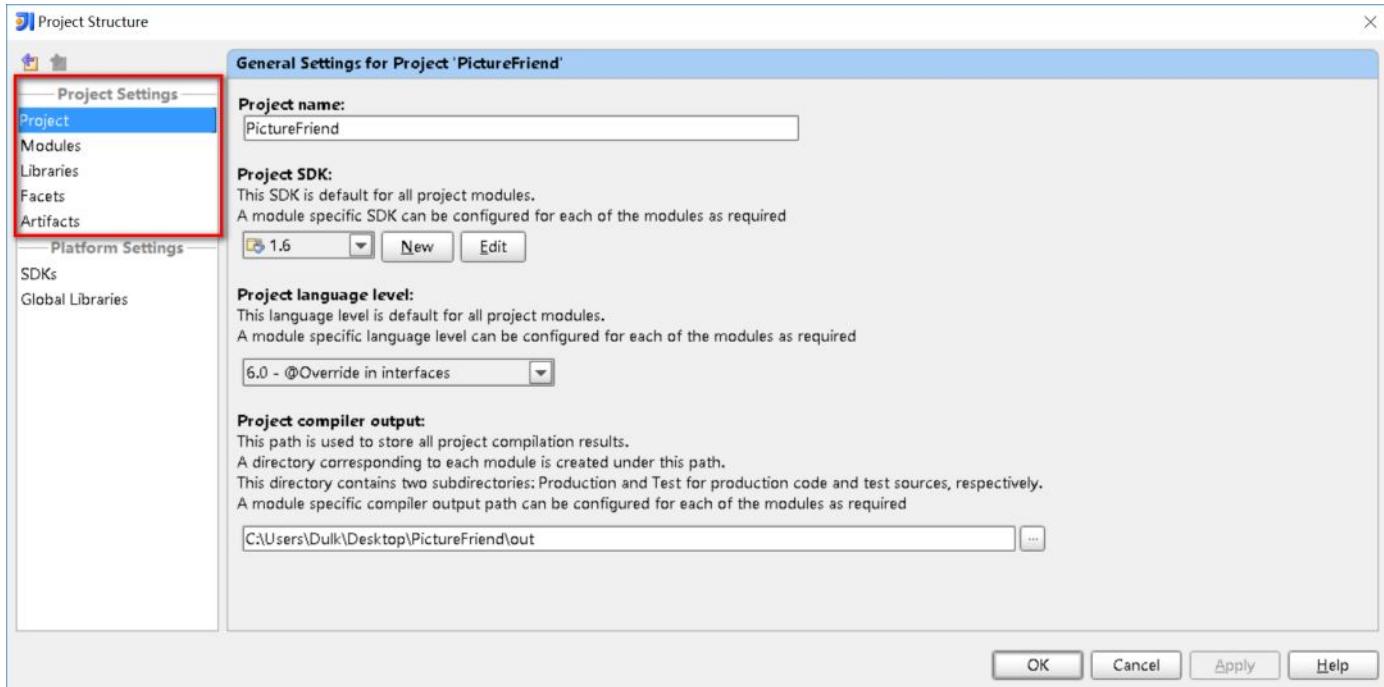
不会 IntelliJ IDEA 项目配置？请收藏这篇文章

Dulk Java后端 3月1日

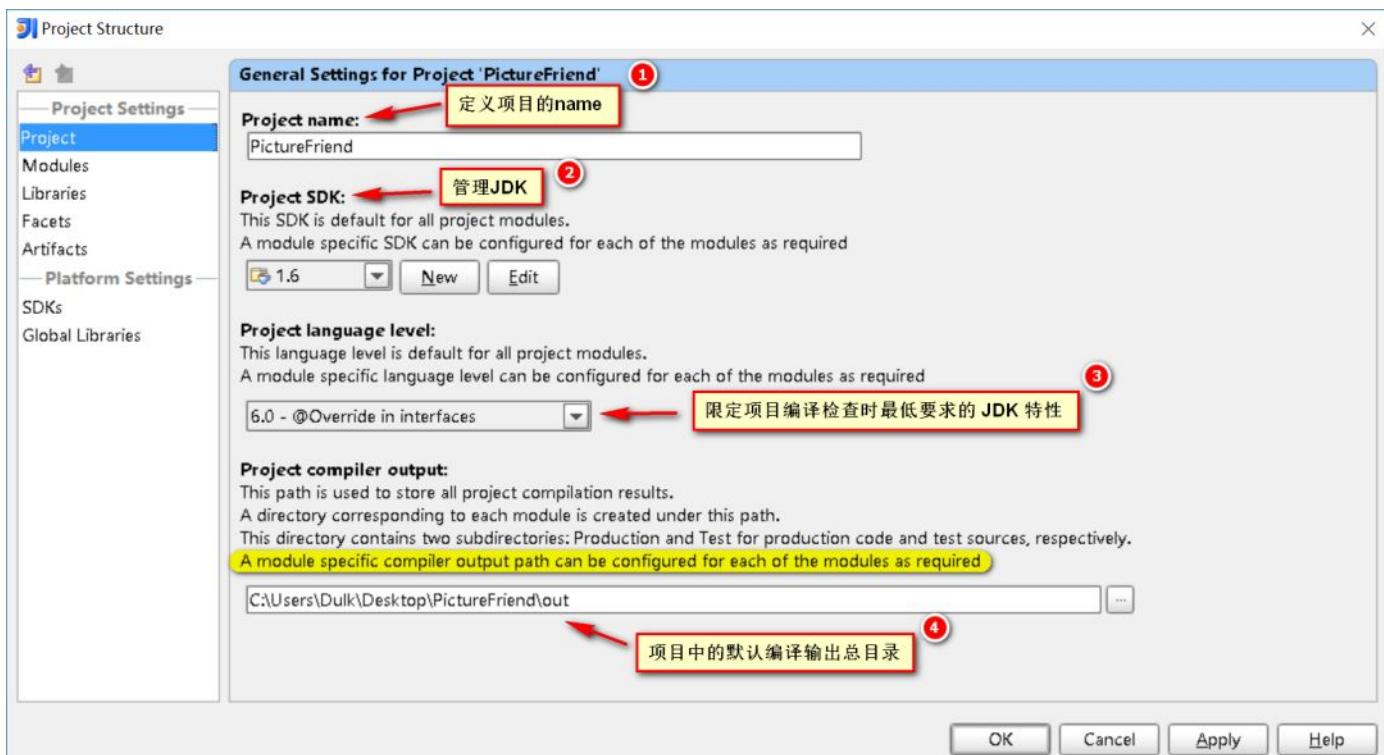


1、项目配置的理解

IDEA 中最重要的各种设置项，就是这个 Project Structure 了，关乎你的项目运行，缺胳膊少腿都不行。最近公司正好也是用之前自己比较熟悉的IDEA而不是Eclipse，为了更深入理解和使用，就找来各种资料再研究一下，这里整理后来个输出。



1.1 Project



定义项目的名称；

2. Project SDK:

设置该项目使用的JDK，也可以在此处新添加其他版本的JDK；

3. Project language level:

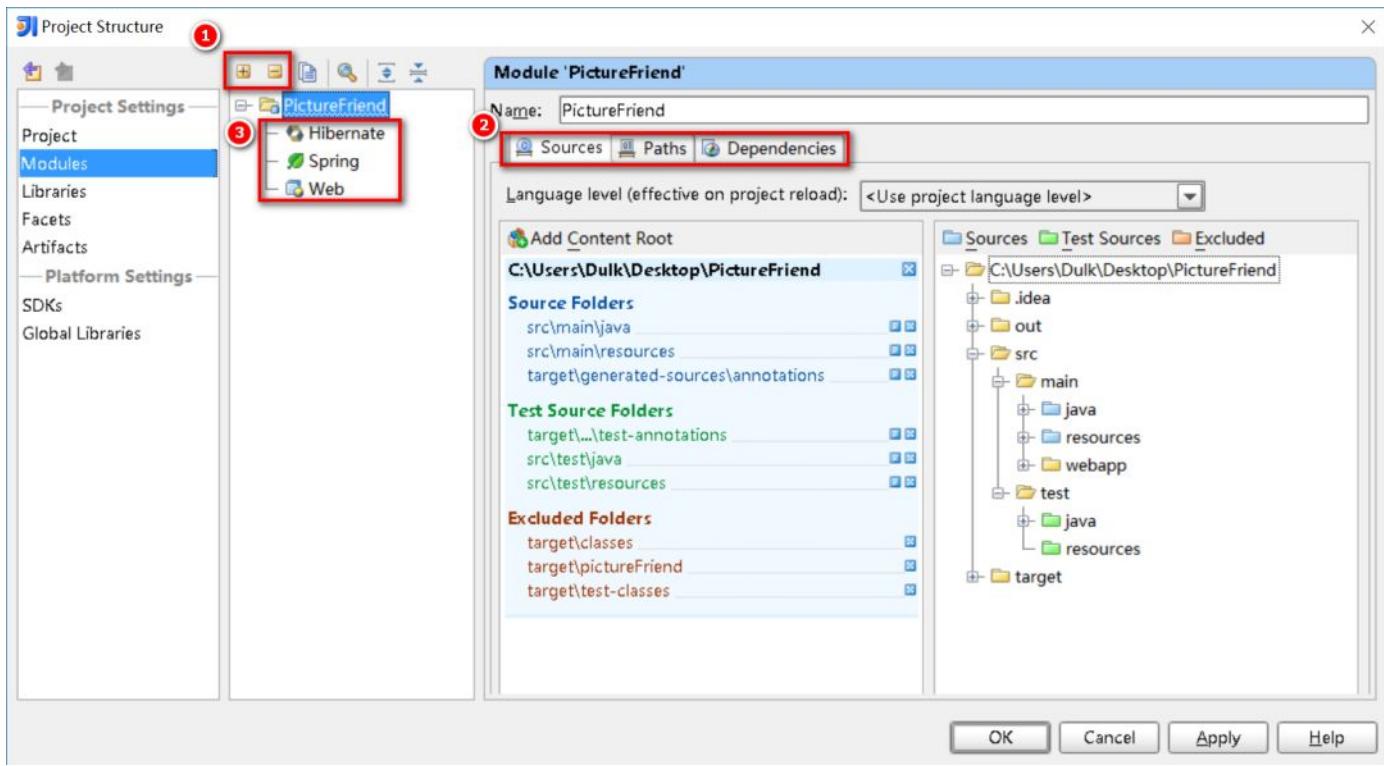
这个和JDK的类似，区别在于，假如你设置了JDK1.8，却只用到1.6的特性，那么这里可以设置语言等级为1.6，这个是限定项目编译检查时最低要求的JDK特性；

4. Project compiler output:

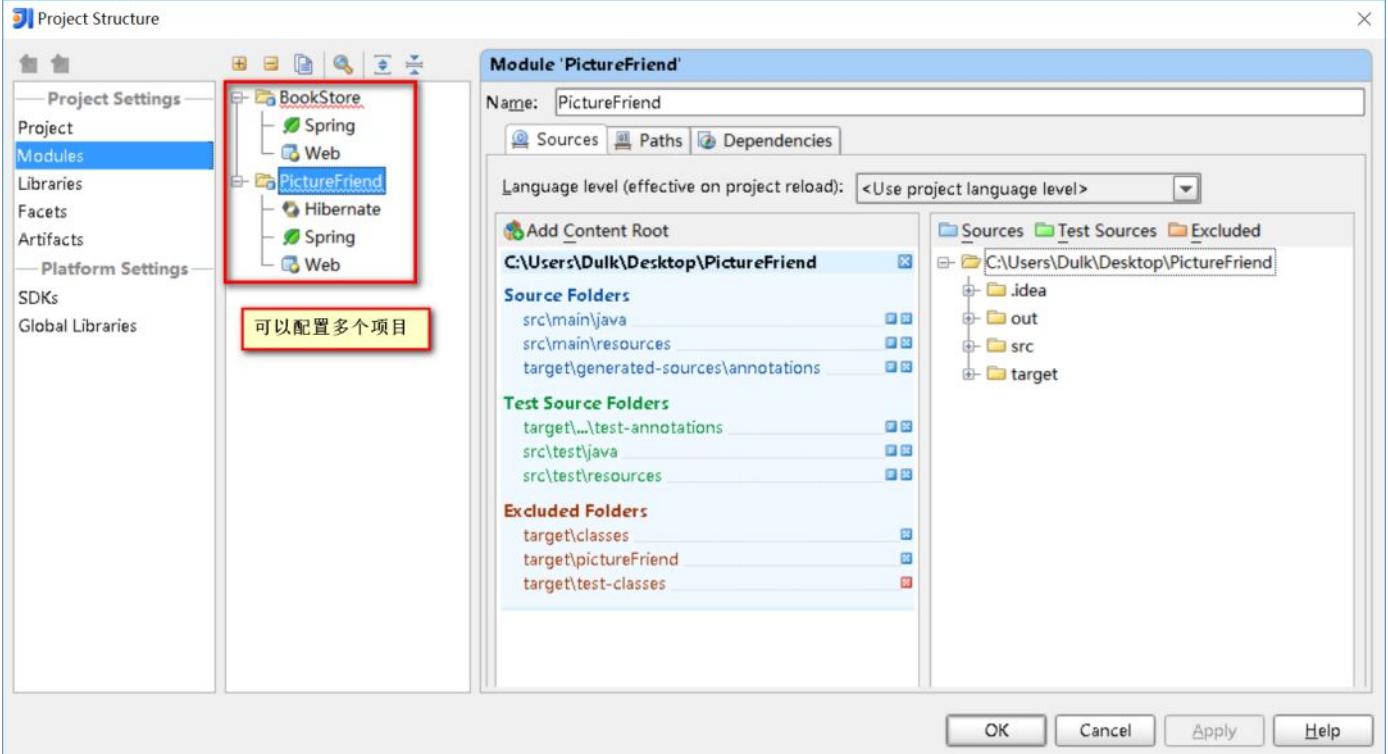
项目中的默认编译输出总目录，如图黄色部分，实际上每个模块可以自己设置特殊的输出目录 (Modules - (project) - Paths - Use module compile output path)，所以这个设置有点鸡肋。

小插曲：更多 IDEA 相关文章可以本公众号（Java后端）回复 技术博文，获取~

1.2 Modules



1.2.1 增删子项目



一个项目中可以有多个子项目，每个子项目相当于一个模块。一般我们项目只是单独的一个，IntelliJ IDEA 默认也是单子项目的形式，所以只需要配置一个模块。

(此处的两个项目引入仅作示例参考)

1.2.2 子项目配置

每个子项目都对应了 Sources、Paths、Dependencies 三大配置选项：

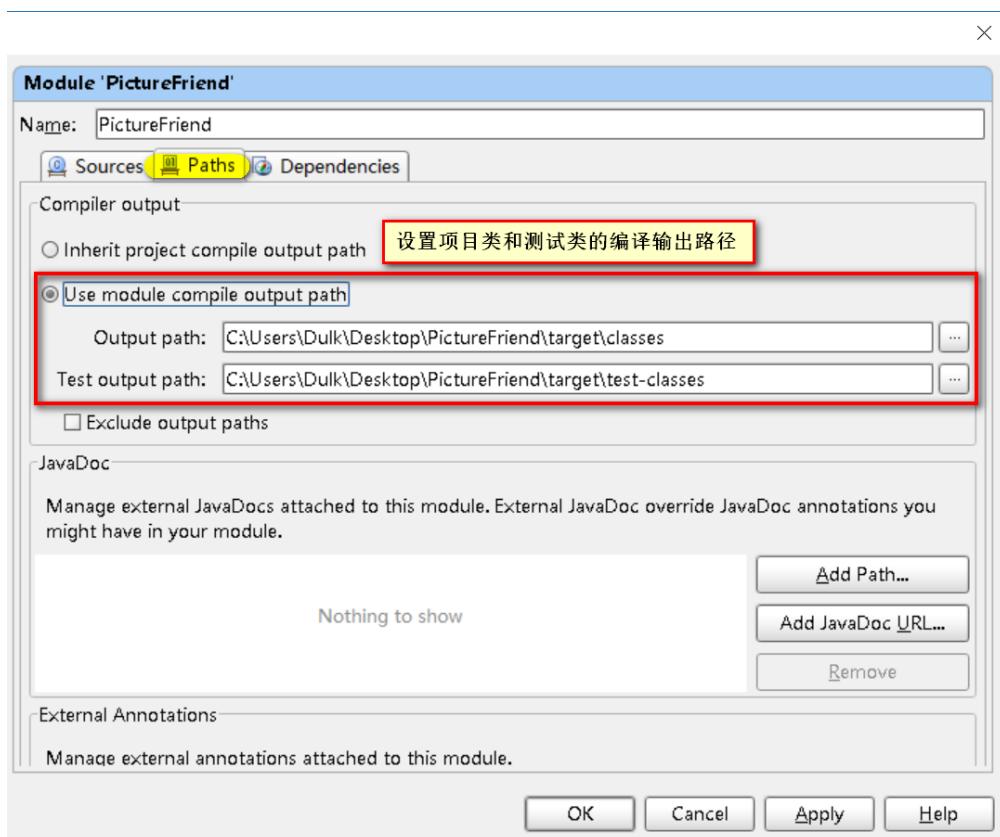
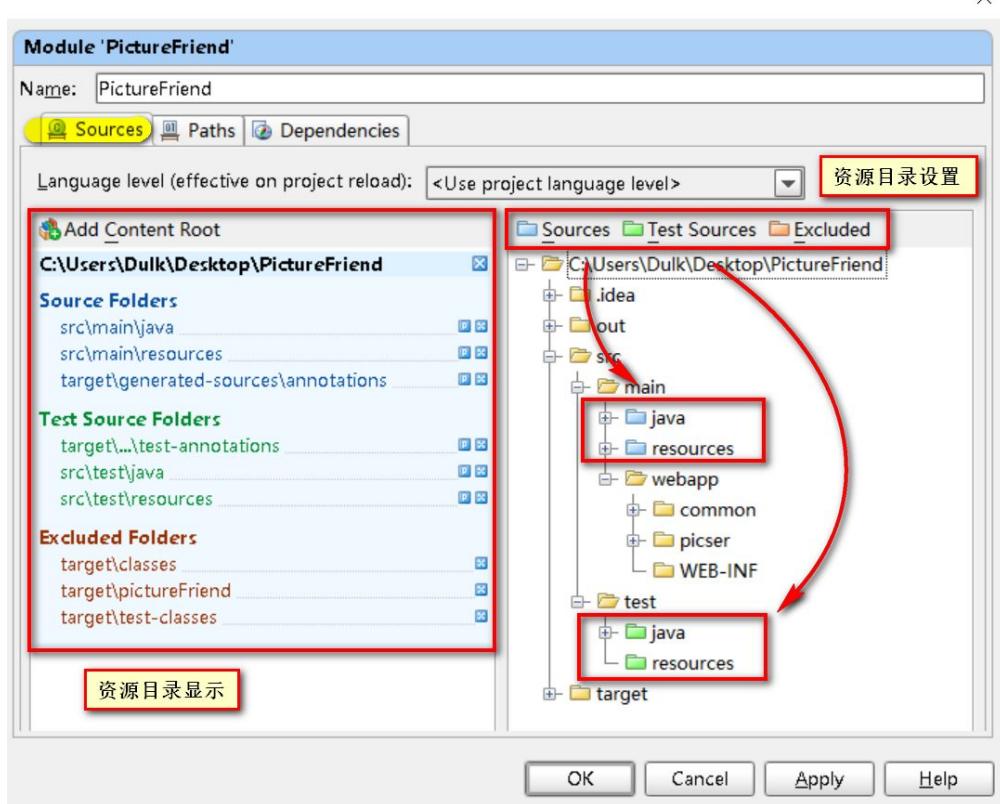
Sources:

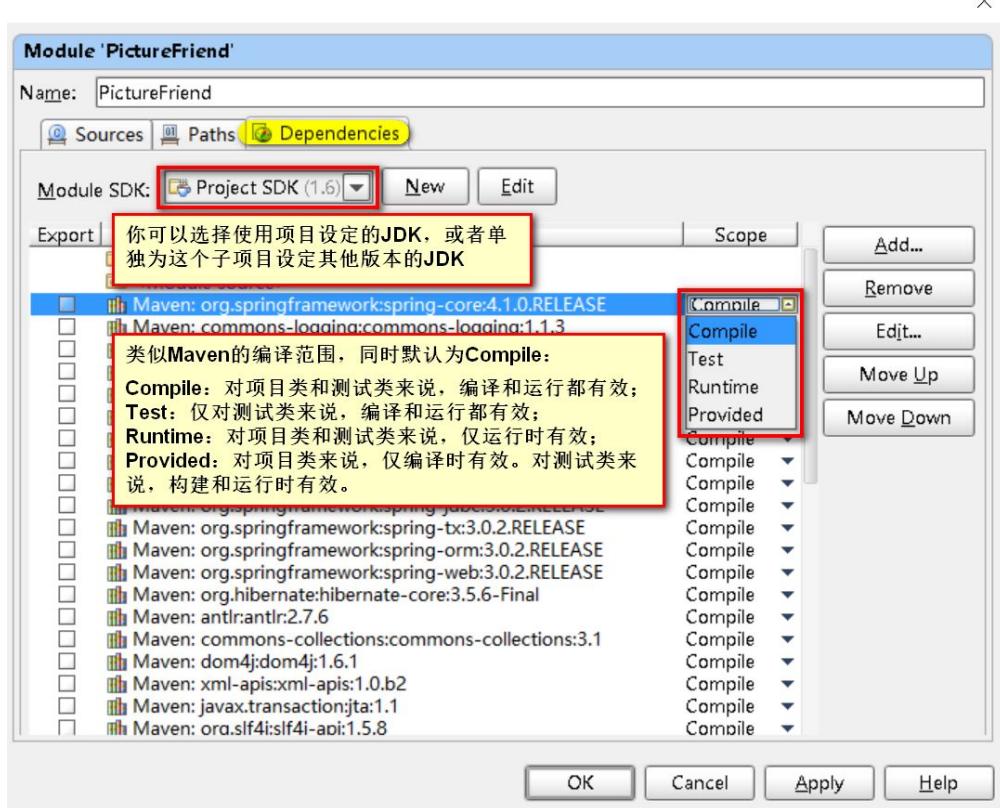
显示项目的目录资源，那些是项目部署的时候需要的目录，不同颜色代表不同的类型；

Paths:

可以指定项目的编译输出目录，即项目类和测试类的编译输出地址（替换掉了Project的默认输出地址）

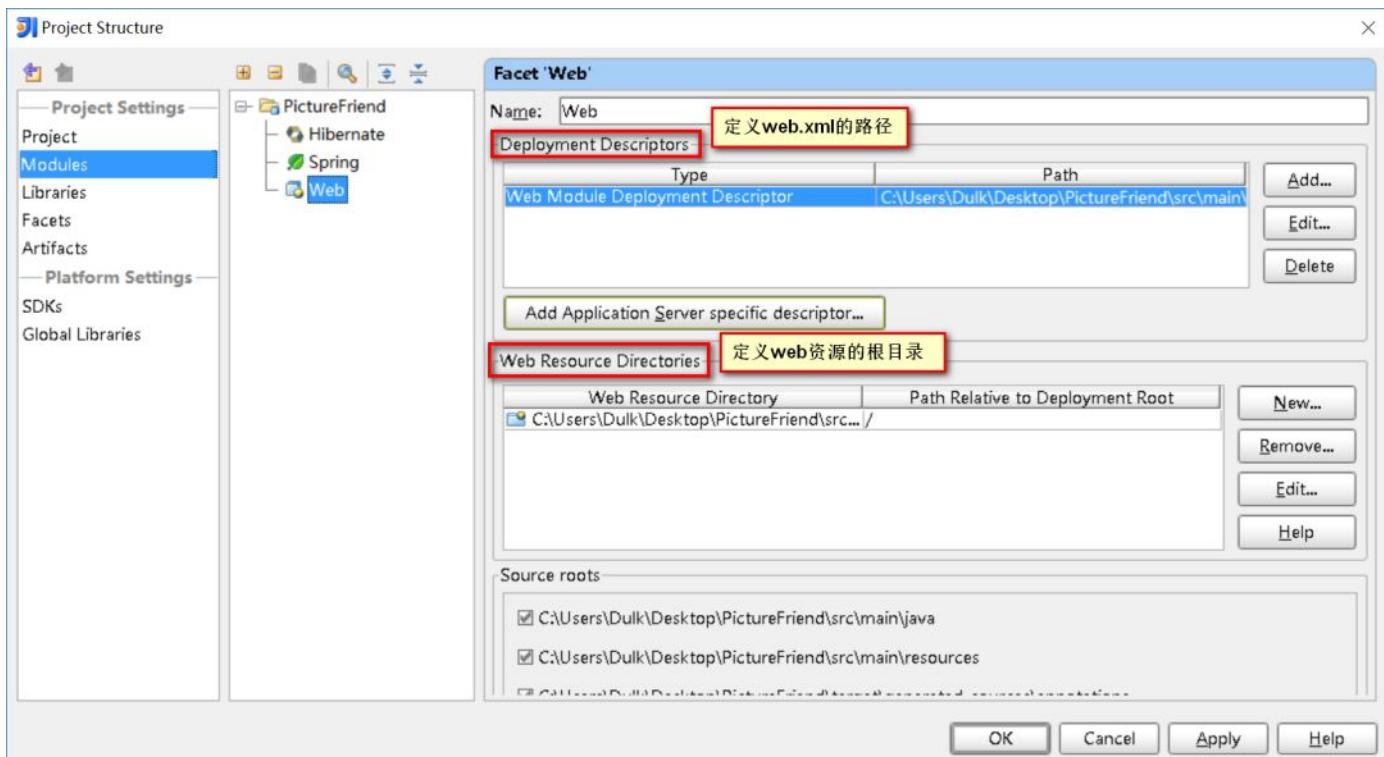
Dependencies: 项目的依赖





1.2.3 增删框架 (Web部署-1)

每个子项目之下都可以定义它所使用的框架，这里重点说明一下Web部分的设置。



1.3 Libraries

这里可以显示所添加的jar包，同时也可以添加jar包，并且可以把多个jar放在一个组里面，类似于jar包整理。

这里默认将每个jar包做为了一个单独的组（未测试，待定）。

1.4 Facets

官方的解释是：

When you select a framework (a facet) in the element selector pane, the settings for the framework are shown in the right-hand part of the dialog.

(当你在左边选择面板点击某个技术框架，右边将会显示这个框架的一些设置)

说实话，并没有感觉到有什么作用。

1.5 Artifacts (Web部署-2)

项目的打包部署设置，这个是项目配置里面比较关键的地方，重点说一下。

先理解下它的含义，来看看官方定义的artifacts：

An artifact is an assembly of your project assets that you put together to test, deploy or distribute your software solution or its part. Examples are a collection of compiled Java classes or a Java application packaged in a Java archive, a Web application as a directory structure or a Web application archive, etc.

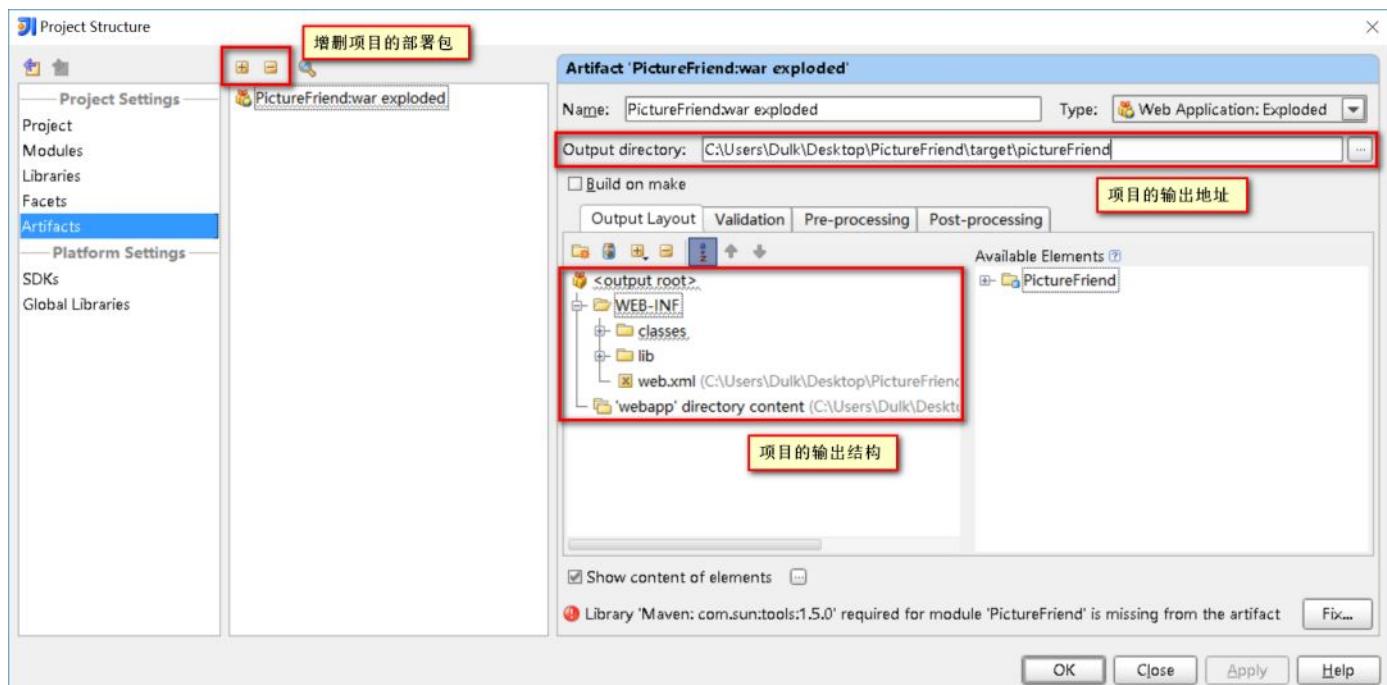
即编译后的Java类，Web资源等的整合，用以测试、部署等工作。再白话一点，就是说某个module要如何打包，例如war exploded、war、jar、ear等等这种打包形式。某个module有了Artifacts就可以部署到应用服务器中了。

jar: Java ARchive, 通常用于聚合大量的Java类文件、相关的元数据和资源(文本、图片等)文件到一个文件，以便分发Java平台应用软件或库；

war: Web application ARchive, 一种JAR文件，其中包含用来分发的JSP、Java Servlet、Java类、XML文件、标签库、静态网页(HTML和相关文件)，以及构成Web应用程序的其他资源；

exploded: 在这里你可以理解为展开，不压缩的意思。也就是war、jar等产出物没压缩前的目录结构。建议在开发的时候使用这种模式，便于修改了文件的效果立刻显现出来。)

默认情况下，IDEA的 Modules 和 Artifacts 的 output目录已经设置好了，不需要更改，打成war包的时候会自动在 WEB-INF目录下生成classes，然后把编译后的文件放进去。



你可能对这里的输出目录不太理解，之前不是配置过了文件编译的输出目录了吗？为什么这里还有一个整合这些资源的目录呢？它又做了哪些事呢？

其实，实际上，当你点击运行tomcat时，默认就开始做以下事情：

编译，IDEA在保存/自动保存后不会做编译，不像Eclipse的保存即编译，因此在运行server前会做一次编译。

编译后class文件存放在指定的项目编译输出目录下（见1.2.2）；

根据artifact中的设定对目录结构进行创建；

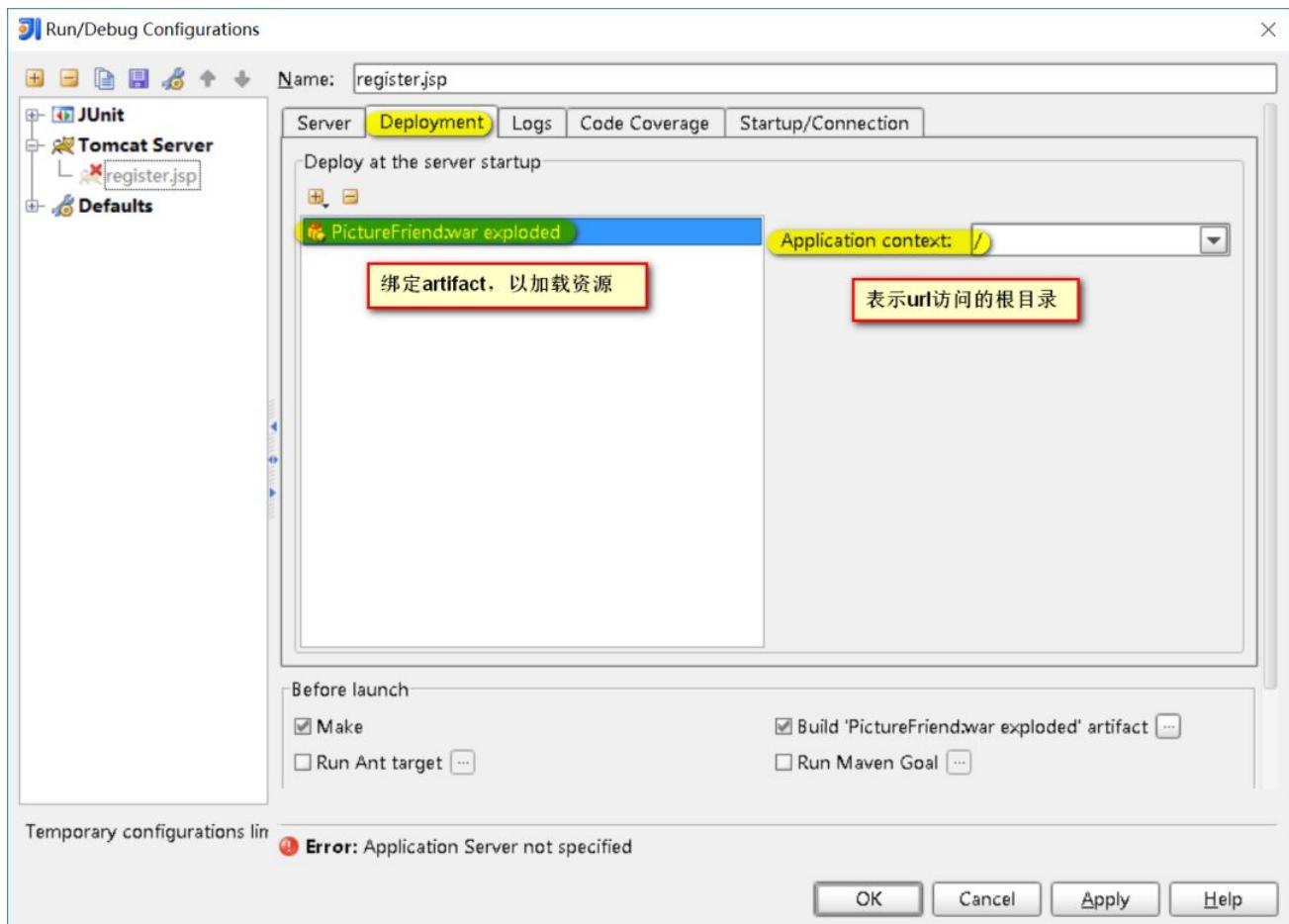
拷贝web资源的根目录下的所有文件到artifact的目录下（见1.2.3）；

拷贝编译输出目录下的classes目录到artifact下的WEB-INF下（见1.2.2）；

拷贝lib目录下所需的jar包到artifact下的WEB-INF下；

运行server，运行成功后，如有需要，会自动打开浏览器访问指定url。

在这里还要注意的是，配置完成的artifact，需要在tomcat中进行添加：



2、参考链接

IntelliJ IDEA 项目相关的几个重要概念介绍

whudoc.qiniudn.com/2016/IntelliJ-IDEA-Tutorial/project-composition-introduce.html

www.jetbrains.com/help/idea/2016.3/dependencies-tab.html

Dependencies Tab

<https://www.jetbrains.com/help/idea/2016.3/facet-page.html>

Facet Page

Working with Artifacts

www.jetbrains.com/help/idea/2016.3/working-with-artifacts.html#artifact_def

IntelliJ IDEA 14.x 中的Facets和Artifacts的区别

www.cnblogs.com/52php/p/5677661.html

IntelliJ使用指南——深入理解IntelliJ的Web部署逻辑

white-crucifix.iteye.com/blog/2070830

IntelliJ IDEA WEB项目的部署配置

my.oschina.net/lujianing/blog/186737

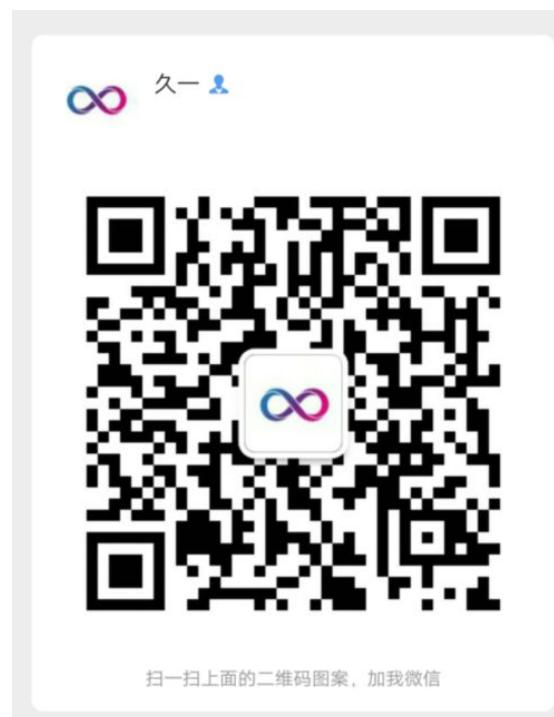
作者 | Dulk

链接 | www.cnblogs.com/deng-cc/p/6416332.html

- END -

如果看到这里，说明你喜欢这篇文章，请[转发、点赞](#)。微信搜索「web_resource」，欢迎添加小编微信「focusoncode」，每日朋友圈更新一篇高质量技术博文(无广告)。

↓ 扫描二维码添加小编 ↓



推荐阅读

[1. 删库跑路！创始人回应了](#)

[2. 高频使用的 Git 命令](#)

[3. 一个依赖搞定 Session 共享](#)

[4. 浅谈 Web 网站架构演变过程](#)



微信搜一搜

Q Java后端

[阅读原文](#)

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

丢掉 Postman ! 我选择了 IDEA REST Client

凯京技术团队 Java后端 2019-11-19

点击上方 Java后端, 选择 [设为星标](#)

优质文章, 及时送达

作者 | 陈凯玲

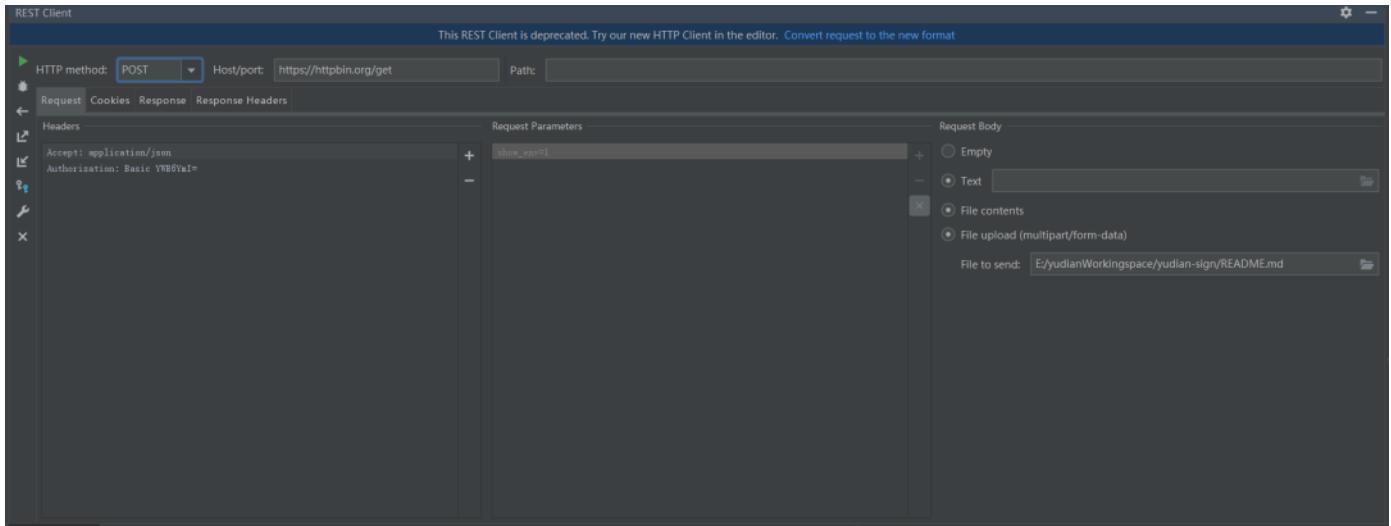
来源 | my.oschina.net/keking/blog/3104972

接口调试是每个软件开发从业者必不可少的一项技能，一个项目的完成，可能接口测试调试的时间比真正开发写代码的时间还要多，几乎是每个开发的日常工作项。所谓工欲善其事必先利其器，在没有尝到IDEA REST真香之前，postman(chrome的一款插件)确实是一个非常不错的选择，具有完备的REST Client功能和请求历史记录功能。但是当使用了IDEA REST之后，postman就可以丢了，因为，IDEA REST Client具有postman的所有功能，而且还有postman没有的功能，继续往下看。

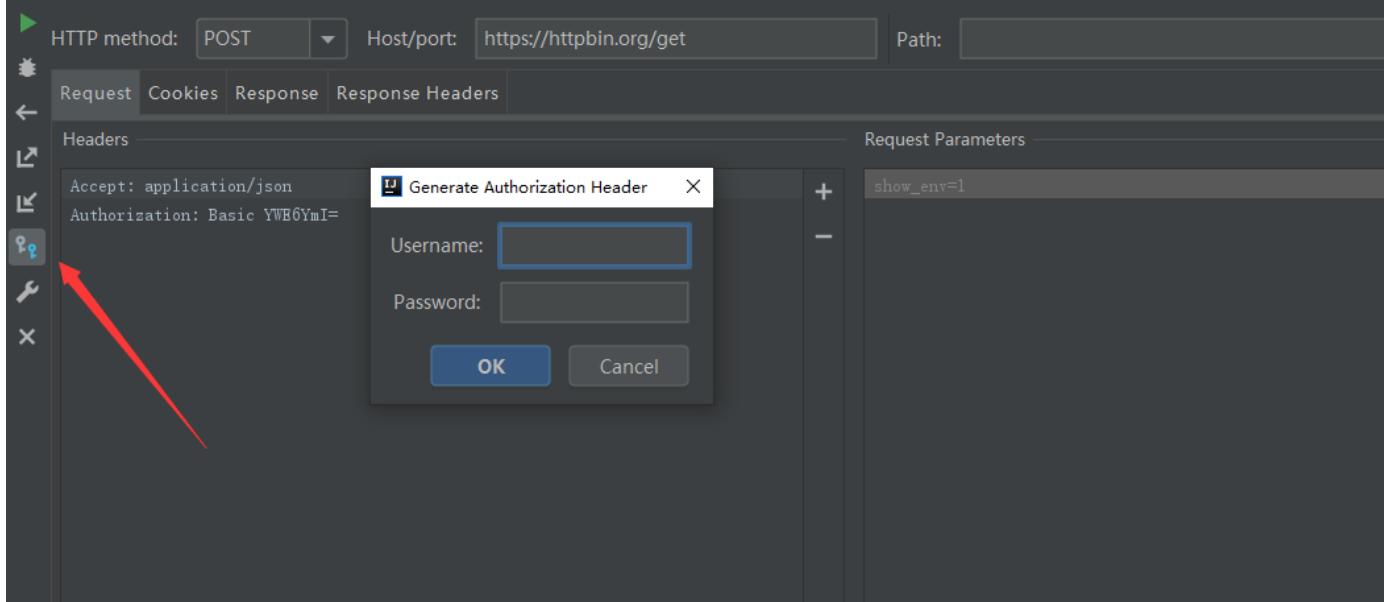
真香定律的原因有如下几个：

1. 首先postman的所有功能IDEA REST Client都具备了，如REST Client控制台和历史请求记录
2. 其次如果能够在一个生产工具里完成开发和调试的事情，干嘛要切换到另一个工具呢
3. 然后IDEA REST Client还支持环境配置区分的功能，以及接口响应断言和脚本化处理的能力
4. IDEA REST Client的请求配置可以用文件配置描述，所以可以跟随项目和项目成员共享

从顶层工具栏依次Tools -> HTTP Client -> Test RESTFUL Web Service 打开后，IDEA REST Client控制台的界面如下样式：



可以看到，这个控制台展示的功能区和postman已经没什么差别了，包括请求方式，请求参数和请求头的填充都已经包含了，特别说明下的是，如果请求的方式是Authorization :Basic这种方式认证的话，可以点击下图所示的按钮，会弹出填充用户名和密码的窗口出来，填完后会自动补充到Authorization 的header里面去



IntelliJ IDEA自动将最近执行的50个请求保存到http-requests-log.http 文件中,该文件存储在项目的.idea / httpRequests / 目录下。使用请求历史记录,您可以快速导航到特定响应并再次发出请求。文件内容大如下图所示,再次发出请求只要点击那个运行按钮即可。如果从请求历史记录再次发出请求,则其执行信息和响应输出的链接将添加到请求历史记录文件的顶部。

```

1 1 ▶ GET https://httpbin.org/get
2   Accept: application/json
3   Authorization: Basic YWE6YmI=
4
5   <> 2019-09-11T015235.200.json
6   ###
7
8 1 ▶ POST https://httpbin.org/get
9   Accept: application/json
10  Authorization: Basic YWE6YmI=
11
12  <> 2019-09-11T015124.405.html
13  |
14  ###

```

上面的历史记录就是一个完整的IDEA REST Client请求脚本,如果你是从控制台触发的,那么可以直接复制历史请求记录的文件放到项目里作为HTTP请求的脚本,给其他成员共享,如果不是,也可以直接新建一个.http或者.rest结尾的文件,IDEA会自动识别为HTTP请求脚本。

Tips: 可以微信搜索: Java后端,关注后加入咱们自己的交流群。

语法部分

演示POST请求

```
### 演示POST请求  
POST {{baseUrl}}/get?show_env=1  
Accept: application/json
```

```
{  
    "name": "a"  
}
```

演示GET请求

```
GET {{baseUrl}}/post  
Content-Type: application/x-www-form-urlencoded  
  
id=999&value=content
```

首先通过###三个井号键来分开每个请求体，然后请求url和header参数是紧紧挨着的，请求参数不管是POST的body传参还是GET的parameter传参，都是要换行的

环境区分

细心的你可能发现了上面示例的代码，没有真实的请求地址，取而代之的，是一个{{baseUrl}}的占位符，这个就是IDEA REST Client真香的地方，支持从指定的配置文件中获取到环境相关的配置参数，不仅baseUrl可以通过占位符替换，一些请求的参数如果和接口环境相关的都可以通过配置文件来区分。

首先在.http的脚本同目录下创建一个名为http-client.private.env.json的文件，然后内容如下，一级的key值时用来区分环境的，比如，dev、uat、pro等，环境下的对象就是一次HTTP请求中能够获取到的环境变量了，你可以直接在请求的HTTP的脚本中通过{{xx}}占位符的方式获取到这里配置的参数

```
{  
    "uat": {  
        "baseUrl": "http://gateway.xxx.cn/",  
        "username": "",  
        "password": ""  
    },  
    "dev": {  
        "baseUrl": "http://localhost:8888/",  
        "username": "",  
        "password": ""  
    }  
}
```

那么在选择执行请求的时候，IDEA就会让你选执行那个环境的配置，如：

```
15     ### 演示POST请求
16
17     POST {{baseUrl}}/post?t?show_env=1
18     Run with 'dev' environment
19     Run with 'uat' environment
20     Run {{baseUrl}}/get
21
22     {
23         "name": "a"
24     }
25     ### 演示GET请求
26
27     GET {{baseUrl}}/post
28     Content-Type: application/x-www-form-urlencoded
29
30     id=999&value=content
```

结果断言

IDEA REST Client可以针对接口的响应值进行脚本化的断言处理，立马从一个接口调试工具上升到测试工具了，比如：

Successful test: check response status is 200

```
GET https://httpbin.org/status/200

> {%
client.test("Request executed successfully", function() {
  client.assert(response.status === 200, "Response status is not 200");
});
%}
```

结果值暂存

试想下这样的场景，当一个系统需要通过认证才能访问的时候，如果用postman的时候，是不是先访问登录接口，然后获得token后，手动粘贴复制到新的调试接口的header参数里面去，这太麻烦了，IDEA REST Client还有一个真香的功能，可以完美解决这个问题，请看下面的脚本：

演示POST请求

```
POST https://httpbin.org/post
Content-Type: application/json

{
  "user": "admin",
  "password": "123456"
}

> {%
  client.global.set("auth_token", response.body.json.token);
%}
```

演示GET请求

```
GET https://httpbin.org/headers
Authorization: Bearer {{auth_token}}
```

在第一个认证的请求结束后，可以在response里拿到返回的token信息，然后我们通过脚本设置到了全局变量里，那么在接下来的接口请求中，就可以直接使用双大括号占位符的方式获取到这个token了

结语

postman有口皆碑，确实是一个非常不错的必备工具，之前给比人推荐这种工具时总是安利他postman。但是，IDEA REST Client也真的很不错，值得尝试一下，后面安利这种工具就切换到IDEA REST Client了，postman反正被我丢掉了。和第三方做接口对接时，项目里必备一个rest-http.http接口请求文件，满足自己的同时也方便了他人。

作者简介：

陈凯玲，2016年5月加入凯京科技。现任凯京科技研发中心架构组经理，救火队队长。独立博客KL博客（<http://www.kailing.pub>）博主。

[阅读原文](#)

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

如何在 IntelliJ IDEA 中使用 Git

Java后端 2019-12-26

点击上方 Java后端, 选择 [设为星标](#)

优质文章, 及时送达

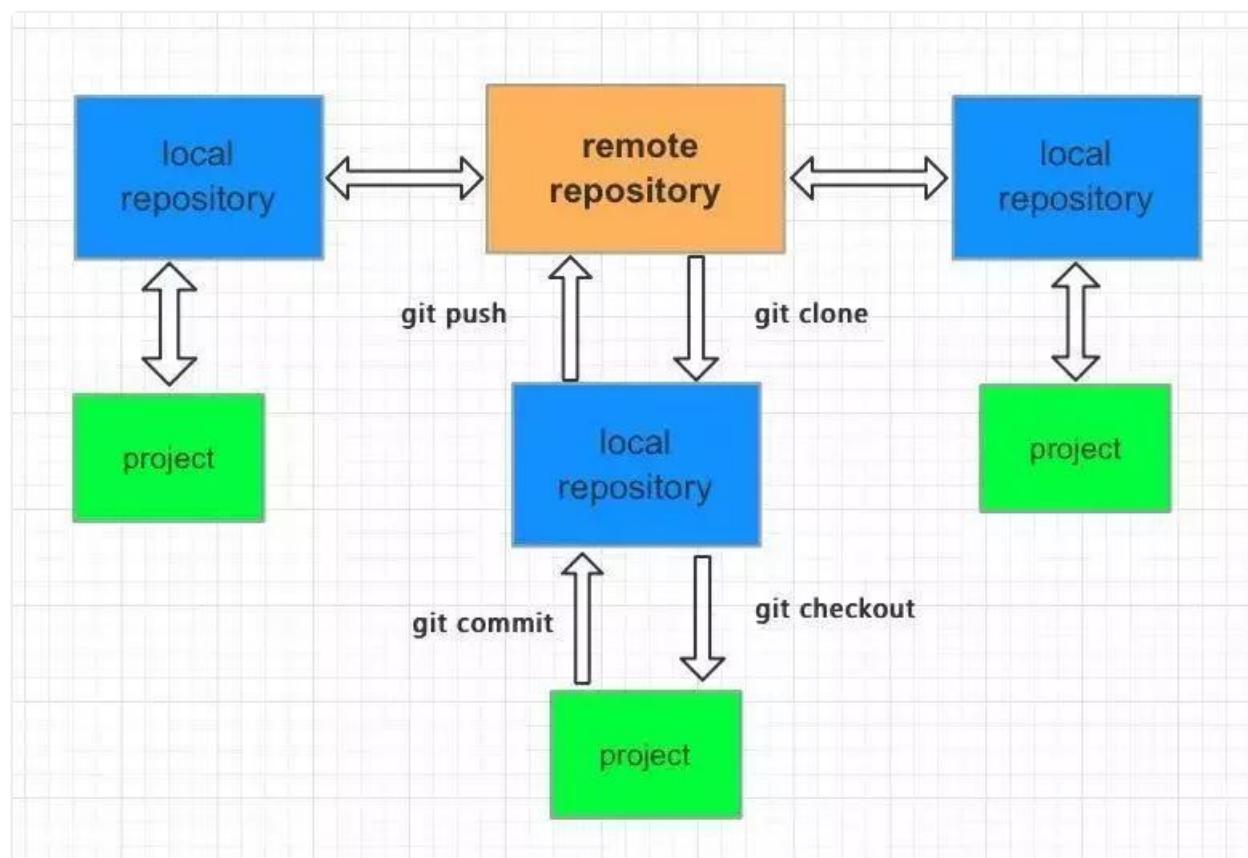
作者: J'KYO

来源: <https://urlify.cn/2YjiEj>

1、Git简介

Git是目前流行的分布式版本管理系统。它拥有两套版本库，本地库和远程库，在不进行合并和删除之类的操作时这两套版本库互不影响。也因此其近乎所有的操作都是本地执行，所以在断网的情况下仍然可以提交代码，切换分支。Git又使用了SHA-1哈希算法确保了在文件传输时变得不完整、磁盘损坏导致数据丢失时能立即察觉到。

Git的基本工作流程：



- git clone：将远程的Master分支代码克隆到本地仓库
- git checkout：切出分支出来开发
- git add：将文件加入库跟踪区
- git commit：将库跟踪区改变的代码提交到本地代码库中
- git push：将本地仓库中的代码提交到远程仓库

git 分支

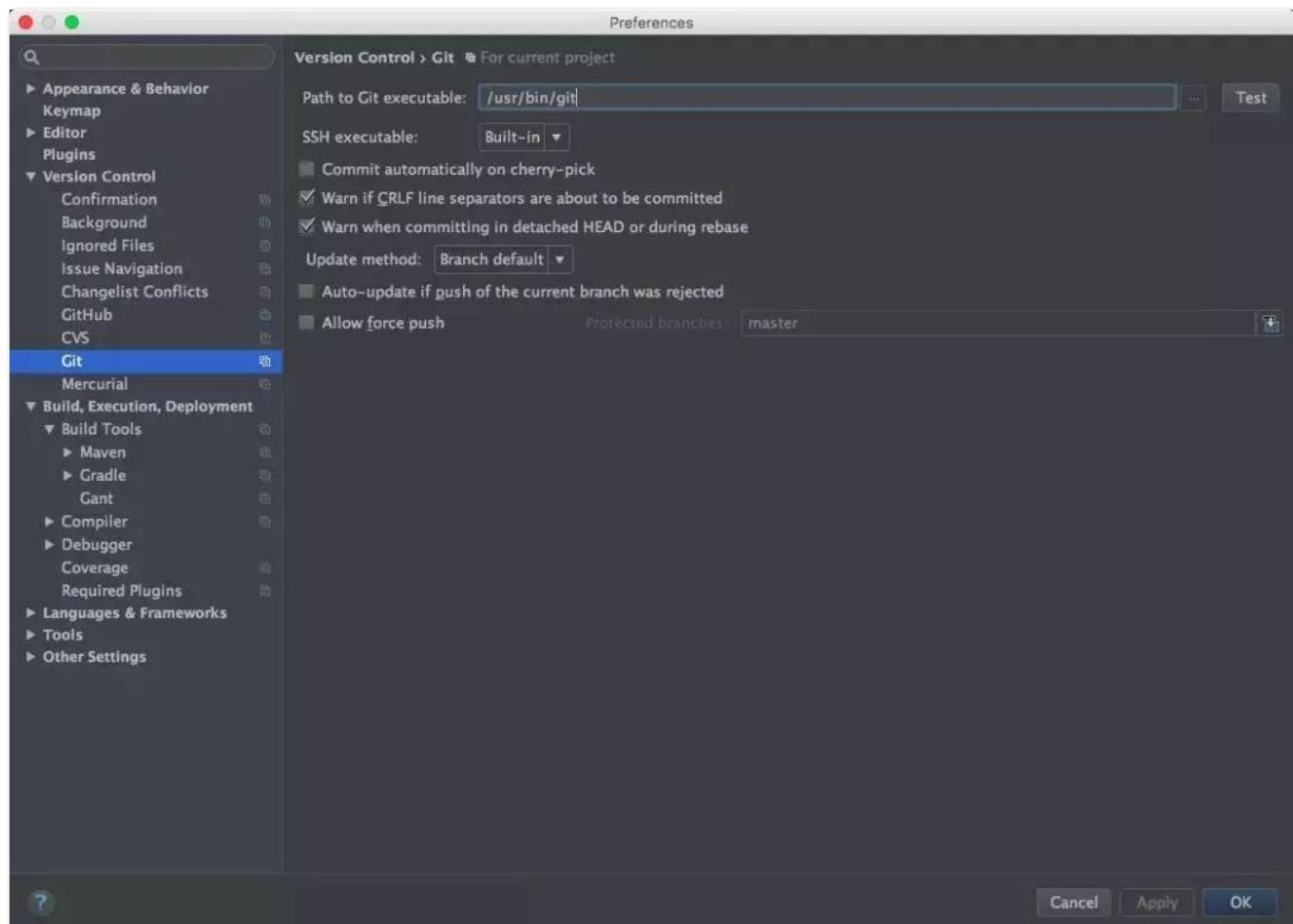
- 主分支
- master分支：存放随时可供生产环境中的部署的代码
- develop分支：存放当前最新开发成果的分支，当代码足够稳定时可以合并到master分支上去。

- 辅助分支
- feature分支：开发新功能使用，最终合并到develop分支或抛弃掉
- release分支：做小的缺陷修正、准备发布版本所需的各项说明信息
- hotfix分支：代码的紧急修复工作

2、Git在IntelliJ IDEA下的使用

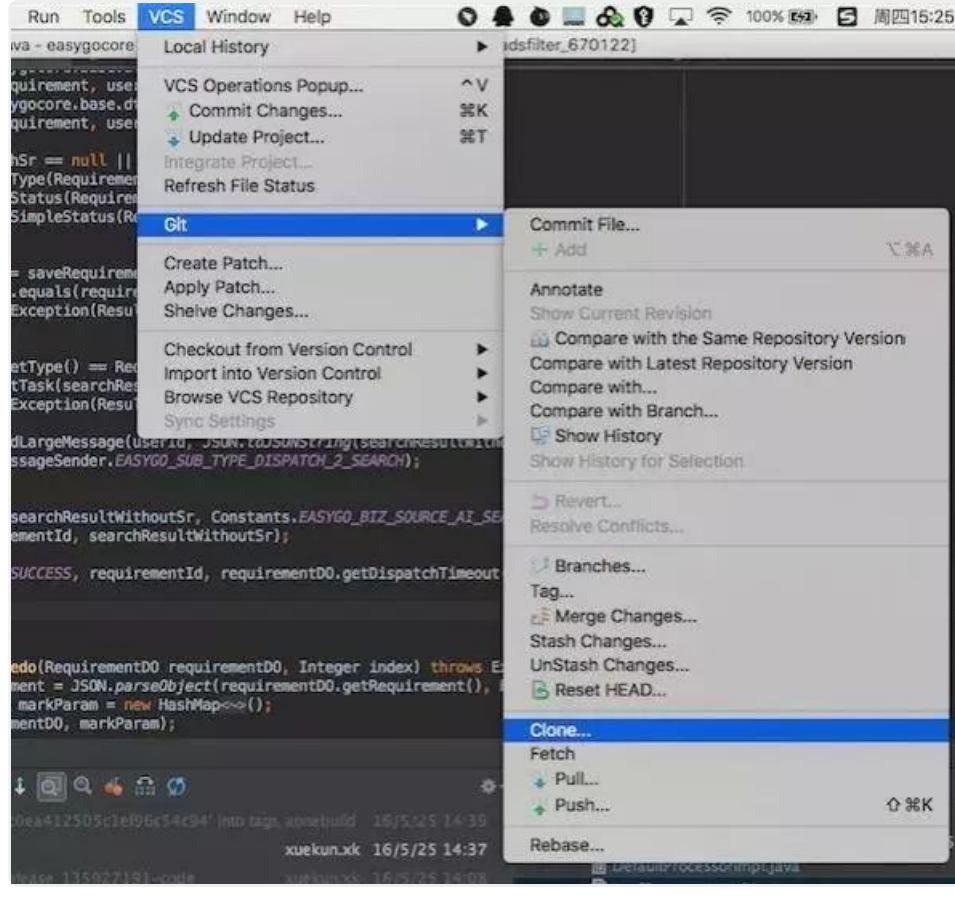
2.1、IntelliJ IDEA下配置Git

本地安装好git，并配置合理的SSH key，具体看这里 IntelliJ IDEA->Performance->Version Control->git 将自己安装git的可执行文件路径填入Path to Git executable，点击Test测试一下

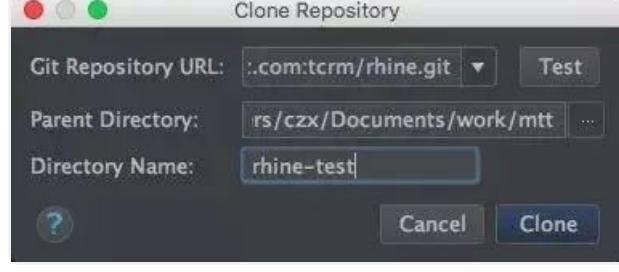


2.2、git clone

VCS->Git->Clone

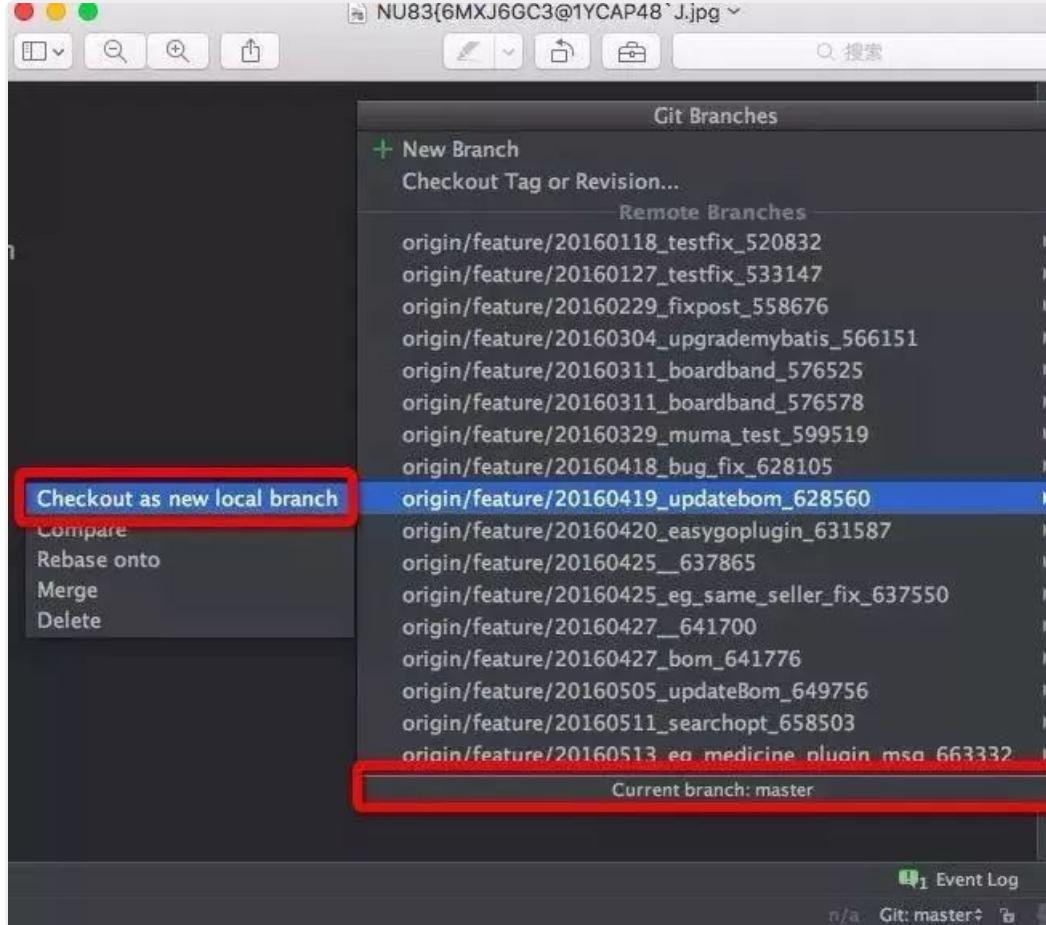


输入你的远程仓库地址,点击测试一下地址是否正确

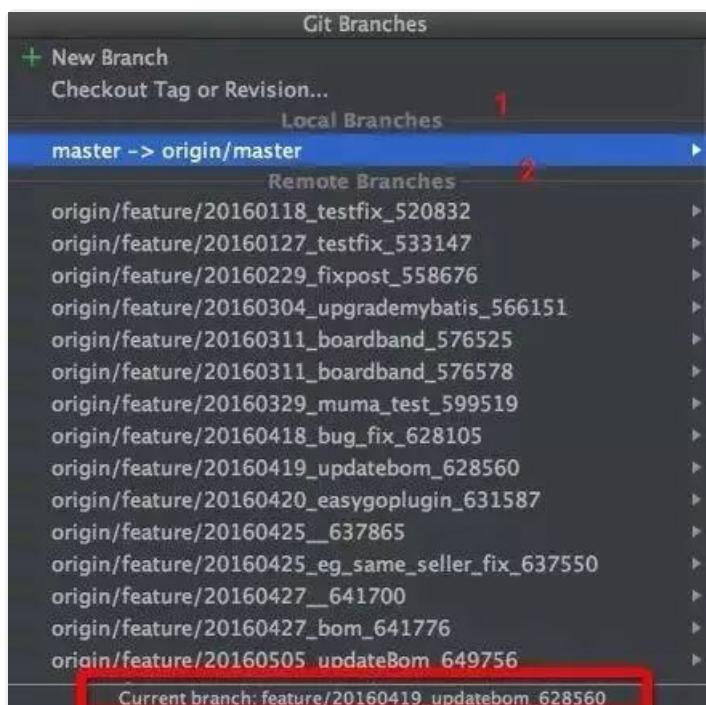


2.3、git checkout

在IntelliJ IDEA右下角有一个git的分支管理, 点击。选择自己需要的分支, checkout出来



checkout出来，会在底端显示当前的分支。其中1显示的为本地仓库中的版本，2为远程仓库中的版本

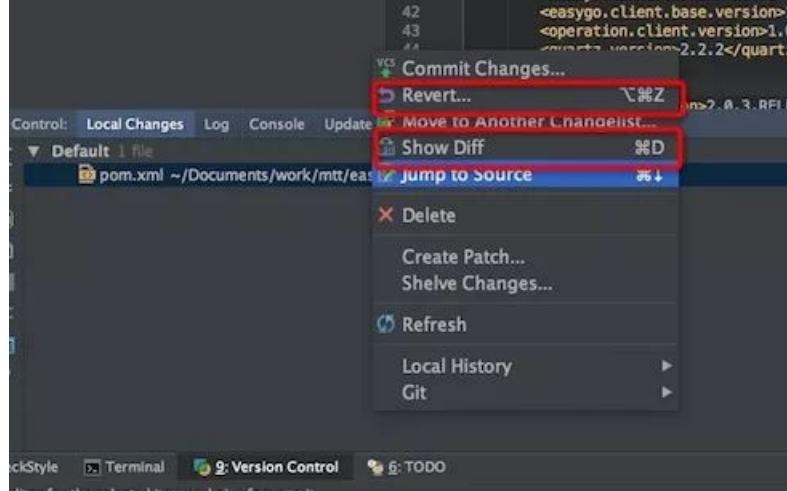


点击IDE的右上角的向下箭头的VCS，将分支的变更同步到本地



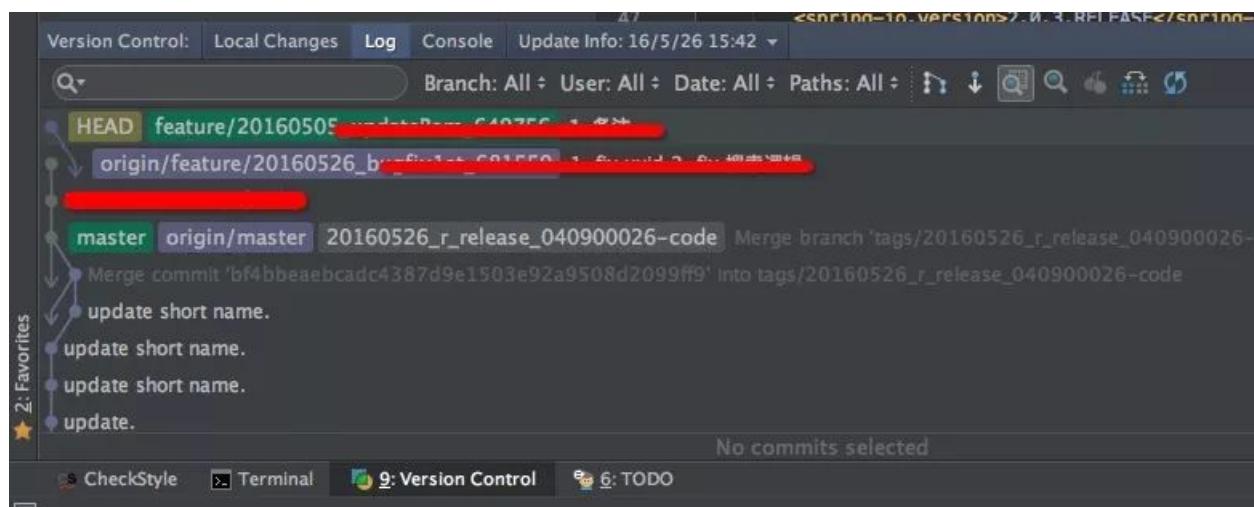
2.4、git diff

在local changes中选中要比对的文件，右键选择show diff便可以查看文件的变动。或者选择Revert放弃文件的改动



2.5、git log

在Version Control下选择Log，可以查看提交历史

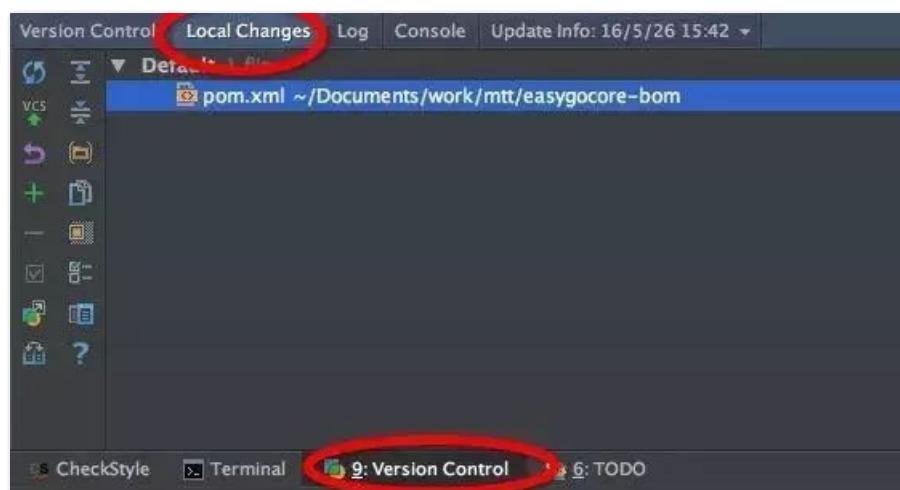


2.6、git commit

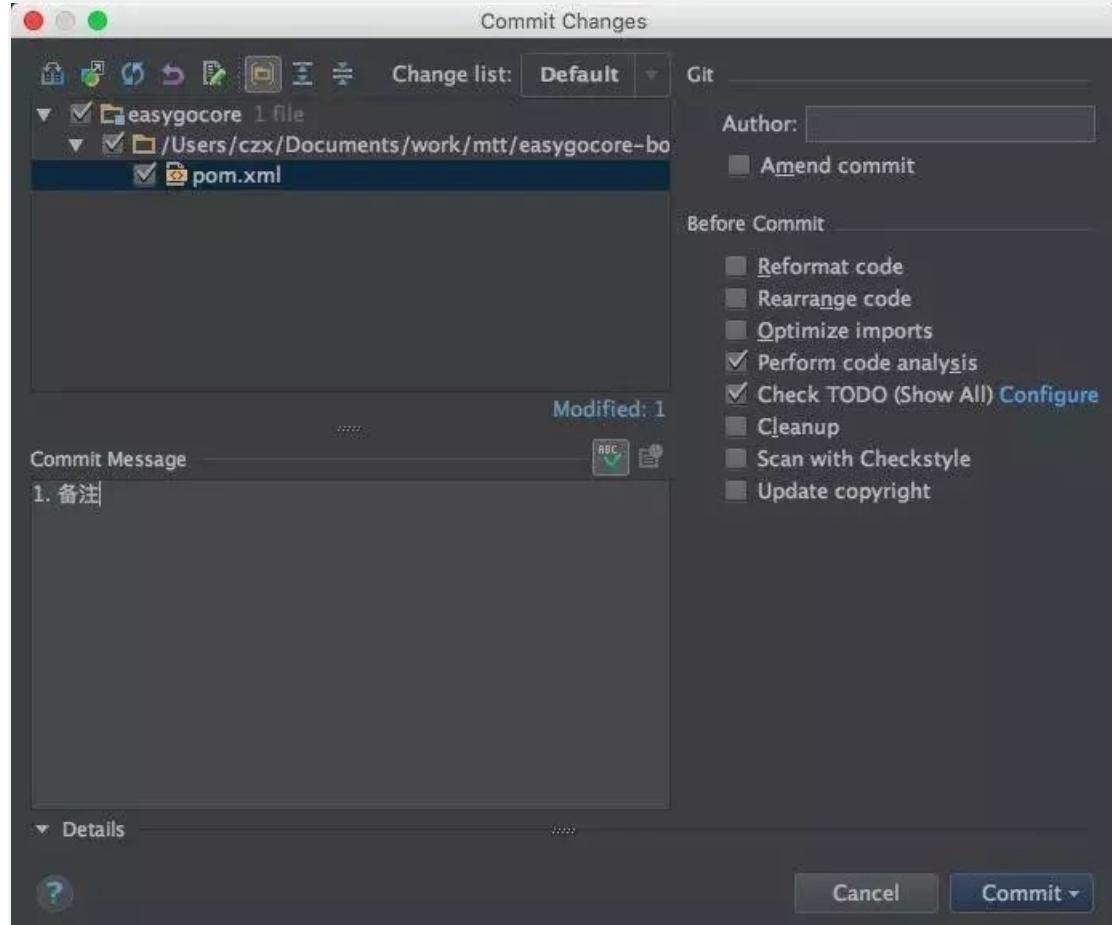
默认导入的工程已经git add加入库跟踪区了

随便修改一下pom.xml文件，其修改的文件会显示在Version Control中的local changes下

欢迎关注公众号：Java后端

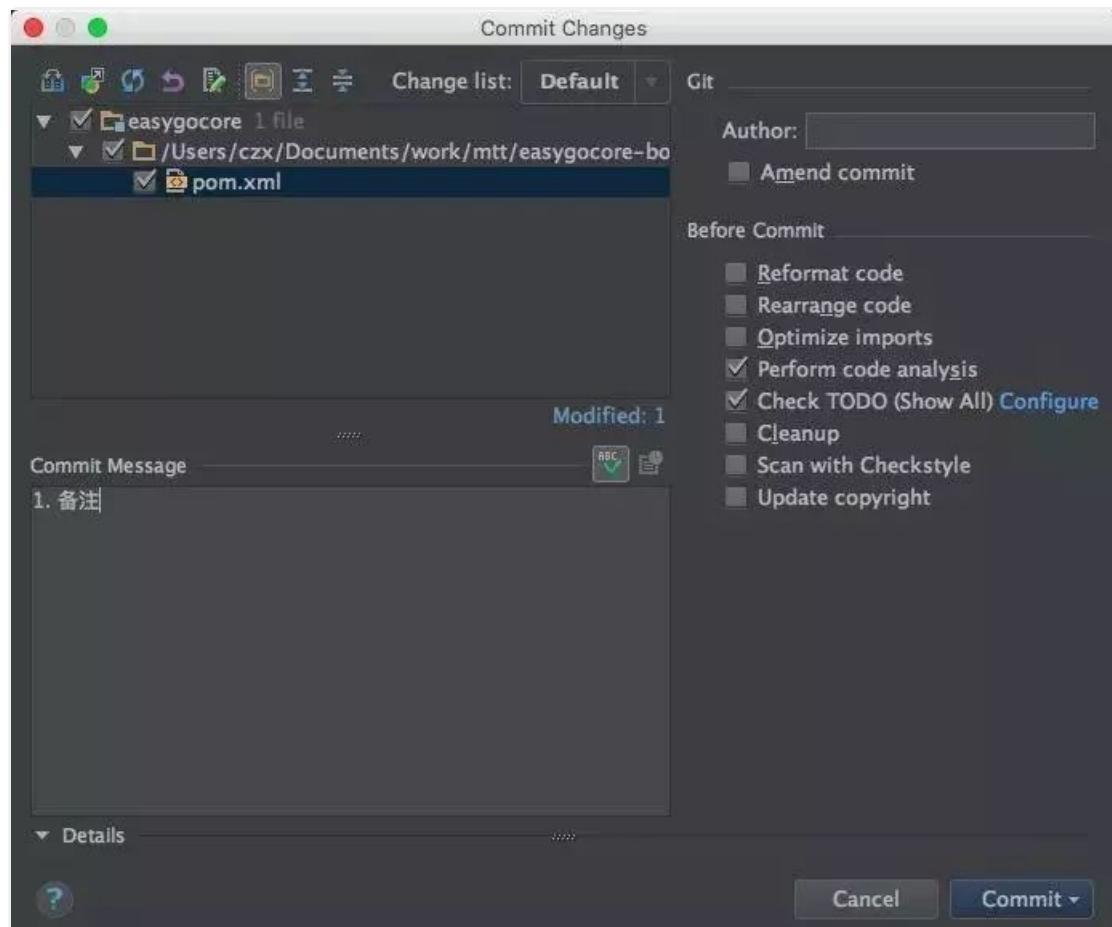


点击IDE右上角的向上箭头的VCS, git commit, 写上日志提交到本地代码库中



2.7、git push

VCS->Git->Push 将本地代码提交到远程仓库



2.8、在Idea命令行使用git

mac下同时按alt+F12,进入idea命令行

常见的命令：

- clone项目 git clone xxxxxxx
- 检查项目状态 git status
- 切换分支并和远程的分支关联 git checkout -b xxx -t origin/xxx
- 拉最新更新 git pull
- 提交更新 git commit -am "备注"
- 合并分支到当前分支，首先切换到需要被合并的分支 git checkout xxx, 再合并 git merge yyyy
- 提交 git push

- END -

推荐阅读

1. Spring 的 Bean 生命周期
2. 12306 又崩溃,买张车票怎么就这么难
3. 发布没有答案的面试题,都是耍流氓
4. 什么是一致性 Hash 算法?
5. 团队开发中 Git 最佳实践



喜欢文章, 点个在看 

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

学生免费使用正版 IDEA

Java后端 2019-08-22

昨天发了个文章，留言区学生党询问如何申请学生特权，对于学生或教师来说，JetBrains开发工具免费提供给学生和教师使用。而且取得一次授权后只需要使用相同的 JetBrains 帐号就可以激活其他产品，不需要重复申请。

1. 打开 <https://www.jetbrains.com/student/>，点击“APPLY NOW”开始申请。

2. 填写姓名，以及学校提供给你的邮箱(edu 后缀邮箱)

JetBrains Products for Learning

Apply with:

UNIVERSITY EMAIL ADDRESS ISIC/ITIC MEMBERSHIP OFFICIAL DOCUMENT

Status: I'm a student
 I'm a teacher

Name: [Redacted] [Redacted]
Our software will be registered to your real name.

Email address: [Redacted].edu.cn
Your valid university email address, e.g. john.smith@mit.edu. We'll send you further instructions.

APPLY FOR FREE PRODUCTS

邮箱地址要 edu.cn 结尾

下一步后，进入邮箱查看（如果收件箱没有找到，请查看是否为误判丢进了“垃圾邮件”）

日期： 2016-09-16 13:04:44

发件人：“JetBrains Account” <no_reply@jetbrains.com> 添加到通讯录 邮件往来 拒收

收件人： [Redacted].edu.cn

主题： JetBrains Educational Pack Confirmation [举报垃圾邮件]

Hi,

You've received this email because your email address was used for registering/updating a JetBrains Educational Pack.

Please follow this link to confirm your intention:

[Confirm Request](#)

Yours truly,

JetBrains Team

<https://www.jetbrains.com>

The Drive to Develop

我就是在“垃圾邮件”中找到的确认信

1. 点击“Confirm Request”进行确认，打开的网站会提示你注册 JetBrains 账号，输入账号与密码后，再次确认。

2. 回到邮箱，刷新，此时点击“Active Educational License”

日期：2016-09-16 13:06:39
发件人：“JetBrains Sales” <sales.us@jetbrains.com> 添加到通讯录 邮件往来 拦收
收件人：
主题：JetBrains Student License Confirmation [举报垃圾邮件]

Dear [REDACTED]

Congratulations! Your JetBrains Student License is confirmed.

To activate your license, use the following link:
[Activate Educational License](#)

After accepting the License Agreement, you will be asked to sign up for a Student JetBrains Account. You will use this account to sign in to JetBrains product(s) whenever you use them.

Happy coding!

Yours truly,
JetBrains Sales Team
<https://www.jetbrains.com>
The Drive to Develop

激活 License

1. 成功的获得授权码。

1 × JetBrains Product Pack for Students Personal license were added. ×

1 License Export licenses into .xls

JetBrains Product Pack for Students License ID: [REDACTED]

Licensed to: [REDACTED] [Download activation code for offline usage](#)

License restriction: For educational use only

Valid through: September 15, 2017

Following products included:

• IntelliJ IDEA Ultimate	• ReSharper	• ReSharper C++	• dotTrace	• dotMemory
• dotCover	• AppCode	• CLion	• PhpStorm	• PyCharm
• RubyMine	• WebStorm	• DataGrip		

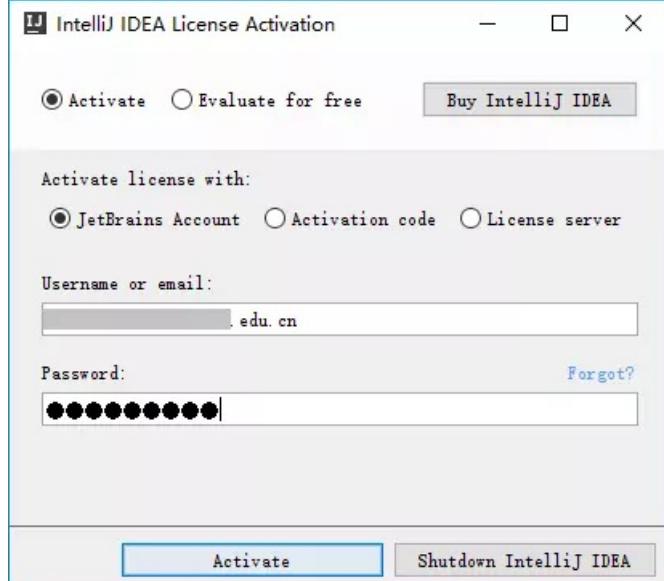
After downloading and installing the software, simply run it and follow the on-screen prompts to sign in with your JetBrains Account.

Can't find your license here? Link your past purchases to your JetBrains Account by providing a license key or domain.

可以看到，JetBrains 开发工具非常多，而且都可以免费使用了

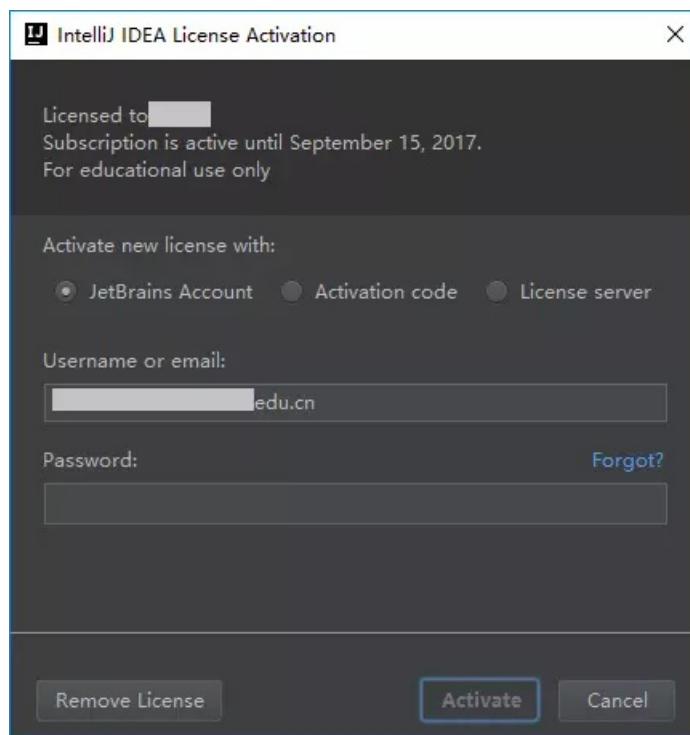
1. 官网下载开发工具，以 IntelliJ 为例

2. 可以直接通过之前注册的 JetBrains 帐号激活产品。



登录即可激活

1. 激活完成, Help->Register验证。



激活成功

1. 申请的 Liscence 有效期为一年, 在到期后, 可以去官网重新认证。

2. 一个 Liscence 可以在多台设备验证, 但同时使用的只能为1台。

3 . 最后, 引用第3条中作者的一句话, “希望各位享受权利的同时不要忘记自己的义务, 不要售卖、转手自己的学生优惠资格, 不要作践自己作为学生的价值”, 谢谢!

链接 :<https://www.jianshu.com/p/2c6977ab00b1>

阅读原文

安利一款 IDEA 中强大的代码生成利器

Java后端 2月6日

点击上方 Java后端, 选择 **设为星标**

优质文章, 及时送达

作者 | Sharehub

链接 | blog.xiaohansong.com/idea-live-templates.html

前言

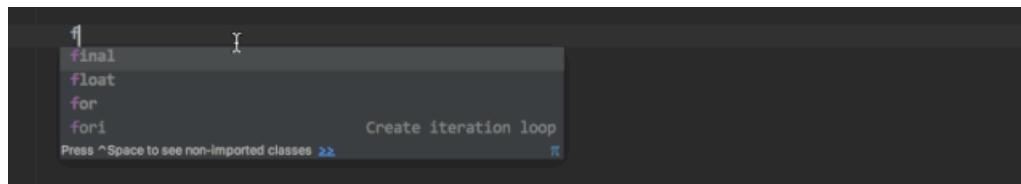
Java 开发过程经常需要编写有固定格式的代码，比如说声明一个私有变量，logger或者bean等等。对于这种小范围的代码生成，我们可以利用 IDEA 提供的 Live Templates功能。刚开始觉得它只是一个简单的Code Snippet，后来发现它支持变量函数配置，可以支持很复杂的代码生成。

下面我来介绍一下Live Templates的用法。

基本使用

IDEA 自带很多常用的动态模板，在 Java 代码中输入fori，回车就会出现

```
for (int i = 0; i < ; i++) {  
}
```



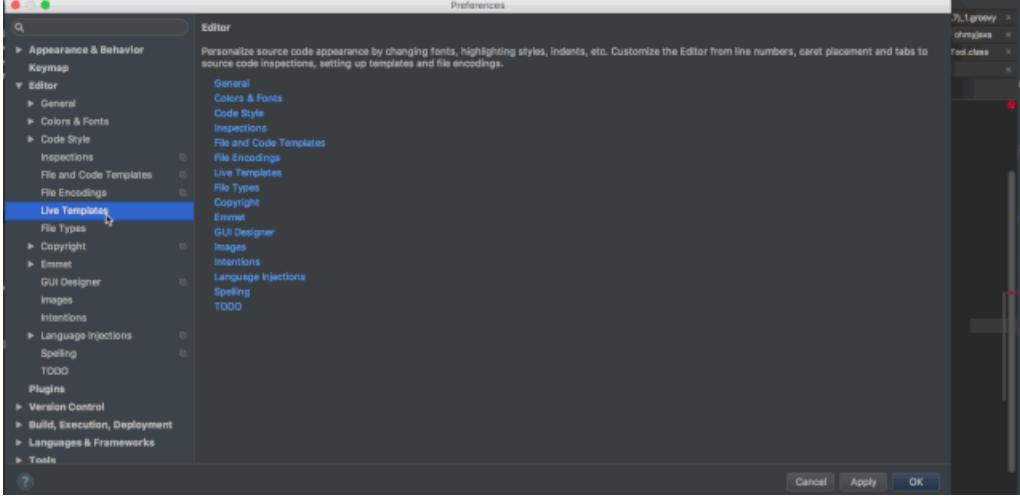
按Tab可以在各个空白处跳转，手动填值。

自定义 Template

官方自带模板毕竟不能满足我们个人编码风格的需要，Live Templates提供了变量函数的方式供我们自定义。

简单用法

新增自定义模板，首先需要填写触发单词（即 Abbreviation），描述是可选的，然后定义模板的上下文，点击define选择 Java，这样在编辑 Java 的时候就会触发当前模板，定义完上下文之后，就可以填写模板了。



下面列举几个我常用的简单模板

```
=====
<out>
-----
System.out.println($END$)
=====

<pfs>
-----
private final static String $varName$ = "$var$";`
```

=====

```
<privateField>
-----
/** 
 * $COMMENT$
```

*/

```
@Getter
@Setter
private $TYPE$ $NAME$;
```

=====

```
<main>
-----

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

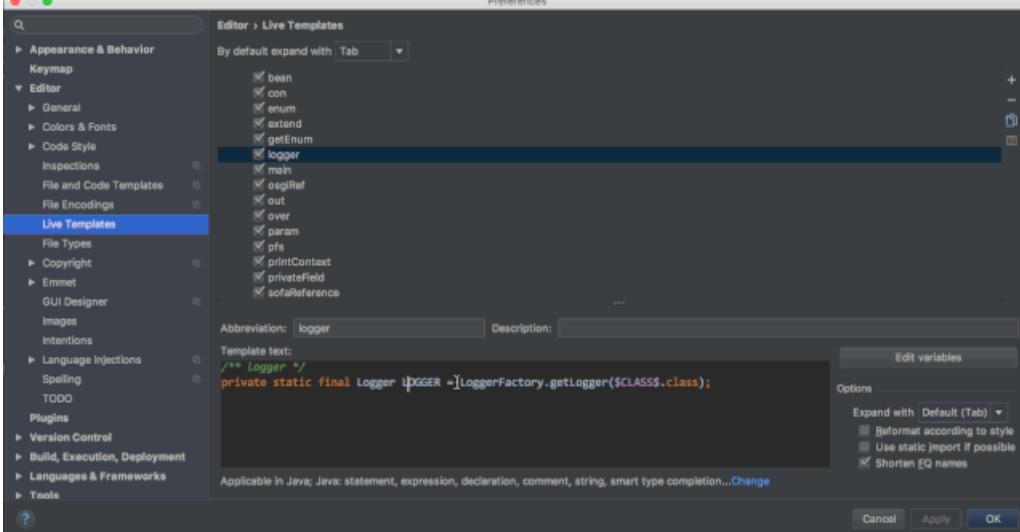
}

```
=====
```

模板支持变量的定义，使用 `$$` 包围的字符表示一个变量。 `END` 是一个特殊的预定义变量，表示光标最后跳转的位置。每个变量的位置都可以跳转过去。

高级用法

如果你用过 vim 的Code Snippet插件，你会发现模板里面是可以执行函数的，强大的 Live Templates当然也支持，而且 IDEA 能够感知代码的语义，比如说当前编辑的函数的参数。但这一点就能够让我们玩出花来。我们从易到难来研究模板函数的功能。



前面我们提到的变量可以绑定函数，配置方式如上图所示。

快速声明变量

声明变量是一个常用的操作，特别是需要声明变量需要加注解，注释的时候，这些代码写起来就很枯燥。下面是我定义的模板：

```
<osgiRef>
-----
/** $END$
 */
@OsgiReference
@Setter
private $TYPE$ $NAME$;
```

乍一看这个模板跟我上面定义的privateField差不多，唯一的不同在于我给这些变量绑定了函数。

- clipboard(): 返回当前粘贴板的字符串
- decapitalize(): 将输入的字符串首字母变为小写

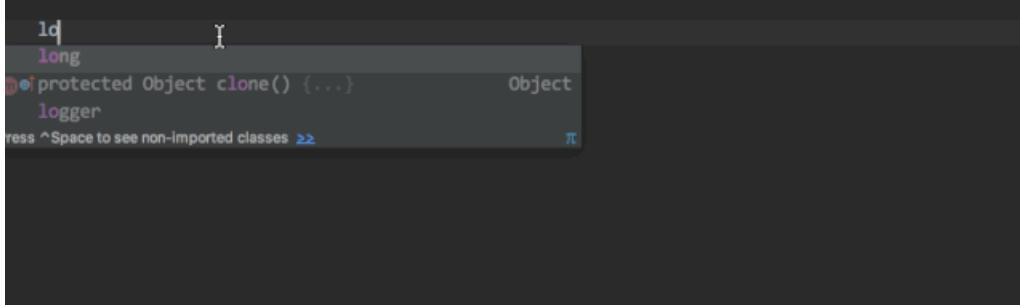
下面我们演示一下，我们先拷贝当前类名，然后输入osgiRef



快速声明 logger

声明 logger 也是一个常用的操作，上面我们是利用了粘贴函数来快速声明变量，现在我们来利用另一个函数className()，顾名思义，它的作用就是返回当前类名。

```
<logger>
-----
/** logger */
private static final Logger LOGGER = LoggerFactory.getLogger($CLASS$.class);
```



最强大的 groovyScript()

如果说上面用到的函数提供的能力有限，不够灵活，那么groovyScript()提供了一切你想要的能力，它支持执行 Groovy 脚本处理输入，然后输出处理后的字符串。

```
groovyScript("code", ...)

| code | 一段Groovy代码或者Groovy脚本代码绝对路径 |
| ... | 可选入参，这些参数会绑定到`_1,_2,_3,..._n`，在Groovy代码中使用。 |
```

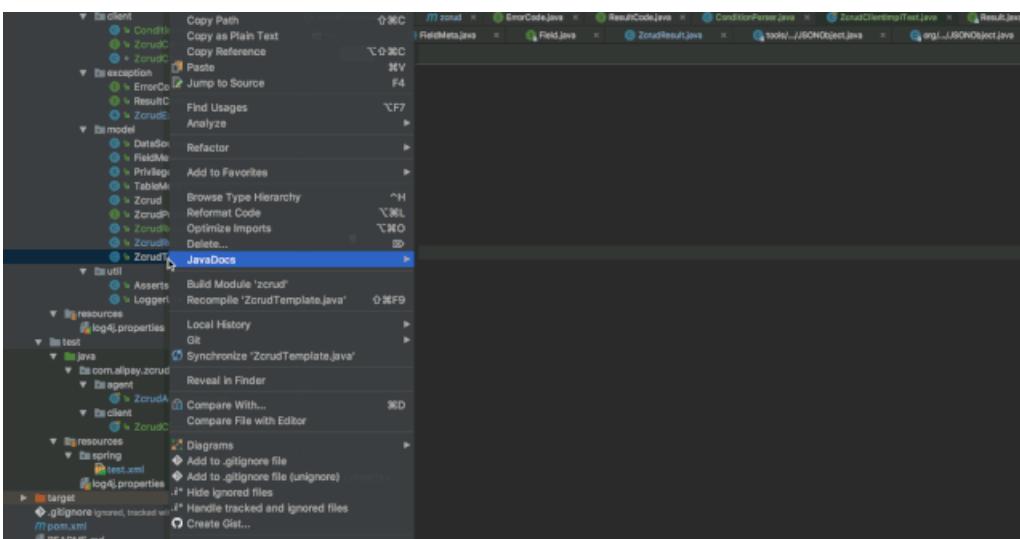
下面我们来看一下它的实际应用。

快速 bean 配置

新增一个服务都要在 Spring 中注册一个 bean，一般这个配置无非就是将指明id和class，由于我们是在 xml 中配置，所以不能利用className()函数，但是我们可以利用clipboard()函数获取到类的全引用，在 IDEA 中我们直接右键类名，点击Copy Reference就行。然后执行 groovy 脚本获取类名。

```
<bean>
-----
<bean id="$id$" class="$REF$"/>
```

id绑定 `decapitalize(groovyScript("_1.tokenize('.')[-1]", clipboard()))`，首先取 `clipboard()` 的值得到类的全引用，然后执行 groovy 代码 `_1.tokenize('.')[-1]`（按.分割为字符串数组，然后取最后一个即可得到类名，然后用 `decapitalize()` 将首字母小写即可得到id。



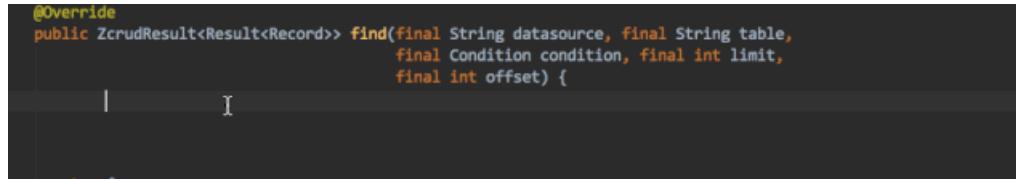
快速打印当前上下文信息

打印错误日志的时候需要打印当前上下文信息的，例如说入参，有时候入参很多的时候，写起来很痛苦，好在有模板函数 `methodParameters()`，返回当前函数参数的列表，当然这个列表我们不能直接使用，需要结合groovyScript对它进行转化。

```
<printContext>
```

```
-----  
LogUtil.$TYPE$(LOGGER, "$MSG$ " + $params$);
```

将params绑定到 `groovyScript("'\\" + _1.collect { it + ' = [' + ' + it + ' + \"]' }.join(' , ') + '\\"", methodParameters())`，就能够自动将当前函数的参数格式化后输出。



总结

上面我们简单介绍了常用的模板函数，其实 IDEA 还有很多其它模板函数，具体参考[Creating and Editing Template Variables](#)。

<https://www.jetbrains.com/help/idea/2016.3/creating-and-editing-template-variables.html>

IDEA 是一个很强大的工具，善用工具能够极大的提高工作效率，将精力投入到关键的事情上，而不是将时间浪费在编写重复代码上面。一些更高级的用法还有待大家去发掘。最后推广一波我写的代码生成插件CodeMaker，好好利用也能节省很多重复编写代码的时间。

推荐阅读

1. 聊聊在阿里远程办公那点事儿
2. 你真的了解 volatile 吗？
3. 程序员才能看懂的动图
4. 如何获取靠谱的新型冠状病毒疫情



微信搜一搜

Java后端

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

手把手教你免费获取正版 IntelliJ IDEA

Java后端 2019-08-21

以下文章来源于IT牧场，作者itmuch



IT牧场

IT牧场公众号，阿里技术专家分享开发、运维、架构相关干货！



前言

近日发了很多 IDEA 相关的干货，有很多读者想用正版 IDEA，又不想付钱，只能想歪门邪道，最后还没有破解成功。我认为 IDEA 是一款非常棒的产品，花点钱购买个正版也是应该的。

你知道吗？其实 IDEA 是可以免费使用的。

IDEA是个人最喜欢的 IDE，它非常智能，懂我的心，极大地提高了个人编程效率；让人爱不释手，欲罢不能。

然而，这是一款收费软件，价格不菲。IDEA价目详见：<https://www.jetbrains.com/idea/buy/#commercial?billing=yearly>。

本文教大家如何 免费，并且 光荣地 使用 正版 IntelliJ IDEA。

IDEA免费开源协议

在 <https://www.jetbrains.com/community/opensource/>，IDEA有一个开源免费协议。简单翻译一下。

申请条款

- 您必须是项目负责人或常规提交者
- 您的OS项目符合

开源定义[1]

您的开源项目可能不提供付费赞助，或从商业公司或组织（非政府组织，教育，研究或政府）获得资金。您不得为您的开源项目提供任何付费支持，咨询或培训服务，也不得分发您的开源软件的付费版本。获得该项目工作报酬的贡献者不符合资格。您的OS项目正在积极开发至少3个月。您的OS项目社区处于活动状态。您定期发布更新的版本。

许可条款

许可证提供1年，并允许在1年内免费升级软件的所有新版本。如果您的项目仍满足要求，可根据要求提供许可证续订。一个许可证可以安装在任意数量的计算机上，但不能在两个或更多计算机上同时使用。许可证仅提供给核心团队开发人员。

许可限制

许可证仅可用于非商业OS开发。请考虑购买单独的许可证以处理商业项目。

该软件的使用仅限于许可用户，无权将软件转让给任何第三方。

申请免费使用

申请门槛

从协议不难看出,你只需在GitHub上准备一个维护超过3个月的项目开源项目,就可以免费使用IDEA 1年了,1年到期后,可以按照此步骤再申请一次。

这是一个良好的闭环:

有开源项目,所以能申请免费使用IDEA;

有了IDEA神器,又可以更好地维护开源项目……

申请

到 <https://www.jetbrains.com/shop/eformopensource?product=ALL> 即可提交申请。

Open Source License Request

JetBrains can support your open source project by providing free All Products Pack license to use for the development of your project. If you are a project lead or a core contributor, please fill out the form below to request this support.

1. Do we know you?

No, we are a new customer
 Yes, we've used JetBrains Open Source license(s) in our project before

2. Tell us about your project

Project name: light-security

Primary language(s): Java C# .NET PHP JavaScript
 Ruby Python Go Kotlin Scala
 C/C++ Objective-C Other

Project age: 4 months(s)

Please note that a project must be in active development for at least 3 months to be eligible for support.

Project website: <https://github.com/eacdy/light-security>

Repository URL: <https://github.com/eacdy/light-security>

Where we can access your repository or browse its contents.

License URL: <https://github.com/eacdy/light-security/blob/master/LICENSE>

A copy of the license terms and conditions for your software.

Country / region: China

No. of required licenses: 1

Please note that the licenses are granted only to active contributors. Their contributions to the project are regular and visible in the repository. Non-code commits are not considered active development.

If you are interested in team tools, please try the free versions of [TeamCity](#), [YouTrack](#), [Usource](#) available on our website. If you need full versions of any of these tools, please indicate this in the Project description field.

Project description: Light Security is a JWT based RBAC security framework.

Provide any additional information to help us better understand what you're working on.

3. Tell us about yourself

Name: ZHOU

Email address: 0000@126.com

I confirm that the email address provided above belongs to me.

A link to your profile on GitHub, etc: <https://github.com/eacdy>

If you don't have a GitHub profile, please provide any link indicating your contribution to the project.

Please note: to apply for free licenses on behalf of an open source project, you should be a project lead or a core contributor to the project.

I confirm that the granted license:
 will be used only for non-commercial open source development;
 will be shared only with the project's active contributors.

I have read and I accept the [JetBrains Account Agreement](#)

APPLY FOR FREE LICENSES

With any additional questions please contact us at opensource@jetbrains.com

IT牧场

点击 **APPLY FOR FREE LICENSES** 按钮,即可看到类似如下的界面:

Open Source Request Confirmation

Your request has been successfully submitted. Thank you.

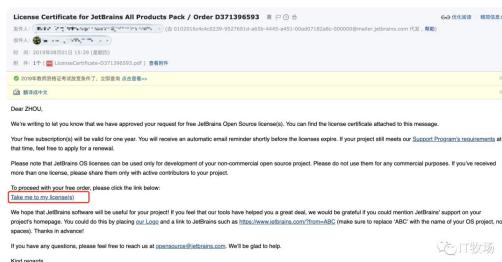
You will soon get an automatic reply with the ID number of your request at 0000@126.com.
We will review your request within a few days. If it is approved, you will receive a separate email with further instructions on how to proceed with free licenses you requested.

[Learn more about JetBrains solutions and meet the developer community](#)

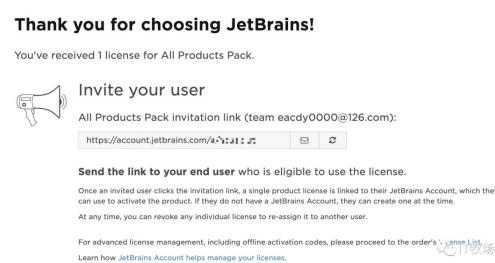
Copyright © 2000-2019 JetBrains s.r.o. | [Help](#) | [Support](#) | [JetBrains Privacy Policy](#) | [JetBrains Account Agreement](#) | Build #2019.07-1739

收取激活码

1、等待1天左右，即可前往申请时填写的邮箱，即可收到激活码了。



2、点击图中的链接，即可进入协议界面，点击 **ACCEPT**，即可看到类似如下的界面：



3、点击图中的链接，并按照提示操作，注册一个账号，或者如果你已经有Jetbrains账号，就直接登录。这一步主要是让你的Jetbrains账号和Liscence绑定。

4、将激活码填入如下界面即可激活IDEA：



5、激活后的效果：



可以看到，已经成功激活了。未来过期后，依照本次操作再执行一次即可。当然我本机电脑还没有升级到2019.2，这个无妨。
你可以先升级，再激活；也可以先激活再升级。

除此之外，学生党可通过校内邮箱申请使用资格，具体方式可以谷歌，非常简单。

References

- [1] 开源定义: <http://opensource.org/docs/osd>
- [2] 开源项目: https://www.jetbrains.com/store/license_opensource.html
- [3] 许可协议: https://www.jetbrains.com/store/license_opensource.html

如果喜欢本篇文章，欢迎[转发、点赞](#)。关注订阅号「Web项目聚集地」，回复「全栈」即可获取 2019 年最新 Java、Python、前端学习视频资源。

推荐阅读

1. 堪称神器的 Chrome 插件
2. Nginx 搭建图片服务器
3. 为什么推荐 Java 程序员使用 Google Guava 编程
4. Linux 最常用命令：解决 95% 以上的问题
5. 前端吐槽的后端接口那些事
6. 数据库不使用外键的 9 个理由



Web项目聚集地

微信扫描二维码，关注我的公众号

喜欢文章，点个在看

文章已于修改

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

提升 10 倍生产力：IDEA 远程一键部署 Spring Boot 到 Docker

陶章好 Java后端 2019-09-29

点击上方 Java后端, 选择“[设为星标](#)”

优质文章, 及时送达

作 者 | 陶章好

链 接 | juejin.im/post/5d026212f265da1b8608828b

上一篇 | [推荐 7 个 Spring Boot 前后端分离项目](#)

IDEA是Java开发利器，Spring Boot是Java生态中最流行的微服务框架，docker是时下最火的容器技术，那么它们结合在一起会产生什么化学反应呢？

一、开发前准备

1. Docker安装

可以参考：<https://docs.docker.com/install/>

2. 配置docker远程连接端口

```
1 vi /usr/lib/systemd/system/docker.service
```

找到 ExecStart，在最后面添加 -H tcp://0.0.0.0:2375，如下图所示

```
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
BindsTo=containerd.service
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock -H tcp://0.0.0.0:2375
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
```

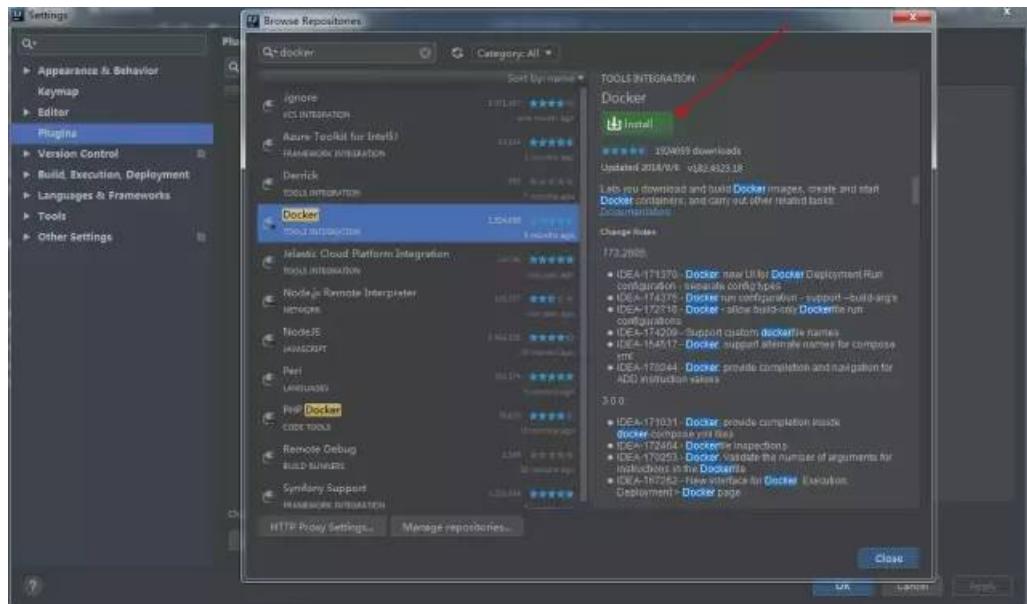
3. 重启docker

```
1 systemctl daemon-reload
2 systemctl start docker
```

4. 开放端口

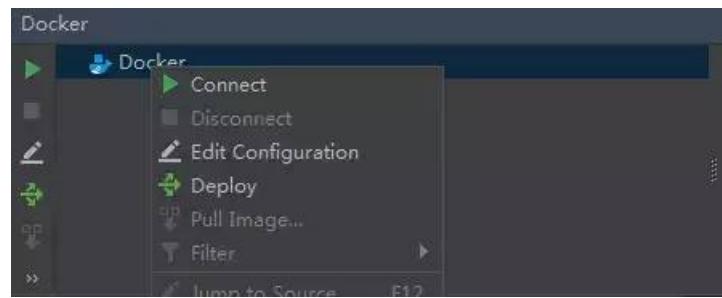
```
1 firewall-cmd --zone=public --add-port=2375/tcp --permanent
```

5. Idea安装插件，重启

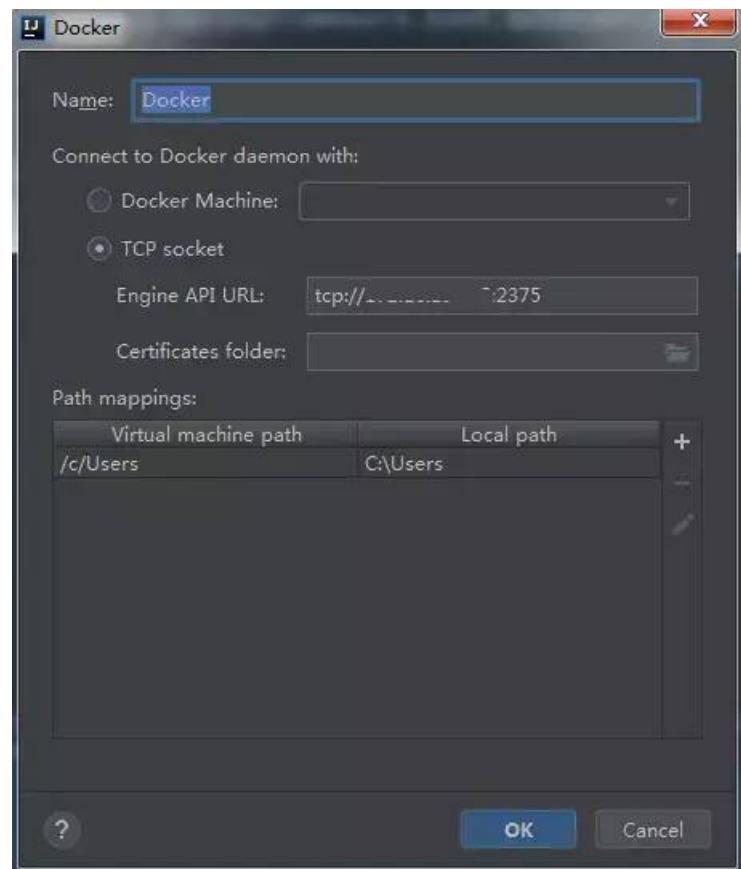


6. 连接远程docker

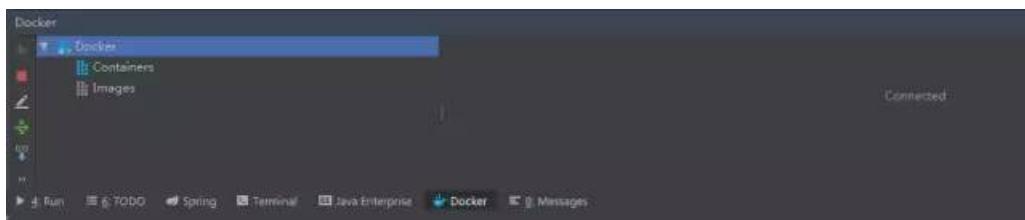
1、编辑配置



2、填远程docker地址



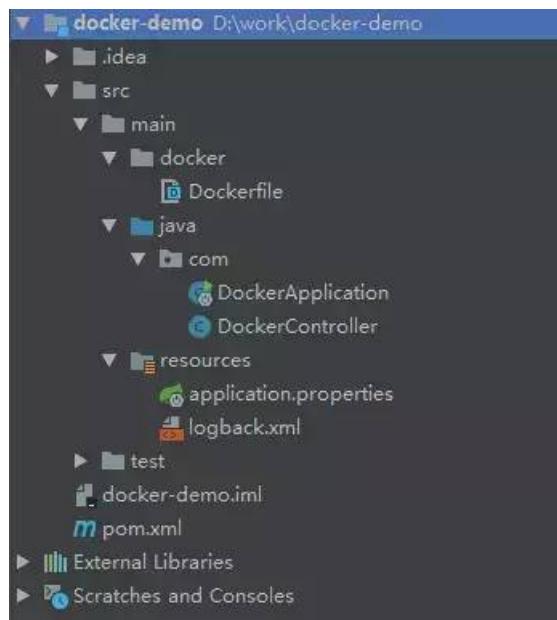
3、连接成功，会列出远程docker容器和镜像



二、新建项目

创建Spring Boot项目

项目结构图



1、配置pom文件

```
1 <properties>
2     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
3 >
4     <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
5 >
6     <docker.image.prefix>com.demo</docker.image.prefix>
7 >
8     <java.version>1.8</java.version>
9 >
10 </properties>
11 >
12 <build>
13 >
14     <plugins>
15 >
16         <plugin>
17 >
18             <groupId>org.springframework.boot</groupId>
19 >
20             <artifactId>spring-boot-maven-plugin</artifactId>
21 >
```

```
21      </plugin>
22  >
23  <plugin
24  >
25      <groupId>com.spotify</groupId>
26  >
27      <artifactId>docker-maven-plugin</artifactId>
28  >
29      <version>1.0.0</version>
30  >
31  <configuration
32  >
33      <dockerDirectory>src/main/docker</dockerDirectory>
34  >
35  <resources
36  >
37      <resource
38  >
39          <targetPath>/</targetPath>
40  >
41          <directory>${project.build.directory}</directory>
42  >
43          <include>${project.build.finalName}.jar</include>
44  >
45      </resource>
46  >
47      </resources>
48  >
49  </configuration>
50  >
51  </plugin>
52  >
53  <plugin
54  >
55      <artifactId>maven-antrun-plugin</artifactId>
56  >
57  <executions
58  >
59      <execution
60  >
61          <phase>package</phase>
62  >
63      <configuration
64  >
65          <tasks
66  >
67              <copy todir="src/main/docker" file="target/${project.artifactId}-${prc
68  >
69          </tasks>
70  >
71      </configuration>
72  >
73  <goals
74  >
```

```
<goal>run</goal>
>
    </goals>
>
    </execution>
>
    </executions>
>
    </plugin>
>

    </plugins>
>
    </build>
>
<dependencies>
    <dependency>
>
        <groupId>org.springframework.boot</groupId>
>
        <artifactId>spring-boot-starter-web</artifactId>
>
        </dependency>
>
    <dependency>
>
        <groupId>org.springframework.boot</groupId>
>
        <artifactId>spring-boot-starter-test</artifactId>
>
        <scope>test</scope>
>
        </dependency>
>
    <dependency>
>
        <groupId>log4j</groupId>
>
        <artifactId>log4j</artifactId>
>
        <version>1.2.17</version>
>
        </dependency>
>
</dependencies>
```

2、在src/main目录下创建docker目录，并创建Dockerfile文件

```
1 FROM openjdk:8-jdk-alpine
2 ADD *.jar app.jar
3 ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/.urandom", "-jar", "/app.jar"]
```

3、在resource目录下创建application.properties文件

```
1 logging.config=classpath:logback.xml  
2 logging.path=/home/developer/app/logs/  
3 server.port=8990
```

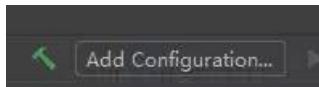
4、创建DockerApplication文件

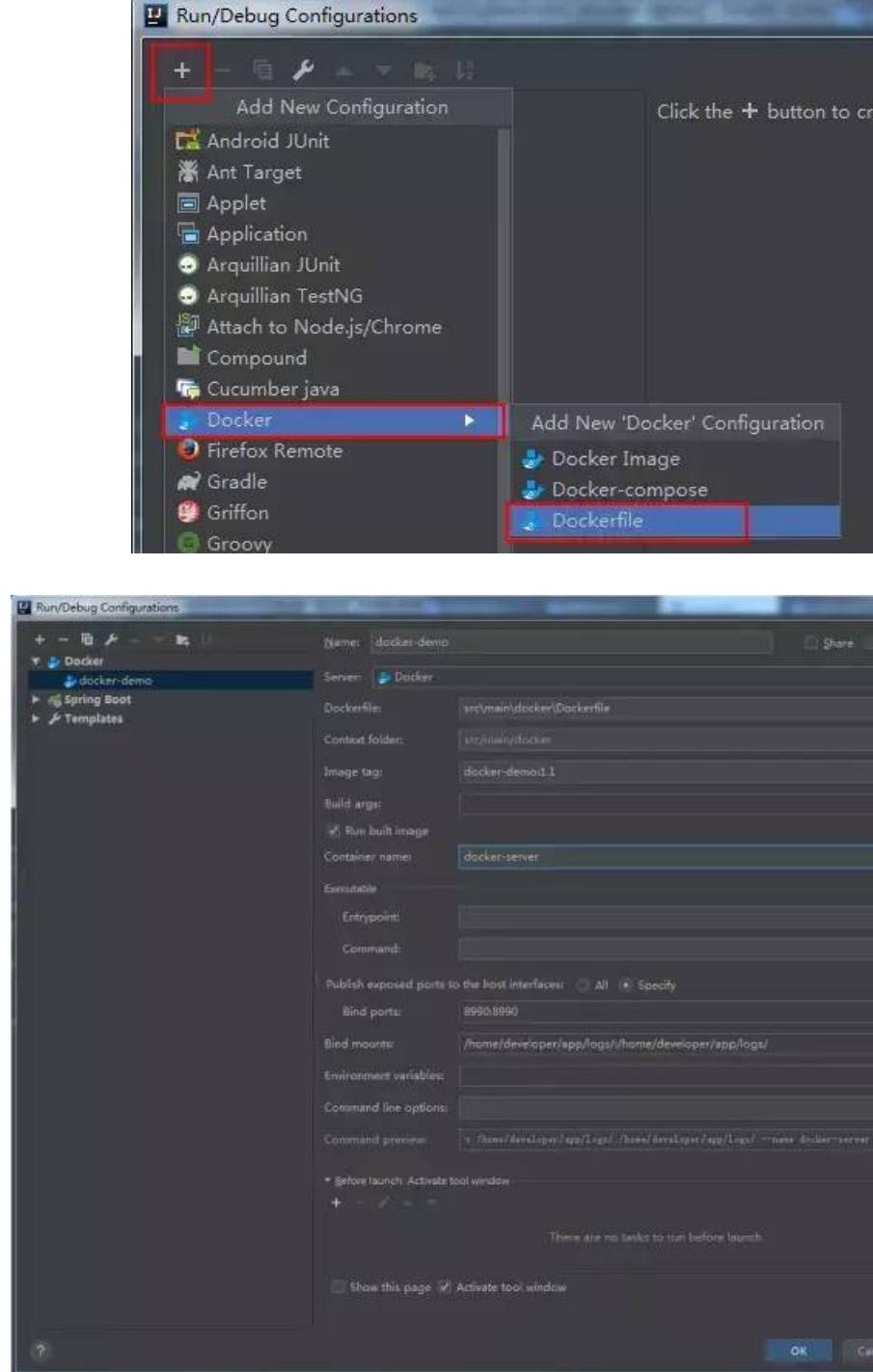
```
1 @SpringBootApplication  
2 public class DockerApplication {  
3     public static void main(String[] args)  
4     {  
5         SpringApplication.run(DockerApplication.class, args);  
6     }  
7 }
```

5、创建DockerController文件

```
1 @RestController  
2 public class DockerController {  
3     static Log log = LogFactory.getLog(DockerController.class)  
4 ;  
5  
6     @RequestMapping("/")  
7     public String index() {  
8         log.info("Hello Docker!")  
9         ;  
10        return "Hello Docker!"  
11    }  
12 }
```

6、增加配置



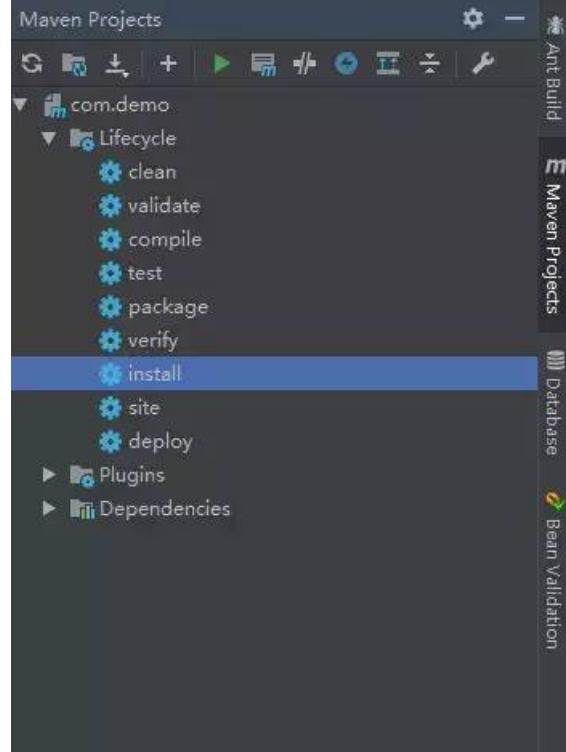


命令解释：

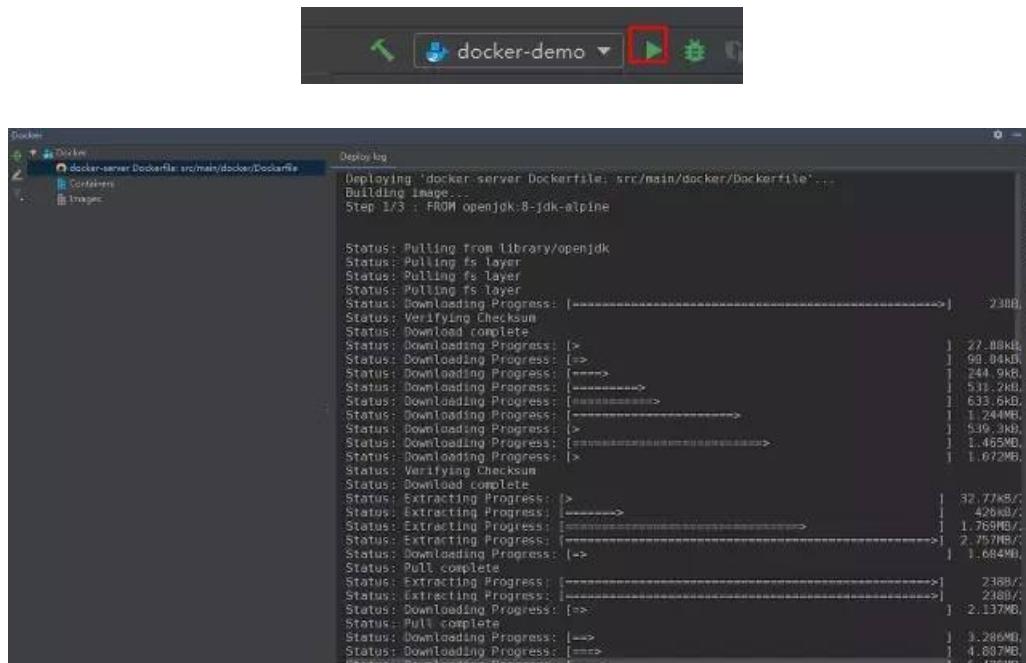
- Image tag : 指定镜像名称和tag，镜像名称为 docker-demo，tag为1.1
- Bind ports : 绑定宿主机端口到容器内部端口。格式为[宿主机端口]:[容器内部端口]
- Bind mounts : 将宿主机目录挂载到容器内部目录中。

格式为[宿主机目录]:[容器内部目录]。这个springboot项目会将日志打印在容器 /home/developer/app/logs/ 目录下，将宿主机目录挂载到容器内部目录后，那么日志就会持久化容器外部的宿主机目录中。

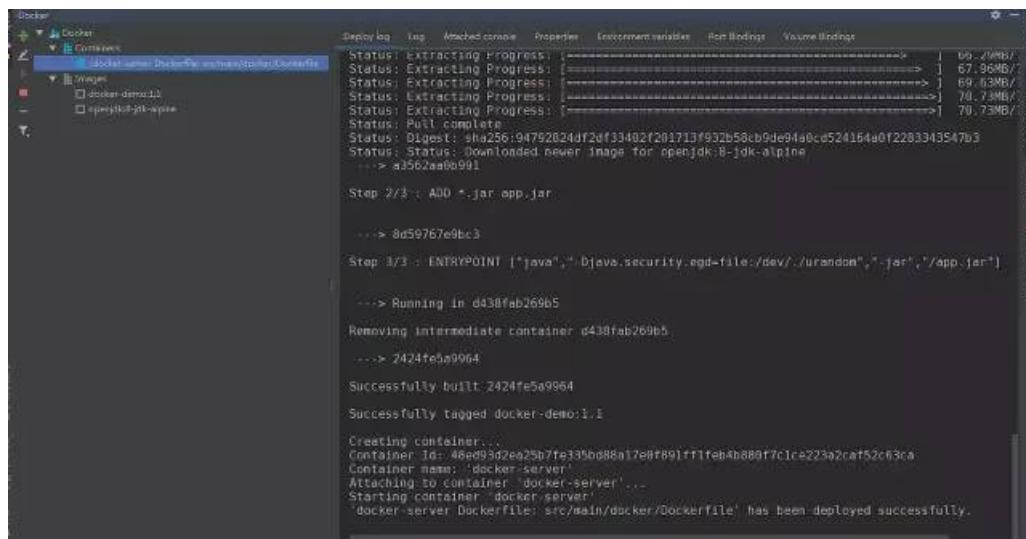
7、Maven打包



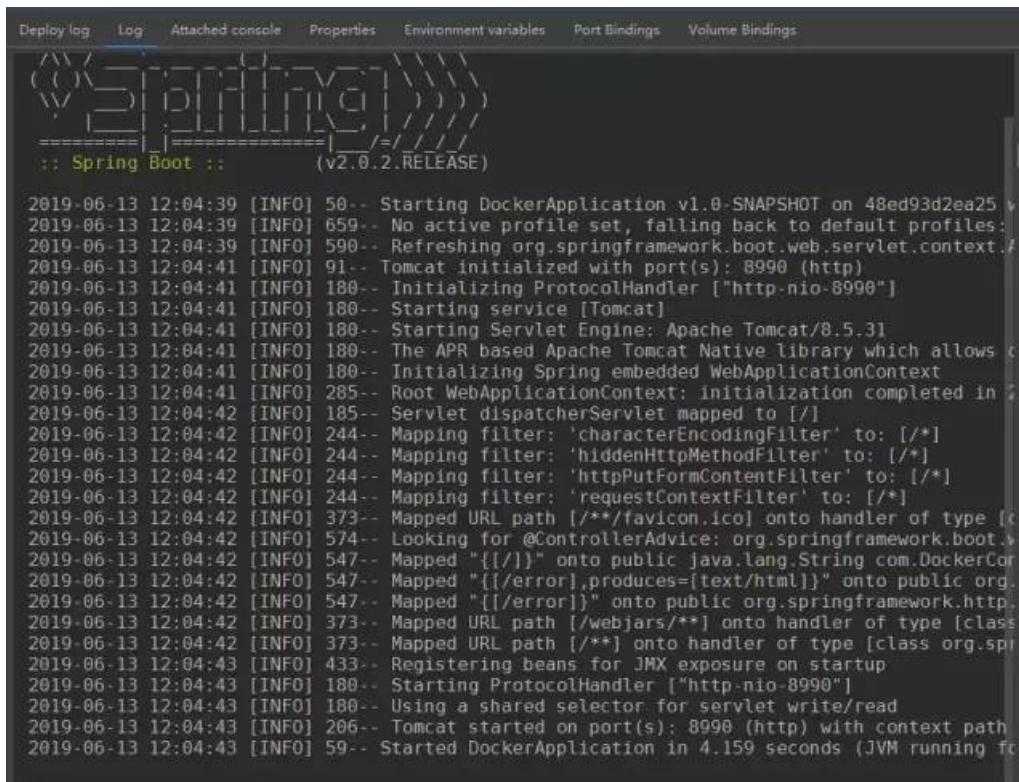
8、运行



先pull基础镜像，然后再打包镜像，并将镜像部署到远程docker运行



9、运行成功



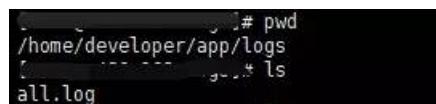
The screenshot shows the Docker tool window in IDEA. The 'Log' tab is selected, displaying the application's startup logs. The logs indicate the application is starting on port 8990, initializing Tomcat, and mapping various URL paths to handlers. The log output ends with the message 'Started DockerApplication in 4.159 seconds (JVM running for 4.159 seconds)'.

```
2019-06-13 12:04:39 [INFO] 50-- Starting DockerApplication v1.0-SNAPSHOT on 48ed93d2ea25 with PID 1 in directory /app
2019-06-13 12:04:39 [INFO] 659-- No active profile set, falling back to default profiles: []
2019-06-13 12:04:39 [INFO] 590-- Refreshing org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext@590533e: startup date [2019-06-13T12:04:39.214+0000]; root of context hierarchy
2019-06-13 12:04:41 [INFO] 91-- Tomcat initialized with port(s): 8990 (http)
2019-06-13 12:04:41 [INFO] 180-- Initializing ProtocolHandler ["http-nio-8990"]
2019-06-13 12:04:41 [INFO] 180-- Starting service [Tomcat]
2019-06-13 12:04:41 [INFO] 180-- Starting Servlet Engine: Apache Tomcat/8.5.31
2019-06-13 12:04:41 [INFO] 180-- The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: [/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.232-b09-0ubuntu1~16.04.1-amd64/lib/amd64]
2019-06-13 12:04:41 [INFO] 180-- Initializing Spring embedded WebApplicationContext
2019-06-13 12:04:41 [INFO] 285-- Root WebApplicationContext: initialization completed in 2 ms
2019-06-13 12:04:42 [INFO] 185-- Servlet dispatcherServlet mapped to [/]
2019-06-13 12:04:42 [INFO] 244-- Mapping filter: 'characterEncodingFilter' to: [//*]
2019-06-13 12:04:42 [INFO] 244-- Mapping filter: 'hiddenHttpMethodFilter' to: [//*]
2019-06-13 12:04:42 [INFO] 244-- Mapping filter: 'httpPutFormContentFilter' to: [//*]
2019-06-13 12:04:42 [INFO] 244-- Mapping filter: 'requestContextFilter' to: [//*]
2019-06-13 12:04:42 [INFO] 373-- Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.boot.web.servlet.error.ErrorMvcHandler]
2019-06-13 12:04:42 [INFO] 574-- Looking for @ControllerAdvice: org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration$WebExceptionHandler
2019-06-13 12:04:42 [INFO] 547-- Mapped "={"/}" onto public java.lang.String com.DockerController.error()
2019-06-13 12:04:42 [INFO] 547-- Mapped "[{/error}], produces=[text/html]}" onto public org.springframework.http.ResponseEntity<org.springframework.web.util.ErrorPage> com.DockerController.error(javax.servlet.http.HttpServletRequest)
2019-06-13 12:04:42 [INFO] 547-- Mapped "[{/error}]" onto public org.springframework.http.ResponseEntity<org.springframework.web.util.ErrorPage> com.DockerController.error()
2019-06-13 12:04:42 [INFO] 373-- Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.handler.BeanNameUrlHandler]
2019-06-13 12:04:42 [INFO] 373-- Mapped URL path [/**] onto handler of type [class org.springframework.boot.web.servlet.error.ErrorMvcHandler]
2019-06-13 12:04:43 [INFO] 433-- Registering beans for JMX exposure on startup
2019-06-13 12:04:43 [INFO] 180-- Starting ProtocolHandler ["http-nio-8990"]
2019-06-13 12:04:43 [INFO] 180-- Using a shared selector for servlet write/read
2019-06-13 12:04:43 [INFO] 206-- Tomcat started on port(s): 8990 (http) with context path /
2019-06-13 12:04:43 [INFO] 59-- Started DockerApplication in 4.159 seconds (JVM running for 4.159 seconds)
```

10、浏览器访问



11、日志查看



```
[# pwd
/home/developer/app/logs
[ # ls
all.log]
```

自此，通过IDEA部署Spring Boot项目到Docker成功！难以想象，部署一个Java web项目竟然如此简单方便！

-END-

如果看到这里，说明你喜欢这篇文章，请[转发、点赞](#)。微信搜索「web_resource」，关注后回复「进群」或者扫描下方二维码即可进入无广告交流群。

↓ 扫描二维码进群 ↓



扫一扫上面的二维码图案，加我微信

推荐阅读

1. Java后端优质文章整理
2. 7 个开源的 Spring Boot 前后端分离项目
3. 如何设计 API 接口, 实现统一格式返回?
4. 花 20 分钟, 再来梳理一下 Git 基础知识
5. 在 Spring Boot 中, 如何干掉 if else



Java后端

长按识别二维码，关注我的公众号

喜欢文章，点个在看 

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

收藏了！IntelliJ IDEA 2019 快捷键开发手册

Java后端 2019-09-17

点击上方**蓝色字体**, 选择“标星公众号”

优质文章, 第一时间送达

来源：mamicode.com/info-detail-2540162.html

古人有云：工欲善其事，必先利其器，要是只是手握利器，而不能发挥其最大的效益，那无异于赤手空拳，对敌对垒。那古人所云，未得其精髓，只能为碎语闲言尔。

自动代码

常用的有fori/sout/psvm+Tab即可生成循环、System.out、main方法等boilerplate样板代码。

例如要输入for(User user : users)只需输入user.for+Tab；

再比如，要输入Date birthday = user.getBirthday()只需输入user.getBirthday().var+Tab即可。

代码标签输入完成后，按Tab，生成代码。

1. Ctrl+Alt+O 优化导入的类和包
2. Alt+Insert 生成代码(如get, set方法, 构造函数等) 或者右键 (Generate)
3. fori/sout/psvm + Tab
4. Ctrl+Alt+T 生成try catch 或者 Alt+enter
5. CTRL+ALT+T 把选中的代码放在 TRY{} IF{} ELSE{} 里
6. Ctrl + O 重写方法
7. Ctrl + I 实现方法
8. Ctrl+shift+U 大小写转化
9. ALT+回车 导入包, 自动修正
10. ALT+/ 代码提示
11. CTRL+J 自动代码
12. Ctrl+Shift+J, 整合两行为一行
13. CTRL+空格 代码提示
14. CTRL+SHIFT+SPACE 自动补全代码
15. CTRL+ALT+L 格式化代码
16. CTRL+ALT+I 自动缩进
17. CTRL+ALT+O 优化导入的类和包
18. ALT+INSERT 生成代码(如GET, SET方法, 构造函数等)
19. CTRL+E 最近更改的代码
20. CTRL+ALT+SPACE 类名或接口名提示
21. CTRL+P 方法参数提示
22. CTRL+Q, 可以看到当前方法的声明
23. Shift+F6 重构-重命名 (包、类、方法、变量、甚至注释等)
24. Ctrl+Alt+V 提取变量

查询快捷键

1. Ctrl+Shift+Backspace可以跳转到上次编辑的地方
2. CTRL+ALT+ left/right 前后导航编辑过的地方
3. ALT+7 靠左窗口显示当前文件的结构
4. Ctrl+F12 浮动显示当前文件的结构
5. ALT+F7 找到你的函数或者变量或者类的所有引用到的地方
6. CTRL+ALT+F7 找到你的函数或者变量或者类的所有引用到的地方
7. Ctrl+Shift+Alt+N 查找类中的方法或变量
8. 双击SHIFT 在项目的所有目录查找文件
9. Ctrl+N 查找类
10. Ctrl+Shift+N 查找文件
11. CTRL+G 定位行
12. CTRL+F 在当前窗口查找文本
13. CTRL+SHIFT+F 在指定窗口查找文本
14. CTRL+R 在当前窗口替换文本
15. CTRL+SHIFT+R 在指定窗口替换文本
16. ALT+SHIFT+C 查找修改的文件
17. CTRL+E 最近打开的文件
18. F3 向下查找关键字出现位置
19. SHIFT+F3 向上一个关键字出现位置
20. 选中文本，按Alt+F3，高亮相同文本，F3逐个往下查找相同文本
21. F4 查找变量来源
22. CTRL+SHIFT+O 弹出显示查找内容
23. Ctrl+W 选中代码，连续按会有其他效果
24. F2 或Shift+F2 高亮错误或警告快速定位
25. Ctrl+Up/Down 光标跳转到第一行或最后一行下
26. Ctrl+B 快速打开光标处的类或方法
27. CTRL+ALT+B 找所有的子类
28. CTRL+SHIFT+B 找变量的类
29. Ctrl+Shift+上下键 上下移动代码
30. Ctrl+Alt+ left/right 返回至上次浏览的位置
31. Ctrl+X 删除行
32. Ctrl+D 复制行
33. Ctrl+/ 或 Ctrl+Shift+/ 注释（// 或者/…/）
34. Ctrl+H 显示类结构图
35. Ctrl+Q 显示注释文档
36. Alt+F1 查找代码所在位置
37. Alt+1 快速打开或隐藏工程面板
38. Alt+ left/right 切换代码视图
39. ALT+ ↑ / ↓ 在方法间快速移动定位
40. CTRL+ALT+ left/right 前后导航编辑过的地方
41. Ctrl+Shift+Backspace可以跳转到上次编辑的地方
42. Alt+6 查找TODO

其他快捷键

1. SHIFT+ENTER 另起一行
2. CTRL+Z 倒退(撤销)
3. CTRL+SHIFT+Z 向前(取消撤销)

4. **CTRL+ALT+F12** 资源管理器打开文件夹
5. **ALT+F1** 查找文件所在目录位置
6. **SHIFT+ALT+INSERT** 竖编辑模式
7. **CTRL+F4** 关闭当前窗口
8. **Ctrl+Alt+V**, 可以引入变量。例如: new String(); 自动导入变量定义
9. **Ctrl+~**, 快速切换方案 (界面外观、代码风格、快捷键映射等菜单)

调试快捷键

其实常用的就是**F8 F7 F9** 最值得一提的就是**Drop Frame** 可以让运行过的代码从头再来。

1. **alt+F8** debug时选中查看值
2. **Alt+Shift+F9**, 选择 Debug
3. **Alt+Shift+F10**, 选择 Run
4. **Ctrl+Shift+F9**, 编译
5. **Ctrl+Shift+F8**, 查看断点
6. **F7**, 步入
7. **Shift+F7**, 智能步入
8. **Alt+Shift+F7**, 强制步入
9. **F8**, 步过
10. **Shift+F8**, 步出
11. **Alt+Shift+F8**, 强制步过
12. **Alt+F9**, 运行至光标处
13. **Ctrl+Alt+F9**, 强制运行至光标处
14. **F9**, 恢复程序
15. **Alt+F10**, 定位到断点

重构

1. **Ctrl+Alt+Shift+T**, 弹出重构菜单
2. **Shift+F6**, 重命名
3. **F6**, 移动
4. **F5**, 复制
5. **Alt+Delete**, 安全删除
6. **Ctrl+Alt+N**, 内联

十大IntelliJ IDEA快捷键

IntelliJ IDEA中有很多快捷键让人爱不释手, stackoverflow上也有一些有趣的讨论。每个人都有自己的最爱, 想排出个理想的榜单还真是困难。

以前也整理过IntelliJ的快捷键, 这次就按照我日常开发时的使用频率, 简单分类列一下我最喜欢的十大快捷-神-键吧。 [微信搜索 web_resource 关注后回复 Java, 送你2019最新Java资源。](#)

1 智能提示

IntelliJ首当其冲的当然就是Intelligence智能! 基本的代码提示用**Ctrl+Space**, 还有更智能地按类型信息提示**Ctrl+Shift+Space**, 但因为IntelliJ总是随着我们敲击而自动提示, 所以很多时候都不会手动敲这两个快捷键(除非提示框消失)

了)。

用F2/ Shift+F2移动到有错误的代码，Alt+Enter快速修复(即Eclipse中的Quick Fix功能)。当智能提示为我们自动补全方法名时，我们通常要自己补上行尾的反括号和分号，当括号嵌套很多层时会很麻烦，这时我们只需敲Ctrl+Shift+Enter就能自动补全末尾的字符。而且不只是括号，例如敲完if/for时也可以自动补上{}花括号。

最后要说一点，IntelliJ能够智能感知Spring、Hibernate等主流框架的配置文件和类，以静制动，在看似“静态”的外表下，智能地扫描理解你的项目是如何构造和配置的。[微信搜索 web_resource 关注后回复 Java，送你2019最新Java资源](#)。

2 重构

IntelliJ重构是另一完爆Eclipse的功能，其智能程度令人瞠目结舌，比如提取变量时自动检查到所有匹配同时提取成一个变量等。尤其看过《重构-改善既有代码设计》之后，有了IntelliJ的配合简直是令人大呼过瘾！也正是强大的智能和重构功能，使IntelliJ下的TDD开发非常顺畅。

切入正题，先说一个无敌的重构功能大汇总快捷键Ctrl+Shift+Alt+T，叫做Refactor This。按法有点复杂，但也符合IntelliJ的风格，很多快捷键都要双手完成，而不像Eclipse不少最有用的快捷键可以潇洒地单手完成(不知道算不算Eclipse的一大优点)，但各位用过Emacs的话就会觉得也没什么了(非Emacs黑)。

此外，还有些最常用的重构技巧，因为太常用了，若每次都在Refactor This菜单里选的话效率有些低。比如Shift+F6直接就是改名，Ctrl+Alt+V则是提取变量。关注Java技术栈微信公众号，在后台回复关键字：IDEA，可以获取一份栈长整理的IDEA最新技术干货。[微信搜索 web_resource 关注后回复 Java，送你2019最新Java资源](#)。

3 代码生成

这一点类似Eclipse，虽不是独到之处，但因为日常使用频率极高，所以还是罗列在榜单前面。常用的有for/i/sout/psvm+Tab即可生成循环、System.out、main方法等boilerplate样板代码，用Ctrl+J可以查看所有模板。

后面“辅助”一节中将会讲到Alt+Insert，在编辑窗口中点击可以生成构造函数、toString、getter/setter、重写父类方法等。这两个技巧实在太常用了，几乎每天都要生成一堆main、System.out和getter/setter。

另外，IntelliJ IDEA 13中加入了后缀自动补全功能(Postfix Completion)，比模板生成更加灵活和强大。例如要输入for(User user : users)只需输入user.for+Tab。再比如，要输入Date birthday = user.getBirthday();只需输入user.getBirthday().var+Tab即可。

4 编辑

编辑中不得不说的一大神键就是能够自动按语法选中代码的Ctrl+W以及反向的Ctrl+Shift+W了。此外，Ctrl+Left/Right移动光标到前/后单词，Ctrl+[]移动到前/后代码块，这些类Vim风格的光标移动也是一大亮点。以上Ctrl+Left/Right/[]加上Shift的话就能选中跳跃范围内的代码。Alt+Forward/Backward移动到前/后方法。还有些非常普通的像Ctrl+Y删除行、Ctrl+D复制行、Ctrl+折叠代码就不多说了。

关于光标移动再多扩展一点，除了IntelliJ本身已提供的功能外，我们还可以安装ideaVim或者emacsIDEAs享受到Vim的快速移动和Emacs的AceJump功能(超爽！)。

另外，IntelliJ的书签功能也是不错的，用Ctrl+Shift+Num定义1-10书签(再次按这组快捷键则是删除书签)，然后通过Ctrl+Num跳转。这避免了多次使用前/下一编辑位置Ctrl+Left/Right来回跳转的麻烦，而且此快捷键默认与Windows热键冲突(默认多了Alt，与Windows改变显示器显示方向冲突，一不小心显示器就变成倒着显式的了，囧啊)。[微信搜索 web_resource 关注后回复 Java，送你2019最新Java资源](#)。

5 查找打开

类似Eclipse，IntelliJ的Ctrl+N/Shift+N可以打开类或资源，但IntelliJ更加智能一些，我们输入的任何字符都将看作模糊匹配，省却了Eclipse中还有输入*的麻烦。最新版本的IDEA还加入了Search Everywhere功能，只需按Shift+Shift即可在一个弹出框中搜索任何东西，包括类、资源、配置项、方法等等。

类的继承关系则可用Ctrl+H打开类层次窗口，在继承层次上跳转则用Ctrl+B/Ctrl+Alt+B分别对应父类或父方法定义和子类或子方法实现，查看当前类的所有方法用Ctrl+F12。

要找类或方法的使用也很简单，Alt+F7。要查找文本的出现位置就用Ctrl+F/Ctrl+Shift+F在当前窗口或全工程中查找，再配合F3/Shift+F3前后移动到下一匹配处。

IntelliJ更加智能的又一佐证是在任意菜单或显示窗口，都可以直接输入你要找的单词，IntelliJ就会自动为你过滤。关注Java技术栈微信公众号，在后台回复关键字：IDEA，可以获取一份栈长整理的IDEA最新技术干货。

6 其他辅助

以上这些神键配上一些辅助快捷键，即可让你的双手90%以上的时间摆脱鼠标，专注于键盘仿佛在进行钢琴表演。这些不起眼却是至关重要的最后一块拼图有：

Ø 命令：Ctrl+Shift+A可以查找所有IntelliJ的命令，并且每个命令后面还有其快捷键。所以它不仅是一大神键，也是查找学习快捷键的工具。

Ø 新建：Alt+Insert可以新建类、方法等任何东西。

Ø 格式化代码：格式化import列表Ctrl+Alt+O，格式化代码Ctrl+Alt+L。

Ø 切换窗口：Alt+Num，常用的有1-项目结构，3-搜索结果，4/5-运行调试。Ctrl+Tab切换标签页，Ctrl+E/Ctrl+Shift+E打开最近打开过的或编辑过的文件。

Ø 单元测试：Ctrl+Alt+T创建单元测试用例。

Ø 运行：Alt+Shift+F10运行程序，Shift+F9启动调试，Ctrl+F2停止。

Ø 调试：F7/F8/F9分别对应Step into，Step over，Continue。

此外还有些我自定义的，例如水平分屏Ctrl+|等，和一些神奇的小功能Ctrl+Shift+V粘贴很早以前拷贝过的，Alt+Shift+Insert进入到列模式进行按列选中。

Ø Top #10切来切去：Ctrl+Tab

Ø Top #9选你所想：Ctrl+W

Ø Top #8代码生成：Template/Postfix +Tab

Ø Top #7发号施令：Ctrl+Shift+A

Ø Top #6无处藏身：Shift+Shift

Ø Top #5自动完成：Ctrl+Shift+Enter

Ø Top #4创造万物：Alt+Insert

太难割舍，前三名并列吧！

Ø Top #1智能补全：Ctrl+Shift+Space

Ø Top #1自我修复：Alt+Enter

Ø Top #1重构一切：Ctrl+Shift+Alt+T

CTRL+ALT+ left/right 前后导航编辑过的地方 Ctrl+Shift+Backspace可以跳转到上次编辑的地方

如果看到这里，说明你喜欢这篇文章，帮忙[转发](#)一下吧，感谢。微信搜索「web_resource」，关注后回复「进群」即可进入无广告技术交流群。

- [1. 技术群群友的一次阿里面试经历](#)
- [2. Spring 中的 18 个注解，你会几个？](#)
- [3. 寓教于乐，用玩游戏的方式学习 Git](#)
- [4. 在浏览器输入 URL 回车之后发生了什么？](#)
- [5. 接私活必备的 10 个开源项目](#)



Web项目聚集地

微信扫描二维码，关注我的公众号

喜欢文章，点个在看 

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

收藏了！IntelliJ IDEA 快捷键 Windows 版本

满风 Java后端 3月8日



作者 | 满风

链接 | my.oschina.net/dyyweb/blog/494504

前言：常用快捷键

IntelliJ IDEA编辑器大受欢迎的原因之一是它的智能提示和丰富的快捷键，在日常开发中熟练的使用快捷键会大大提升开发的效率，本篇文章就笔者日常开发中的总结，把常用的、好用的快捷键做一个列表，方便查阅。喜欢本文的朋友可以转发朋友圈，方便翻阅。

提示：Mac 版本的快捷键在本期推文次条

1. 自动代码

常用的有for/i/sout/psvm+Tab即可生成循环、System.out、main方法等boilerplate样板代码，例如要输入for(User user : users)只需输入user.for+Tab。

再比如：要输入Date birthday = user.getBirthday();只需输入user.getBirthday().var+Tab即可。代码标签输入完成后，按Tab，生成代码。

Ctrl+Alt+O 优化导入的类和包

Alt+Insert 生成代码(如get, set方法, 构造函数等) 或者右键(Generate)

for/i/sout/psvm + Tab

Ctrl+Alt+T 生成try catch 或者 Alt+enter

CTRL+ALT+T 把选中的代码放在 TRY{} IF{} ELSE{} 里

Ctrl + O 重写方法

Ctrl + I 实现方法

Ctrl+shift+U 大小写转化

ALT+回车 导入包,自动修正

ALT+/ 代码提示

CTRL+J 自动代码

Ctrl+Shift+J, 整合两行为一行

CTRL+空格 代码提示

CTRL+SHIFT+SPACE 自动补全代码

CTRL+ALT+L 格式化代码

CTRL+ALT+I 自动缩进

CTRL+ALT+O 优化导入的类和包

ALT+INSERT 生成代码(如GET,SET方法,构造函数等)

CTRL+E 最近更改的代码

CTRL+ALT+SPACE 类名或接口名提示

CTRL+P 方法参数提示

CTRL+Q, 可以看到当前方法的声明

Shift+F6 重构-重命名 (包、类、方法、变量、甚至注释等)

Ctrl+Alt+V 提取变量

2. 查询快捷键

Ctrl+Shift+Backspace 可以跳转到上次编辑的地方

CTRL+ALT+ left/right 前后导航编辑过的地方

ALT+7 靠左窗口显示当前文件的结构

Ctrl+F12 浮动显示当前文件的结构

ALT+F7 找到你的函数或者变量或者类的所有引用到的地方

CTRL+ALT+F7 找到你的函数或者变量或者类的所有引用到的地方

Ctrl+Shift+Alt+N 查找类中的方法或变量

双击SHIFT 在项目的所有目录查找文件

Ctrl+N 查找类

Ctrl+Shift+N 查找文件

CTRL+G 定位行

CTRL+F 在当前窗口查找文本

CTRL+SHIFT+F 在指定窗口查找文本

CTRL+R 在当前窗口替换文本

CTRL+SHIFT+R 在指定窗口替换文本

ALT+SHIFT+C 查找修改的文件

CTRL+E 最近打开的文件

F3 向下查找关键字出现位置

SHIFT+F3 向上一个关键字出现位置

选中文本, 按Alt+F3 , 高亮相同文本, F3逐个往下查找相同文本

F4 查找变量来源

CTRL+SHIFT+O 弹出显示查找内容

Ctrl+W 选中代码, 连续按会有其他效果

F2 或Shift+F2 高亮错误或警告快速定位

Ctrl+Up/Down 光标跳转到第一行或最后一行下

Ctrl+B 快速打开光标处的类或方法

CTRL+ALT+B 找所有的子类

CTRL+SHIFT+B 找变量的类

Ctrl+Shift+上下键 上下移动代码

Ctrl+Alt+ left/right 返回至上次浏览的位置

Ctrl+X 删除行

Ctrl+D 复制行

Ctrl+/ 或 Ctrl+Shift+/ 注释(// 或者/*...*/)

Ctrl+H 显示类结构图

Ctrl+Q 显示注释文档

Alt+F1 查找代码所在位置

Alt+1 快速打开或隐藏工程面板

Alt+ left/right 切换代码视图

ALT+ ↑ / ↓ 在方法间快速移动定位

CTRL+ALT+ left/right 前后导航编辑过的地方

Ctrl+Shift+Backspace 可以跳转到上次编辑的地方

Alt+6 查找TODO

3. 其他快捷键

SHIFT+ENTER 另起一行

CTRL+Z 倒退(撤销)

CTRL+SHIFT+Z 向前(取消撤销)

CTRL+ALT+F12 资源管理器打开文件夹

ALT+F1 查找文件所在目录位置

SHIFT+ALT+INSERT 竖编辑模式

CTRL+F4 关闭当前窗口

Ctrl+Alt+V, 可以引入变量。例如: new String(); 自动导入变量定义

Ctrl+~, 快速切换方案(界面外观、代码风格、快捷键映射等菜单)

4. svn快捷键

ctrl+k 提交代码到SVN

ctrl+t 更新代码

5. 调试快捷键

其实常用的 就是F8 F7 F9, 最值得一提的就是Drop Frame 可以让运行过的代码从头再来:

alt+F8 debug时选中查看值

Alt+Shift+F9, 选择 Debug

Alt+Shift+F10, 选择 Run

Ctrl+Shift+F9, 编译

Ctrl+Shift+F8, 查看断点

F7, 步入

Shift+F7, 智能步入

Alt+Shift+F7, 强制步入

F8, 步过

Shift+F8, 步出

Alt+Shift+F8, 强制步过

Alt+F9, 运行至光标处

Ctrl+Alt+F9, 强制运行至光标处

F9, 恢复程序

Alt+F10, 定位到断点

6. 重构

Ctrl+Alt+Shift+T, 弹出重构菜单

Shift+F6, 重命名

F6, 移动

F5, 复制

Alt+Delete, 安全删除

Ctrl+Alt+N, 内联

7. 十大 IntelliJ IDEA 快捷键

IntelliJ IDEA 中有很多快捷键让人爱不释手, stackoverflow 上也有一些有趣的讨论。每个人都有自己的最爱, 想排出个理想的榜单还真是困难。

以前也整理过 IntelliJ 的快捷键, 这次就按照我日常开发时的使用频率, 简单分类列一下我最喜欢的十大快捷-神-键吧。

1. 智能提示:

IntelliJ 首当其冲的当然就是 Intelligence 智能! 基本的代码提示用 Ctrl+Space, 还有更智能地按类型信息提示 Ctrl+Shift+Space, 但因为 IntelliJ 总是随着我们敲击而自动提示, 所以很多时候都不会手动敲这两个快捷键(除非提示框消失了)。用 F2/ Shift+F2 移动到有错误的代码, Alt+Enter 快速修复(即 Eclipse 中的 Quick Fix 功能)。

当智能提示为我们自动补全方法名时, 我们通常要自己补上行尾的反括号和分号, 当括号嵌套很多层时会很麻烦, 这时我们只需敲 Ctrl+Shift+Enter 就能自动补全末尾的字符。而且不只是括号, 例如敲完 if/for 时也可以自动补上 {} 花括号。

最后要说一点, IntelliJ 能够智能感知 Spring、Hibernate 等主流框架的配置文件和类, 以静制动, 在看似“静态”的外表下, 智能地扫描理解你的项目是如何构造和配置的。

2. 重构:

IntelliJ 重构是另一完爆 Eclipse 的功能, 其智能程度令人瞠目结舌, 比如提取变量时自动检查到所有匹配同时提取成一个变量等。尤其看过《重构-改善既有代码设计》之后, 有了 IntelliJ 的配合简直是令人大呼过瘾! 也正是强大的智能和重构功能, 使 IntelliJ 下的 TDD 开发非常顺畅。

切入正题, 先说一个无敌的重构功能大汇总快捷键 Ctrl+Shift+Alt+T, 叫做 Refactor This。按法有点复杂, 但也符合 IntelliJ 的风格, 很多快捷键都要双手完成, 而不像 Eclipse 不少最有用的快捷键可以潇洒地单手完成(不知道算不算 Eclipse 的一大优点), 但各位用过 Emacs 的话就会觉得也没什么了(非 Emacs 黑)。此外, 还有些最常用的重构技巧, 因为太常用了, 若每次都在 Refactor This 菜单里选的话效率有些低。比如 Shift+F6 直接就是改名, Ctrl+Alt+V 则是提取变量。

3 代码生成：

这一点类似Eclipse，虽不是独到之处，但因为日常使用频率极高，所以还是罗列在榜单前面。常用的有for/i/sout/psvm+Tab即可生成循环、System.out、main方法等boilerplate样板代码，用Ctrl+J可以查看所有模板。后面“辅助”一节中将会讲到Alt+Insert，在编辑窗口中点击可以生成构造函数、toString、getter/setter、重写父类方法等。这两个技巧实在太常用了，几乎每天都要生成一堆main、System.out和getter/setter。

另外，IntelliJ IDEA 13中加入了后缀自动补全功能(Postfix Completion)，比模板生成更加灵活和强大。例如要输入for(User user : users)只需输入user.for+Tab。再比如，要输入Date birthday = user.getBirthday();只需输入user.getBirthday().var+Tab即可。

4. 编辑：

编辑中不得不说的一大神键就是能够自动按语法选中代码的Ctrl+W以及反向的Ctrl+Shift+W了。此外，Ctrl+Left/Right移动光标到前/后单词，Ctrl+[/]移动到前/后代码块，这些类Vim风格的光标移动也是一大亮点。以上Ctrl+Left/Right/[]加上Shift的话就能选中跳跃范围内的代码。Alt+Forward/Backward移动到前/后方法。还有些非常普通的像Ctrl+Y删除行、Ctrl+D复制行、Ctrl+</>折叠代码就不多说了。

关于光标移动再多扩展一点，除了IntelliJ本身已提供的功能外，我们还可以安装ideaVim或者emacsIDEAs享受到Vim的快速移动和Emacs的AceJump功能(超爽！)。另外，IntelliJ的书签功能也是不错的，用Ctrl+Shift+Num定义1-10书签(再次按这组快捷键则是删除书签)，然后通过Ctrl+Num跳转。这避免了多次使用前/下一编辑位置Ctrl+Left/Right来回跳转的麻烦，而且此快捷键默认与Windows热键冲突(默认多了Alt，与Windows改变显示器显示方向冲突，一不小心显示器就变成倒着显式的了，冏啊)。

5 查找打开：

类似Eclipse，IntelliJ的Ctrl+N/Ctrl+Shift+N可以打开类或资源，但IntelliJ更加智能一些，我们输入的任何字符都将看作模糊匹配，省却了Eclipse中还有输入*的麻烦。最新版本的IDEA还加入了Search Everywhere功能，只需按Shift+Shift即可在一个弹出框中搜索任何东西，包括类、资源、配置项、方法等等。

类的继承关系则可用Ctrl+H打开类层次窗口，在继承层次上跳转则用Ctrl+B/Ctrl+Alt+B分别对应父类或父方法定义和子类或子方法实现，查看当前类的所有方法用Ctrl+F12。

要找类或方法的使用也很简单，Alt+F7。要查找文本的出现位置就用Ctrl+F/Ctrl+Shift+F在当前窗口或全工程中查找，再配合F3/Shift+F3前后移动到下一匹配处。

IntelliJ更加智能的又一佐证是在任意菜单或显示窗口，都可以直接输入你要找的单词，IntelliJ就会自动为你过滤。

另外，更多 IDEA 相关干货，如插件教程、Bug调试教程可以在公众号：Java后端后台回复 技术博文 获取。

6. 其他辅助：

以上这些神键配上一些辅助快捷键，即可让你的双手90%以上的时间摆脱鼠标，专注于键盘仿佛在进行钢琴表演。这些不起眼却是至关重要的最后一块拼图有：

命令：Ctrl+Shift+A可以查找所有IntelliJ的命令，并且每个命令后面还有其快捷键。所以它不仅是一大神键，也是查找学习快捷键的工具。

新建:Alt+Insert可以新建类、方法等任何东西。

格式化代码:格式化import列表Ctrl+Alt+O, 格式化代码Ctrl+Alt+L。

切换窗口:Alt+Num, 常用的有1-项目结构, 3-搜索结果, 4/5-运行调试。Ctrl+Tab切换标签页, Ctrl+E/Ctrl+Shift+E打开最近打开过的或编辑过的文件。

单元测试:Ctrl+Alt+T创建单元测试用例。

运行:Alt+Shift+F10运行程序, Shift+F9启动调试, Ctrl+F2停止。

调试:F7/F8/F9分别对应Step into, Step over, Continue。

此外还有些我自定义的, 例如水平分屏Ctrl+|等, 和一些神奇的小功能Ctrl+Shift+V粘贴很早以前拷贝过的, Alt+Shift+Insert进入到列模式进行按列选中。

自己最常用的十个快捷键:

切来切去:Ctrl+Tab

选你所想:Ctrl+W

代码生成:Template/Postfix +Tab

发号施令:Ctrl+Shift+A

无处藏身:Shift+Shift

自动完成:Ctrl+Shift+Enter

创造万物:Alt+Insert

智能补全:Ctrl+Shift+Space

自我修复:Alt+Enter

重构一切:Ctrl+Shift+Alt+T

CTRL+ALT+ left/right 前后导航编辑过的地方

Ctrl+Shift+Backspace可以跳转到上次编辑的地

题图、编辑由公众号「Java后端」制作, 转载请署名此句话和原作者.

推荐阅读

[1. 盘点那些改变过世界的代码](#)

[2. 基于token的多平台身份认证架构设计](#)

[3. 少侠! 如何写一手好 SQL ?](#)

[4. 写给大忙人看的操作系统](#)



微信搜一搜

Q Java后端

[阅读原文](#)

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

收藏了！盘点 IntelliJ IDEA 那些不为人知的小技巧

Sam哥哥 Java后端 2019-10-13

点击上方 Java后端, 选择设为星标

技术博文, 及时送达

作者 | Sam哥哥

链接 | blog.csdn.net/linsongbin1

[上一篇：从零搭建创业公司后台技术栈](#)



IntelliJ IDEA真是越用越觉得它强大，它总是在我们写代码的时候，不时给我们来个小惊喜。出于对IntelliJ IDEA的喜爱，我决定写一个与其相关的专栏或者系列，把一些好用的IntelliJ IDEA技巧分享给大家。本文是这个系列的第一篇，主要介绍一些你可能不知道的但是又实用的小技巧。

我们可以使用【Presentation Mode】，将IDEA弄到最大，可以让你只关注一个类里面的代码，进行毫无干扰的coding。

可以使用Alt+V快捷键，弹出View视图，然后选择Enter Presentation Mode。效果如下：

```
package spring.cloud.config;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@EnableConfigServer
@SpringBootApplication
public class ConfigApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConfigApplication.class, args);
    }
}
```

<https://blog.csdn.net/linsongbin1>

这个模式的好处就是，可以让你更加专注，因为你只能看到特定某个类的代码。可能读者会问，进入这个模式后，我想看其他类的代码怎么办？这个时候，就要考验你快捷键的熟练程度了。你可以使用CTRL+E弹出最近使用的文件。又或者使用CTRL+N和CTRL+SHIFT+N定位文件。

如何退出这个模式呢？很简单，使用ALT+V弹出view视图，然后选择Exit Presentation Mode 即可。但是我强烈建议你不要这么做，因为你是可以在Enter Presentation Mode模式下在IDEA里面做任何事情的。当然前提是，你对IDEA足够熟练。

如果你使用IDEA在编写JSON字符串的时候，然后要一个一个\去转义双引号的话，就实在太不应该了，又烦又容易出错。在IDEA可以使用Inject language帮我们自动转义双引号。

```
package spring.cloud.config;

public class DemoIDEA {
    public static void main(String[] args) {
        String jsonString = "";
    }
}
```

https://blog.csdn.net/linsongbin1

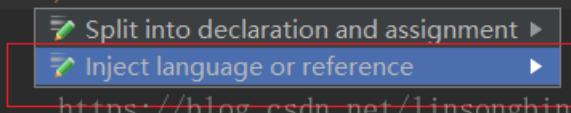
先将焦点定位到双引号里面，使用alt+enter快捷键弹出inject language视图，并选中

Inject language or reference。

```
package spring.cloud.config;

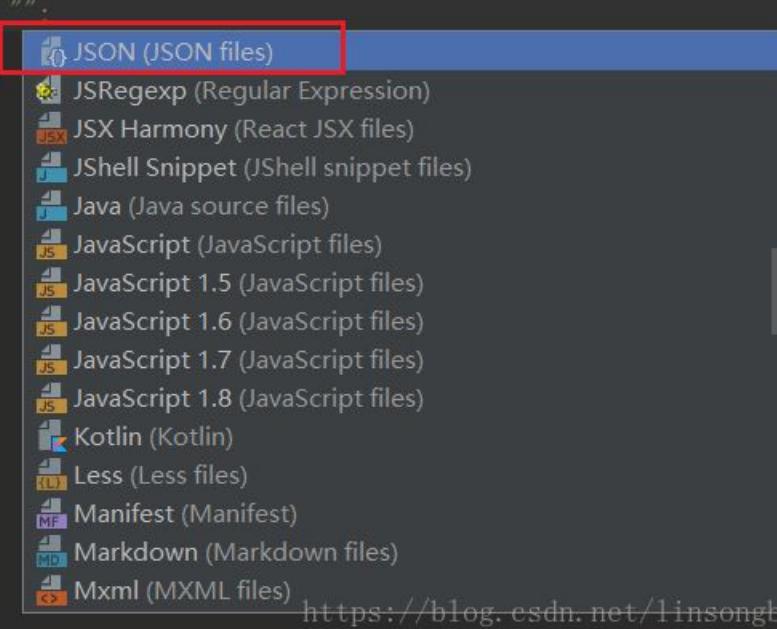
public class DemoIDEA {
    public static void main(String[] args) {
        String jsonString = "";
    }
}
```

https://blog.csdn.net/linsongbin1



选择后，切记，要直接按下enter回车键，才能弹出inject language列表。在列表中选择 json组件。

```
public class DemoIDEA {
    public static void main(String[] args) {
        String jsonString = "";
    }
}
```

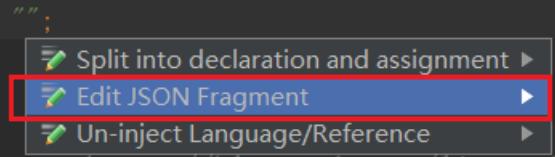


https://blog.csdn.net/linsongbin1

选择完后。鼠标焦点自动会定位在双引号里面，这个时候你再次使用alt+enter就可以看到

```
public class DemoIDEA {
    public static void main(String[] args) {
        //language=JSON
        String jsonString = "";
    }
}
```

https://blog.csdn.net/linsongbin1



选中Edit JSON Fragment并回车，就可以看到编辑JSON文件的视图了。

The screenshot shows an IDE interface with a code editor and a JSON fragment editor. In the code editor, a Java class `DemoIDEA` is defined with a `main` method containing a string assignment:

```
public class DemoIDEA {  
    public static void main(String[] args) {  
        //language=JSON  
        String jsonString = "{\"name\":\"Sam哥哥\", \"age\":\"90后啦\"}";  
    }  
}
```

The JSON string is highlighted with a red box. Below the code editor, a tab labeled "JSON Fragment (DemoIDEA.java:151).json" is open, showing the JSON content:

```
[{"name": "Sam哥哥", "age": "90后啦"}]
```

The entire JSON fragment is also highlighted with a red box. At the bottom right of the screen, there is a URL: <https://blog.csdn.net/linsongbin1>.

可以看到IDEA确实帮我们自动转义双引号了。如果要退出编辑JSON信息的视图，只需要使用ctrl+F4快捷键即可。

Inject language可以支持的语言和操作多到你难以想象，读者可以自行研究。

假设有下面的场景，某个类的名字在project视图里被挡住了某一部分。

The screenshot shows the IntelliJ IDEA project structure. The "Project" tool window on the left lists packages and files. A class named `ConfigApplication` is selected in the code editor. In the project tree, the class name is partially obscured by a red box, specifically the part after "ConfigAppl". The code editor shows the following Java code:

```
package spring.cloud.config;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.config.server.EnableConfigServer;  
@EnableConfigServer  
@SpringBootApplication  
public class ConfigApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(ConfigApplication.class, args);  
    }  
}
```

The URL at the bottom right is <https://blog.csdn.net/linsongbin1>.

你想完整的看到类的名字，该怎么做。一般都是使用鼠标来移动分割线，但是这样子效率太低了。可以使用alt+1把鼠标焦点定位到project视图里，然后直接使用ctrl+shift+左右箭头来移动分割线。微信搜索 web_resource 关注获取更多推送

ctrl+shift+enter其实是表示为您收尾的意思，不只是用来给代码加分号的。比如说：

```
package spring.cloud.config;

public class DemoIDEA {
    public static void main(String[] args) {
        String s = "test";
        if (s == null)
    }
}
```

<https://blog.csdn.net/linsongbin1>

这段代码，我们还需要为if语句加上大括号才能编译通过，这个时候你直接输入ctrl+shift+enter，IDEA会自动帮你收尾，加上大括号的。

IDEA的重构功能非常强大，但是也有时候，在单个类里面，如果只是想批量修改某个文本，大可不必使用到重构的功能。比如说：

```
public class DemoIDEA {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    public void get(RabbitTemplate rabbitTemplate) {
        rabbitTemplate.convertAndSend( object: "tempString");
    }

    public void get2(RabbitTemplate rabbitTemplate) {
        rabbitTemplate.convertAndSend( object: "tempString");
    }
}
```

<https://blog.csdn.net/linsongbin1>

上面的代码中，有5个地方用到了rabbitTemplate文本，如何批量修改呢？

首先是使用ctrl+w选中rabbitTemplate这个文本,然后依次使用5次alt+j快捷键，逐个选中，这样五个文本就都被选中并且高亮起来了，这个时候就可以直接批量修改了。

```
public class DemoIDEA {

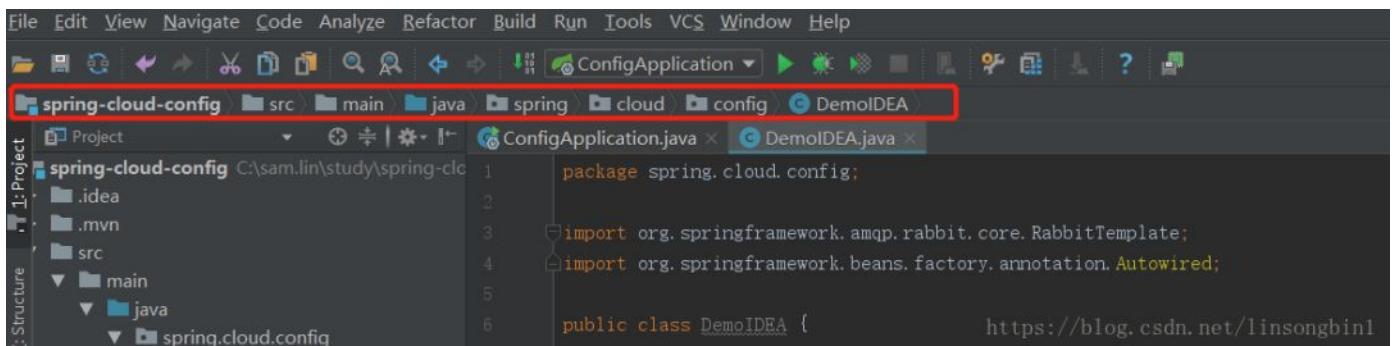
    @Autowired
    private RabbitTemplate rabbitTemplateTest;

    public void get(RabbitTemplate rabbitTemplateTest) {
        rabbitTemplateTest.convertAndSend( object: "tempString");
    }

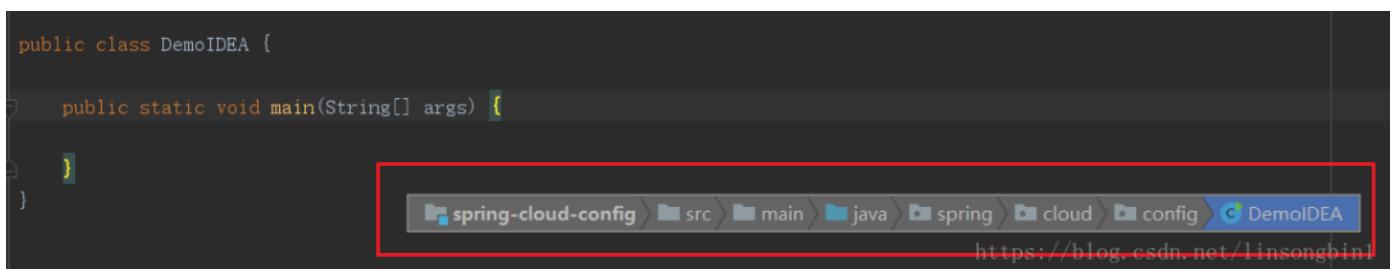
    public void get2(RabbitTemplate rabbitTemplateTest) {
        rabbitTemplateTest.convertAndSend( object: "tempString");
    }
}
```

<https://blog.csdn.net/linsongbin1>

去掉导航栏，因为平时用的不多。

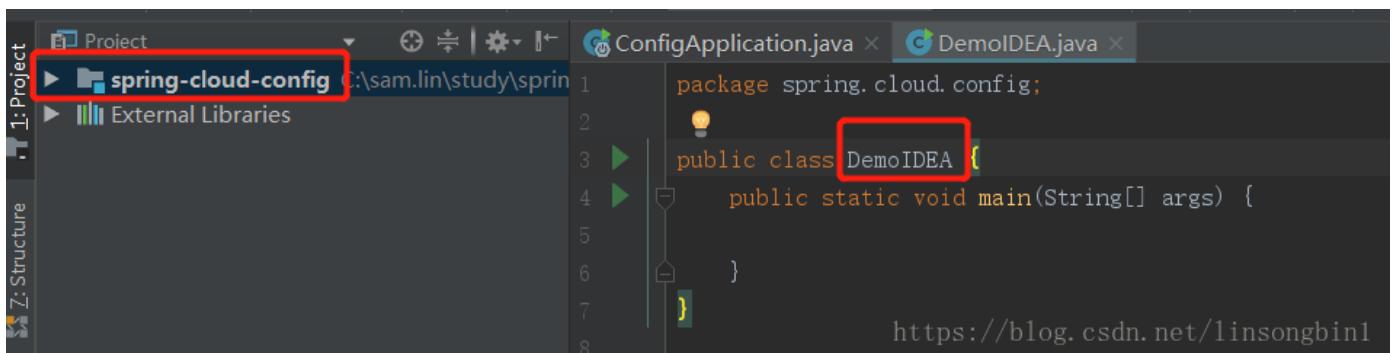


可以把红色的导航栏去掉，让IDEA显得更加干净整洁一些。使用alt+v，然后去掉Navigation bar即可。去掉这个导航栏后，如果你偶尔还是要用的，直接用alt+home就可以临时把导航栏显示出来。微信搜索 web_resource 关注获取更多推送



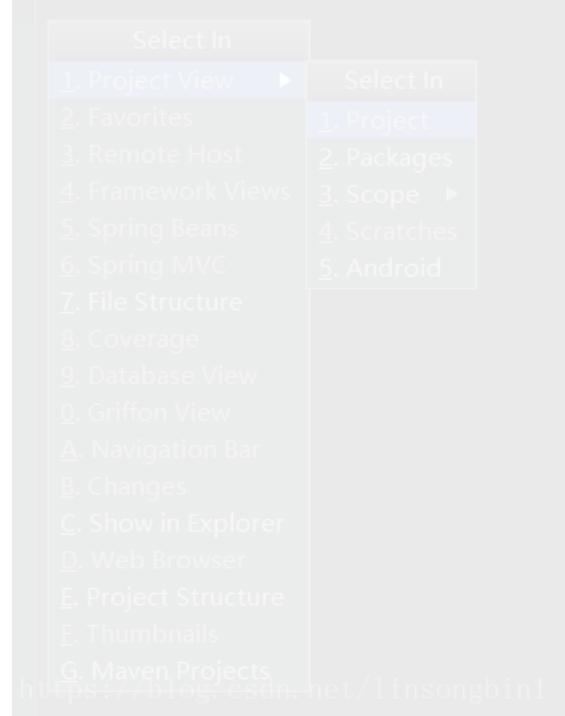
如果想让这个临时的导航栏消失的话，直接使用esc快捷键即可。

当工程里的包和类非常多的时候，有时候我们想知道当前类在project视图里是处在哪个位置。



上面图中的DemoIDEA里，你如何知道它是在spring-cloud-config工程里的哪个位置呢？

可以先使用alt+F1，弹出Select in视图，然后选择Project View中的Project，回车，就可以立刻定位到类的位置了。



那如何从project跳回代码里呢？可以直接使用esc退出project视图，或者直接使用F4,跳到代码里。

如果你依稀记得某个方法名字几个字母，想在IDEA里面找出来，可以怎么做呢？

直接使用ctrl+shift+alt+n，使用symbol来查找即可。

比如说：

```
package spring.cloud.config;

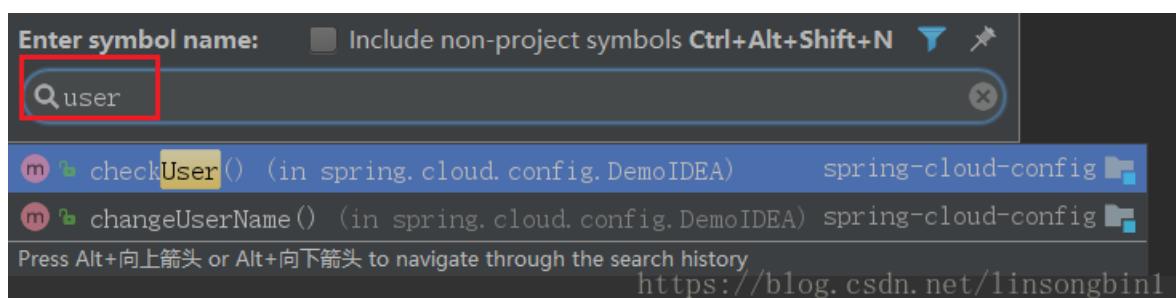
import org.springframework.stereotype.Service;

@Service
public class DemoIDEA {

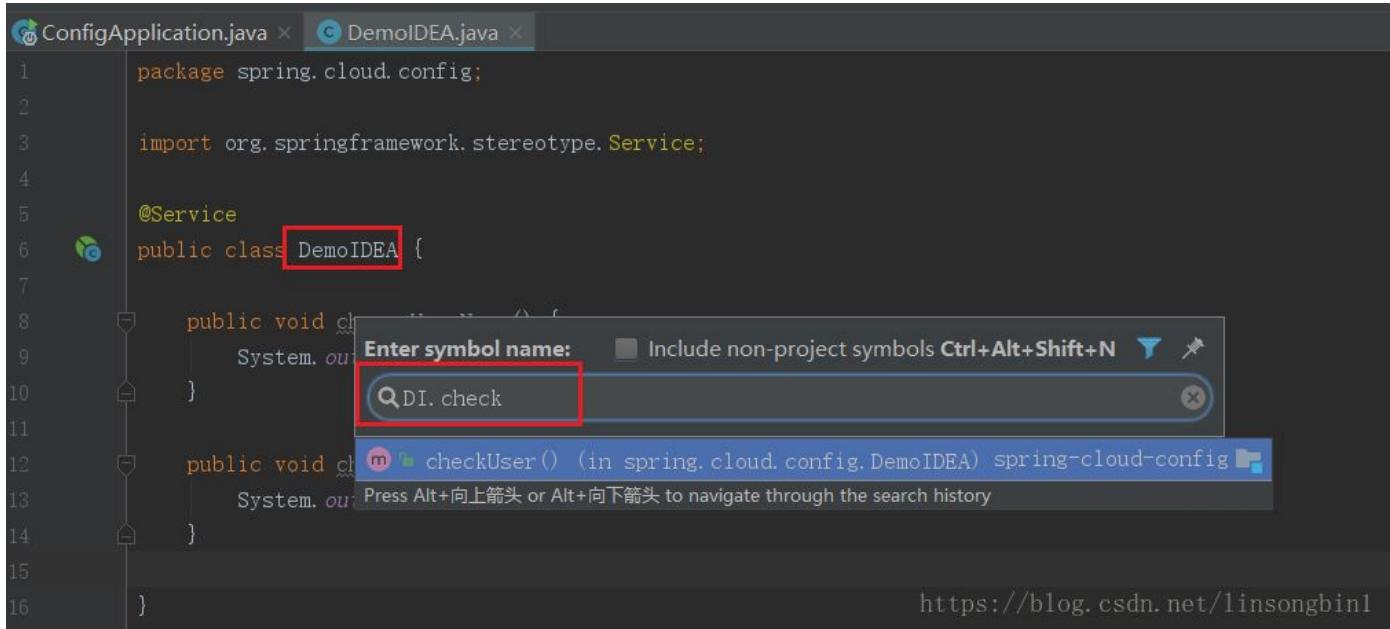
    public void changeUserName() {
        System.out.println("change");
    }

    public void checkUser() {
        System.out.println("change");
    }
}
```

你想找到checkUser方法。直接输入user即可。

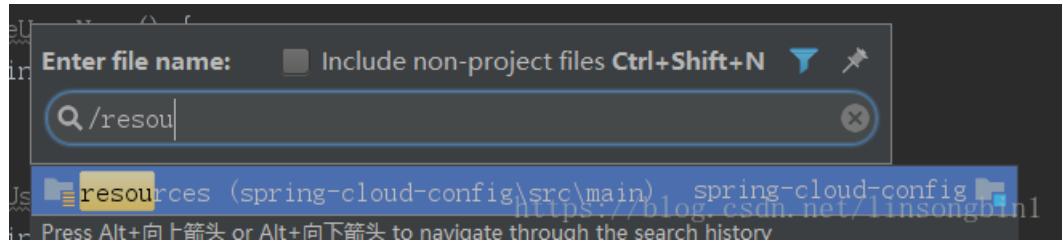


如果你记得某个业务类里面有某个方法，那也可以使用首字母找到类，然后加个.，再输入方法名字也是可以的。

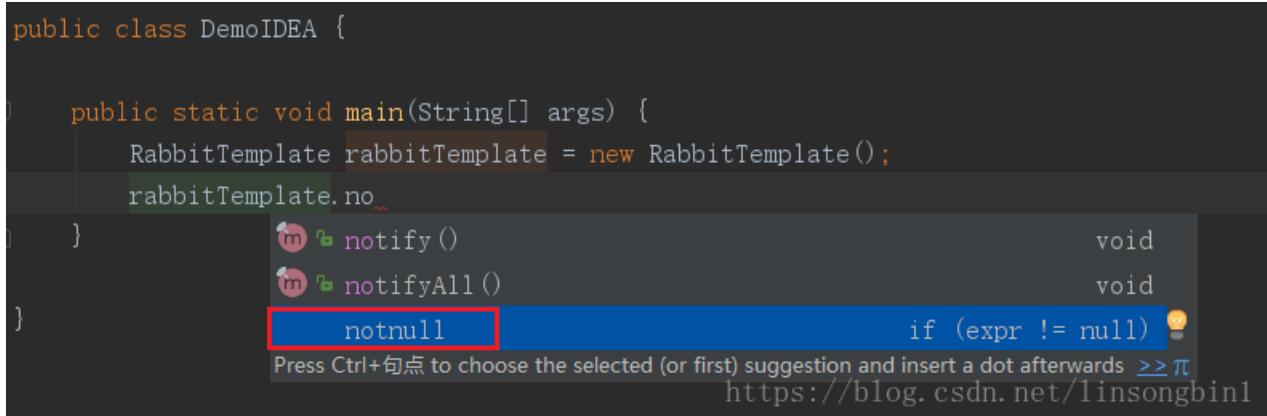


The screenshot shows an IDE interface with two tabs open: 'ConfigApplication.java' and 'DemoIDEA.java'. In the code editor, the cursor is positioned inside a class definition. A search bar at the top says 'Enter symbol name:' with the placeholder 'DI. check'. Below the search bar, a tooltip displays the method 'checkUser()' from the 'DemoIDEA' class, with the text 'Press Alt+向上箭头 or Alt+向下箭头 to navigate through the search history'. The URL 'https://blog.csdn.net/linsongbin1' is visible in the bottom right corner.

使用ctrl+shift+n后，使用/，然后输入目录名字即可。



自动生成not null这种if判断，在IDEA里有很多种办法，其中一种办法你可能没想到。



The screenshot shows an IDE interface with a code editor containing a 'main' method. The cursor is at the end of a line where 'rabbitTemplate' is used. A tooltip appears with several suggestions: 'notify()', 'notifyAll()', and 'notnull'. The 'notnull' suggestion is highlighted with a red box. Below the suggestions, a note says 'Press Ctrl+句点 to choose the selected (or first) suggestion and insert a dot afterwards >> π'. The URL 'https://blog.csdn.net/linsongbin1' is visible in the bottom right corner.

当我们使用rabbitTemplate. 后，直接输入notnull并回车，IDEA就好自动生成if判断了。



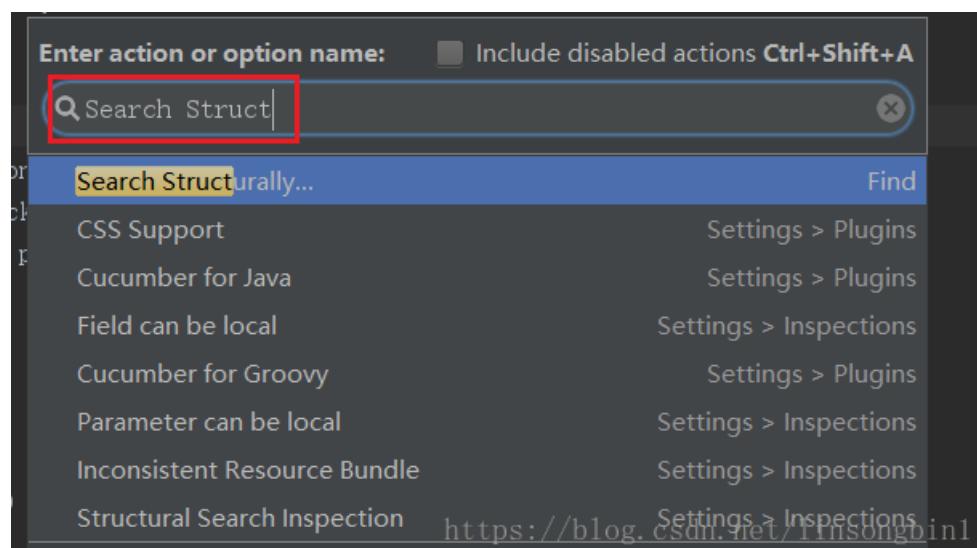
The screenshot shows an IDE interface with a code editor containing a 'main' method. The code includes a 'rabbitTemplate' variable and an 'if' statement: 'if (rabbitTemplate != null) { }'. A red box highlights the entire 'if' block. The URL 'https://blog.csdn.net/linsongbin1' is visible in the bottom right corner.

这个也是我非常喜欢的一个功能，可以根据模板来找到与模板匹配的代码块。比如说：

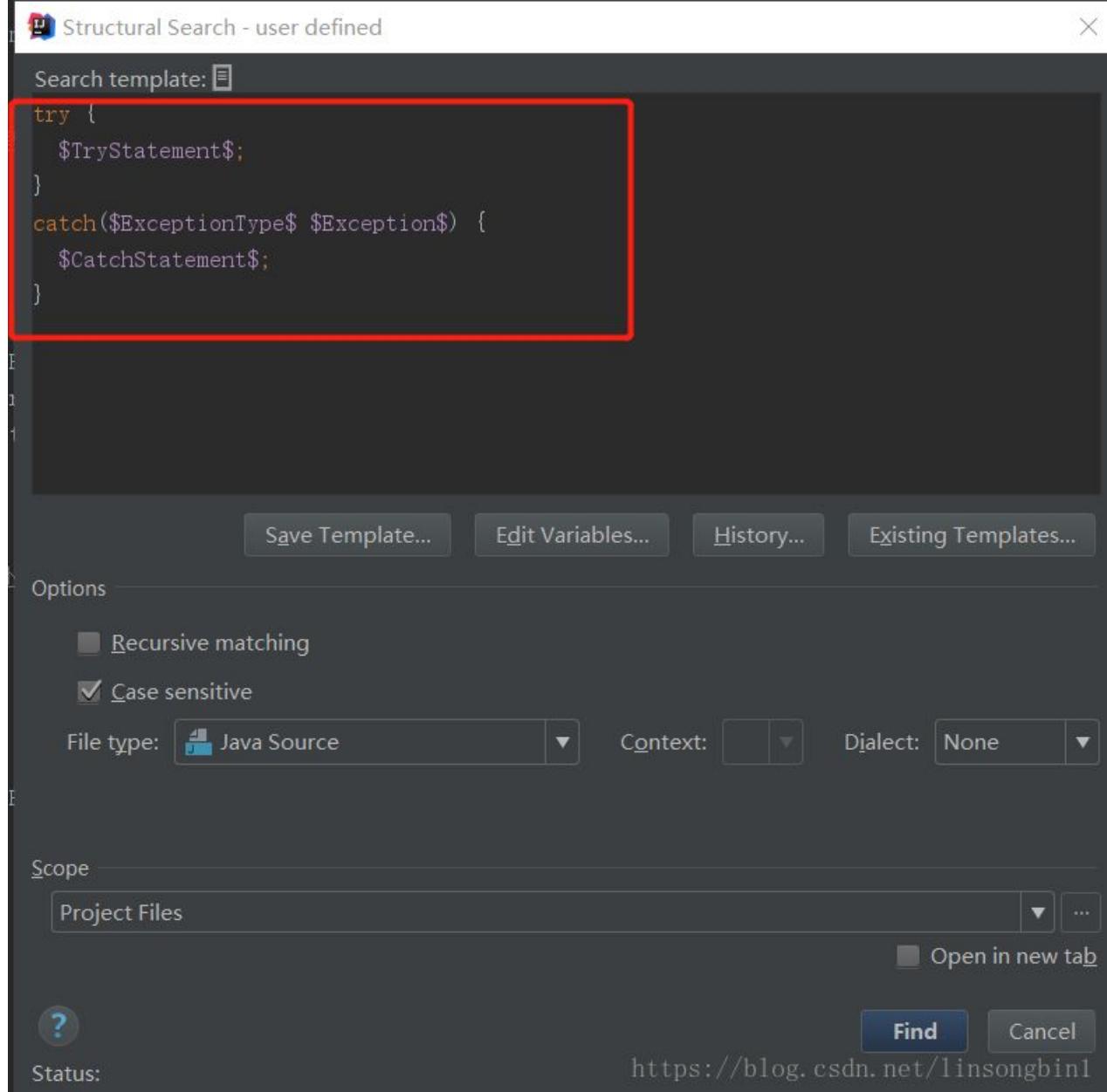
catch语句里没有处理异常,是极其危险的。我们可以IDEA里面方便找到所有这样的代码。

```
public class DemoIDEA {  
  
    public static void main(String[] args) {  
        //有处理异常  
        try {  
            if (1 == 1) {  
  
            }  
        }  
        catch (Exception e) {  
            e.printStackTrace();  
            System.out.println("error");  
        }  
  
        //没有处理异常  
        try {  
            if (1 == 1) {  
  
            }  
        }  
        catch (Exception e) {  
  
        }  
    }  
}  
  
https://blog.csdn.net/linsongbin1
```

首先使用ctrl+shift+A快捷键弹出action框,然后输入Search Struct



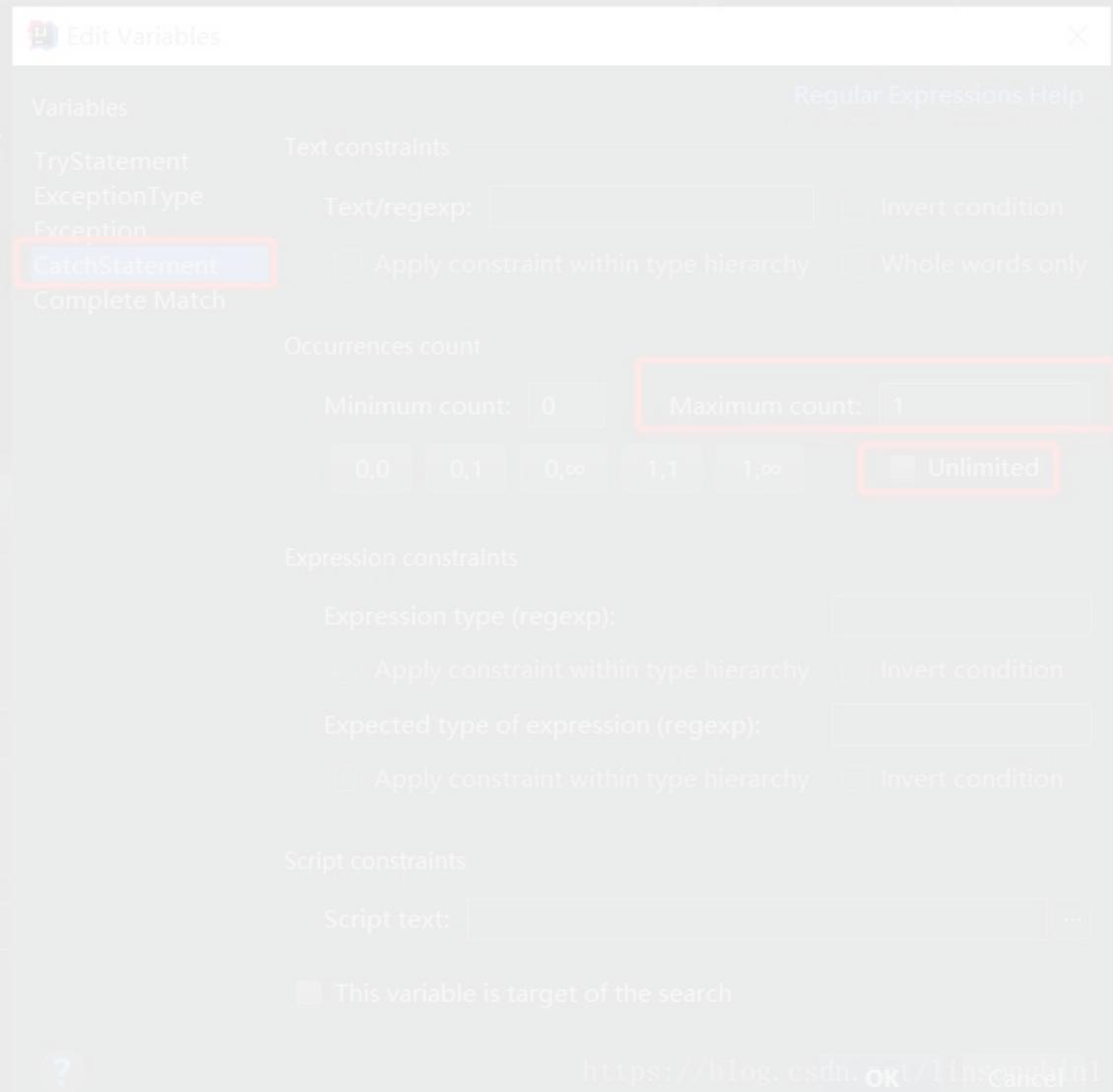
选择Search Structurally后,回车,跳转到模板视图。



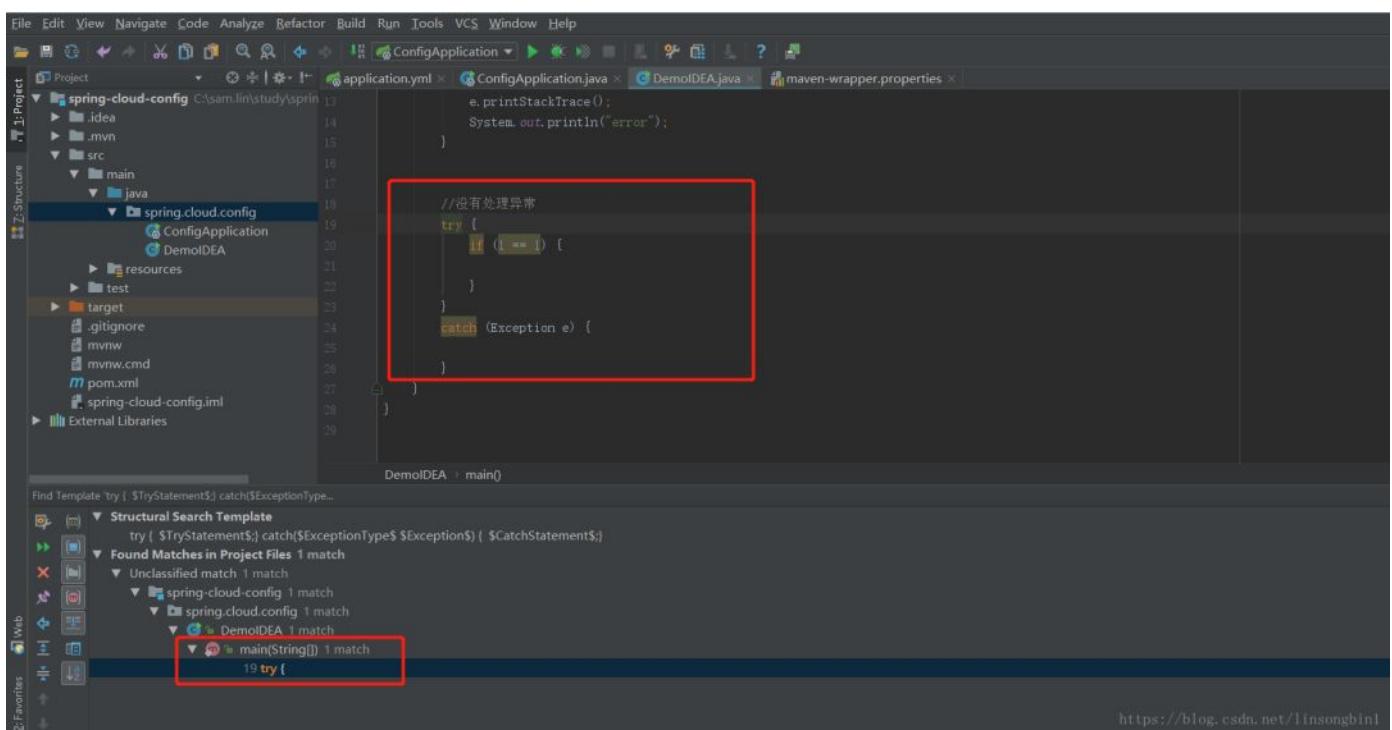
点击Existing

Templates按钮，选择try模板。为了能找出catch里面没有处理异常的代码块，我们需要配置一下CatchStatement的Maximum count的值，将其设置为1。

点击Edit Variables按钮，在界面修改Maximum count的值。



最后点击find按钮，就可以找出catch里面没有处理异常的代码了。



- END -

如果看到这里，说明你喜欢这篇文章，请[转发、点赞](#)。微信搜索「web_resource」，关注后回复「进群」或者扫描下方二维码即可进入无广告交流群。

↓ 扫描二维码进群 ↓



推荐阅读

1. 从零搭建创业公司后台技术栈
2. 如何阅读 Java 源码？
3. 某小公司RESTful、前后端分离的实践
4. 该如何弥补 GitHub 功能缺陷？
5. 团队开发中 Git 最佳实践



喜欢文章, 点个在看

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

效率 Max！IDEA 会飞？只因我装了这 12 个插件

Java后端 2019-10-04

点击上方 Java后端, 选择 [设为星标](#)

优质文章, 及时送达

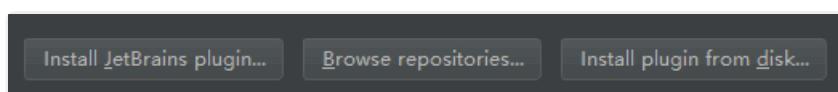
[上一篇:彻底理解 Cookie, Session, Token](#)

今天介绍一下IDEA的一些炫酷的插件，IDEA强大的插件库不仅能给我们带来一些开发的便捷，还能体现我们的与众不同。有了插件的辅佐，开发效率会大大提升，这就是为什么我的 IDEA 会飞！

1. 插件的安装

打开setting文件选择Plugins选项

- Ctrl + Alt + S
- File -> Setting

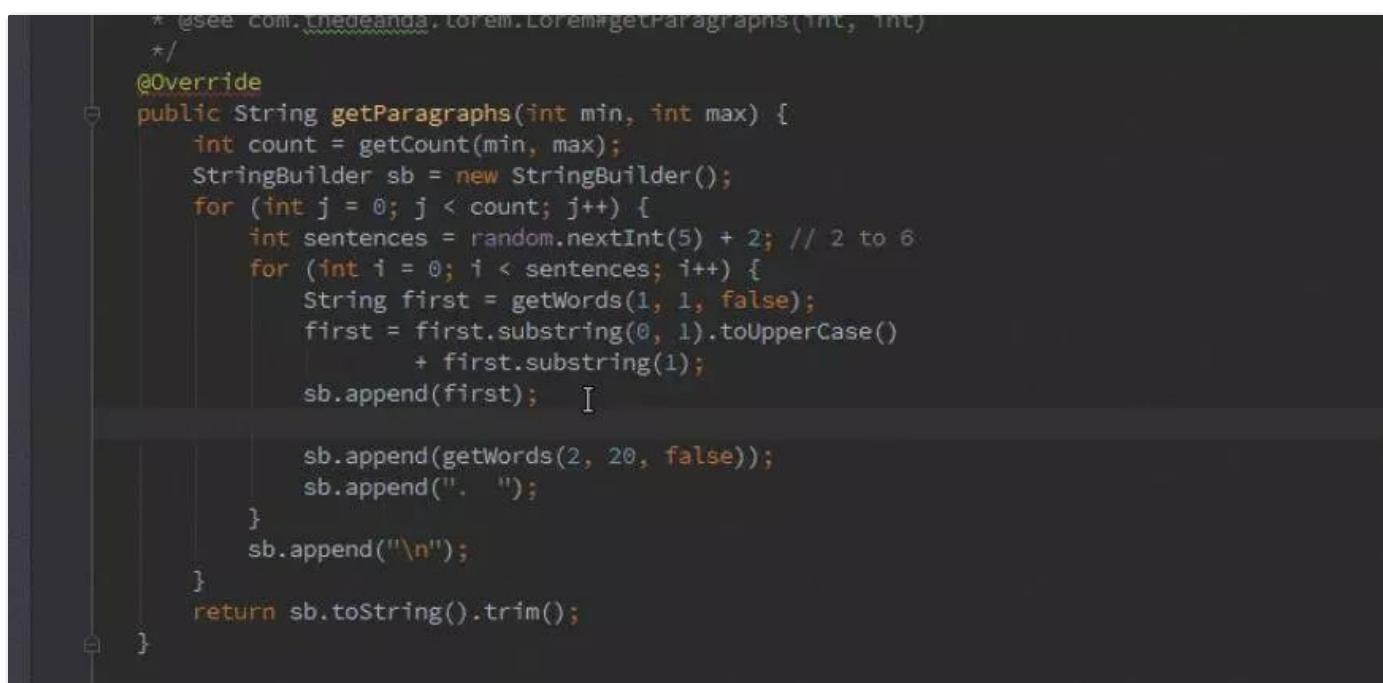


分别是安装JetBrains插件，第三方插件，本地已下载的插件包。详情见往期关于settings的文章。

2. 各种插件

1. activate-power-mode 和 Power mode II

根据Atom的插件activate-power-mode的效果移植到IDEA上



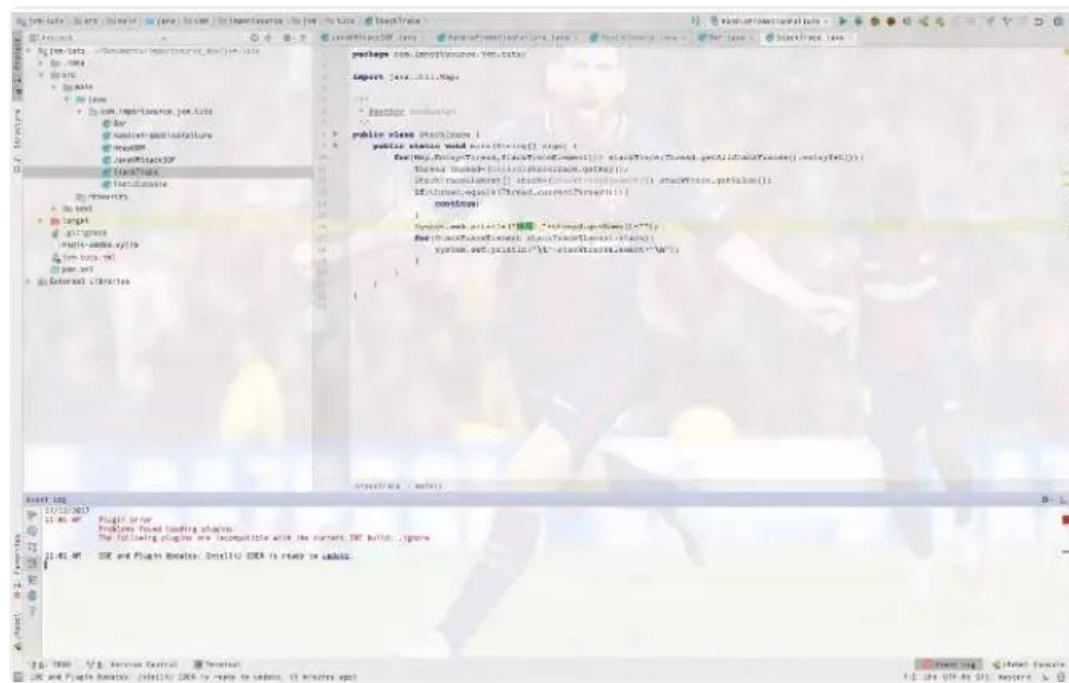
```
* @see com.thedeanda.lorem.Lorem#getParagraphs(int, int)
*/
@Override
public String getParagraphs(int min, int max) {
    int count = getCount(min, max);
    StringBuilder sb = new StringBuilder();
    for (int j = 0; j < count; j++) {
        int sentences = random.nextInt(5) + 2; // 2 to 6
        for (int i = 0; i < sentences; i++) {
            String first = getWords(1, 1, false);
            first = first.substring(0, 1).toUpperCase()
                + first.substring(1);
            sb.append(first); I

            sb.append(getWords(2, 20, false));
            sb.append(". ");
        }
        sb.append("\n");
    }
    return sb.toString().trim();
}
```

写代码是整个屏幕都在抖动，activate-power-mode是白的的，Power mode II色彩更酷炫点。欢迎关注公众号 Java后端 获取更多推送。

2. Background Image Plus

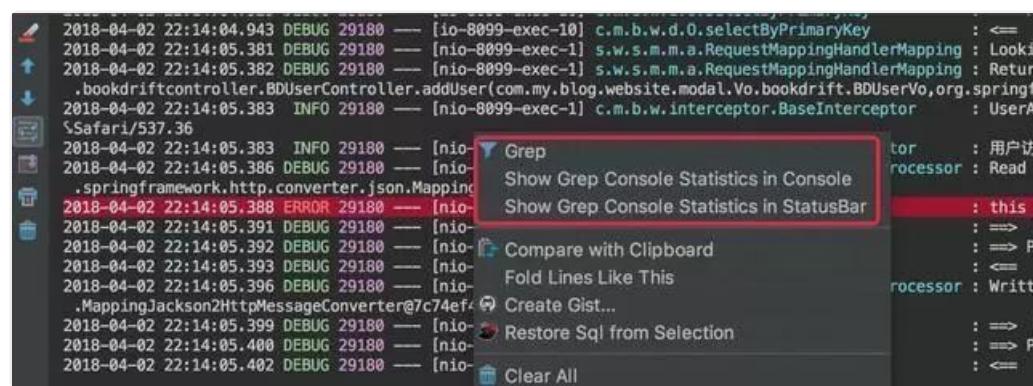
idea背景修改插件，让你的idea与众不同，可以设置自己喜欢的图片作为code背景。



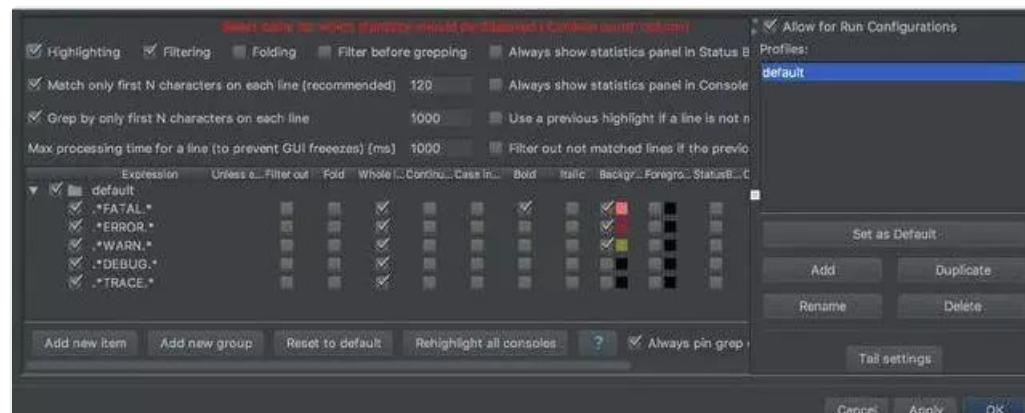
安装成功之后重启，菜单栏的View标签>点击Set Background Image(没安装插件是没有这个标签的)，在弹框中路由选择到本地图片，点击OK即可。

3. Grep console

自定义日志颜色，idea控制台可以彩色显示各种级别的log，安装完成后，在console中右键就能打开。



并且可以设置不同的日志级别的显示样式。



可以直接根据关键字搜索你想要的，搜索条件是支持正则表达式的。

4.Free Mybatis plugin

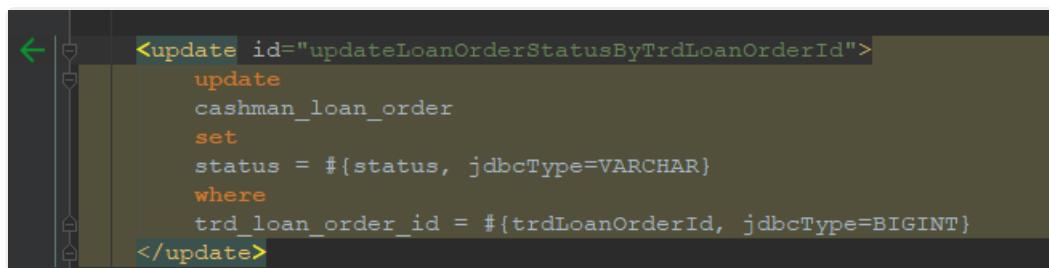
mybatis 插件，让你的mybatis.xml像java代码一样编辑。我们开发中使用mybatis时时长需要通过mapper接口查找对应的xml中的sql语句，该插件方便了我们的操作。欢迎关注公众号 Java后端 获取更多推送。

安装完成重启IDEA之后，我们会看到code左侧或多出一列绿色的箭头，点击箭头我们就可以直接定位到xml相应文件的位置。

mapper

```
→ public interface LoanOrderMapper {  
→     int insert(LoanOrder record);  
→     LoanOrder selectByPrimaryKey(Long id);  
→     LoanOrder selectByTrdOrderId(@Param("trdLoanOrderId") Long trdLoanOrderId);
```

xml



```
<update id="updateLoanOrderStatusByTrdLoanOrderId">  
    update  
    cashman_loan_order  
    set  
    status = #{status, jdbcType=VARCHAR}  
    where  
    trd_loan_order_id = #{trdLoanOrderId, jdbcType=BIGINT}  
</update>
```

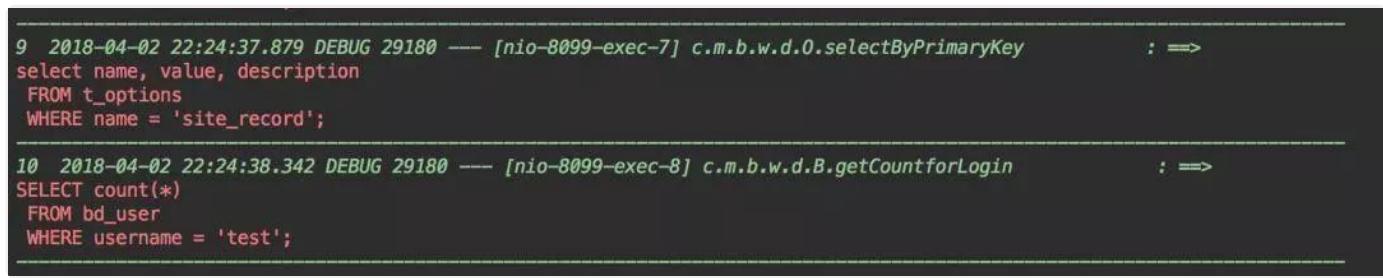
5.MyBatis Log Plugin

Mybatis现在是java中操作数据库的首选，在开发的时候，我们都会把Mybatis的脚本直接输出在console中，但是默认的情况下，输出的脚本不是一个可以直接执行的。

```
c.m.b.w.d.0.selectByPrimaryKey : ==> Preparing: select name, value, description from t_options where name = ?  
c.m.b.w.d.0.selectByPrimaryKey : ==> Parameters: site_record(String)  
c.m.b.w.d.0.selectByPrimaryKey : <== Total: 1
```

如果我们想直接执行，还需要在手动转化一下。欢迎关注公众号 Java后端 获取更多推送。

MyBatis Log Plugin 这款插件是直接将Mybatis执行的sql脚本显示出来，无需处理，可以直接复制出来执行的，如图：

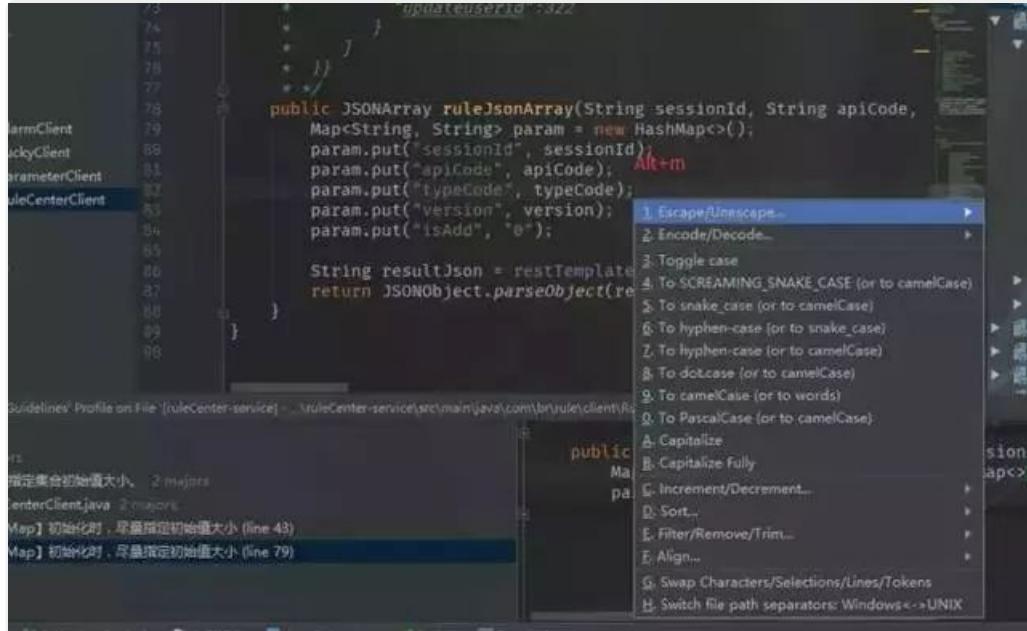


```
9 2018-04-02 22:24:37.879 DEBUG 29180 --- [nio-8099-exec-7] c.m.b.w.d.0.selectByPrimaryKey : ==>  
select name, value, description  
FROM t_options  
WHERE name = 'site_record';  
  
10 2018-04-02 22:24:38.342 DEBUG 29180 --- [nio-8099-exec-8] c.m.b.w.d.B.getCountforLogin : ==>  
SELECT count(*)  
FROM bd_user  
WHERE username = 'test';
```

执行程序后，我们可以很清晰的看到我们执行了哪些sql脚本，而且脚本可以执行拿出来运行。

6.String Manipulation

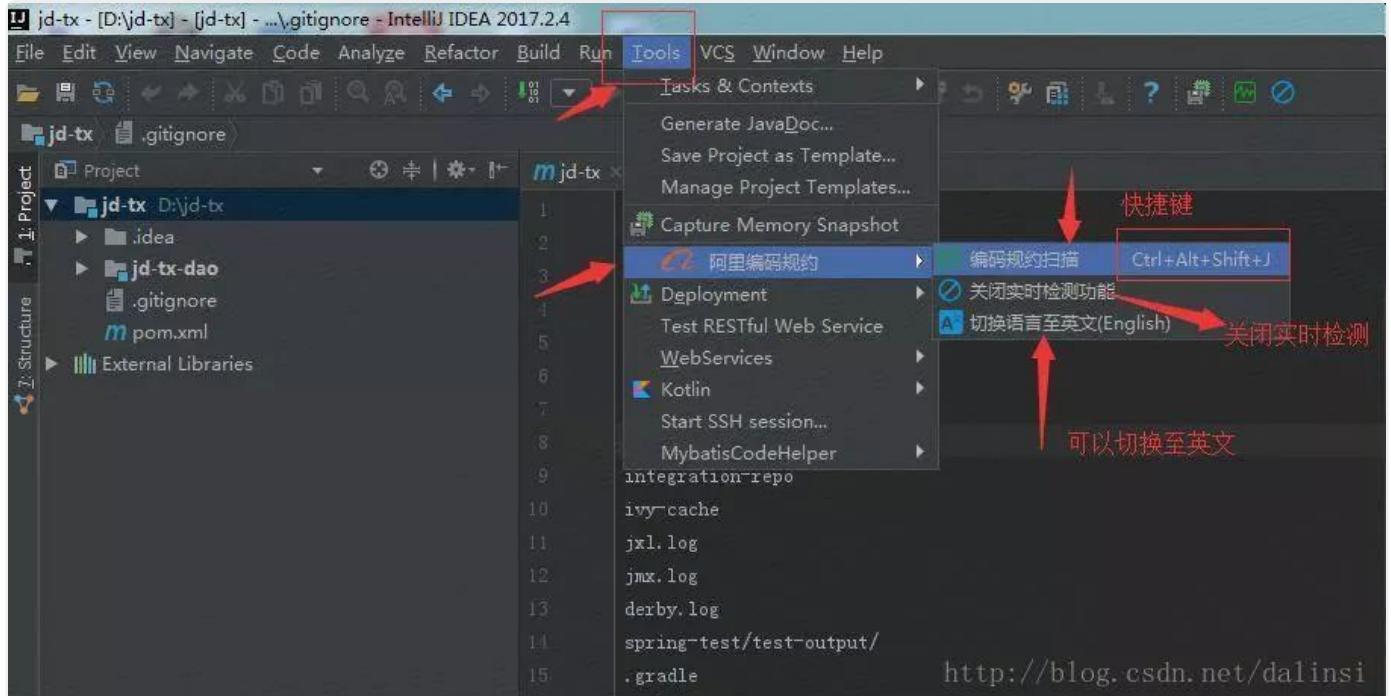
强大的字符串转换工具。使用快捷键，Alt+m。



- 切换样式 (camelCase, hyphen-lowercase, HYPHEN-UPPERCASE, snakecase, SCREAMINGSNAKE_CASE, dot.case, words lowercase, Words Capitalized, PascalCase)
- 转换为SCREAMINGSNAKECASE (或转换为camelCase)
- 转换为 snake_case (或转换为camelCase)
- 转换为dot.case (或转换为camelCase)
- 转换为hyphen-case (或转换为camelCase)
- 转换为hyphen-case (或转换为snake_case)
- 转换为camelCase (或转换为Words)
- 转换为camelCase (或转换为lowercase words)
- 转换为PascalCase (或转换为camelCase)
- 选定文本大写
- 样式反转

7. Alibaba Java Coding Guidelines

阿里巴巴代码规范检查插件，当然规范可以参考《阿里巴巴Java开发手册》。



8. Lombok

Java语言，每次写实体类的时候都需要写一大堆的setter, getter，如果bean中的属性一旦有修改、删除或增加时，需要重新生成或删除get/set等方法，给代码维护增加负担，这也是Java被诟病的一种原因。Lombok则为我们解决了这些问题，使用了lombok的注解(@Setter,@Getter,@ToString,@@RequiredArgsConstructor,@EqualsAndHashCode或@Data)之后，就不需要编写或生成get/set等方法，很大程度上减少了代码量，而且减少了代码维护的负担。欢迎关注公众号 Java后端 获取更多推送。

安装完成之后，在应用Lombok的时候注意别忘了需要添加依，maven为例：

```
1 <dependency>
2   <groupId>org.projectlombok</groupId>
3   <artifactId>lombok</artifactId>
4 </dependency>
```

```
1 @Setter
2 @Getter
3 @ToString
4 @EqualsAndHashCode
5 public class People {
6     private String name;
7     private int age;
8     private String male;
9 }
```

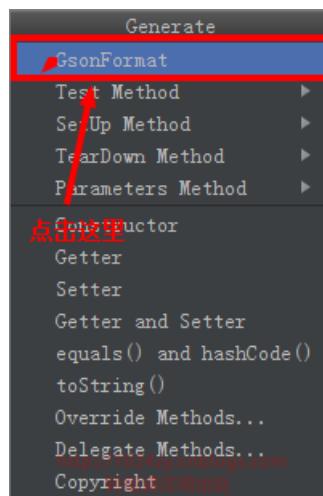
9. Key promoter

Key promoter 是IntelliJ IDEA的快捷键提示插件，会统计你鼠标点击某个功能的次数，提示你应该用什么快捷键，帮助记忆快捷键，等熟悉了之后可以关闭掉这个插件。

10.Gsonformat

可根据json数据快速生成java实体类。

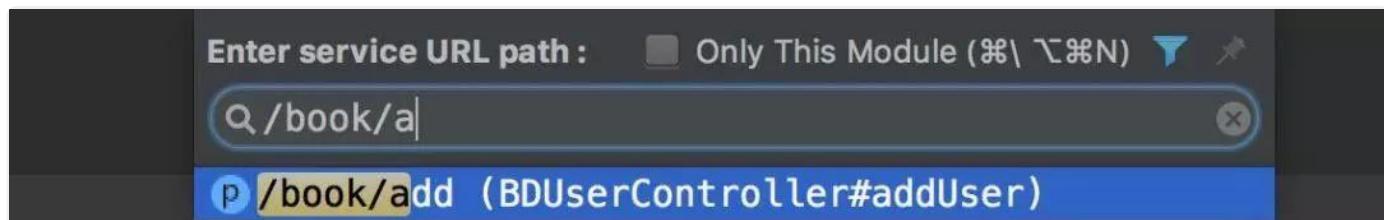
自定义个javaBean(无任何内容，就一个空的类)，复制你要解析的Json，然后alt+insert弹出如下界面或者使用快捷键 Alt+S，在里面粘贴刚刚复制的Json，点击OK即可。



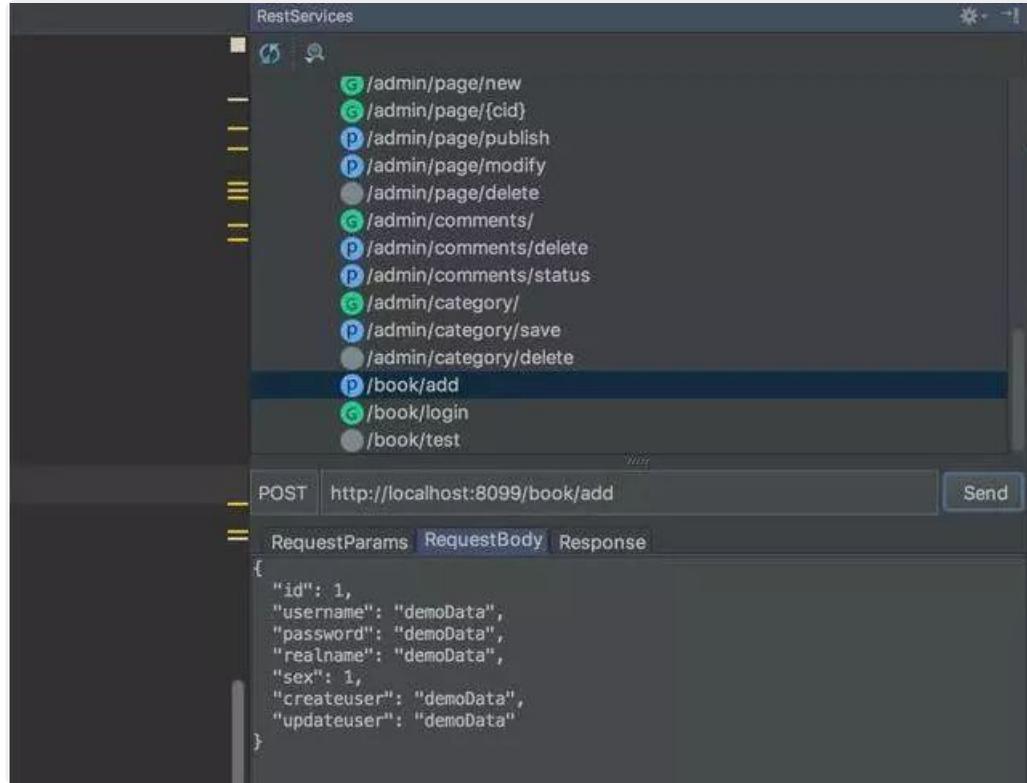
11.Restful toolkit

Spring MVC网页开发的时候，我们都是通过requestmapping的方式来定义页面的URL地址的，为了找到这个地址我们一般都是cmd+shift+F的方式进行查找，大家都知道，我们URL的命名一个是类requestmapping+方法requestmapping，查找的时候还是有那么一点不方便的，restful toolkit就能很方便的帮忙进行查找。欢迎关注公众号 Java后端 获取更多推送。

例如：我要找到/user/add 对应的controller,那么只要Ctrl+斜杠，（图片来自于网络）



就能直接定位到我们想要的controller。这个也是真心方便，当然restful toolkit还为我们提供的其他的功能。根据我们的controller帮我们生成默认的测试数据，还能直接调用测试，这个可以是解决了我们每次postman调试数据时，自己傻傻的组装数据的操作，这个更加清晰，比在console找数据包要方便多了。欢迎关注公众号 Java后端 获取更多推送。



12. JRebel

JRebel是一种热部署生产力工具，修改代码后不用重新启动程序，所有的更改便可以生效。它跳过了Java开发中常见的重建、重新启动和重新部署周期。

更多插件推荐

插件名称	插件介绍	官网地址
Gitee	开源中国的码云插件	https://plugins.jetbrains.com/plugin/8383-gitee
Alibaba Java Coding Guidelines	阿里巴巴出的代码规范检查插件	https://plugins.jetbrains.com/plugin/10046-alibaba-java-coding-guidelines
IDE Features Trainer	IntelliJ IDEA 官方出的学习辅助插件	https://plugins.jetbrains.com/plugin/8554-pr-idea
Key promoter	快捷键提示	https://plugins.jetbrains.com/plugin/4455-pr-idea
Grep Console	自定义设置控制台输出颜色	https://plugins.jetbrains.com/idea/plugin/7125-grep-console
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

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
Markdown Navigator	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
Translation	翻译插件	https://github.com/YiiGuxing/TranslationPlugin
Maven Helper	Maven 辅助插件	https://plugins.jetbrains.com/plugin/7179-maven-helper
Properties to YAML Converter	把 Properties 的配置格式改为 YAML 格式	https://plugins.jetbrains.com/plugin/8000-properties-to-yaml-converter
Git Flow Integration	Git Flow 的图形界面操作	https://plugins.jetbrains.com/plugin/7315-git-flow-integration
Rainbow Brackets	对各个对称括号进行着色，方便查看	https://github.com/izhangzhihao/intellij-rainbow-brackets
MybatisX	mybatis 框架辅助（免费）	https://plugins.jetbrains.com/plugin/10119-mybatisx
Lombok Plugin	Lombok 功能辅助插件	https://plugins.jetbrains.com/plugin/6317-lombok-plugin
.ignore	各类版本控制忽略文件生成工具	https://plugins.jetbrains.com/plugin/7495-ignore
mongo4idea	mongo客户端	https://github.com/dboissier/mongo4idea
iedis	redis客户端	https://plugins.jetbrains.com/plugin/9228-iedis
GenerateAllSetter	new POJO类的快速生成 set 方法	https://plugins.jetbrains.com/plugin/9360-generateallsetter

作者 | jajian

来源 | www.cnblogs.com/jajian/p/8081658.html

- END -

如果看到这里，说明你喜欢这篇文章，请[转发](#)、[点赞](#)。微信搜索「web_resource」，关注后回复「进群」或者扫描下方二维码即可进入无广告交流群。

↓ 扫描二维码进群 ↓



推荐阅读

1. Java后端优质文章整理
2. 彻底理解 Cookie, Session, Token
3. 这 26 条, 你赞同几个?
4. 7 个开源的 Spring Boot 前后端分离项目
5. 如何设计 API 接口, 实现统一格式返回?



Java后端

长按识别二维码, 关注我的公众号

喜欢文章, 点个在看 

阅读原文

声明: pdf仅供学习使用, 一切版权归原创公众号所有; 建议持续关注原创公众号获取最新文章, 学习愉快!

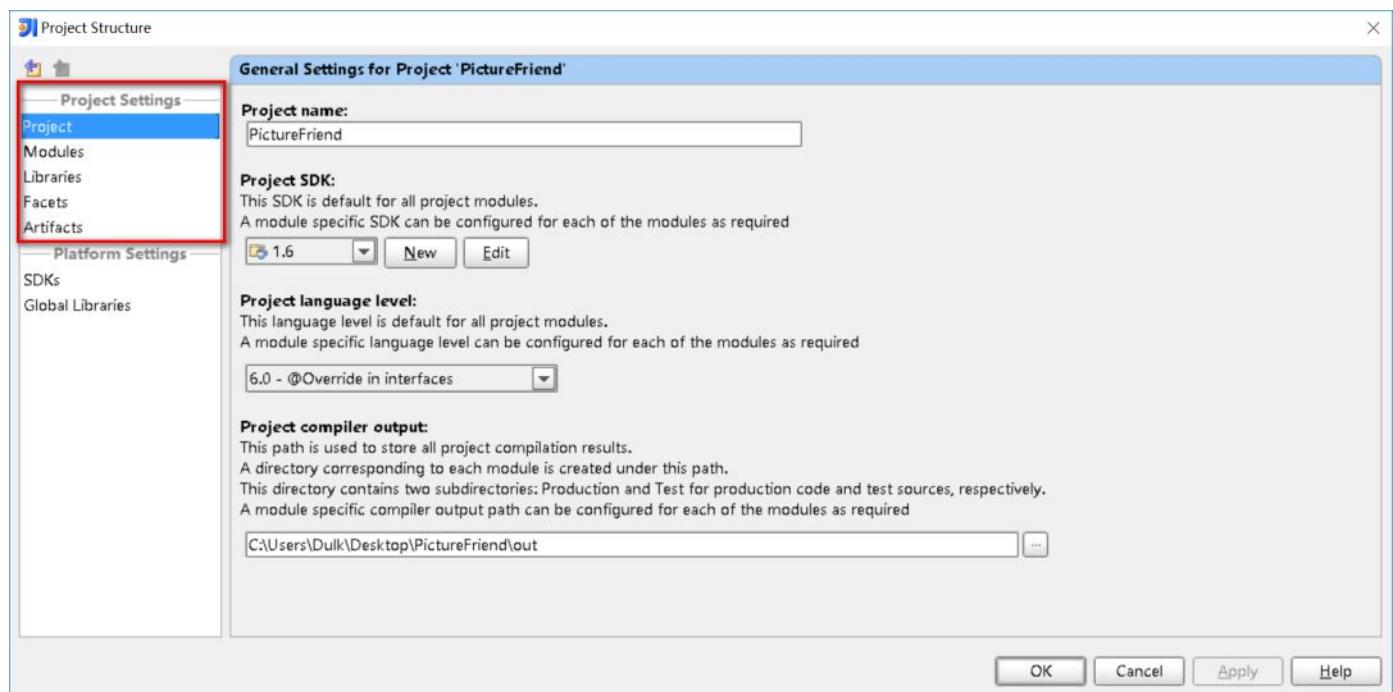
理解 IntelliJ IDEA 的项目配置和 Web 部署

Dulk Java后端 2019-10-23

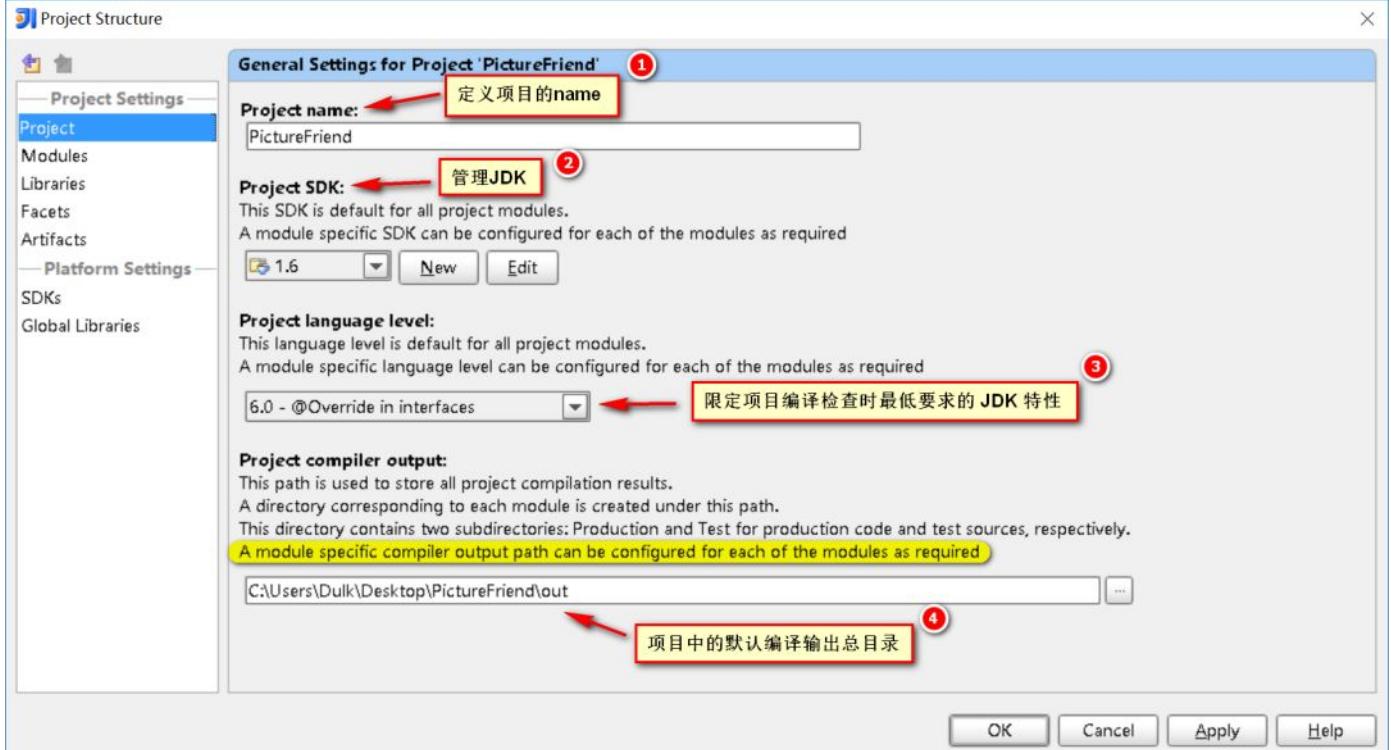


1、项目配置的理解

IDEA 中最重要的各种设置项，就是这个 Project Structure 了，关乎你的项目运行，缺胳膊少腿都不行。最近公司正好也是用之前自己比较熟悉的IDEA而不是Eclipse，为了更深入理解和使用，就找来各种资料再研究一下，这里整理后来个输出。



1.1 Project



1. Project name:

定义项目的名称；

2. Project SDK:

设置该项目使用的JDK，也可以在此处新添加其他版本的JDK；

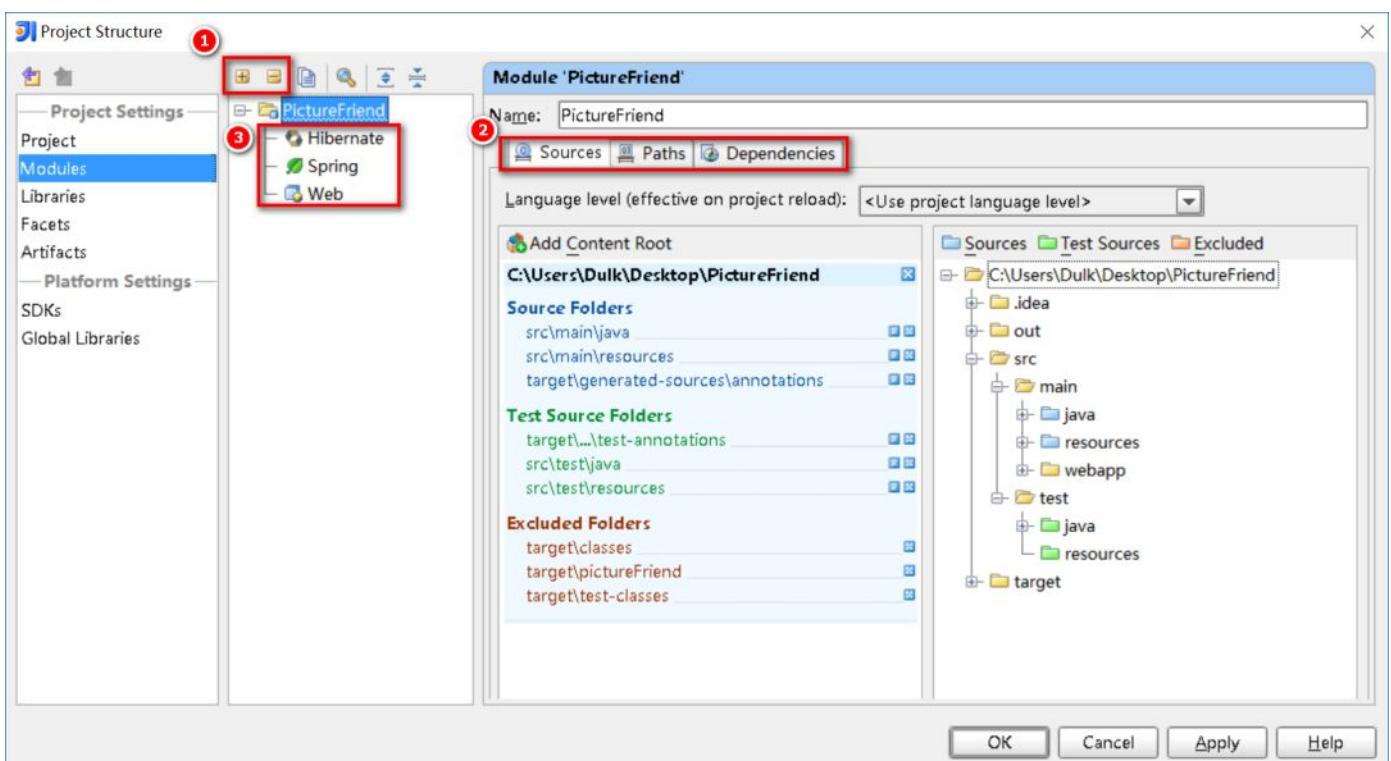
3. Project language level:

这个和JDK的类似，区别在于，假如你设置了JDK1.8，却只用到1.6的特性，那么这里可以设置语言等级为1.6，这个是限定项目编译检查时最低要求的JDK特性；

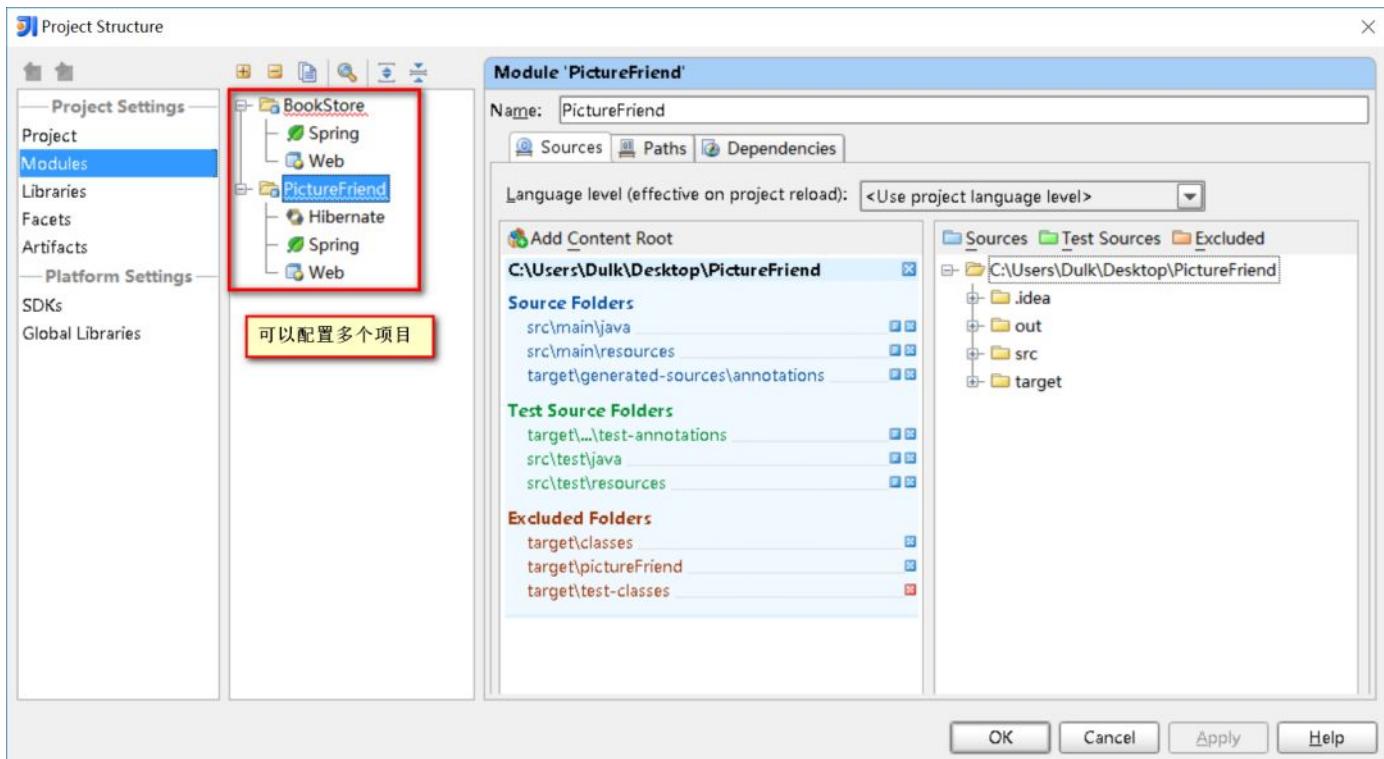
4. Project compiler output:

项目中的默认编译输出总目录，如图黄色部分，实际上每个模块可以自己设置特殊的输出目录 (Modules - (project) - Paths - Use module compile output path)，所以这个设置有点鸡肋。

1.2 Modules



1.2.1 增删子项目



一个项目中可以有多个子项目，每个子项目相当于一个模块。一般我们项目只是单独的一个，IntelliJ IDEA 默认也是单子项目的形式，所以只需要配置一个模块。

(此处的两个项目引入仅作示例参考)

1.2.2 子项目配置

每个子项目都对应了Sources、Paths、Dependencies 三大配置选项：

- **Sources:**

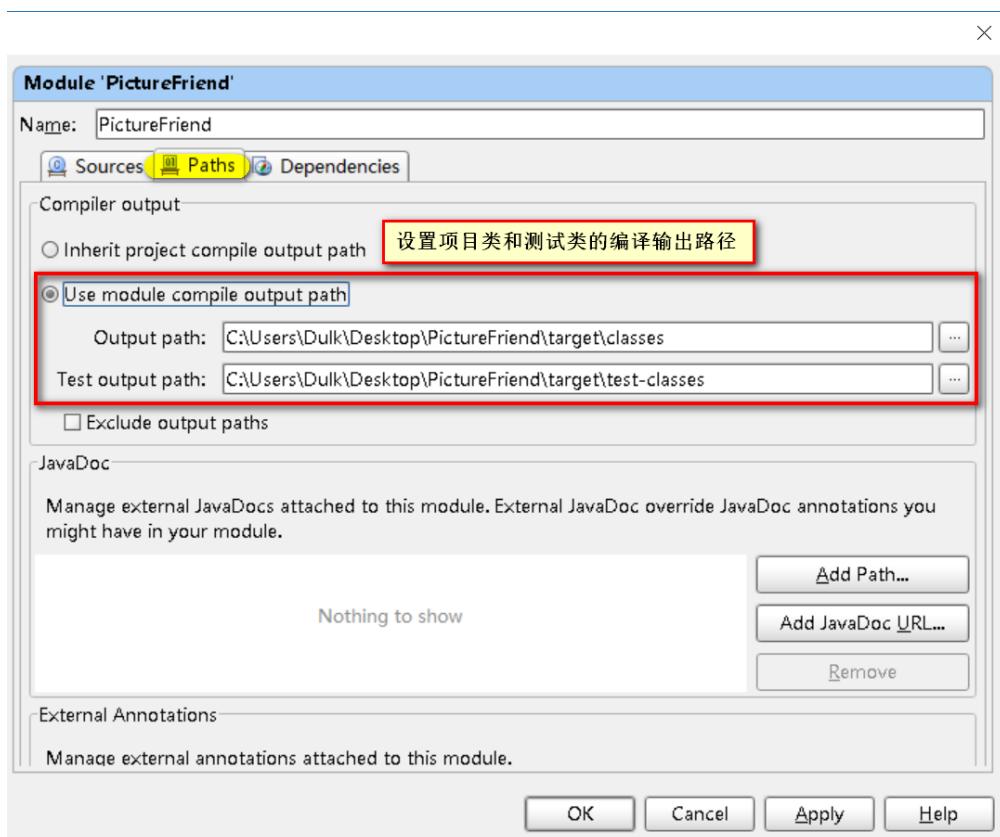
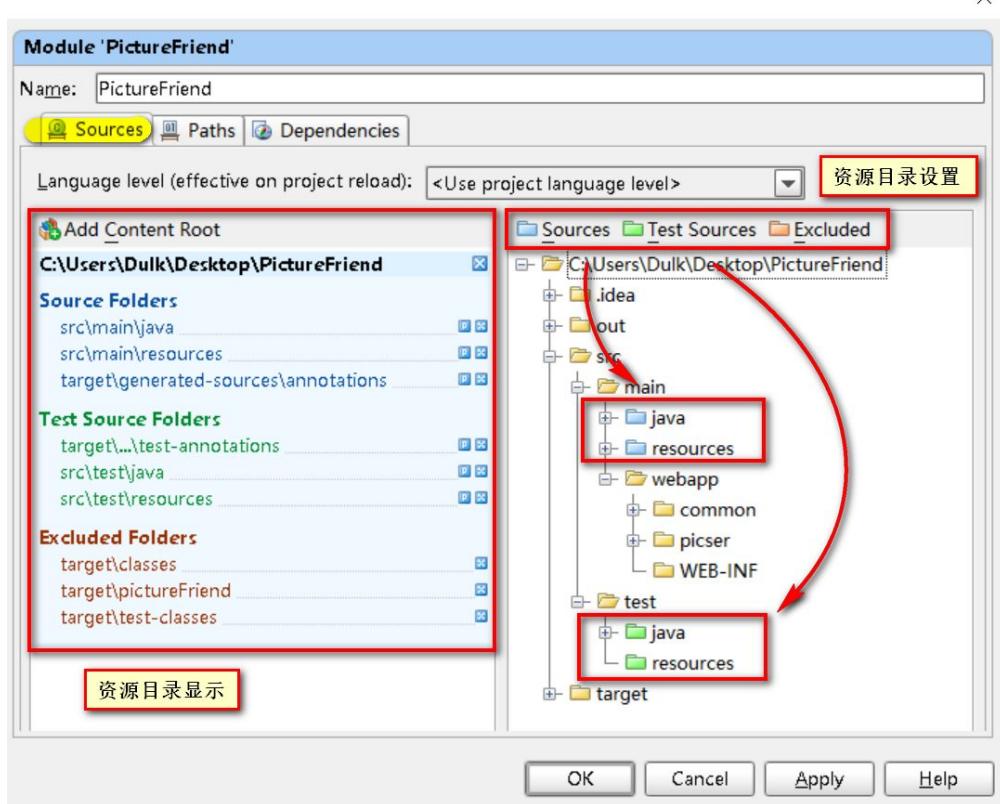
显示项目的目录资源，那些是项目部署的时候需要的目录，不同颜色代表不同的类型；

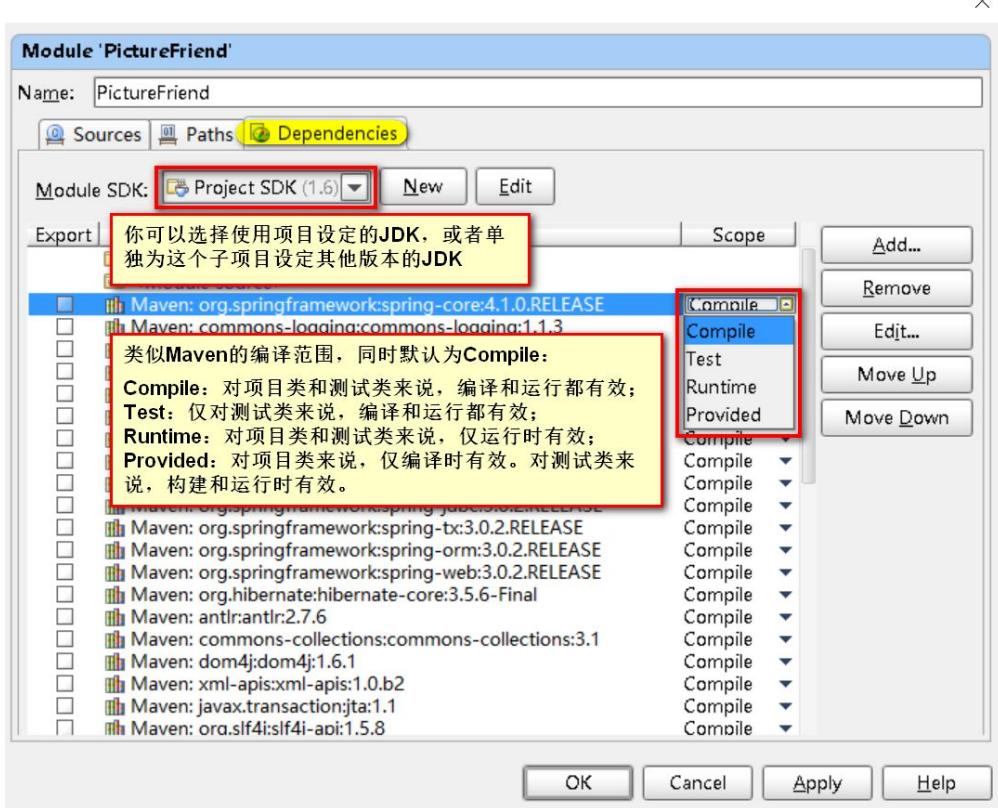
- **Paths:**

可以指定项目的编译输出目录，即项目类和测试类的编译输出地址（替换掉了Project的默认输出地址）

- **Dependencies:**

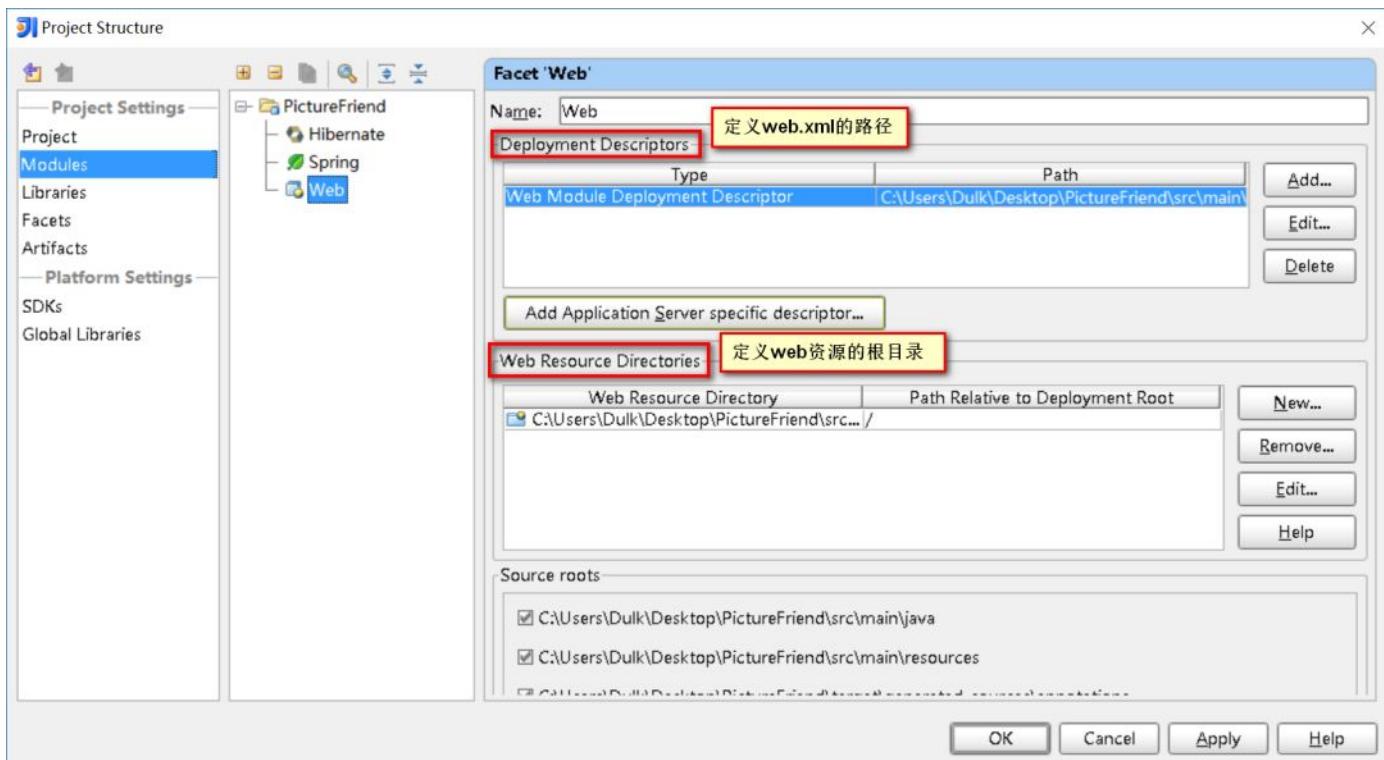
项目的依赖





1.2.3 增删框架 (Web部署-1)

每个子项目之下都可以定义它所使用的框架，这里重点说明一下Web部分的设置。



1.3 Libraries

这里可以显示所添加的jar包，同时也可以添加jar包，并且可以把多个jar放在一个组里面，类似于jar包整理。

这里默认将每个jar包做为了一个单独的组（未测试，待定）。

1.4 Facets

官方的解释是：

When you select a framework (a facet) in the element selector pane, the settings for the framework are shown in the right-hand part of the dialog.

(当你在左边选择面板点击某个技术框架，右边将会显示这个框架的一些设置)

说实话，并没有感觉到有什么作用。

1.5 Artifacts (Web部署-2)

项目的打包部署设置，这个是项目配置里面比较关键的地方，重点说一下。

先理解下它的含义，来看看官方定义的artifacts：

An artifact is an assembly of your project assets that you put together to test, deploy or distribute your software solution or its part. Examples are a collection of compiled Java classes or a Java application packaged in a Java archive, a Web application as a directory structure or a Web application archive, etc.

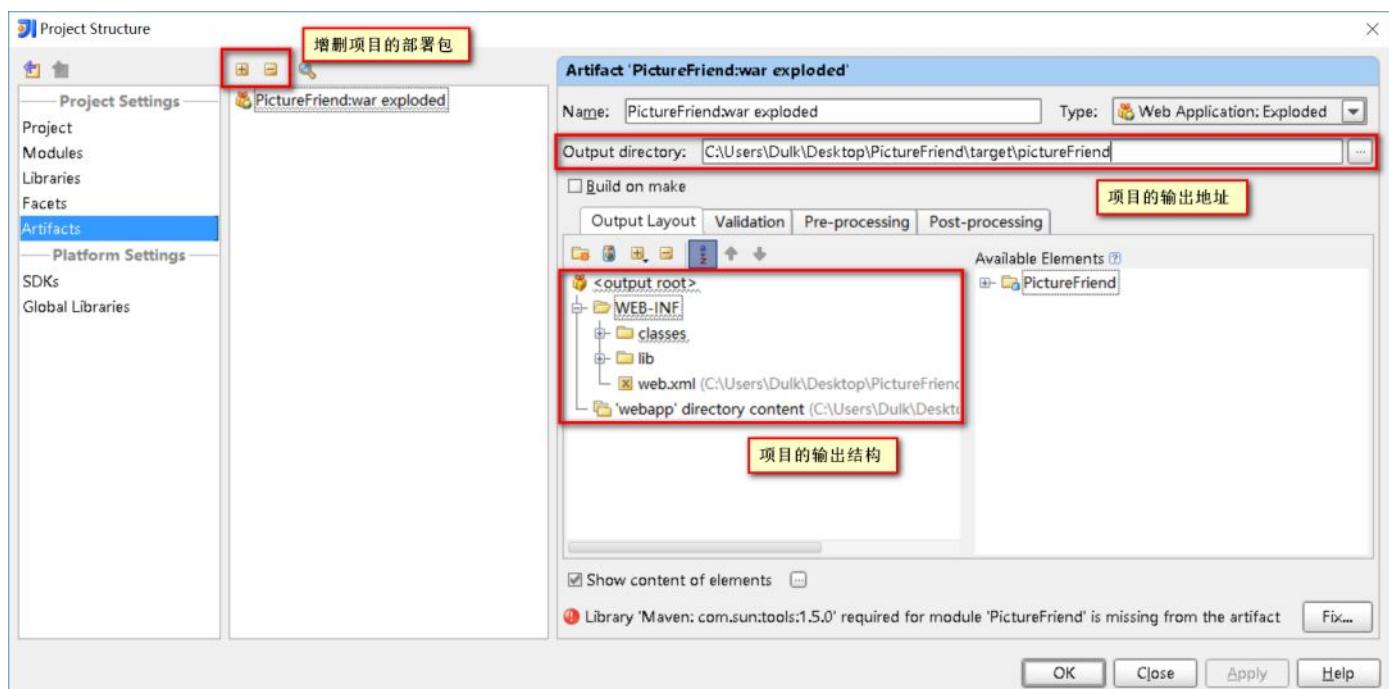
即编译后的Java类，Web资源等的整合，用以测试、部署等工作。再白话一点，就是说某个module要如何打包，例如war exploded、war、jar、ear等等这种打包形式。某个module有了Artifacts就可以部署到应用服务器中了。

(jar: Java ARchive, 通常用于聚合大量的Java类文件、相关的元数据和资源(文本、图片等)文件到一个文件,以便分发Java平台应用软件或库;

war: Web application ARchive, 一种JAR文件,其中包含用来分发的JSP、Java Servlet、Java类、XML文件、标签库、静态网页(HTML和相关文件),以及构成Web应用程序的其他资源;

exploded: 在这里你可以理解为展开，不压缩的意思。也就是war、jar等产出物没压缩前的目录结构。建议在开发的时候使用这种模式，便于修改了文件的效果立刻显现出来。)

默认情况下，IDEA的 Modules 和 Artifacts 的 output目录已经设置好了，不需要更改，打成war包的时候会自动在 WEB-INF目录下生成classes，然后把编译后的文件放进去。

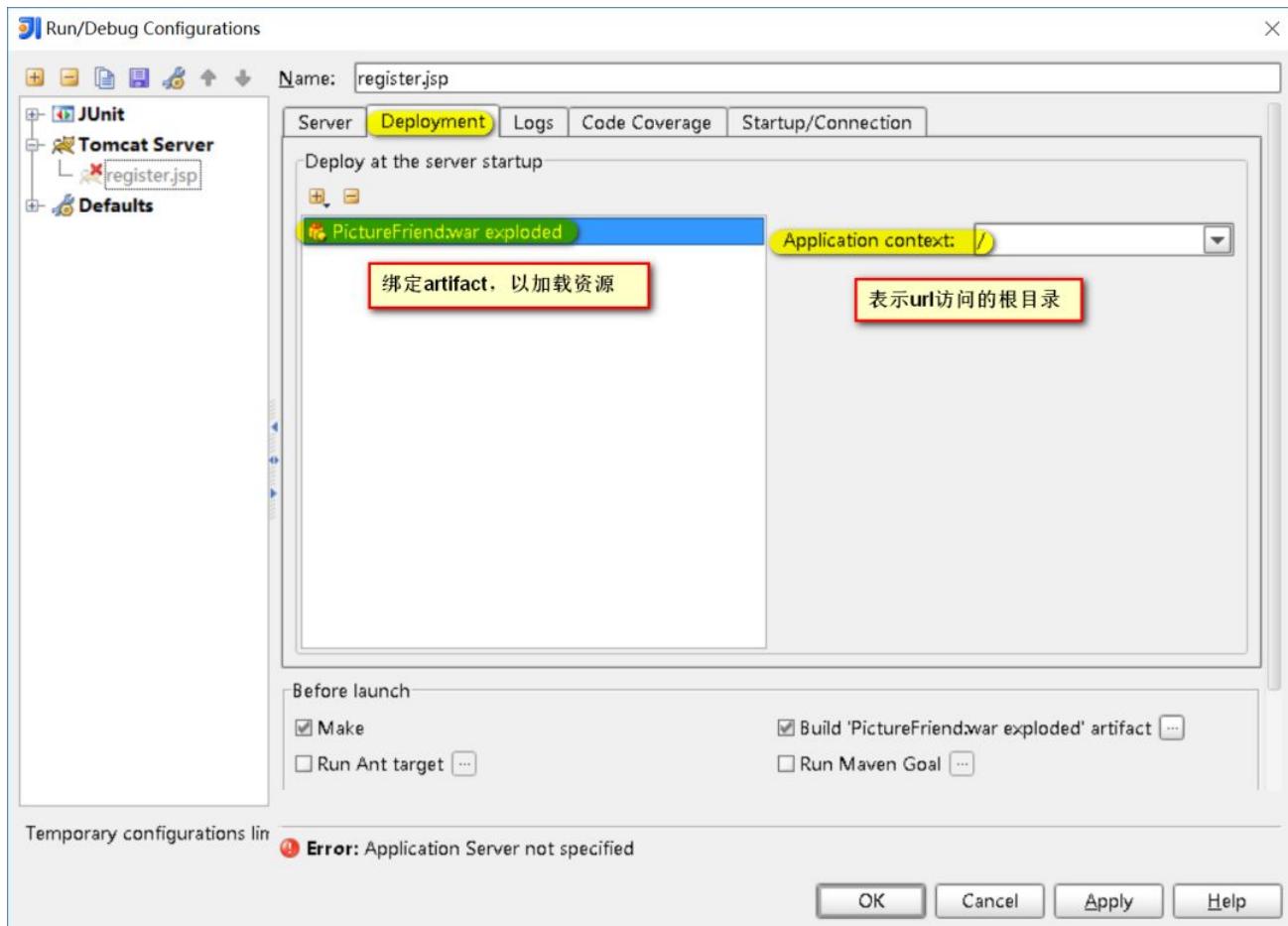


你可能对这里的输出目录不太理解，之前不是配置过了文件编译的输出目录了吗？为什么这里还有一个整合这些资源的目录呢？它又做了哪些事呢？

其实，实际上，当你点击运行tomcat时，默认就开始做以下事情：

- 编译，IDEA在保存/自动保存后不会做编译，不像Eclipse的保存即编译，因此在运行server前会做一次编译。
编译后class文件存放在指定的项目编译输出目录下（见1.2.2）；
- 根据artifact中的设定对目录结构进行创建；
- 拷贝web资源的根目录下的所有文件到artifact的目录下（见1.2.3）；
- 拷贝编译输出目录下的classes目录到artifact下的WEB-INF下（见1.2.2）；
- 拷贝lib目录下所需的jar包到artifact下的WEB_INF下；
- 运行server，运行成功后，如有需要，会自动打开浏览器访问指定url。

在这里还要注意的是，配置完成的artifact，需要在tomcat中进行添加：



2、参考链接

- [1]www.jetbrains.com/help/idea/2016.3/dependencies-tab.html?search=project%20structure
- [2]www.jetbrains.com/help/idea/2016.3/working-with-artifacts.html#artifact_def
- [3]www.cnblogs.com/52php/p/5677661.html
- [5]white-crucifix.iteye.com/blog/2070830
- [6]my.oschina.net/lujianing/blog/186737

阅读原文

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

试试 IDEA 解决 Maven 依赖冲突的高能神器！

桔子 Java后端 2019-12-13

点击上方 Java后端, 选择 [设为星标](#)

优质文章, 及时送达

作者 | 桔子

链接 | segmentfault.com/a/1190000017542396

1、何为依赖冲突

Maven是个很好用的依赖管理工具，但是再好的东西也不是完美的。Maven的依赖机制会导致Jar包的冲突。举个例子，现在你的项目中，使用了两个Jar包，分别是A和B。现在A需要依赖另一个Jar包C，B也需要依赖C。但是A依赖的C的版本是1.0，B依赖的C的版本是2.0。这时候，Maven会将这1.0的C和2.0的C都下载到你的项目中，这样你的项目中就存在了不同版本的C，这时Maven会依据依赖路径最短优先原则，来决定使用哪个版本的Jar包，而另一个无用的Jar包则未被使用，这就是所谓的依赖冲突。

在大多数时候，依赖冲突可能并不会对系统造成什么异常，因为Maven始终选择了一个Jar包来使用。但是，不排除在某些特定条件下，会出现类似找不到类的异常，所以，只要存在依赖冲突，在我看来，最好还是解决掉，不要给系统留下隐患。

2、解决方法

解决依赖冲突的方法，就是使用Maven提供的标签，标签需要放在标签内部，就像下面这样：

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.10.0</version>
  <exclusions>
    <exclusion>
      <artifactId>log4j-api</artifactId>
      <groupId>org.apache.logging.log4j</groupId>
    </exclusion>
  </exclusions>
</dependency>
```

log4j-core 本身是依赖了 log4j-api 的，但是因为一些其他的模块也依赖了 log4j-api，并且两个 log4j-api 版本不同，所以我们使用标签排除掉 log4j-core 所依赖的 log4j-api，这样Maven就不会下载 log4j-core 所依赖的 log4j-api 了，也就保证了我们的项目中只有一个版本的 log4j-api。

3、Maven Helper

看到这里，你可能会有一个疑问。如何才能知道自己的项目中哪些依赖的Jar包冲突了呢？Maven Helper这个IntelliJ IDEA的插件帮我们解决了这个问题。插件的安装方法我就不讲了，既然你都会Maven了，我相信你也是会安装插件的。

在插件安装好之后，我们打开pom.xml文件，在底部会多出一个**Dependency Analyzer**选项

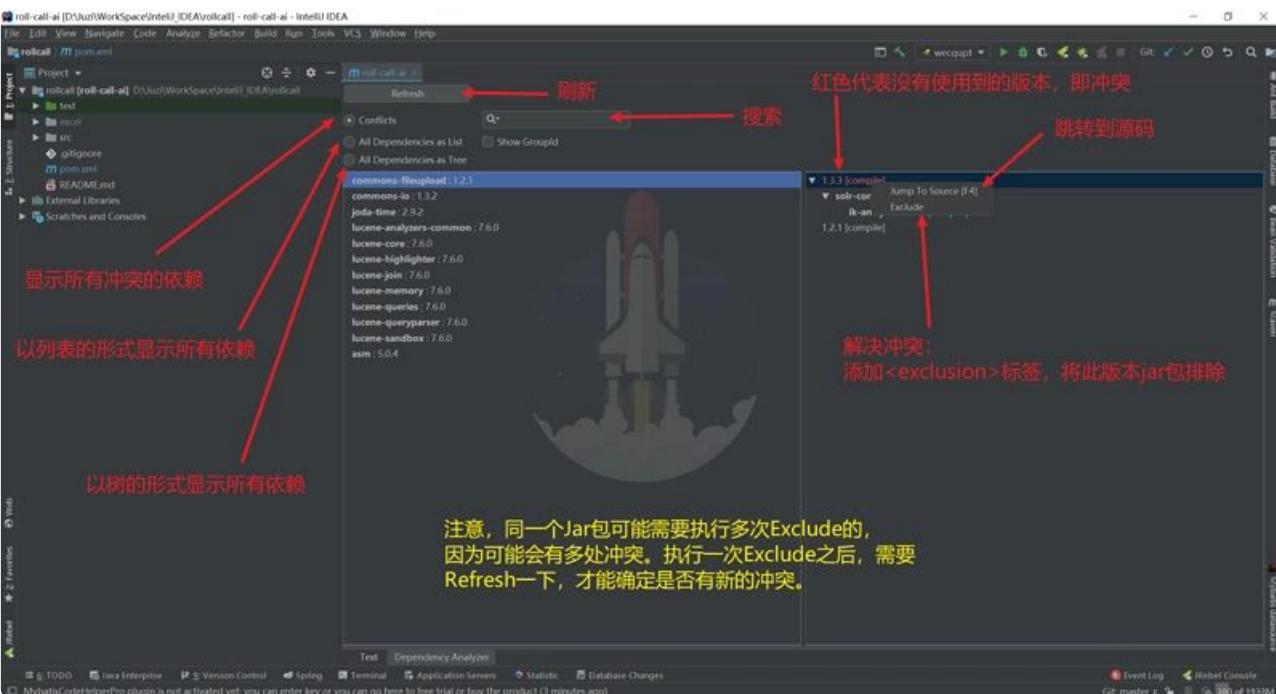
The screenshot shows the IntelliJ IDEA interface with the Maven Dependencies tab selected. A specific dependency entry is highlighted in red, indicating a conflict. The code snippet shows multiple versions of the same artifact being listed.

```

<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>easyexcel</artifactId>
    <version>${easyexcel.version}</version>
</dependency>
<!--junit测试websocket使用，删掉单元测试会报错-->
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-websocket</artifactId>
    <version>8.5.15</version>
</dependency>
<dependency>
    <groupId>com.sun.mail</groupId>
    <artifactId>javax.mail</artifactId>
    <version>1.6.2</version>
</dependency>
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.1.0.Alpha2</version>
    <exclusions>
        <exclusion>
            <artifactId>validation-api</artifactId>
            <groupId>javax.validation</groupId>
        </exclusion>
    </exclusions>
</dependency>

```

点开这个选项



找到冲突, 点击右键, 然后选择Exclude即可排除冲突版本的Jar包。

4、小技巧

除了使用Maven Helper查看依赖冲突, 也可以使用IDEA提供的方法——Maven依赖结构图, 打开Maven窗口, 选择 Dependencies, 然后点击那个图标 (Show Dependencies) 或者使用快捷键 (Ctrl+Alt+Shift+U), 即可打开Maven依赖关系结构图

roll-call-ai [D:\Juzh\WorkSpace\IntelliJ_IDE\roll-call-ai] - roll-call-ai - IntelliJ IDEA

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

roll-call-ai pom.xml

Project Structure

roll-call-ai [roll-call-ai] D:\Juzh\WorkSpace\IntelliJ_IDE\roll-call-ai

src test excel src .gitignore pom.xml README.md External Libraries Scratches and Consoles

627 </exclusions>

628 </dependency>

629 <!--生成二进制的依赖 -->

630 <dependency>

631 <groupId>net.glxn.qrcode</groupId>

632 <artifactId>javase</artifactId>

633 <version>2.0</version>

634 </dependency>

635 <dependency>

636 <groupId>com.alibaba</groupId>

637 <artifactId>easyexcel</artifactId>

638 <version>\${easyexcel.version}</version>

639 </dependency>

640 <!--junit测试用，删掉单元测试 -->

641 <dependency>

642 <groupId>org.apache.tomcat.embed</groupId>

643 <artifactId>tomcat-embed-websocket</artifactId>

644 <version>8.5.15</version>

645 </dependency>

646 <dependency>

647 <groupId>com.sun.mail</groupId>

648 <artifactId>javax.mail</artifactId>

649 <version>1.6.2</version>

650 </dependency>

651 <dependency>

652 <groupId>org.hibernate.validator</groupId>

653 <artifactId>hibernate-validator</artifactId>

654 <version>5.4.2.Final</version>

655 </dependency>

656 <dependency>

657 <groupId>org.hibernate.validator</groupId>

658 <artifactId>hibernate-validator</artifactId>

659 </dependency>

project dependencies dependency

Text Dependency Analyzer

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

roll-call-ai pom.xml

Maven

Profiles

sun Maven Webapp

Lifecycle

Plugins

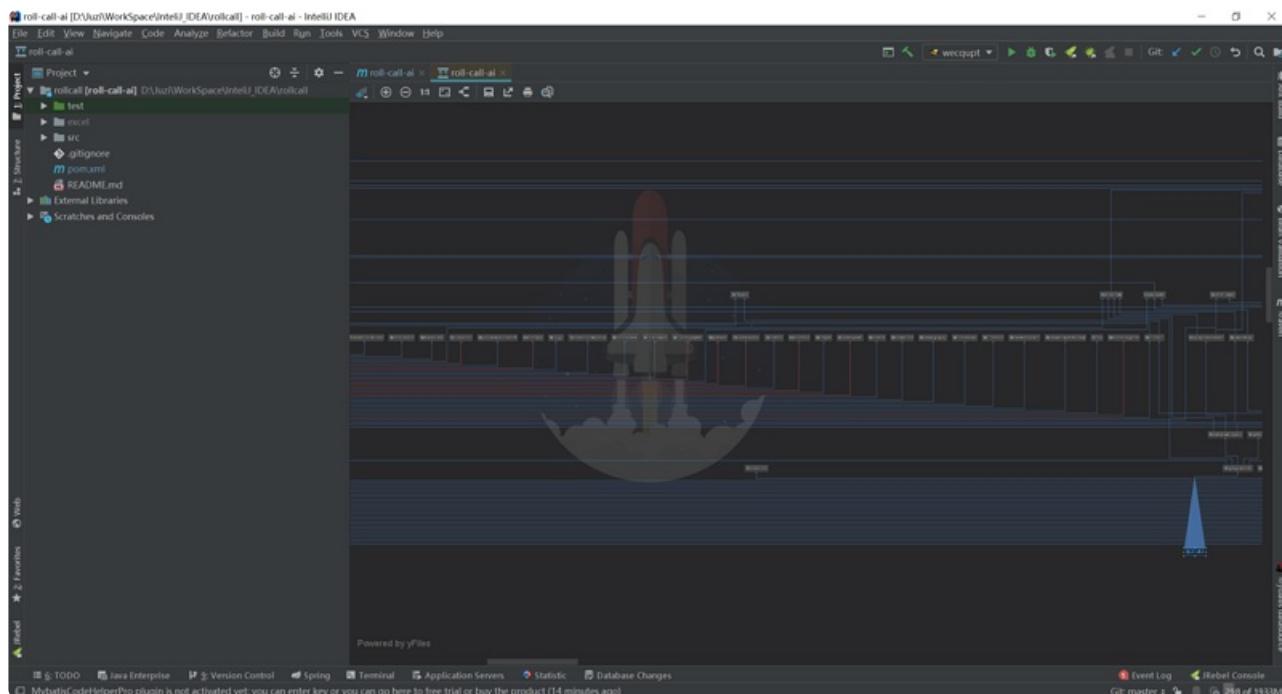
Dependencies

Event Log Rebel Console

MybatisHelperPro plugin is not activated yet, you can enter key or you can go here to free trial or buy the product (11 minutes ago)

651:1 GRF:1 UFT:0 4 spans Git: master % 429 of 1933M

在图中，我们可以看到有一些红色的实线，这些红色实线就是依赖冲突，蓝色实线则是正常的依赖。



来源：<http://suo.im/6brHfY>

【END】

如果看到这里，说明你喜欢这篇文章，请[转发、点赞](#)。微信搜索「web_resource」，关注后回复「进群」或者扫描下方二维码即可进入无广告交流群。

↓ 扫描二维码进群 ↓



扫一扫上面的二维码图案，加我微信

推荐阅读

1. [我把废旧 Android 手机改造成了 Linux 服务器](#)
2. [动画: 一个浏览器是如何工作的?](#)
3. [为什么你学不会递归?](#)
4. [一个女生不主动联系你还有机会吗?](#)
5. [团队开发中 Git 最佳实践](#)



喜欢文章, 点个在看

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！

这几个 IntelliJ IDEA 高级调试技巧，用了都说爽！

十光年 Java后端 2019-12-24

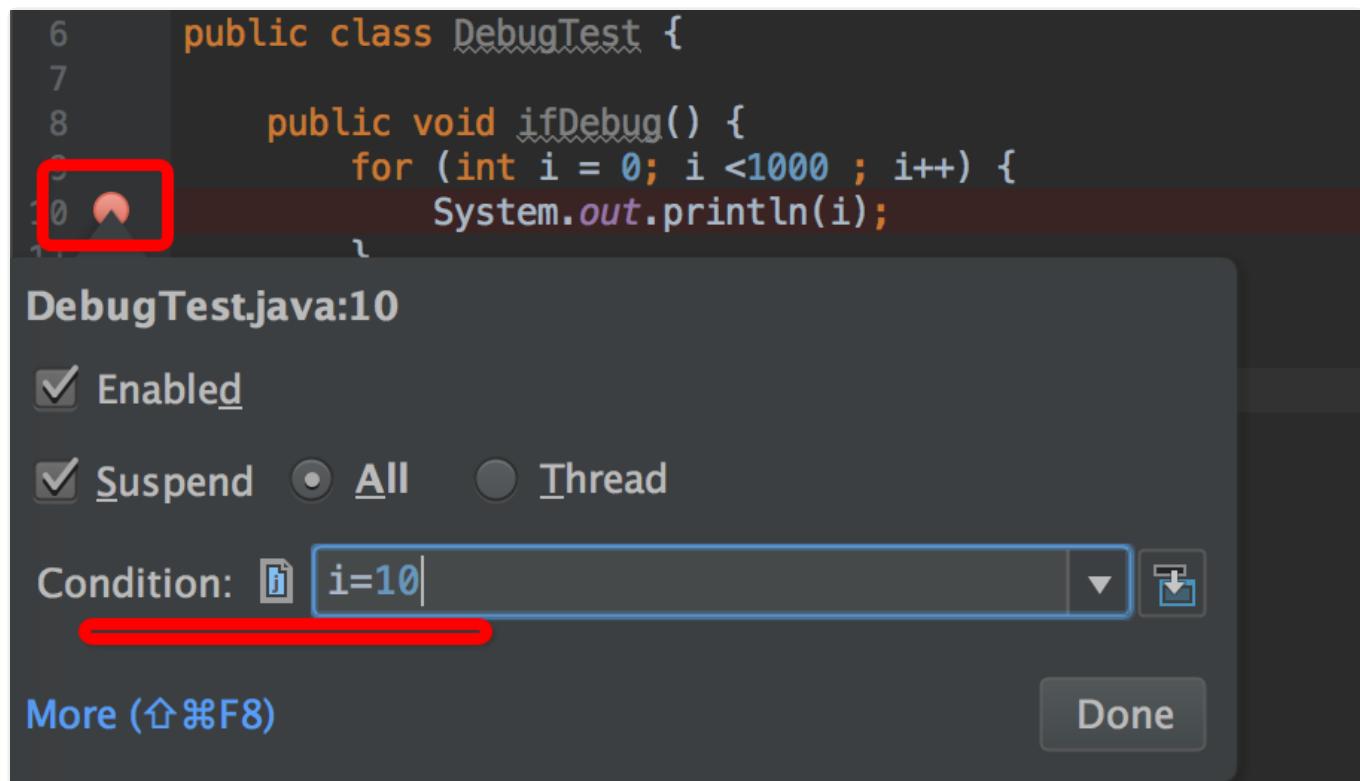
点击上方 Java后端, 选择 [设为星标](#)

优质文章, 及时送达

来源:<https://dwz.cn/zMaNp9Kf>

一、条件断点

循环中经常用到这个技巧，比如：遍历1个大List的过程中，想让断点停在某个特定值。



参考上图，在断点的位置，右击断点旁边的小红点，会出来一个界面，在Condition这里填入断点条件即可，这样调试时，就会自动停在i=10的位置



二、回到"上一步"

该技巧最适合特别复杂的方法套方法的场景，好不容易跑起来，一不小心手一抖，断点过去了，想回过头看看刚才的变量值，如果不知道该技巧，只能再跑一遍。

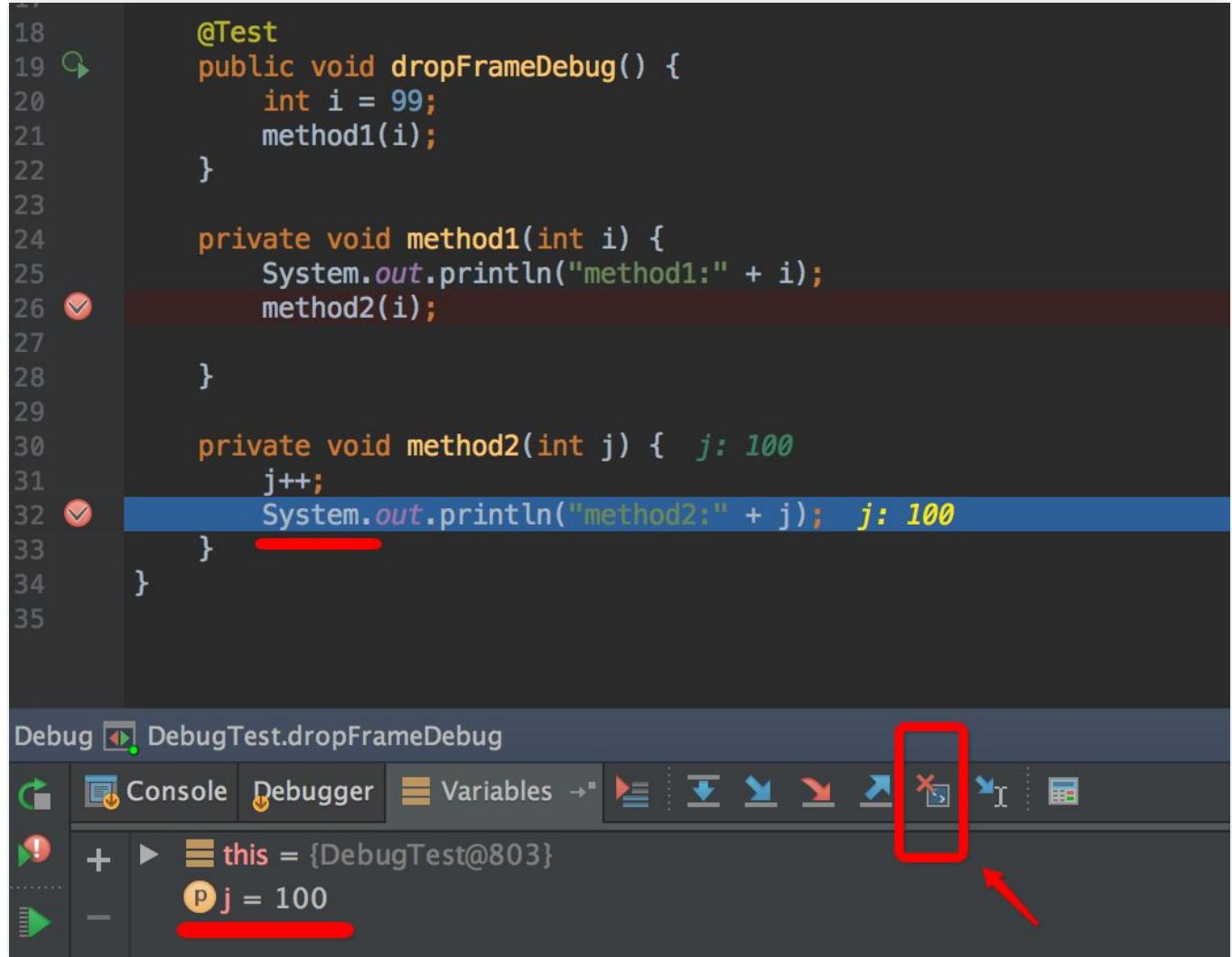
```
18     @Test
19     public void dropFrameDebug() {
20         int i = 99;
21         method1(i);
22     }
23
24     private void method1(int i) {
25         System.out.println("method1:" + i);
26         method2(i);
27     }
28
29     private void method2(int j) { j: 100
30         j++;
31         System.out.println("method2:" + j); j: 100
32     }
33 }
34
35
```

Debug DebugTest.dropFrameDebug

Console Debugger Variables

+ this = {DebugTest@803}

p j = 100



参考上图，method1方法调用method2，当前断点的位置j=100，点击上图红色箭头位置的Drop Frame图标后，时间穿越了

The screenshot shows a Java code editor and a debugger interface. The code editor displays the following Java code:

```
19     public void dropFrameDebug() {  
20         int i = 99;  
21         method1(i);  
22     }  
23  
24     private void method1(int i) { i: 99  
25         System.out.println("method1:" + i);  
26         method2(i); i: 99  
27     }  
28  
29     private void method2(int j) {  
30         j++;  
31         System.out.println("method2:" + j);  
32     }  
33  
34 }  
35
```

The debugger interface at the bottom shows the current state of variables:

Variable	Type	Value
this	{DebugTest@803}	
i	int	99

回到了method1刚开始调用的时候，变量i变成了99，没毛病吧，老铁们，是不是很6：)

注：好奇心是人类进步的阶梯，如果想知道为啥这个功能叫Drop Frame，而不是类似Back To Previous 之类的，可以去翻翻JVM的书，JVM内部以栈帧为单位保存线程的运行状态，drop frame即扔掉当前运行的栈帧，这样当前“指针”的位置，就自然到了上一帧的位置。

三、多线程调试

多线程同时运行时，谁先执行，谁后执行，完全是看CPU心情的，无法控制先后，运行时可能没什么问题，但是调试时就比较麻烦了，最明显的就是断点乱跳，一会儿停这个线程，一会儿停在另一个线程，比如下图：



```
@Test
public void multiThreadTest() {
    new Thread(() -> {
        System.out.println("1.卧枝商恨低");
    }, "菩提树下的杨过").start();

    new Thread(() -> {
        System.out.println("2.卧梅又闻花");
    }, "天空中的飞鸟").start();

    System.out.println("3.要问卧似水");
    System.out.println("4.倚头答春绿");
}
```

如果想希望下一个断点位置是第2句诗句，可能要失望了：

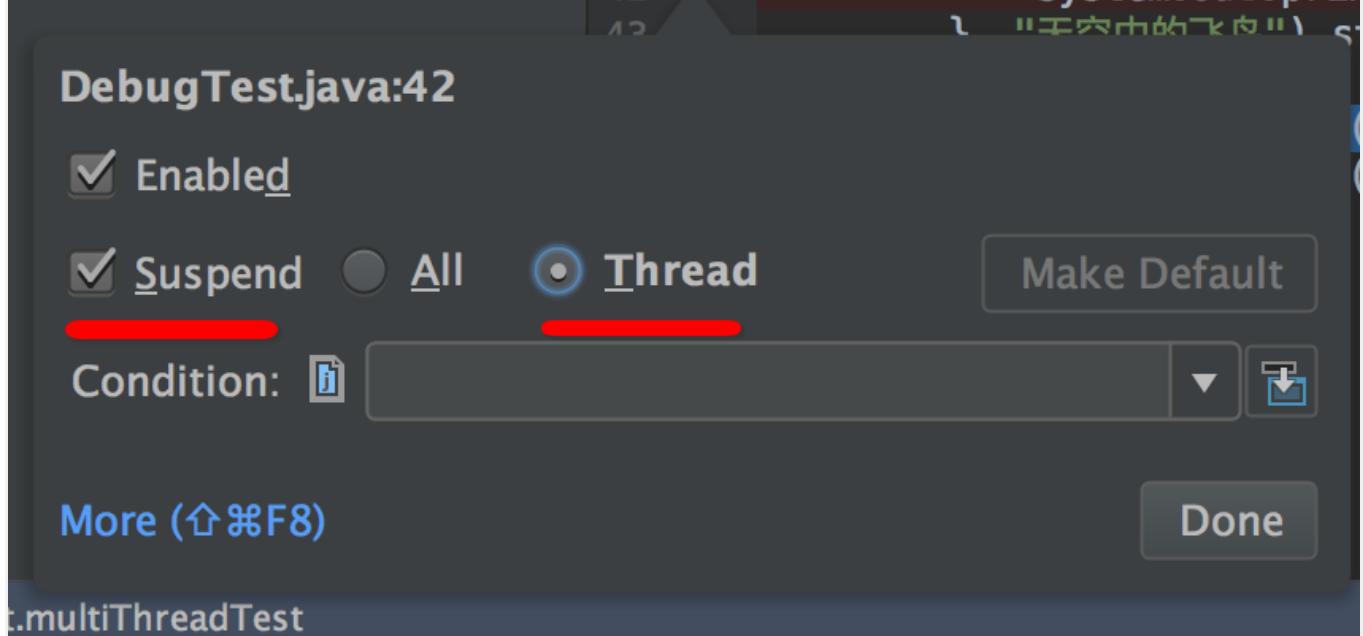


```
@Test
public void multiThreadTest() {
    new Thread(() -> {
        System.out.println("1.卧枝商恨低");
    }, "菩提树下的杨过").start();

    new Thread(() -> {
        System.out.println("2.卧梅又闻花");
    }, "天空中的飞鸟").start();

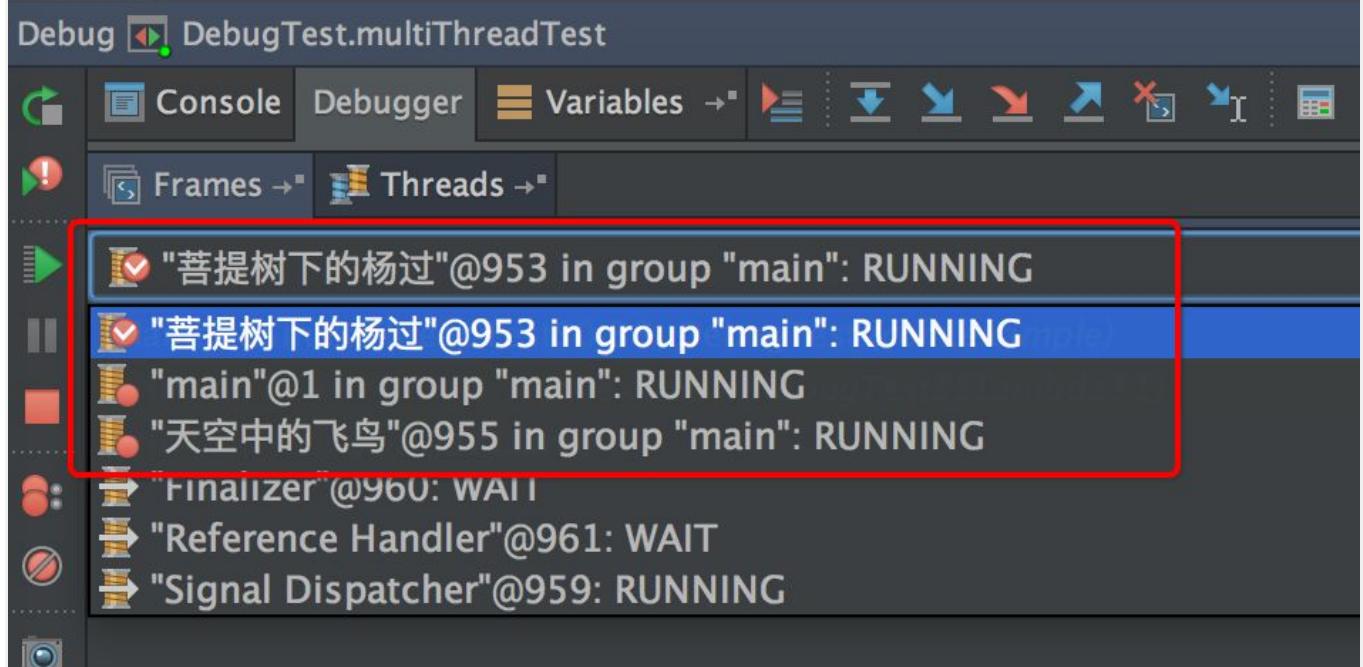
    System.out.println("3.要问卧似水");
    System.out.println("4.倚头答春绿");
}
```

如果想让线程在调试时，想按自己的愿意来，让它停在哪个线程就停在哪个线程，可以在图中3个断点的小红点上右击，

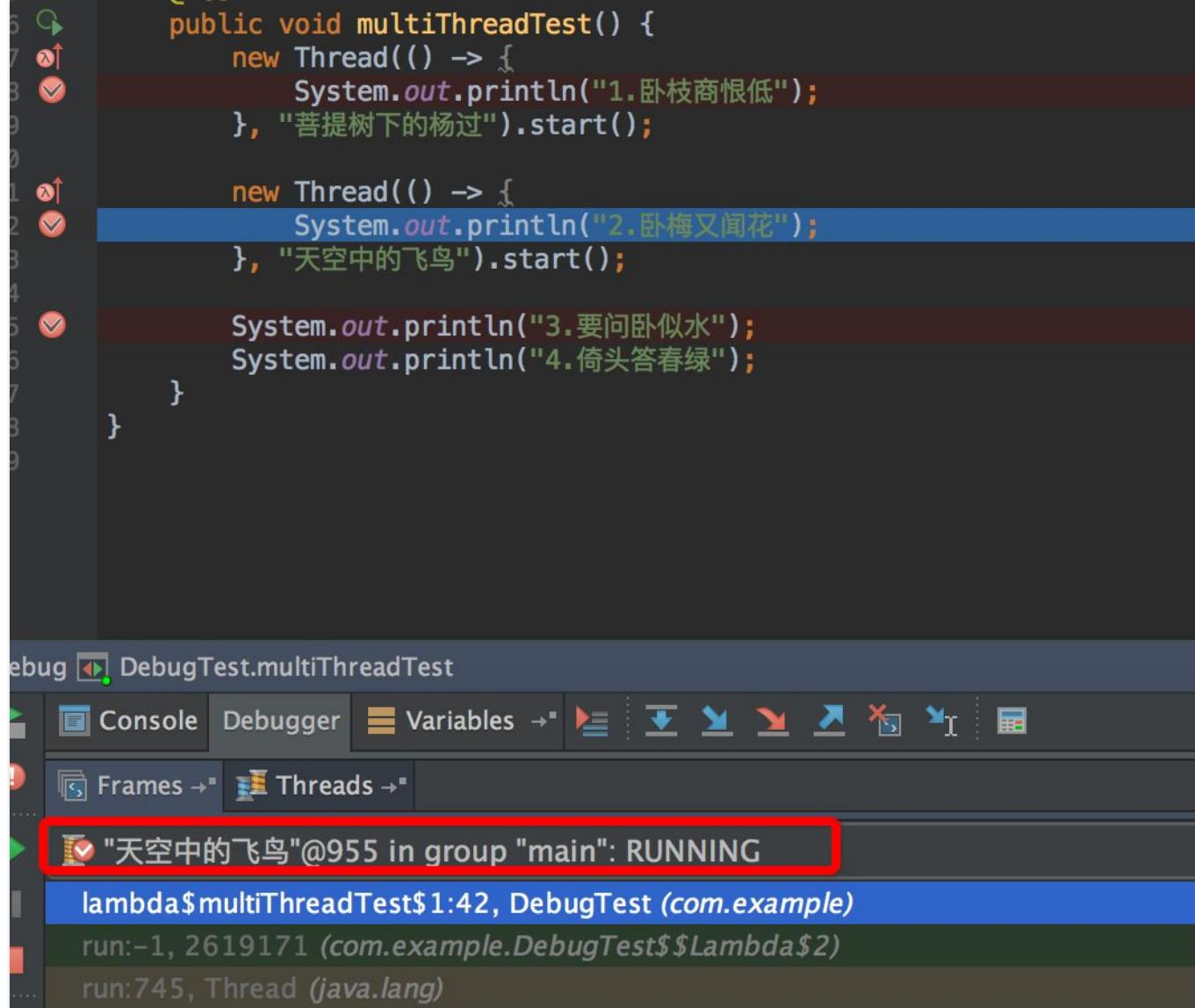


即：Suspend挂起的条件是按每个线程来，而非All。把这3个断点都这么设置后，再来一发试试

```
34
35
36 @Test
37 public void multiThreadTest() {
38     new Thread(() -> {
39         System.out.println("1.卧枝商恨低");
40         }, "菩提树下的杨过").start();
41
42     new Thread(() -> {
43         System.out.println("2.卧梅又闻花");
44         }, "天空中的飞鸟").start();
45
46     System.out.println("3.要问卧似水");
47     System.out.println("4.倚头答春绿");
48 }
49
```



注意上图中的红框位置，断点停下来时，这个下拉框可以看到各个线程（注：给线程起个容易识别的名字是个好习惯！），我们可以选择线程“天空中的飞鸟”



断点如愿停在了第2句诗。

四、远程调试

这也是一个装B的利器，本机不用启动项目，只要有源代码，可以在本机直接远程调试服务器上的代码，打开姿势如下：

1、项目启动时，先允许远程调试

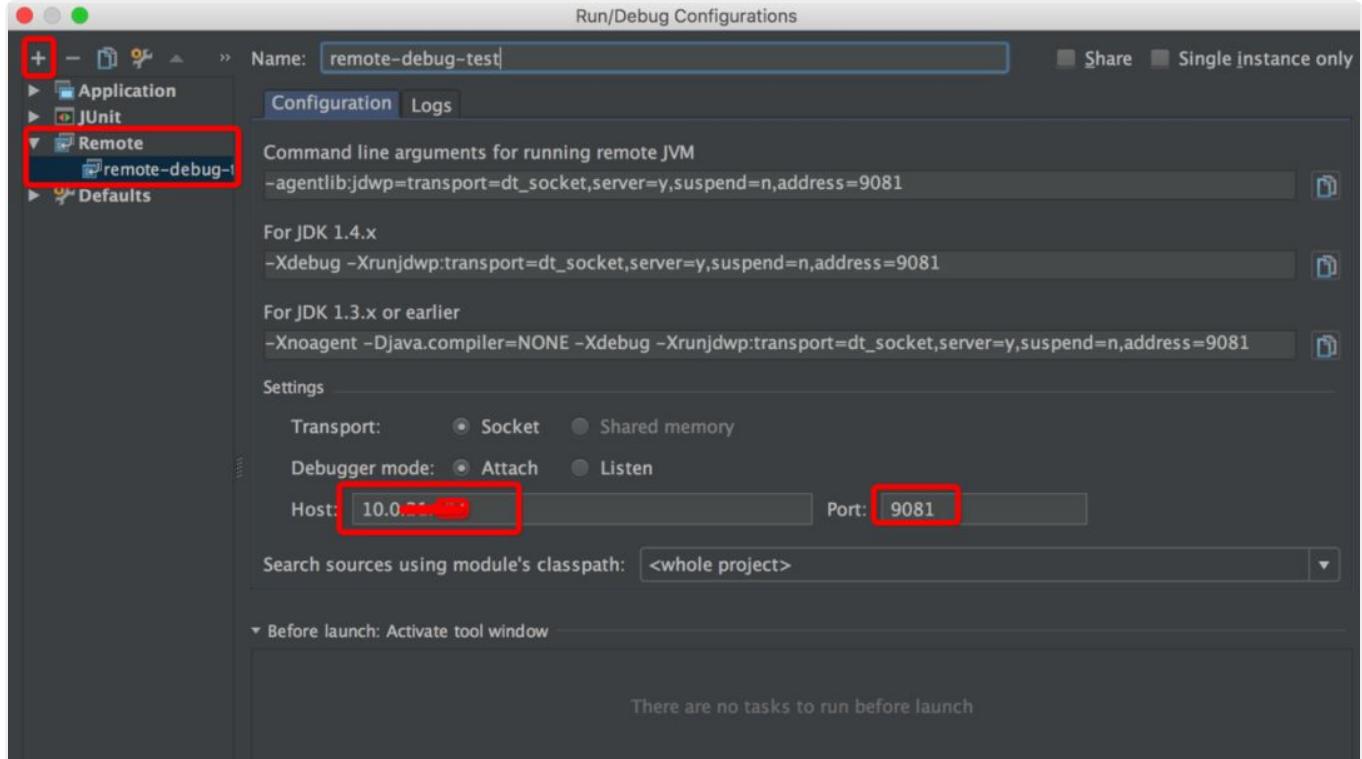
```
java -server -Xms512m -Xmx512m -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=9081 -Djava.ext.dirs=. ${main_class}
```

起作用的就是

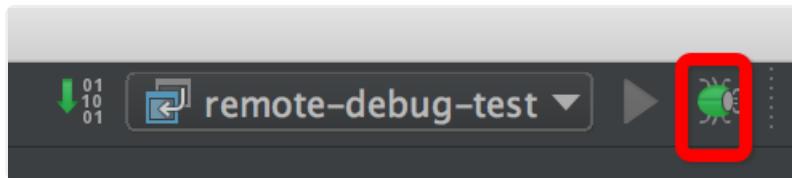
```
-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=9081
```

注意：远程调试从技术上讲，就是在本机与远程建立scoket通讯，所以端口不要冲突，而且本机要允许访问远程端口，另外这一段参数，放要在-jar 或 \${main_class}的前面

2、idea中设置远程调试



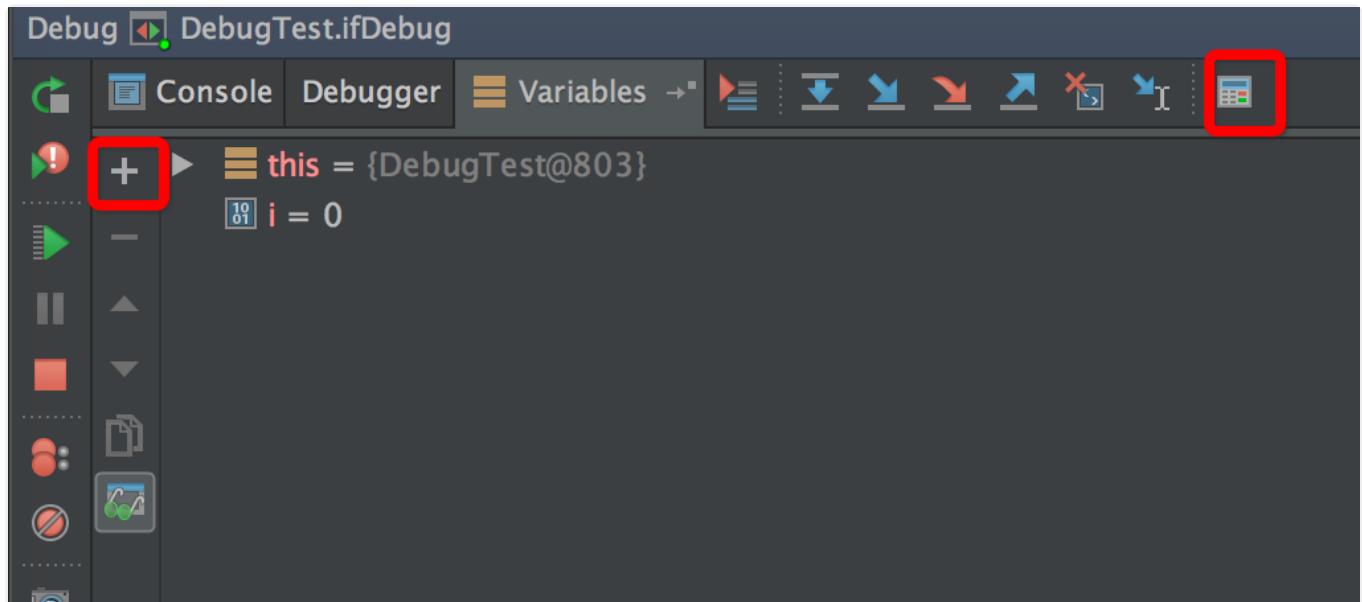
然后就可以调试了



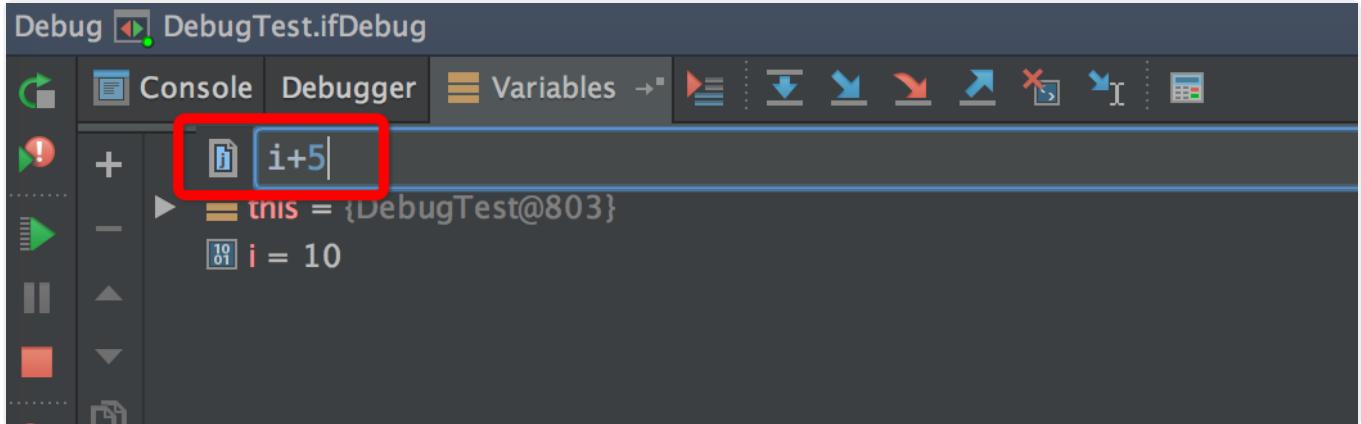
前提是本机有项目的源代码，在需要的地方打个断点，然后访问一个远程的url试试，断点就会停下来。

五、临时执行表达式/修改变量的运行值

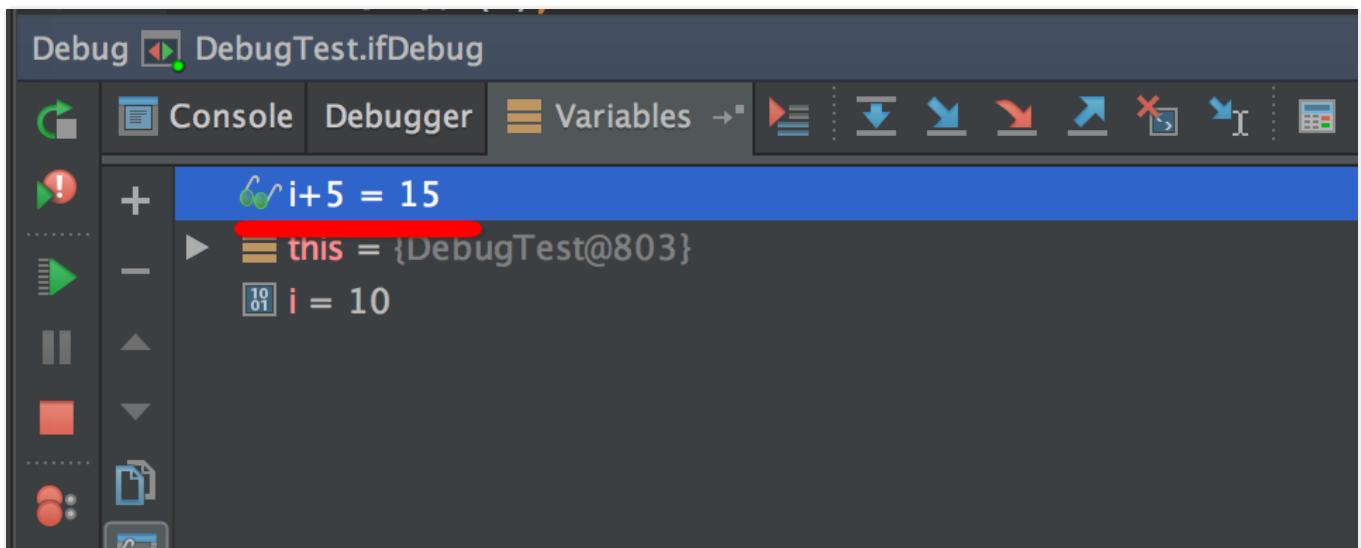
调试时，可以临时执行一些表达式，参考下图：点击这两个图标中的任何一个都可以



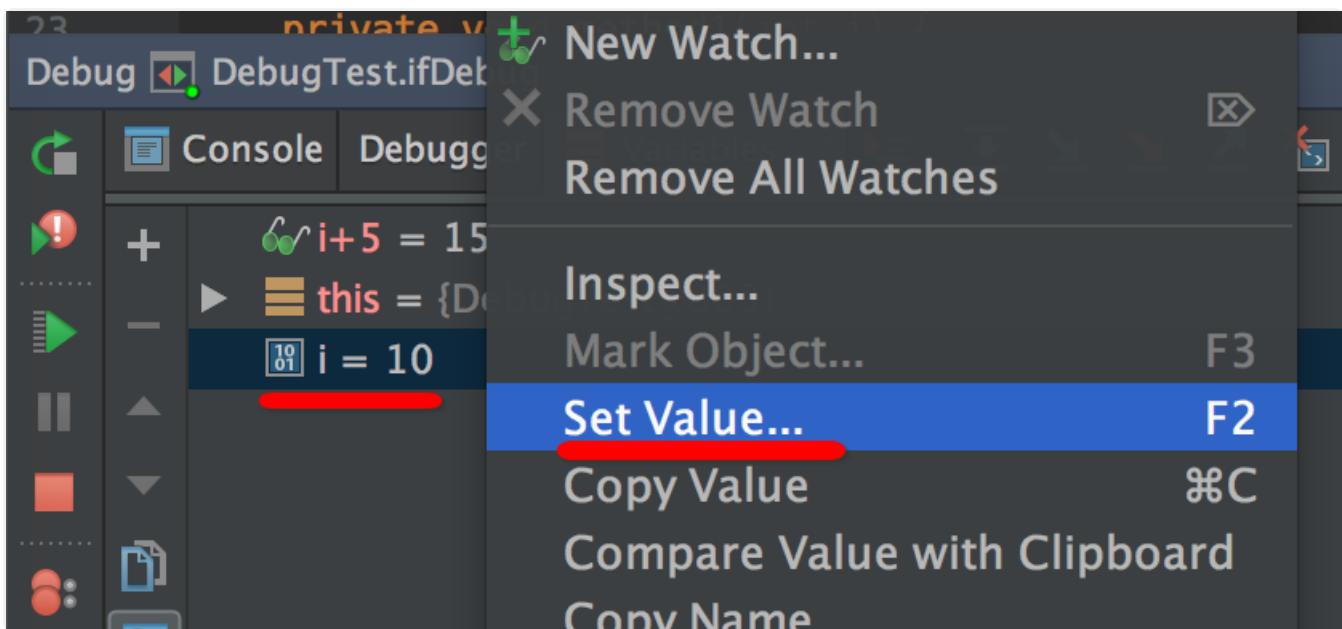
点击+号后，就可以在新出现的输入框里输入表达式，比如*i+5*



然后回车，马上就能看到结果



当然，如果调试时，想动态修改变量的值，也很容易，在变量上右击，然后选择Set Value，剩下的事，地球人都知道。



善用上述调试技巧，相信大家撸起代码来会更有感觉。

-END-

推荐阅读

1. 发布没有答案的面试题，都是耍流氓
2. 8岁上海小学生B站教编程惊动苹果

[3. 我在华为做外包的真实经历！](#)

[4. 什么是一致性 Hash 算法？](#)

[5. 团队开发中 Git 最佳实践](#)



喜欢文章，点个在看 

[阅读原文](#)

声明：pdf仅供学习使用，一切版权归原创公众号所有；建议持续关注原创公众号获取最新文章，学习愉快！