Java开发利器：IDEA的安装与使用（下）

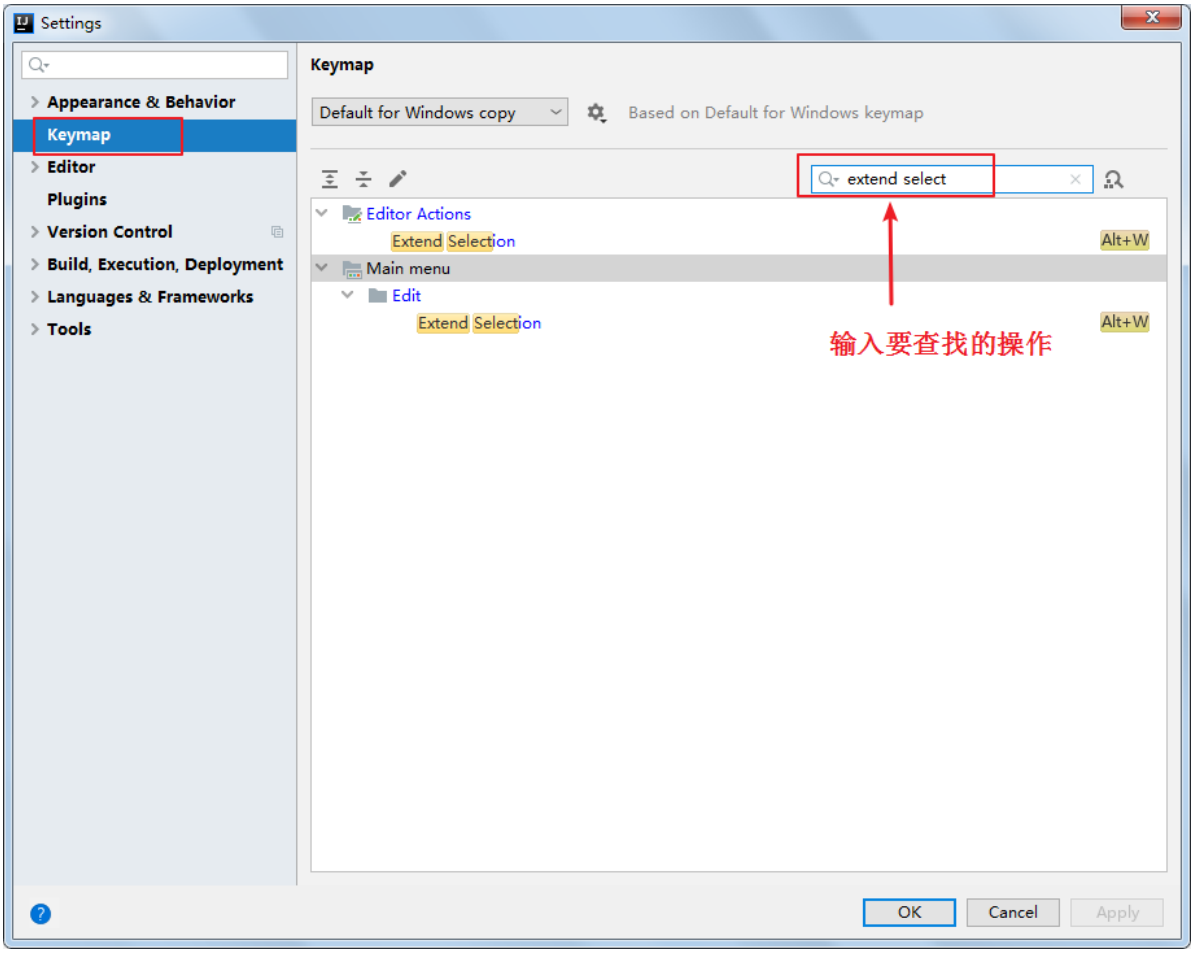
8. 快捷键的使用

8.1 常用快捷键

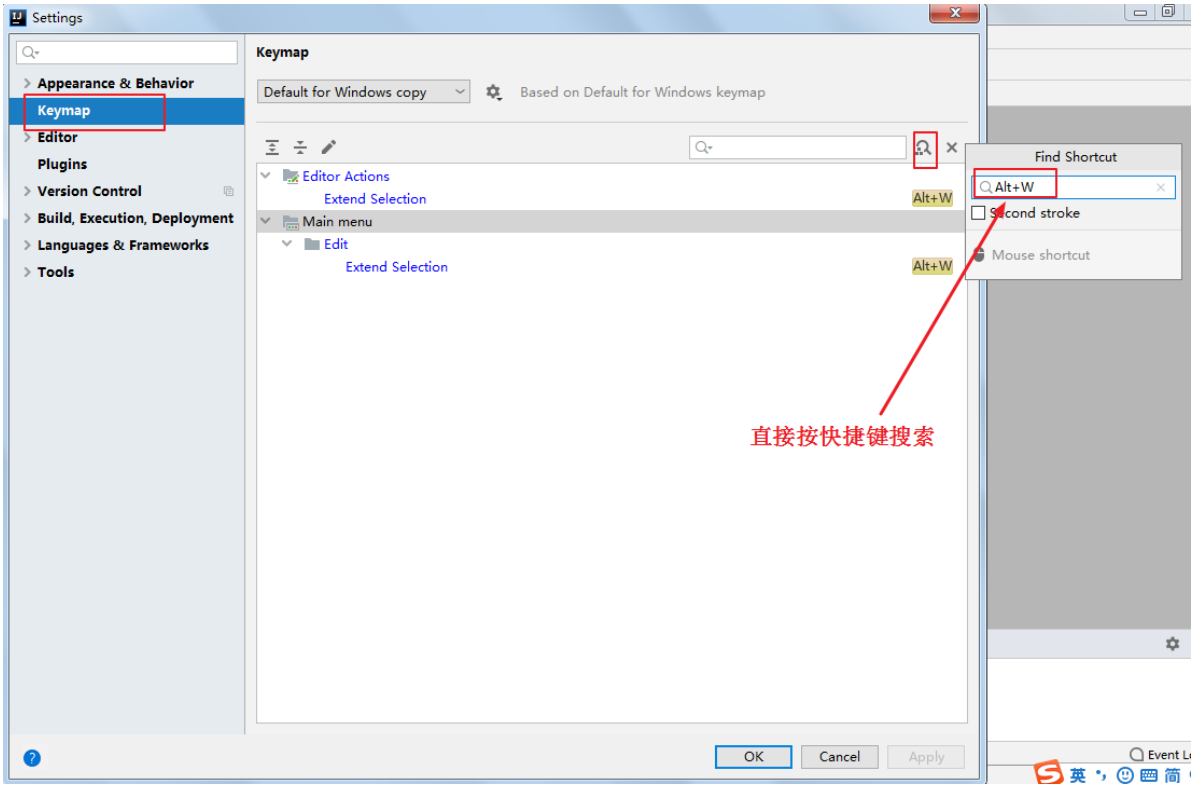
见《`尚硅谷_宋红康_IntelliJ IDEA 常用快捷键一览表.md`》

8.2 查看快捷键

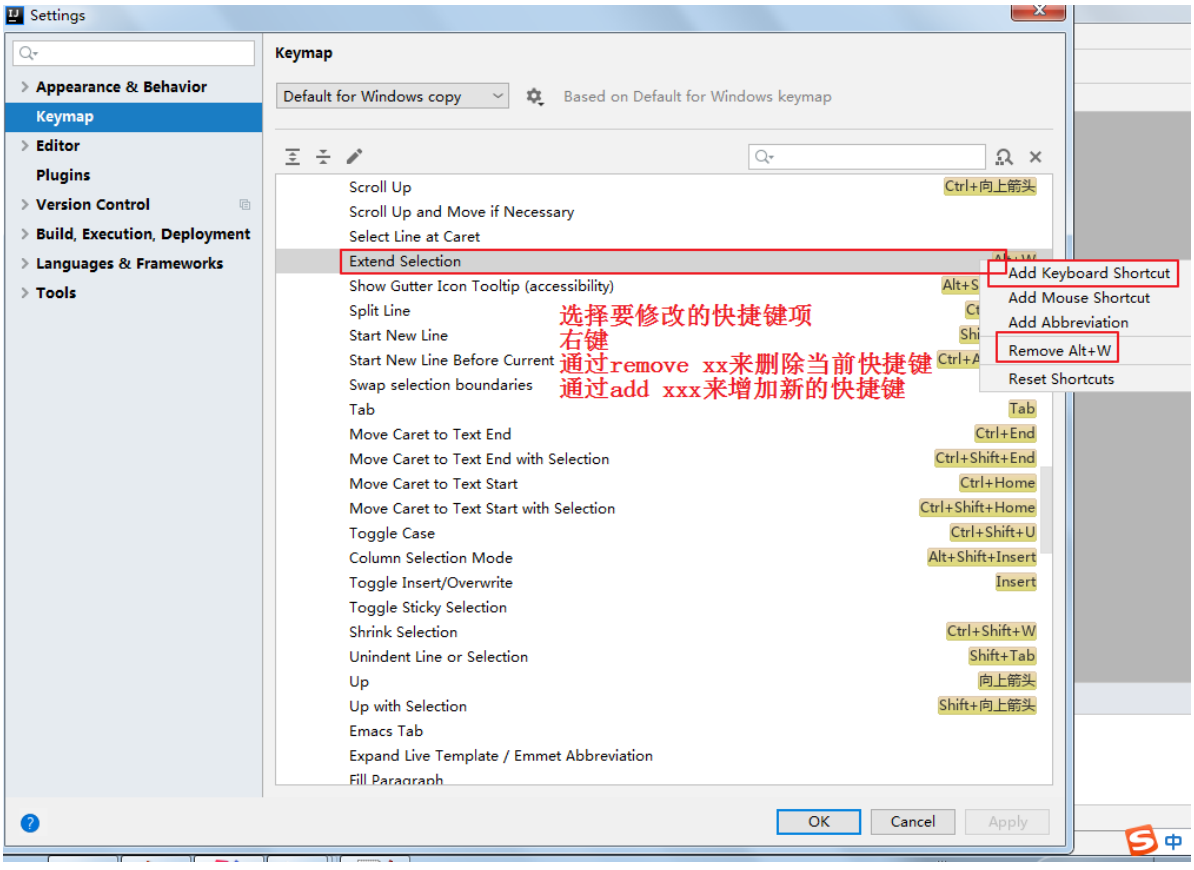
1、已知快捷键操作名，未知快捷键



2、已知快捷键，不知道对应的操作名

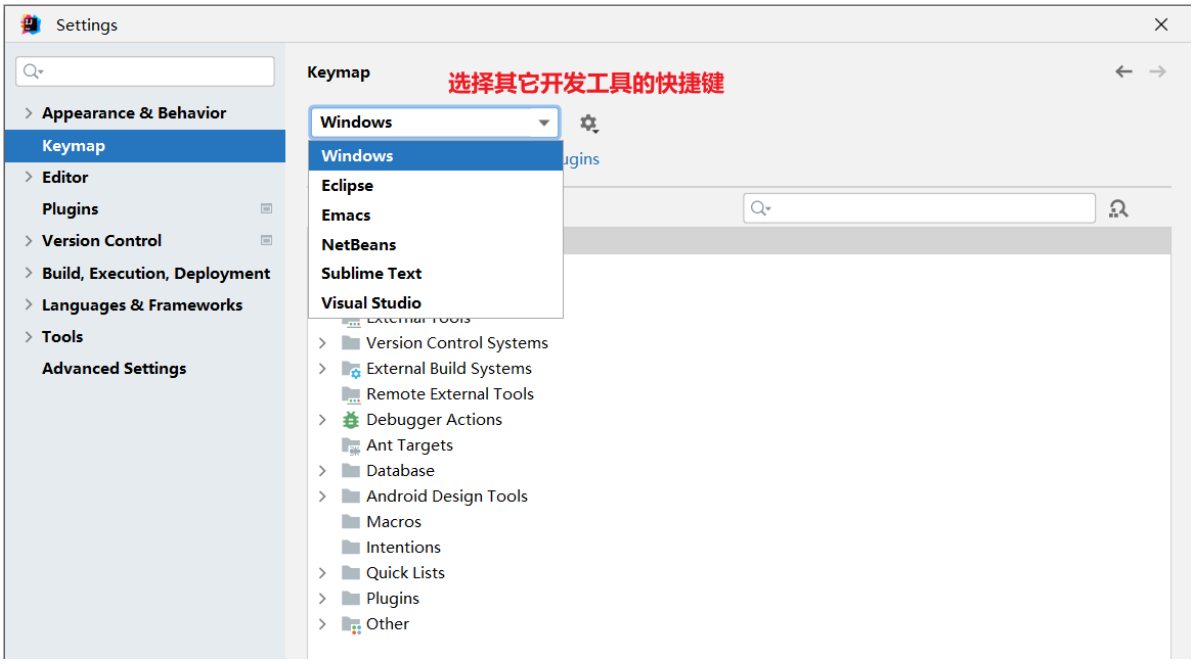


8.3 自定义快捷键



8.4 使用其它平台快捷键

苹果电脑或者是用惯Eclipse快捷的，可以选择其他快捷键插件。



9. IDEA断点调试(Debug)

9.1 为什么需要Debug

编好的程序在执行过程中如果出现错误，该如何查找或定位错误呢？简单的代码直接就可以看出来，但如果代码比较复杂，就需要借助程序调试来查找错误了。

```
运行编写好的程序时，可能出现的几种情况：
> 情况1：没有任何bug,程序执行正确！

====================如果出现如下的三种情况，都又必要使用debug============================
> 情况2：运行以后，出现了错误或异常信息。但是通过日志文件或控制台，显示了异常信息的位置。
> 情况3：运行以后，得到了结果，但是结果不是我们想要的。
> 情况4：运行以后，得到了结果，结果大概率是我们想要的。但是多次运行的话，可能会出现不是我们想要的情况。
        比如：多线程情况下，处理线程安全问题。
```

9.2 Debug的步骤

Debug(调试)程序步骤如下：

1、添加断点

2、启动调试

3、单步执行

4、观察变量和执行流程，找到并解决问题

1、添加断点

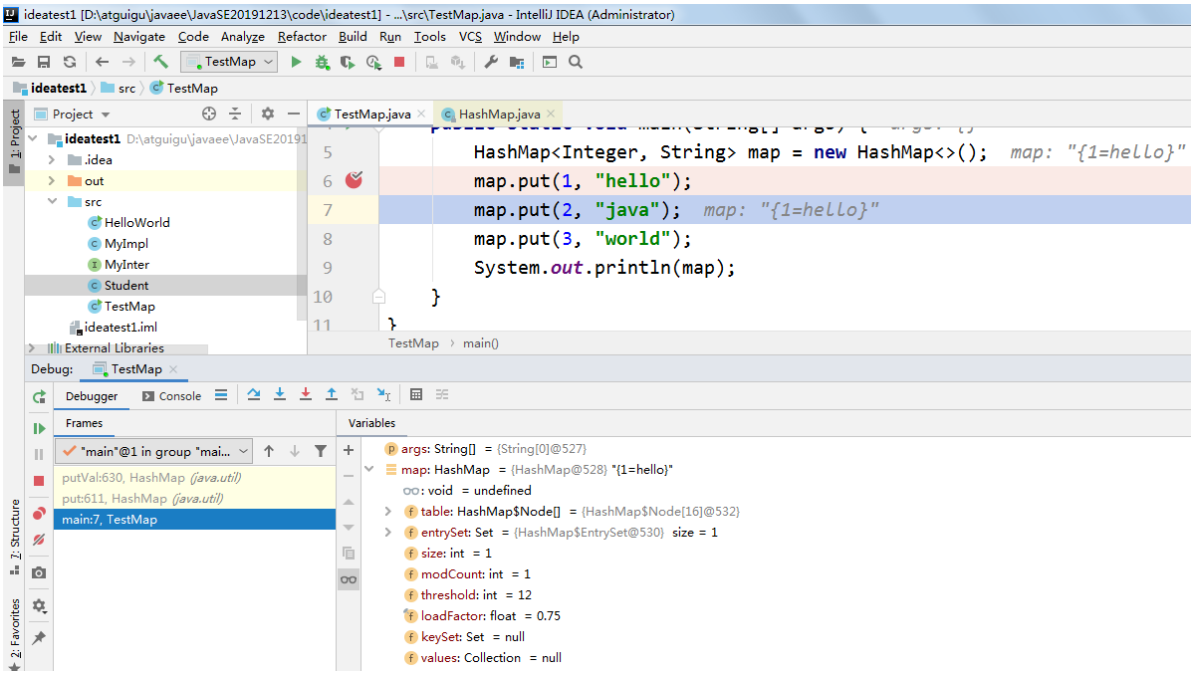
在源代码文件中，在想要设置断点的代码行的前面的标记行处，单击鼠标左键就可以设置断点，在相同位置再次单击即可取消断点。

```java
import java.util.HashMap;

public class TestMap {
    public static void main(String[] args) {
        HashMap<Integer, String> map = new HashMap<>();
        map.put(1, "hello");
        map.put(2, "java");
        map.put(3, "world");
        System.out.println(map);
    }
}
```

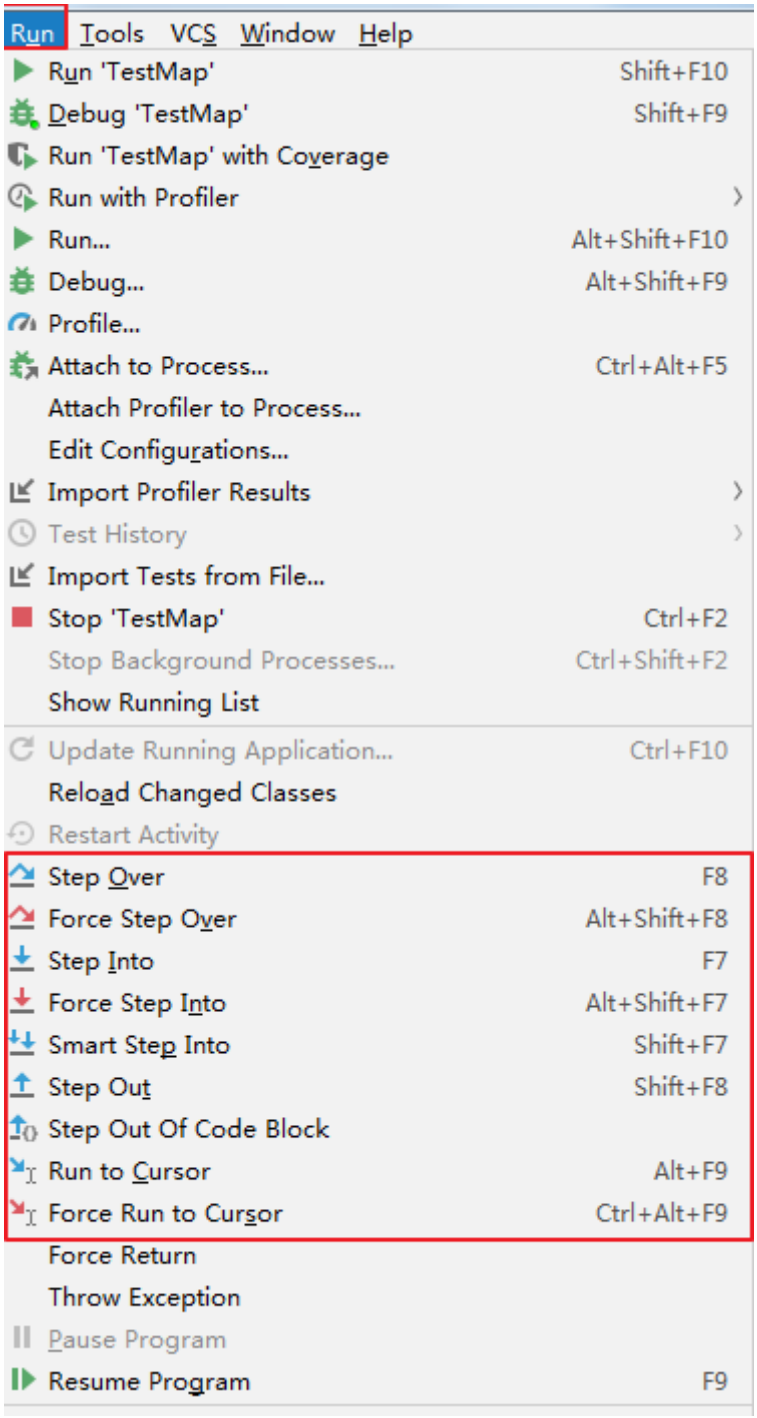
断点

2、启动调试

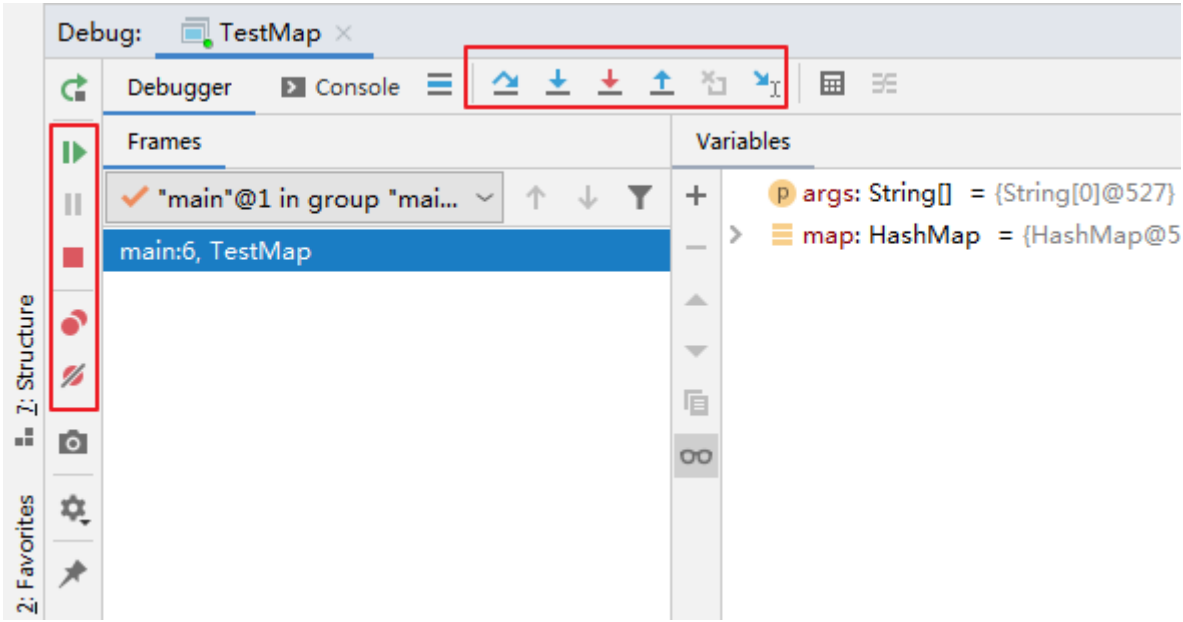
IDEA提供多种方式来启动程序(Launch)的调试，分别是通过菜单(Run –> Debug)、图标(“绿色臭虫”  等等



3、单步调试工具介绍


Run
Tools
VCS
Window
Help
Run 'TestMap' Shift+F10
Debug 'TestMap' Shift+F9
Run 'TestMap' with Coverage
Run with Profiler
Run... Alt+Shift+F10
Debug... Alt+Shift+F9
Profile...
Attach to Process... Ctrl+Alt+F5
Attach Profiler to Process...
Edit Configurations...
Import Profiler Results
Test History
Import Tests from File...
Stop 'TestMap' Ctrl+F2
Stop Background Processes... Ctrl+Shift+F2
Show Running List
Update Running Application... Ctrl+F10
Reload Changed Classes
Restart Activity
Step Over F8
Force Step Over Alt+Shift+F8
Step Into F7
Force Step Into Alt+Shift+F7
Smart Step Into Shift+F7
Step Out Shift+F8
Step Out Of Code Block
Run to Cursor Alt+F9
Force Run to Cursor Ctrl+Alt+F9
Force Return
Throw Exception
Pause Program
Resume Program F9

或



：Step Over（F8）：进入下一步，如果当前行断点是调用一个方法，则不进入当前方法体内

：Step Into（F7）：进入下一步，如果当前行断点是调用一个自定义方法，则进入该方法体内

：Force Step Into（Alt +Shift + F7）：进入下一步，如果当前行断点是调用一个核心类库方法，则进入该方法体内

：Step Out（Shift + F8）：跳出当前方法体

：Run to Cursor（Alt + F9）：直接跳到光标处继续调试

：Resume Program（F9）：恢复程序运行，但如果该断点下面代码还有断点则停在下一个断点上

：Stop（Ctrl + F2）：结束调试

：View Breakpoints（Ctrl + Shift + F8）：查看所有断点

：Mute Breakpoints：使得当前代码后面所有的断点失效， 一下执行到底

> 说明：在Debug过程中，可以动态的下断点。

9.3 多种Debug情况介绍

9.3.1 行断点

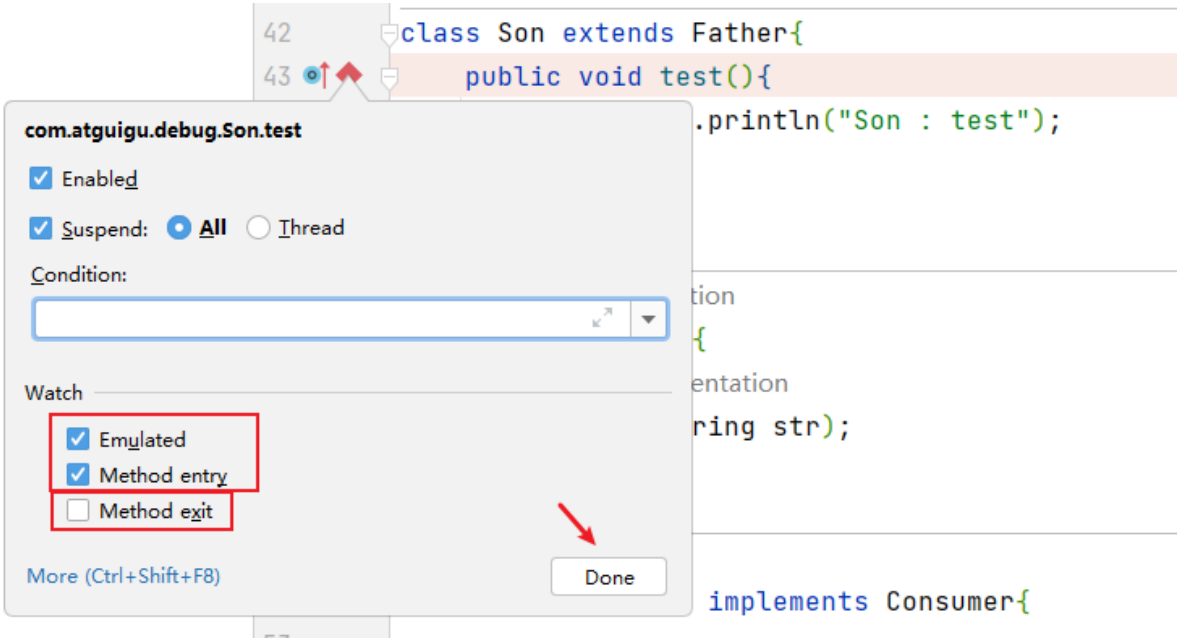
- 断点打在代码所在的行上。执行到此行时，会停下来。

```
package com.atguigu.debug;

/**
 * ClassName: Debug01
 * Package: com.atguigu.debug
 * Description: 演示1: 行断点  &  测试debug各个常见操作按钮
 *
 * @Author: 尚硅谷-宋红康
 * @Create: 2022/10/20 18:44
 * @Version 1.0
 */
public class Debug01 {
    public static void main(String[] args) {
```

```
        //1.
        int m = 10;
        int n = 20;
        System.out.println("m = " + m + ",n = " + n);
        swap(m, n);
        System.out.println("m = " + m + ",n = " + n);

        //2.
        int[] arr = new int[] {1,2,3,4,5};
        System.out.println(arr);//地址值

        char[] arr1 = new char[] {'a','b','c'};
        System.out.println(arr1);//abc
    }

    public static void swap(int m,int n){
        int temp = m;
        m = n;
        n = temp;
    }

}
```

9.3.2 方法断点

- 断点设置在方法的签名上，默认当进入时，断点可以被唤醒。
- 也可以设置在方法退出时，断点也被唤醒



- 在多态的场景下，在父类或接口的方法上打断点，会自动调入到子类或实现类的方法

```
package com.atguigu.debug;

import java.util.HashMap;

/**
 * ClassName: Debug02
 * Package: com.atguigu.debug
 * Description: 演示2:  方法断点
 *
```

```
 * @Author: 尚硅谷-宋红康
 * @Create: 2022/10/20 21:15
 * @Version 1.0
 */
public class Debug02 {
    public static void main(String[] args) {

        //1.
        Son instance = new Son();
        instance.test();
        //2.
        Father instance1 = new Son();
        instance1.test();

        //3.
        Consumer con = new ConsumerImpl();
        con.accept("atguigu");

        //4.
        HashMap map = new HashMap();
        map.put("Tom",12);
        map.put("Jerry",11);
        map.put("Tony",20);
    }
}

class Father{
    public void test(){
        System.out.println("Father : test");
    }
}

class Son extends Father{
    public void test(){
        System.out.println("Son : test");
    }
}

interface Consumer{
    void accept(String str);
}

class ConsumerImpl implements Consumer{

    @Override
    public void accept(String str) {
        System.out.println("ConsumerImple:" + str);
    }
}
```

9.3.3 字段断点

- 在类的属性声明上打断点，默认对属性的修改操作进行监控


```
package com.atguigu.debug;

/**
 * ClassName: Debug03
 * Package: com.atguigu.debug
 * Description: 演示3：字段断点
 *
 * @Author: 尚硅谷-宋红康
 * @Create: 2022/10/20 21:34
 * @Version 1.0
 */
public class Debug03 {
    public static void main(String[] args) {
        Person p1 = new Person(3);

        System.out.println(p1);
    }
}

class Person{
    private int id = 1;
    private String name;
    private int age;

    public Person() {
    }
    {
        id = 2;
    }
    public Person(int id) {
        this.id = id;
    }
```

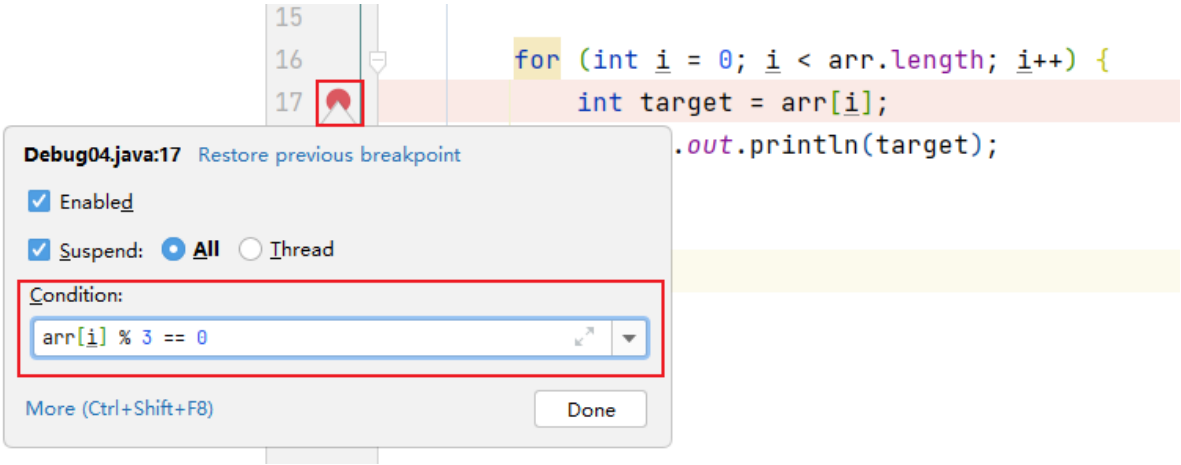
```
    public Person(int id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

9.3.4 条件断点

```
package com.atguigu.debug;

/**
 * ClassName: Debug04
 * Package: com.atguigu.debug
 * Description: 演示4：条件断点
 *
 * @Author: 尚硅谷-宋红康
 * @Create: 2022/10/20 21:49
 * @Version 1.0
 */
```

```java
public class Debug04 {
    public static void main(String[] args) {
        int[] arr = new int[]{1,2,3,4,5,6,7,8,9,10,11,12};

        for (int i = 0; i < arr.length; i++) {
            int target = arr[i];
            System.out.println(target);
        }
    }
}
```

针对上述代码，在满足arr[i] % 3 == 0的条件下，执行断点。

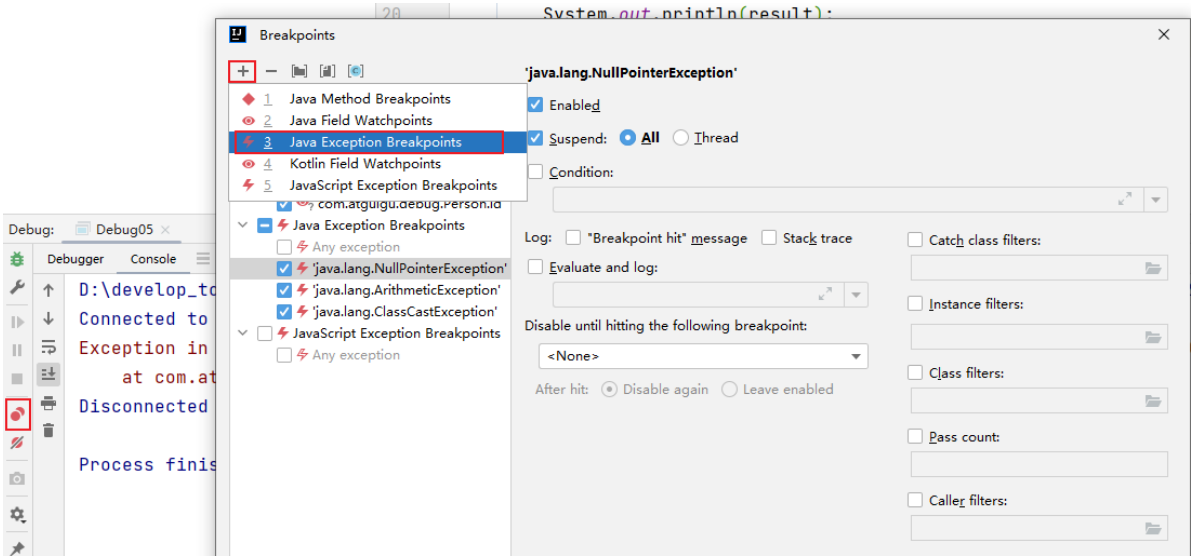


9.3.5 异常断点

- 对异常进行跟踪。如果程序出现指定异常，程序就会执行断点，自动停住。

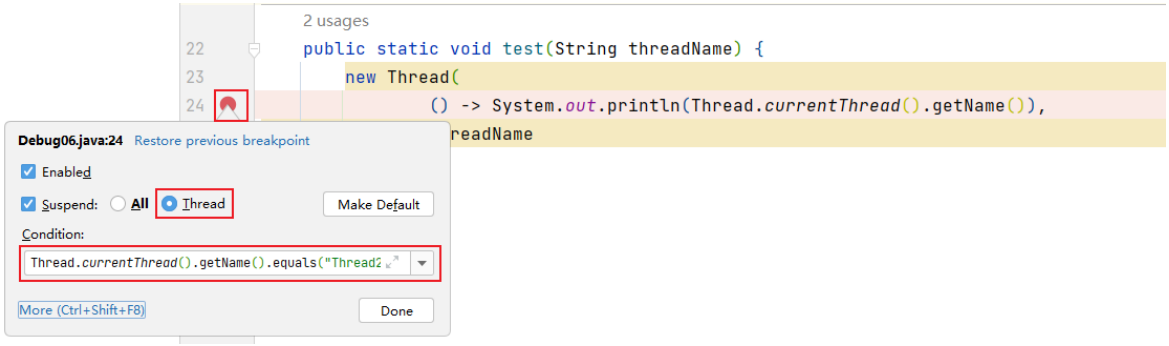
```java
package com.atguigu.debug;

import java.util.Date;

/**
 * ClassName: Debug05
 * Package: com.atguigu.debug
 * Description: 演示5：异常断点
 *
 * @Author: 尚硅谷-宋红康
 * @Create: 2022/10/20 22:01
 * @Version 1.0
 */
public class Debug05 {
    public static void main(String[] args) {

        int m = 10;
        int n = 0;
        int result = m / n;
        System.out.println(result);


//        Person p1 = new Person(1001);
//        System.out.println(p1.getName().toUpperCase());


    }
}
```

通过下图的方式，对指定的异常进行监控：



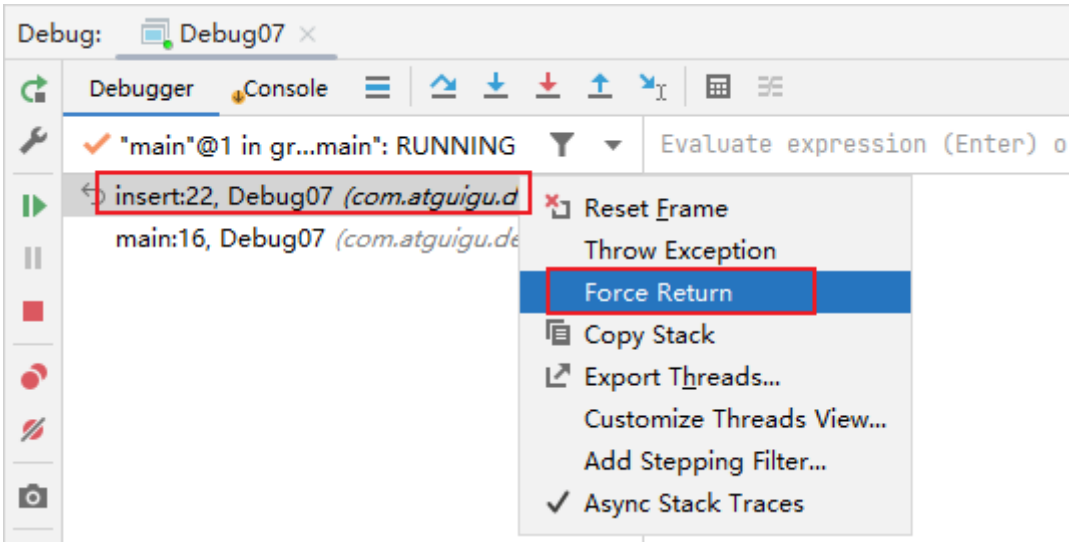
9.3.6 线程调试

```
package com.atguigu.debug;

/**
 * ClassName: Debug06
 * Package: com.atguigu.debug
 * Description: 演示6：线程调试
 *
 * @Author: 尚硅谷-宋红康
 * @Create: 2022/10/20 22:46
 * @Version 1.0
 */
public class Debug06 {

    public static void main(String[] args) {

        test("Thread1");
        test("Thread2");


    }

    public static void test(String threadName) {
        new Thread(
                () -> System.out.println(Thread.currentThread().getName()),
                threadName
        ).start();
    }

}
```



9.3.7 强制结束

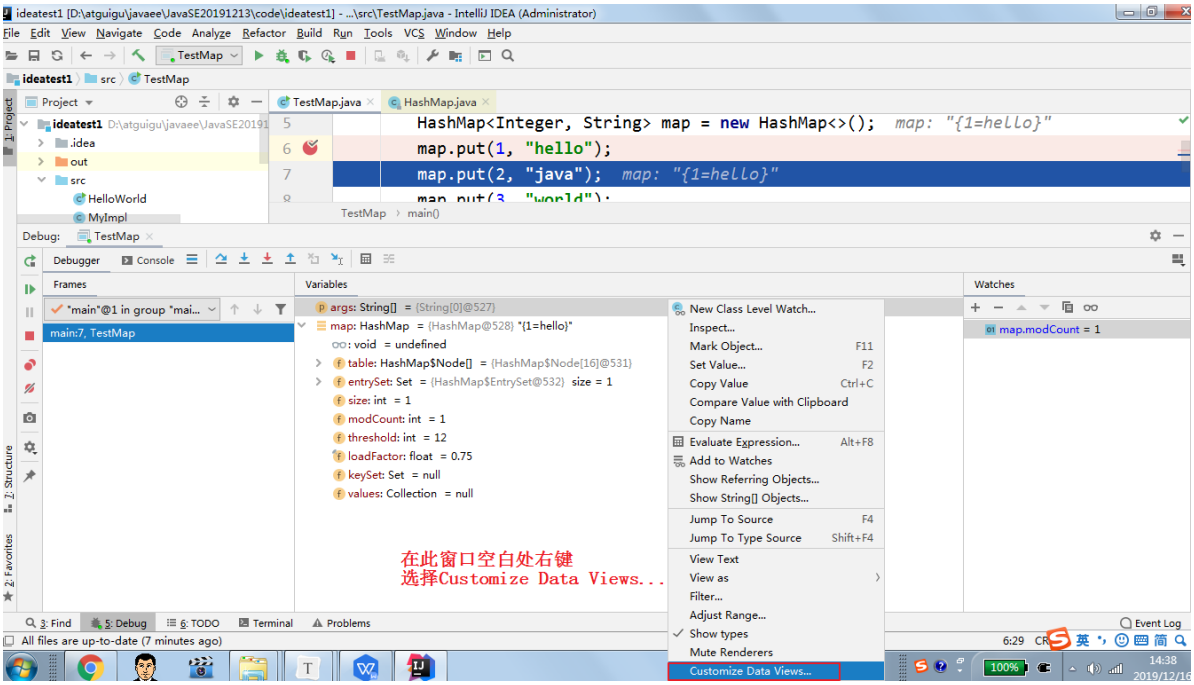
```
package com.atguigu.debug;

/**
 * ClassName: Debug07
 * Package: com.atguigu.debug
 * Description: 演示7：强制结束
 *
 * @Author: 尚硅谷-宋红康
 * @Create: 2022/10/20 23:15
 * @Version 1.0
 */
public class Debug07 {
    public static void main(String[] args) {
        System.out.println("获取请求的数据");
        System.out.println("调用写入数据库的方法");
        insert();
        System.out.println("程序结束");
    }

    private static void insert() {
        System.out.println("进入insert()方法");
        System.out.println("获取数据库连接");
        System.out.println("将数据写入数据表中");
        System.out.println("写出操作完成");
        System.out.println("断开连接");
    }
}
```

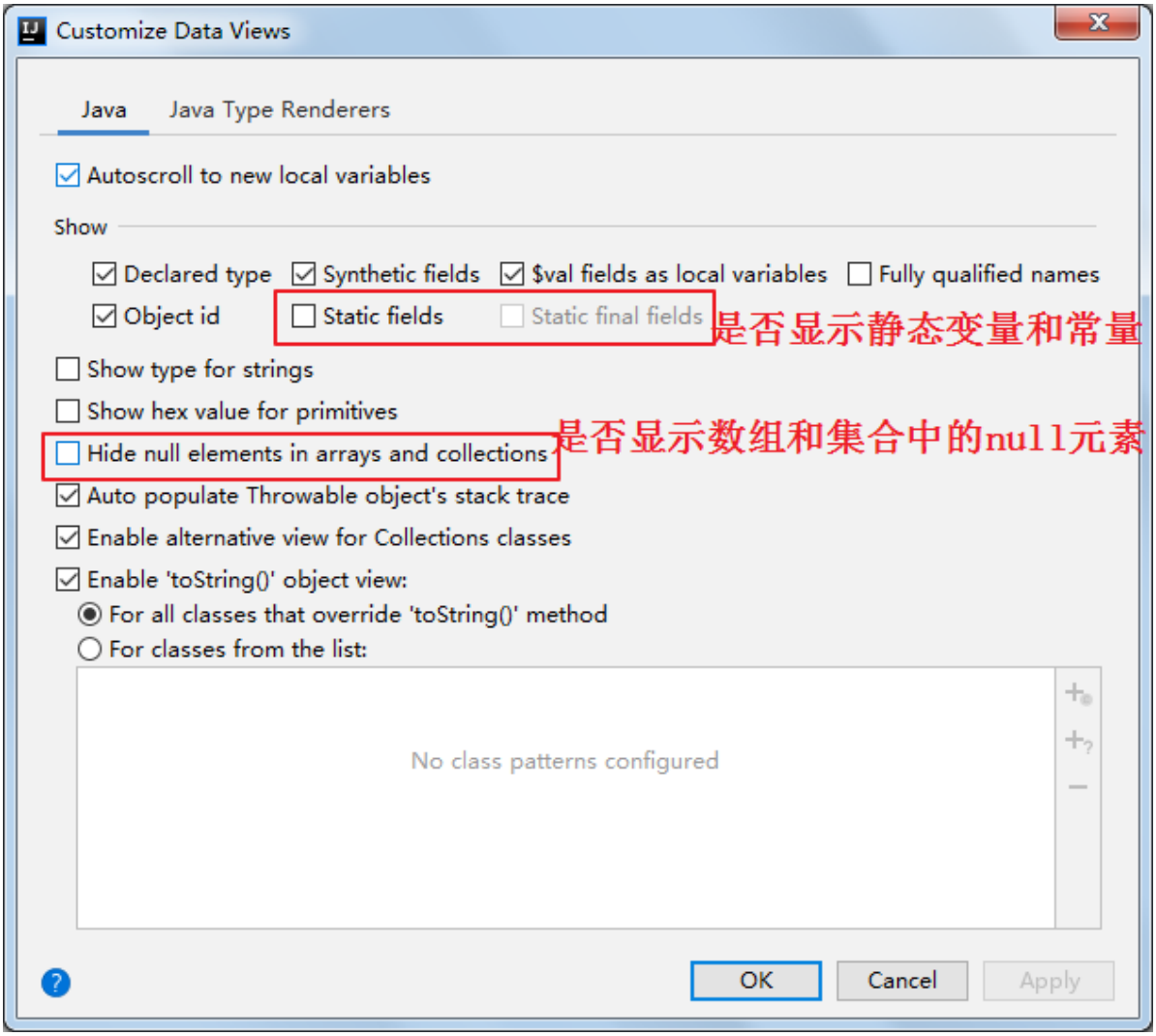


9.4 自定义调试数据视图

```
package com.atguigu.debug;

import java.util.HashMap;

/**
 * ClassName: Debug08
 * Package: com.atguigu.debug
 * Description: 演示8：用户自定义数据视图
 *
 * @Author: 尚硅谷-宋红康
 * @Create: 2022/10/20 23:21
 * @Version 1.0
 */
public class Debug08 {
    public static void main(String[] args) {
        HashMap<Integer,String> map = new HashMap<>();
        map.put(1,"高铁");
        map.put(2,"网购");
        map.put(3,"支付宝");
        map.put(4,"共享单车");

        System.out.println(map);
    }
}
```

设置如下：





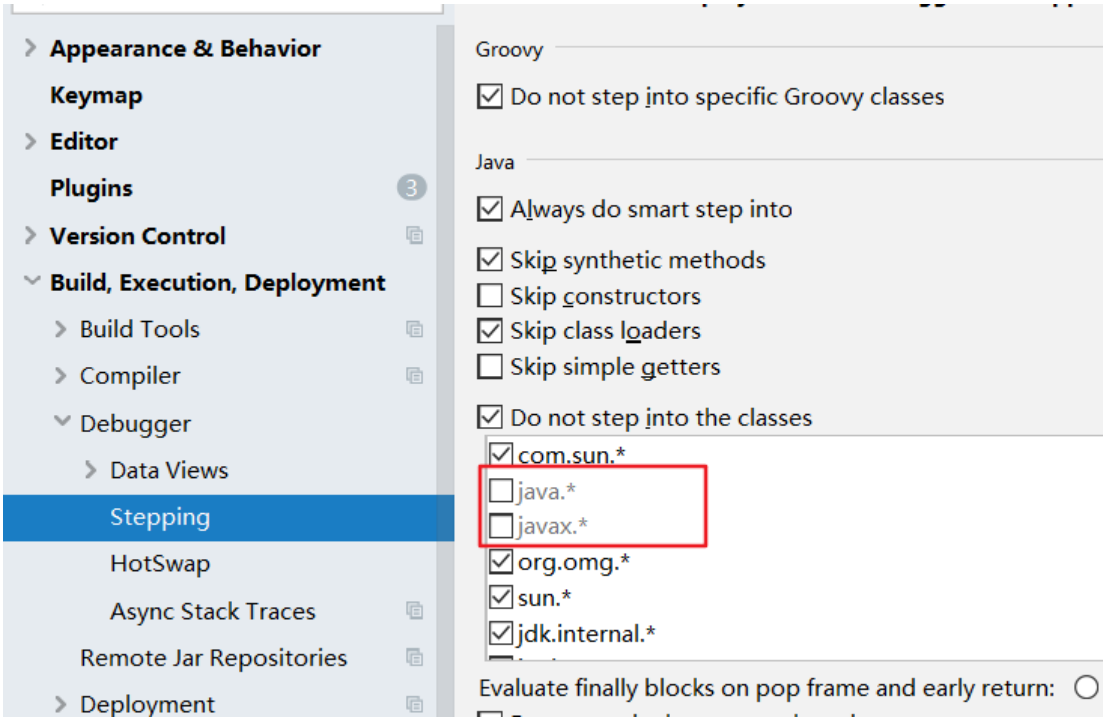
9.5 常见问题

问题：使用Step Into时，会出现无法进入源码的情况。如何解决?

方案1：使用 force step into 即可

方案2：点击Setting -> Build,Execution,Deployment -> Debugger -> Stepping

把Do not step into the classess中的 `java.*`、`javax.*` 取消勾选即可。



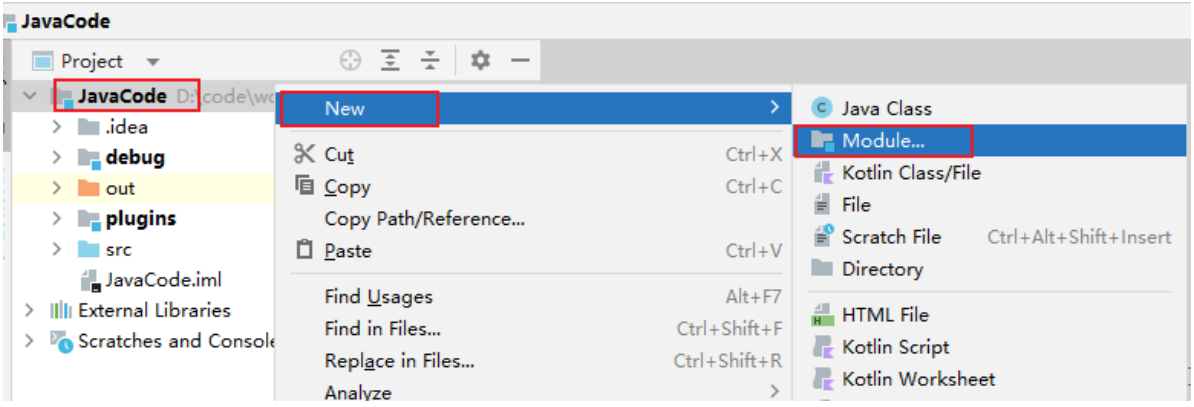
小结：

> 经验：初学者对于在哪里加断点，缺乏经验，这也是调试程序最麻烦的地方，需要一定的经验。
>
> 简单来说，在可能发生错误的代码的前面加断点。如果不会判断，就在程序执行的起点处加断点。

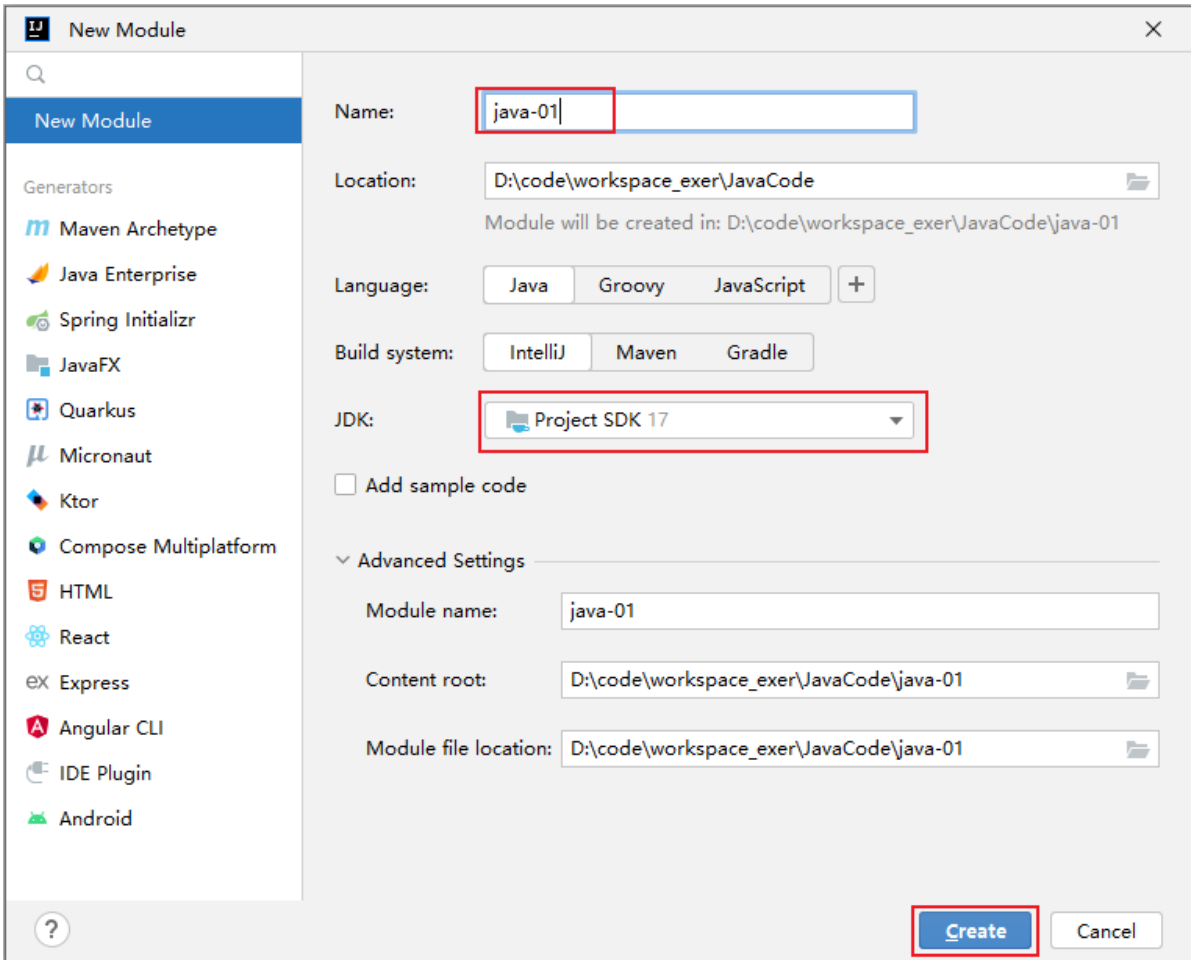
10. 创建不同类型的工程

10.1 创建Java工程

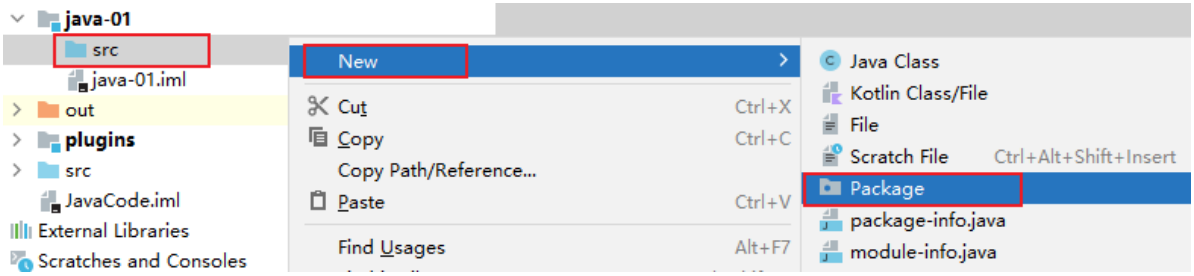
在工程上，右键- New - Module，如下：



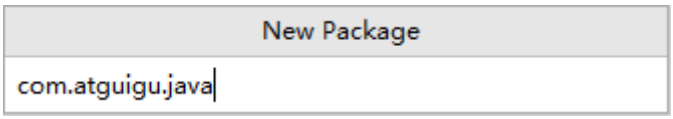
指明Java工程的名称及使用的JDK版本：



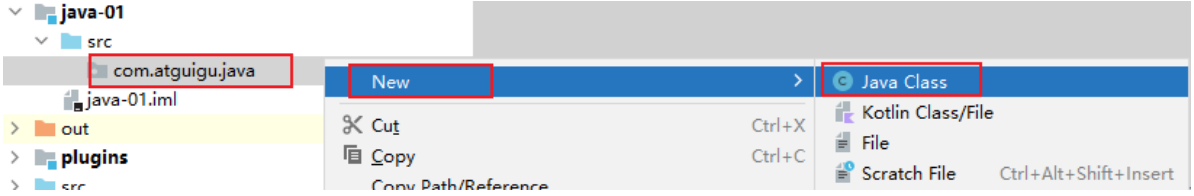
创建包：



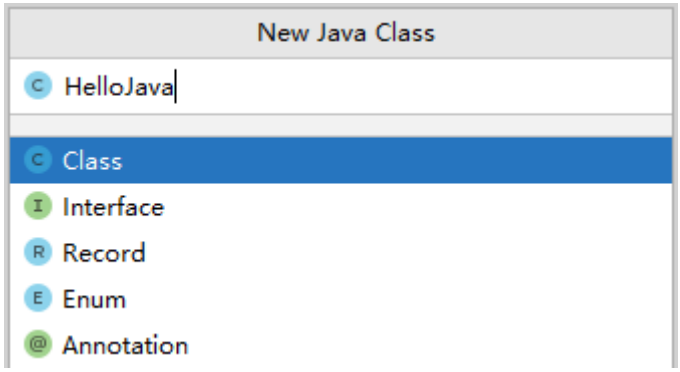
提供包名：



在包下创建类，即可：



提供类名：



测试代码：

HelloJava.java

```java
package com.atguigu.java;

/**
 * ClassName: HelloJava
 * Package: com.atguigu.java
 * Description:
 *
 * @Author: 尚硅谷-宋红康
 * @Create: 2022/10/20 11:44
 * @Version 1.0
 */
public class HelloJava {
    public static void main(String[] args) {
        System.out.println("hello,atguigu!");
    }
}
```

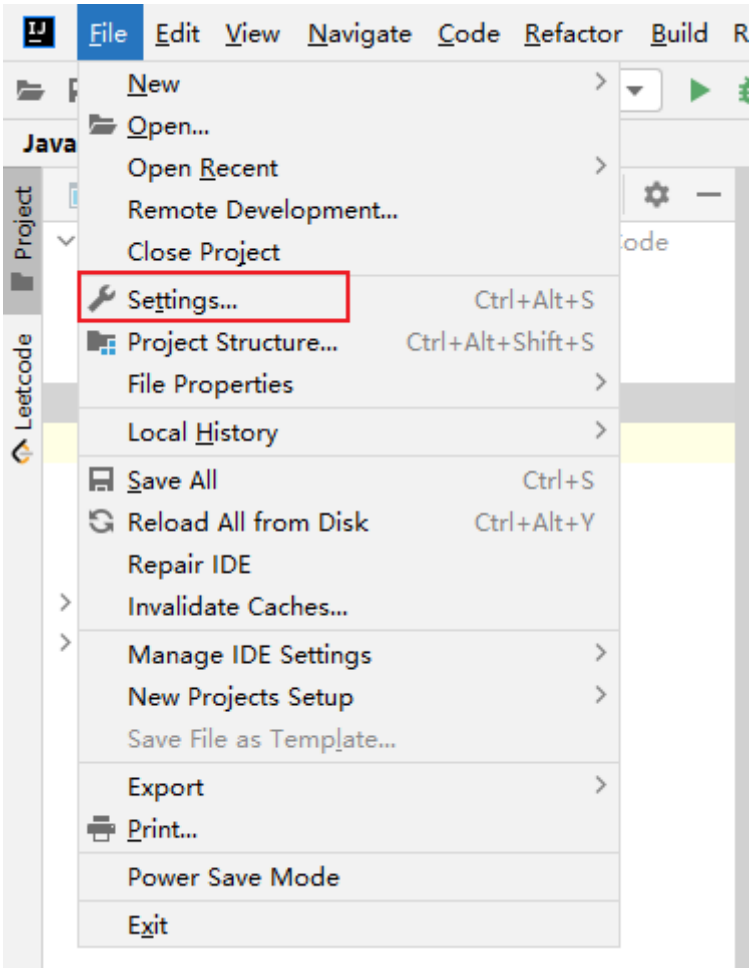
点击运行即可：

10.2 创建Java Web工程

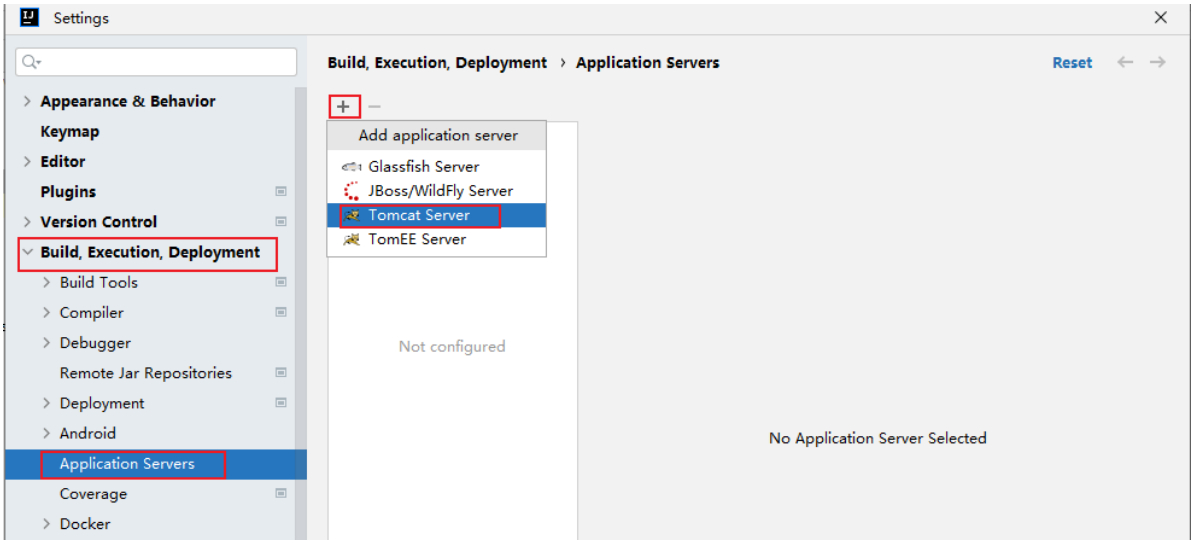
10.2.1 IDEA中配置Tomcat

在IDEA中配置Tomcat之前，需要保证已经安装并配置了Tomcat的环境变量。如果没有安装并配置，可以参考《 尚硅谷_宋红康_Tomcat8.5快速部署.docx 》配置完成以后，在命令行输入： **catalina run** 。能够启动tomcat，则证明安装配置成功。

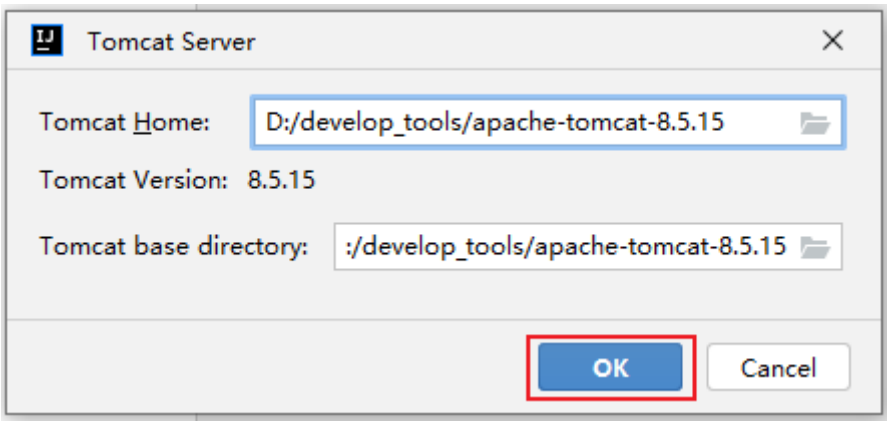
下面看如何在IDEA中配置：



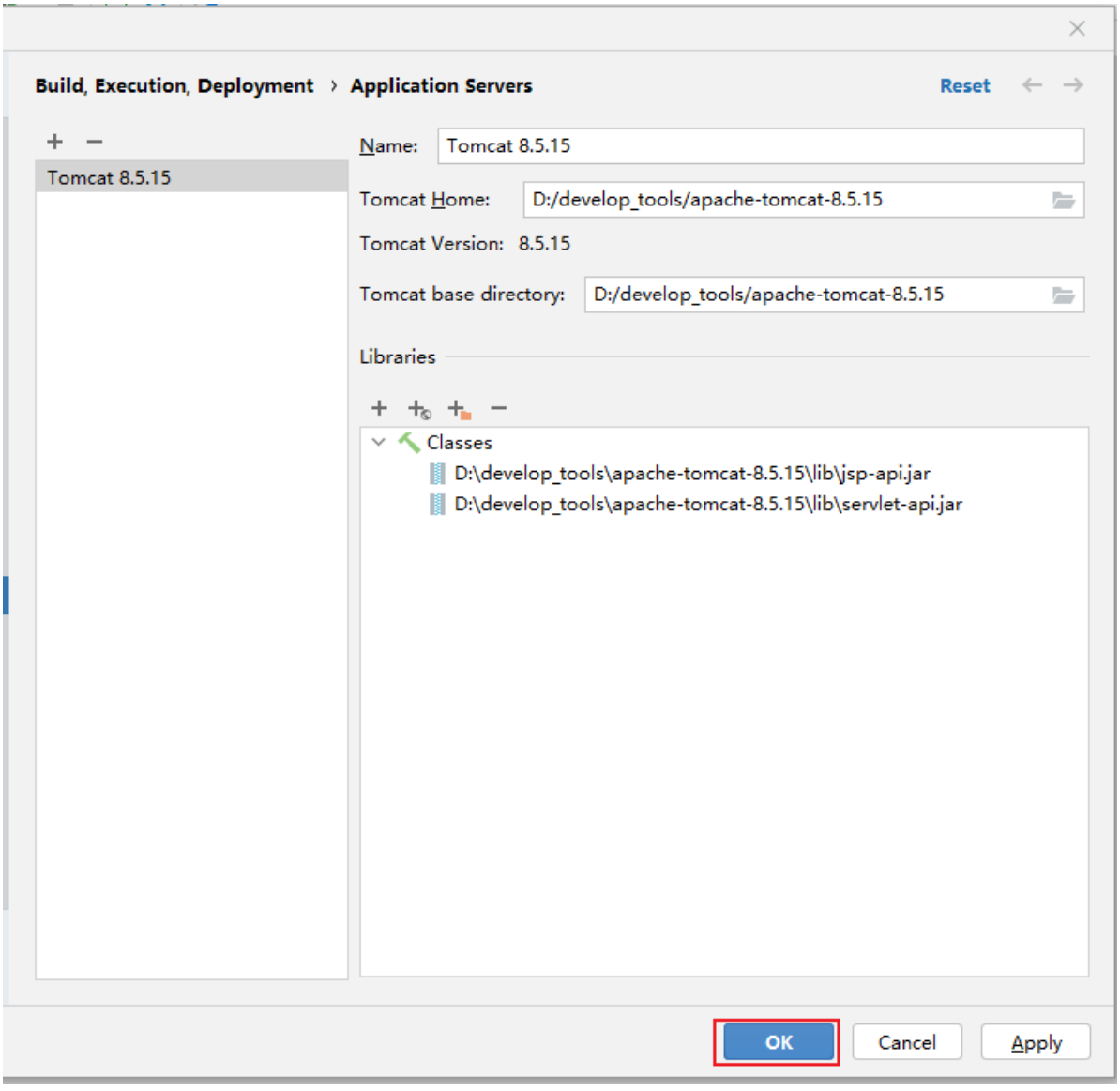
配置Tomcat Server的位置：



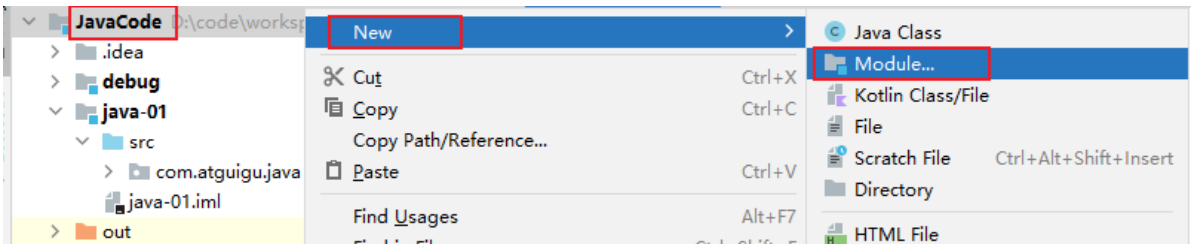
这里配置Tomcat的名称以及配置应用服务器的位置。根据自己Tomcat的安装位置决定。



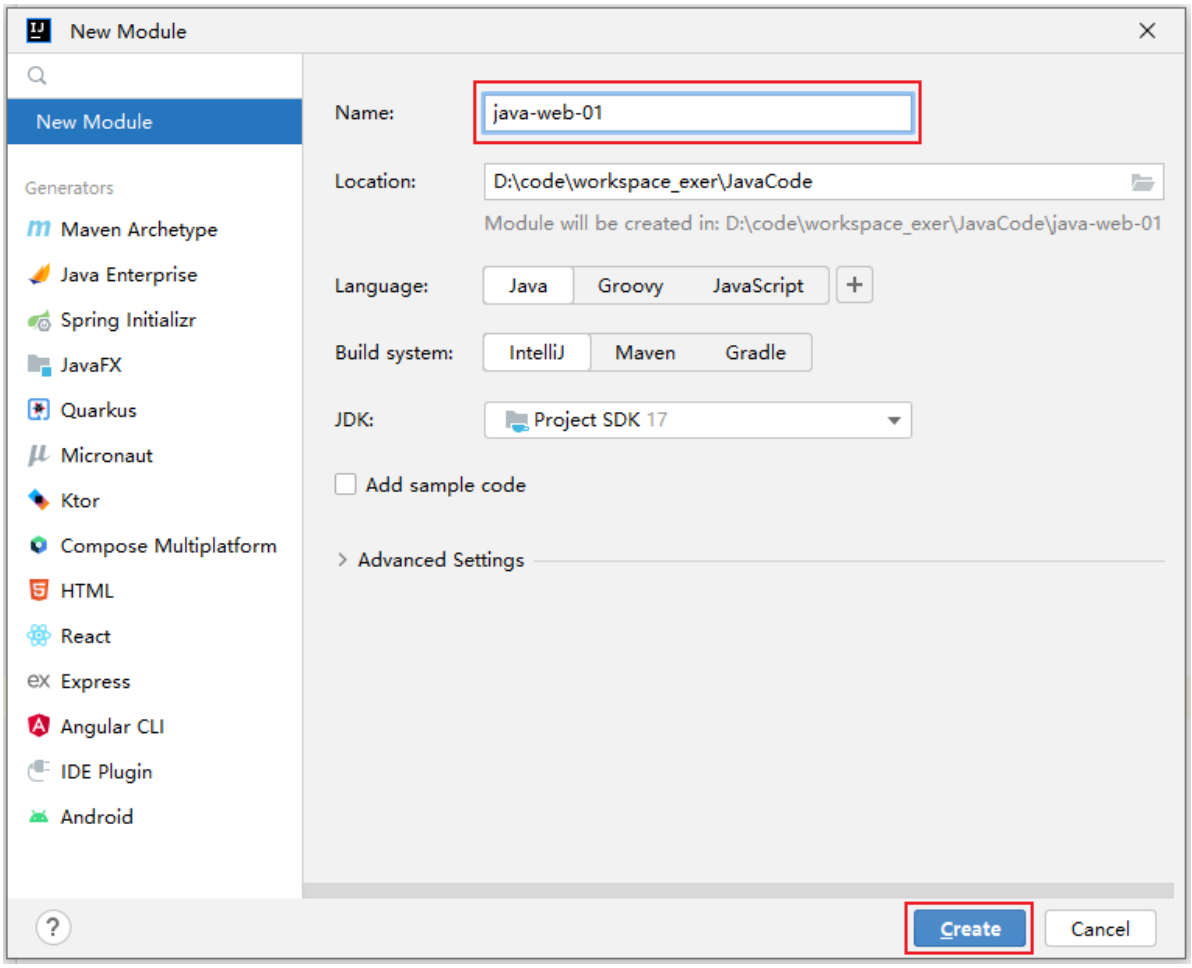
配置好后，如下图所示：



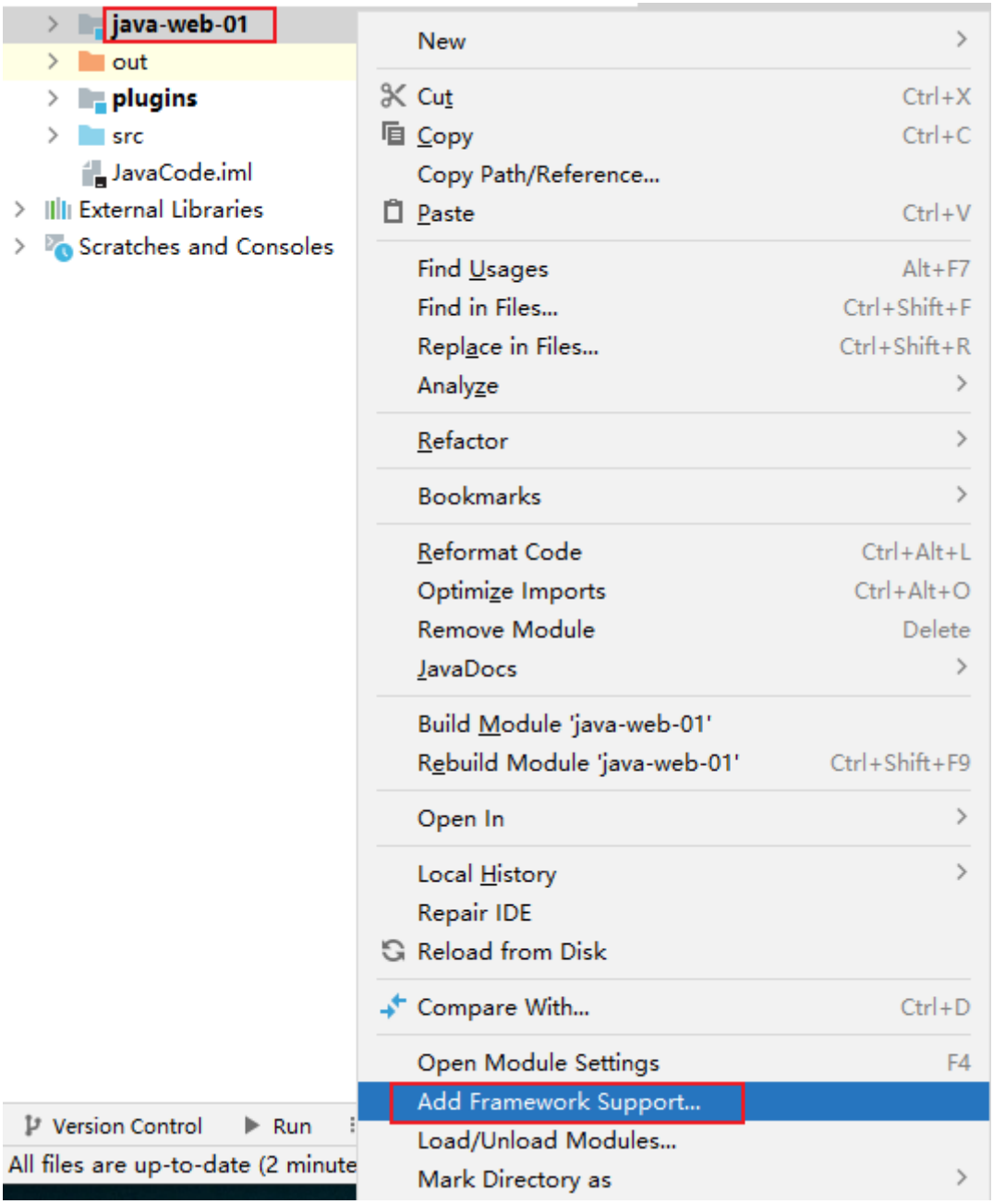
10.2.2 创建Web工程



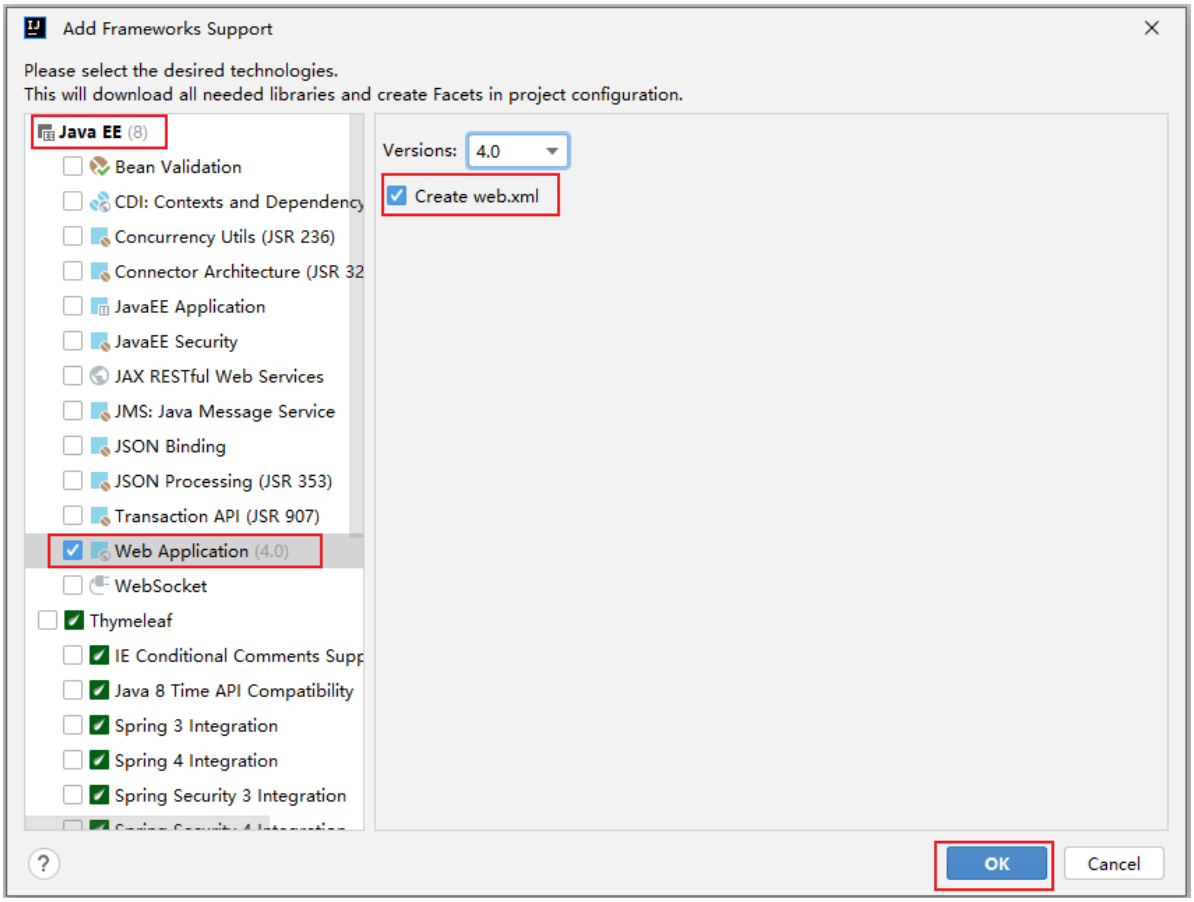
选择New Module，指明当前工程的名称：



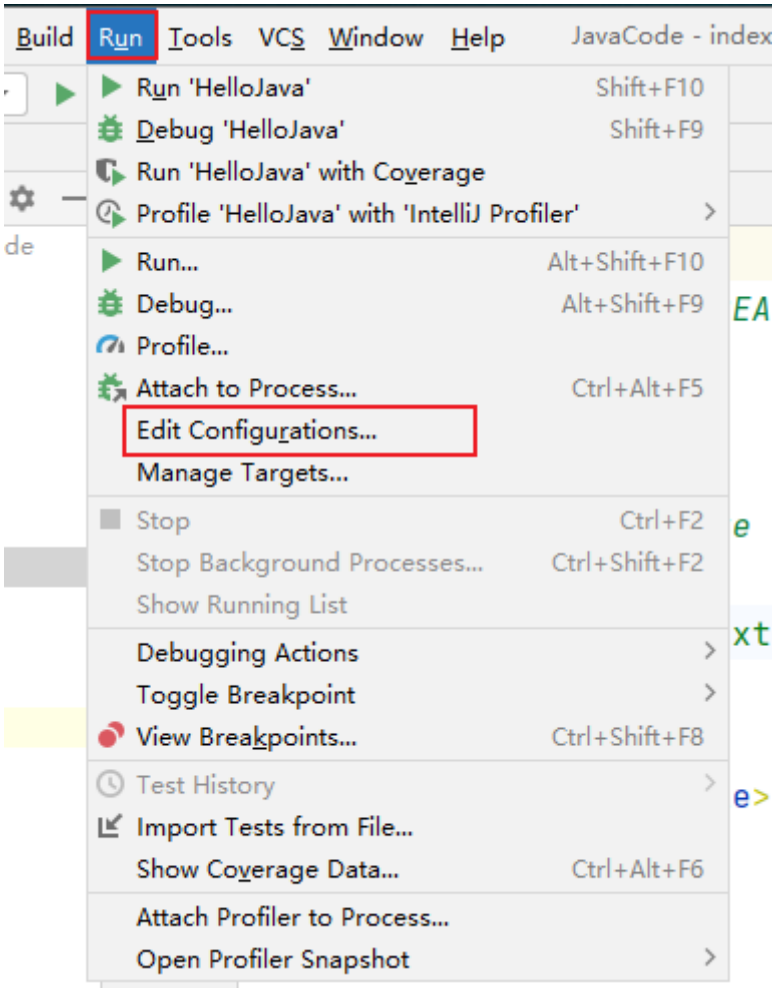
选中当前创建的工程，添加框架支持：

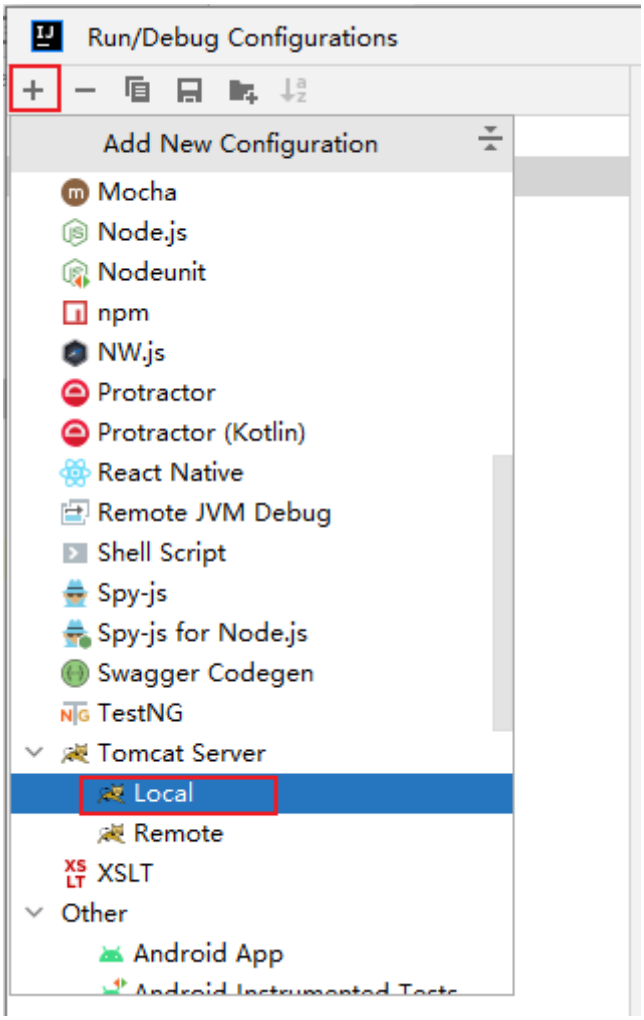


选择：Web Application，选择Create web.xml，如下：

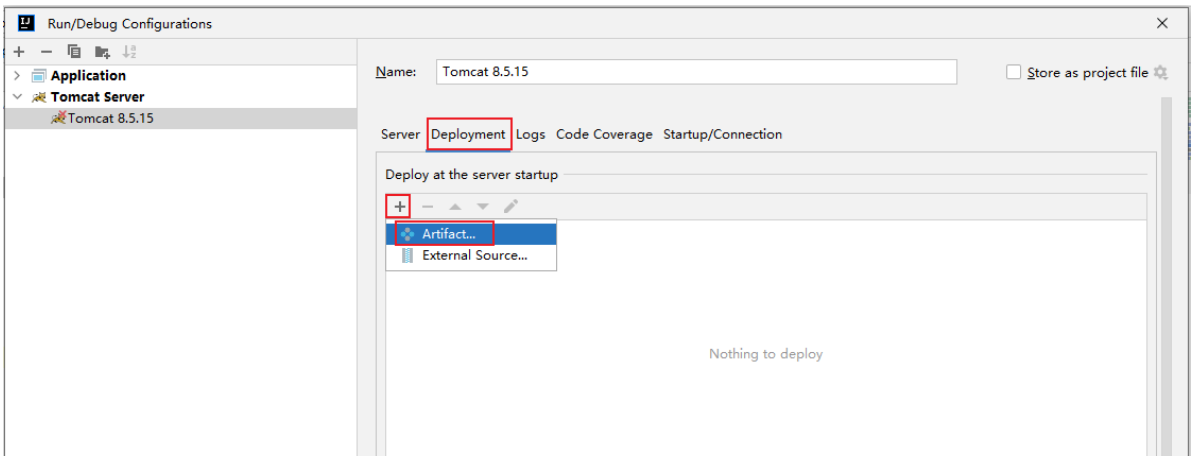


10.2.3 配置web工程并运行

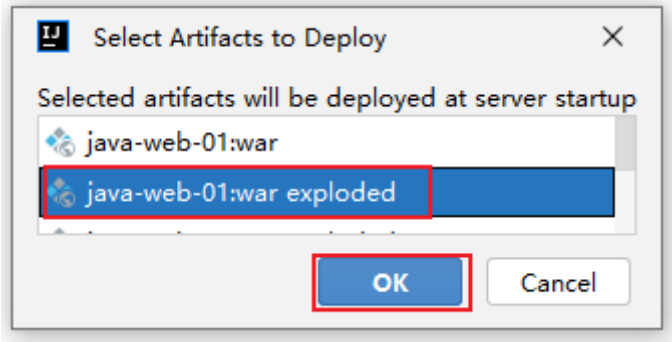




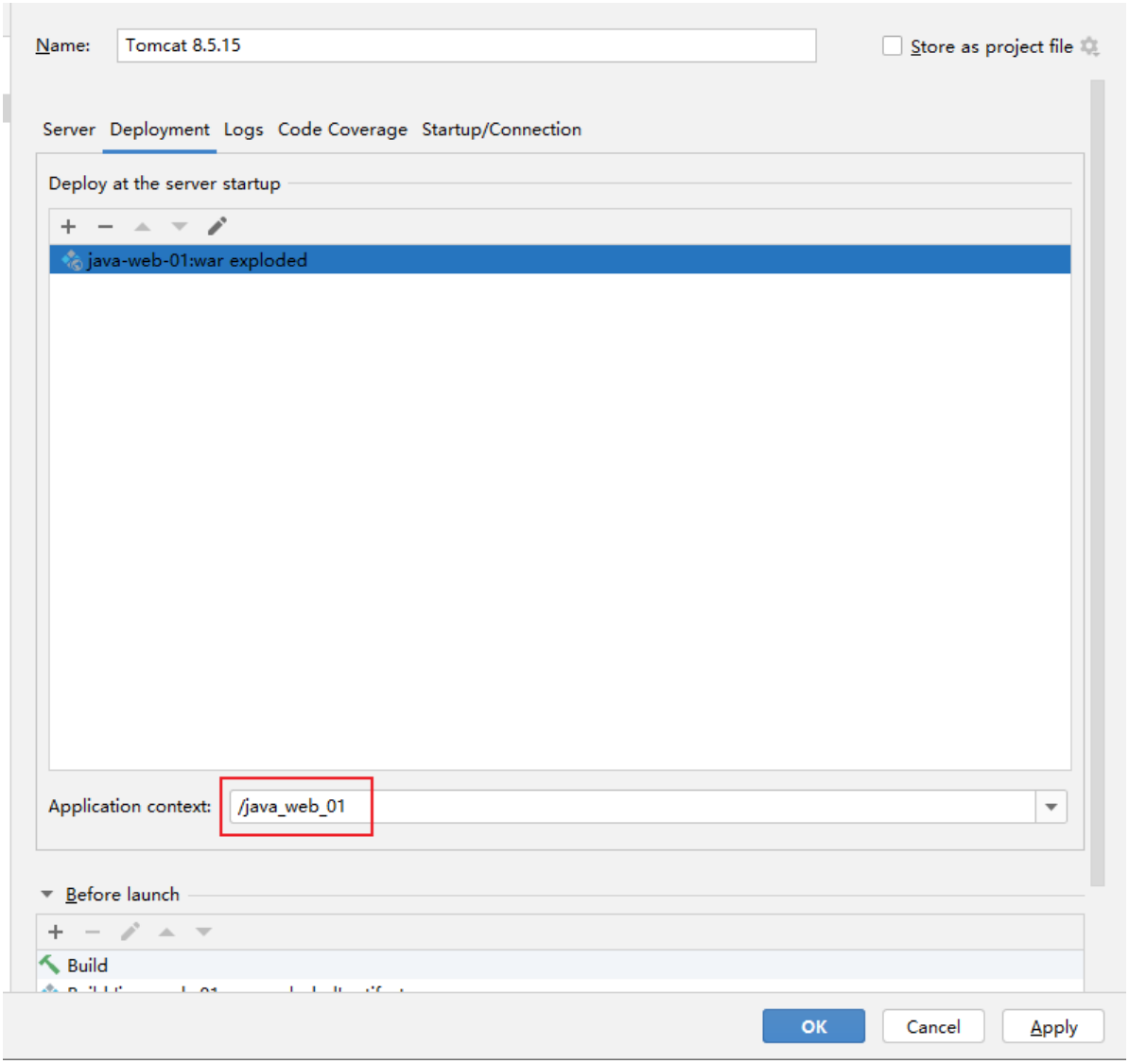
部署当前的web项目：



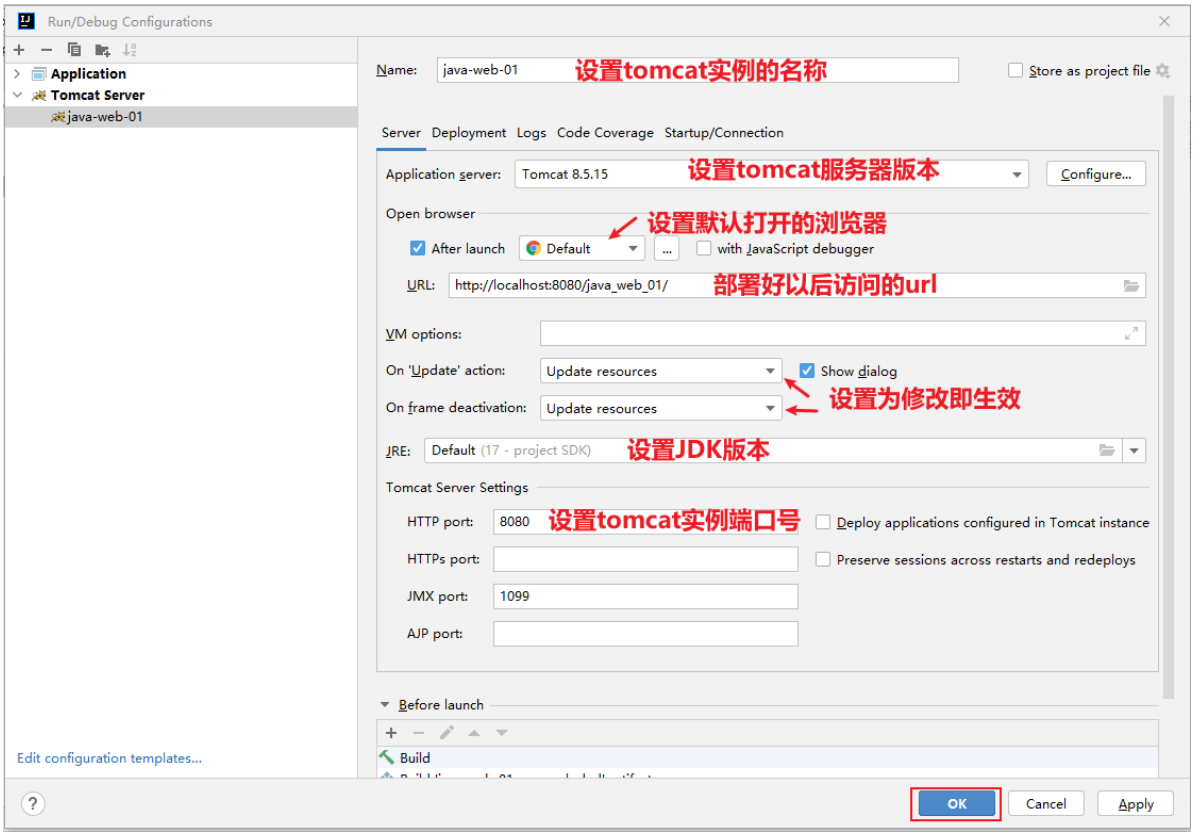
选择第2项：



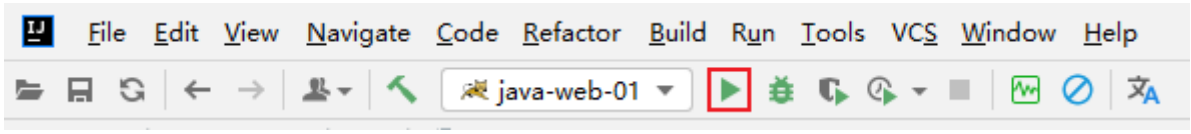
可以修改Application context，如下：



配置当前web工程的详细信息，如下：

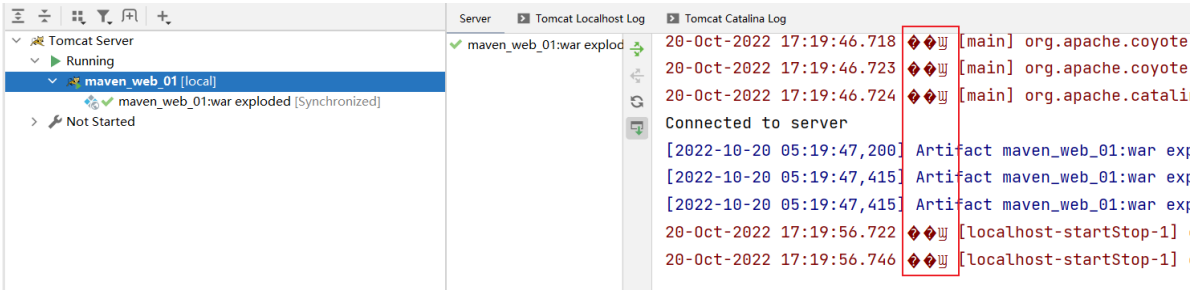


配置好后，可以直接运行：



10.2.4 乱码的解决

如果Tomcat日志出现乱码，需要配置：



解决方案：

1) 点击Help => Edit custom VM Options，在最后面添加

```
-Dfile.encoding=UTF-8
```

2) 在当前Tomcat实例中配置 VM option，添加

```
-Dfile.encoding=UTF-8
```

在第二步的Startup/Connection页签的Run和Debug添加一个key为 `JAVA_TOOL_OPTIONS` ， value为“ `-Dfile.encoding=UTF-8` ”的环境变量

3) 保存后重启IDEA，可以发现控制台中文乱码显示正常了。

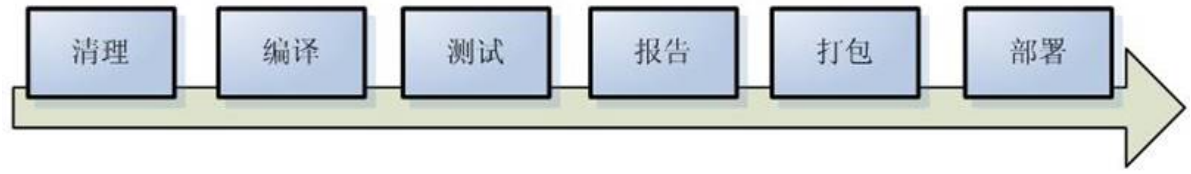
10.3 创建Maven Java工程

10.3.1 Maven的介绍

Maven是一款自动化构建工具，专注服务于Java平台的 项目构建 和 依赖管理 。在JavaEE开发的历史上构建工具的发展也经历了一系列的演化和变迁：

> Make→Ant→Maven→Gradle→其他……

构建环节：

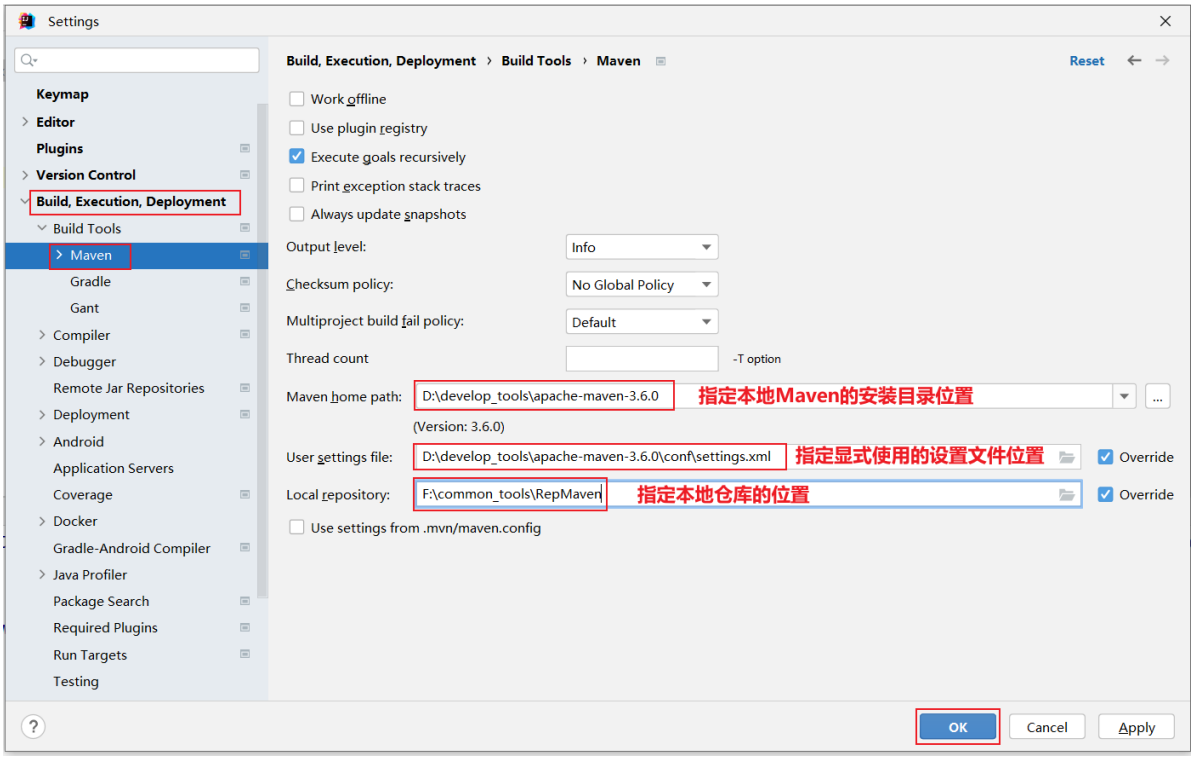


> ①清理：删除以前的编译结果，为重新编译做好准备。

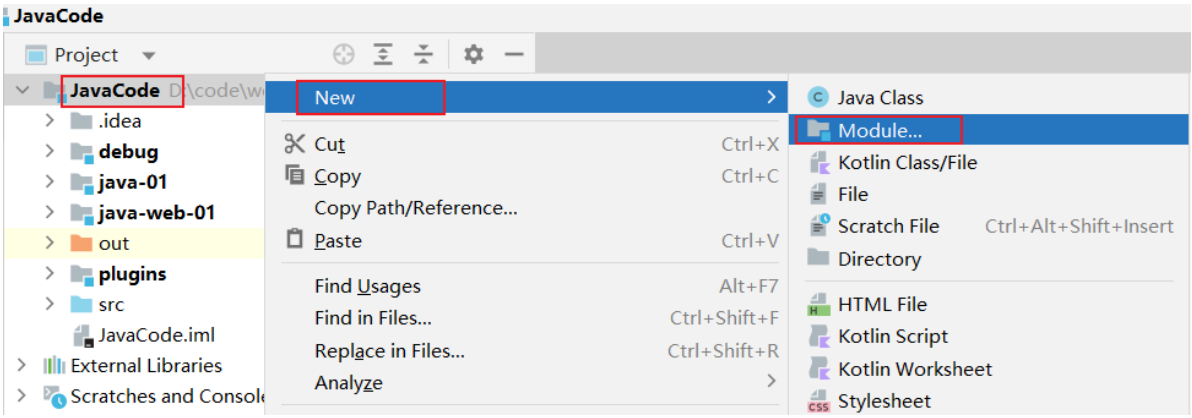
> ②编译：将Java源程序编译为字节码文件。
>
> ③测试：运行单元测试用例程序，确保项目在迭代开发过程中关键点的正确性。
>
> ④报告：测试程序的结果。
>
> ⑤打包：将java项目打成jar包；将Web项目打成war包。
>
> ⑥安装：将jar包或war包安装到本地仓库中。
>
> ⑦部署：将jar或war从Maven仓库中部署到Web服务器上运行。

10.3.2 Maven的配置

maven的下载 – 解压 – 环境变量的配置这里就不赘述了，需要的参考03-资料\05-Maven的配置中的《 尚硅谷_Maven的配置_V2.0.docx 》。下面直接整合Maven。选择自己Maven的目录，和settings文件，然后配置自己的仓库reposiroty。

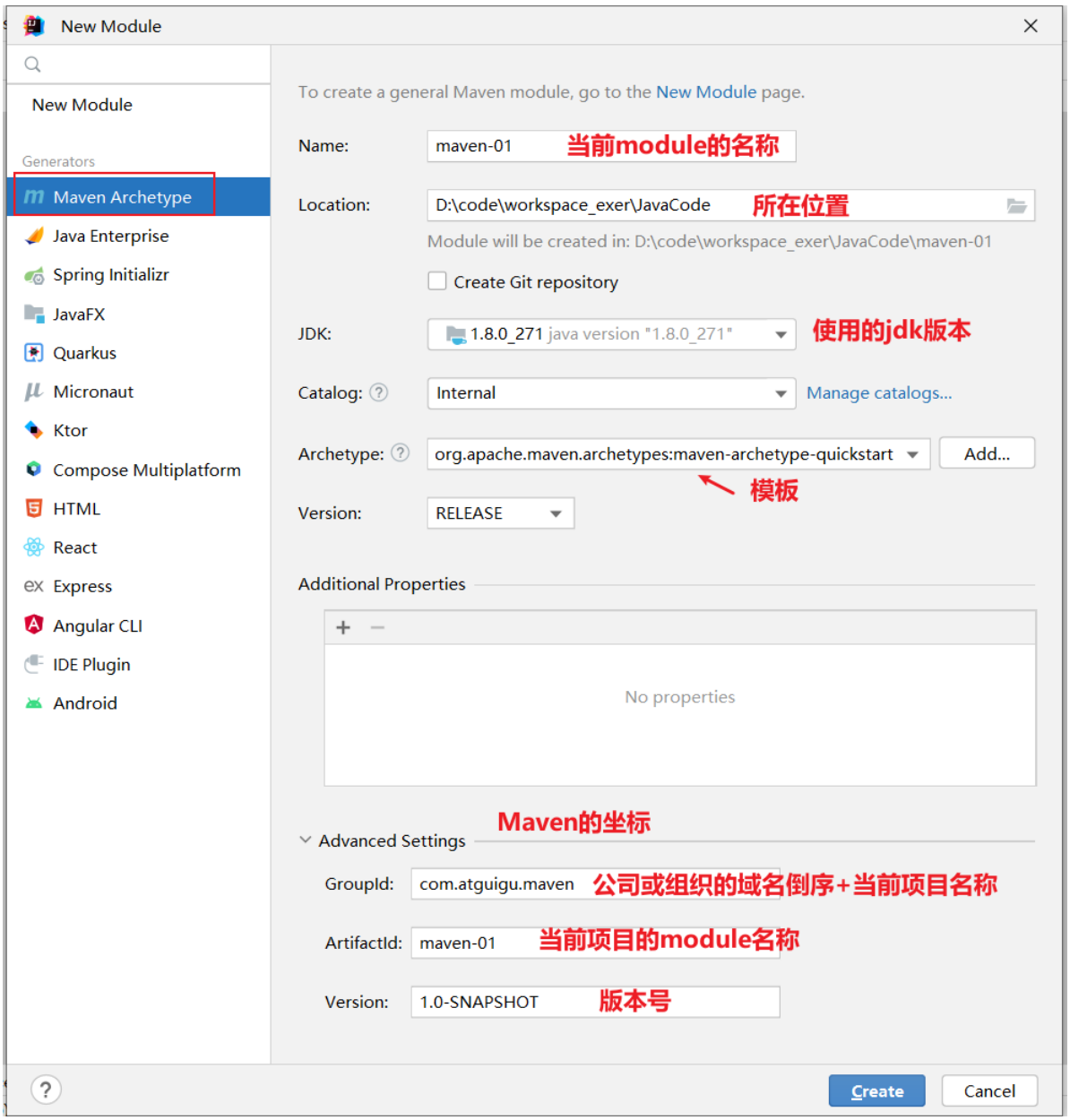


10.3.3 Maven Java工程的创建

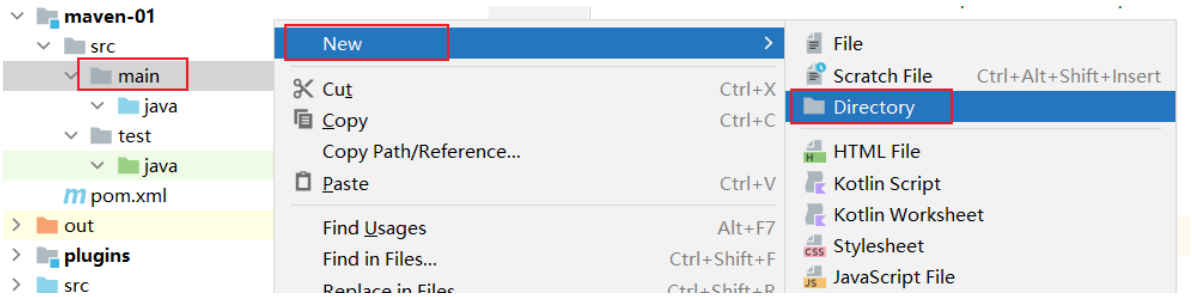


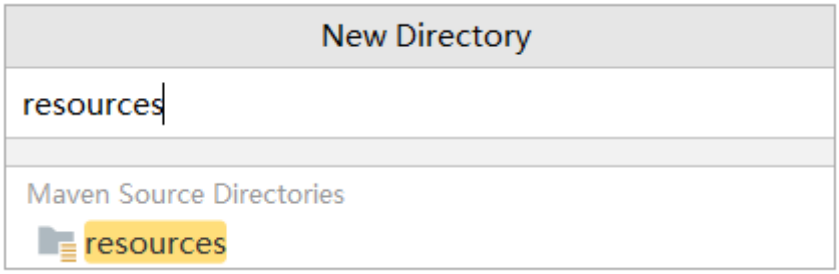
指明当前maven工程的名称、模板等信息。这里要求一个项目组的jdk版本必须一致。

通过坐标，就可以定位仓库中具体的jar包。如下：

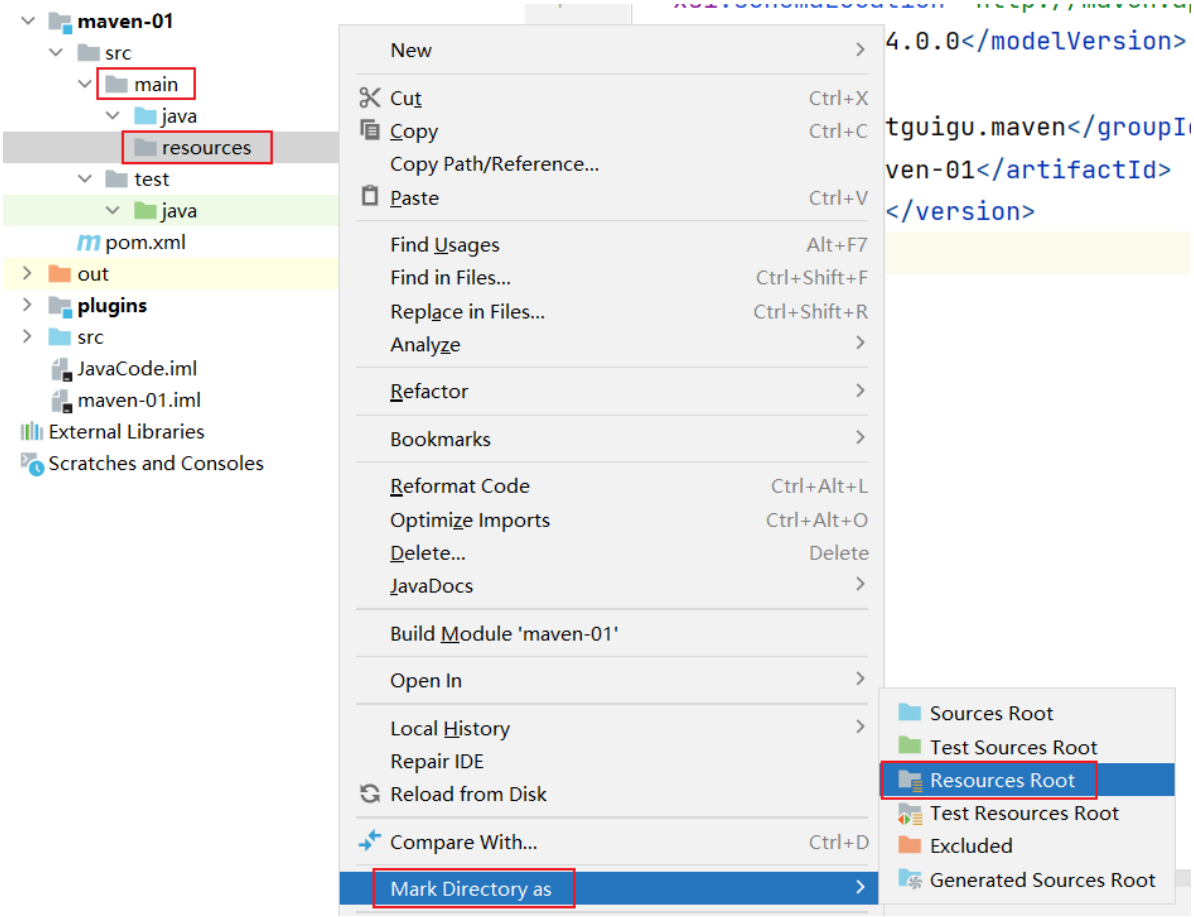


新创建的maven 的java工程缺少相应的resources文件目录，需要创建如下：





指明main下resources的文件目录类型：



类似的操作test目录下，提供resources即可。

这里说明Maven的java工程的目录结构：

```
工程名
    src
    ----main
    --------java
    --------resources
    ----test
    --------java
    --------resources
    pom.xml
```

- main目录用于存放主程序。
- test目录用于存放测试程序。
- java目录用于存放源代码文件。
- resources目录用于存放配置文件和资源文件。

10.3.4 编写代码及测试

第1步：创建Maven的核心配置文件pom.xml

```
<?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.atguigu.maven</groupId>
    <artifactId>maven-01</artifactId>
```

```
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.18</version>
        </dependency>

    </dependencies>

</project>
```

第2步：编写主程序代码

在src/main/java/com/atguigu/java目录下新建文件HelloMaven.java

```
package com.atguigu.java;

/**
 * ClassName: HelloMaven
 * Package: com.atguigu.java
 * Description:
 *
 * @Author: 尚硅谷-宋红康
 * @Create: 2022/10/20 18:20
 * @Version 1.0
 */
public class HelloMaven {
    public String sayHello(String message) {
        return "Hello, " + message + "!";
    }
}
```

第3步：编写测试代码

在/src/test/java/com/atguigu/java目录下新建测试文件HelloMavenTest.java

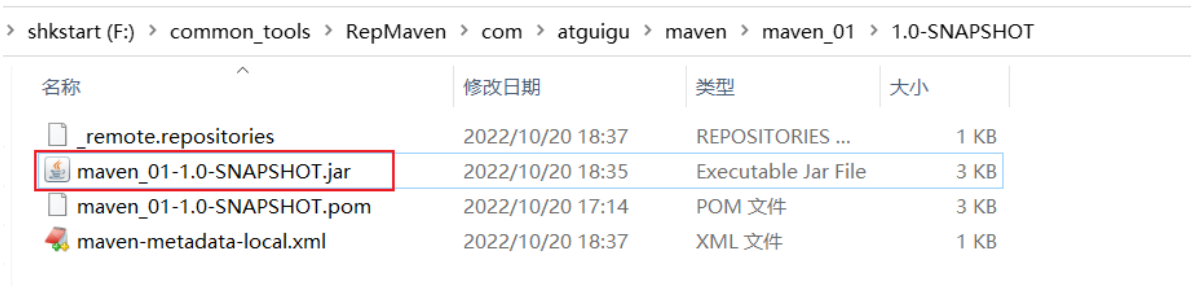
```
package com.atguigu.java;

import org.junit.Test;

/**
 * ClassName: HelloMavenTest
 * Package: com.atguigu.java
 * Description:
 *
 * @Author: 尚硅谷-宋红康
 * @Create: 2022/10/20 18:21
 * @Version 1.0
 */
```

```java
public class HelloMavenTest {

    @Test
    public void testHelloMaven() {
        HelloMaven helloMaven = new HelloMaven();
        System.out.println(helloMaven.sayHello("Maven"));
    }
}
```

第4步：运行几个基本的Maven命令

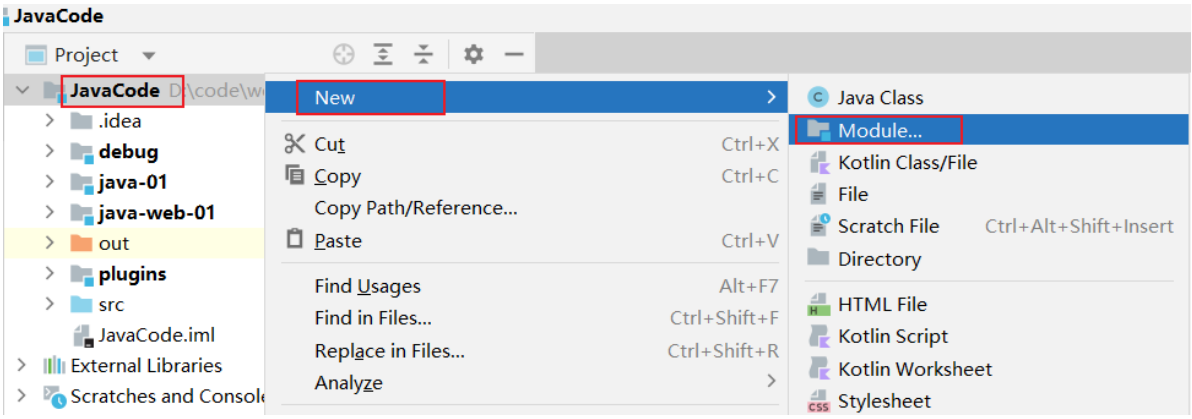
目录下也会有对应的生命周期。其中常用的是：clean、compile、package、install。

比如这里install，如果其他项目需要将这里的模块作为依赖使用，那就可以install。安装到本地仓库的位置。

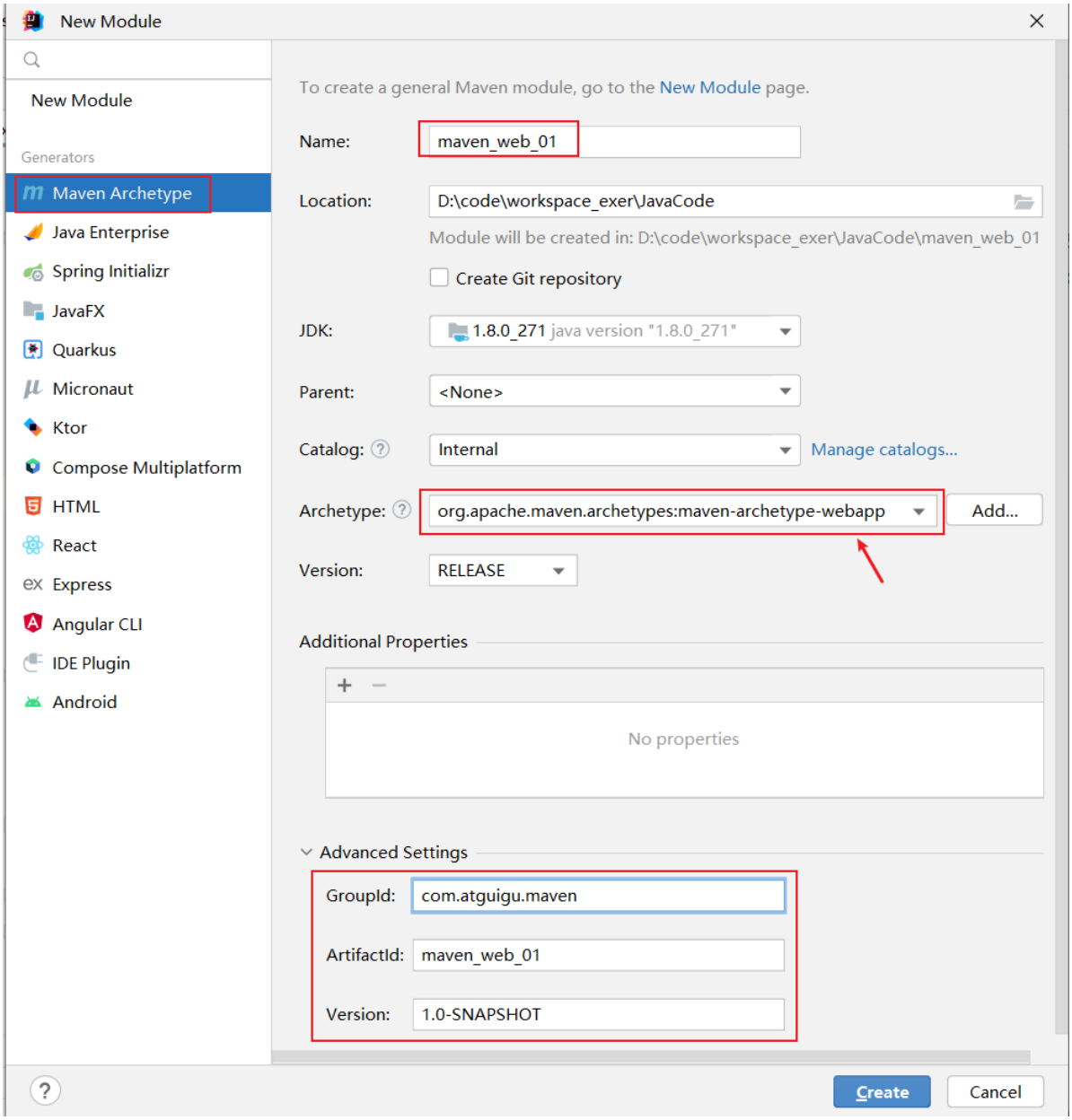


10.4 创建Maven Web工程

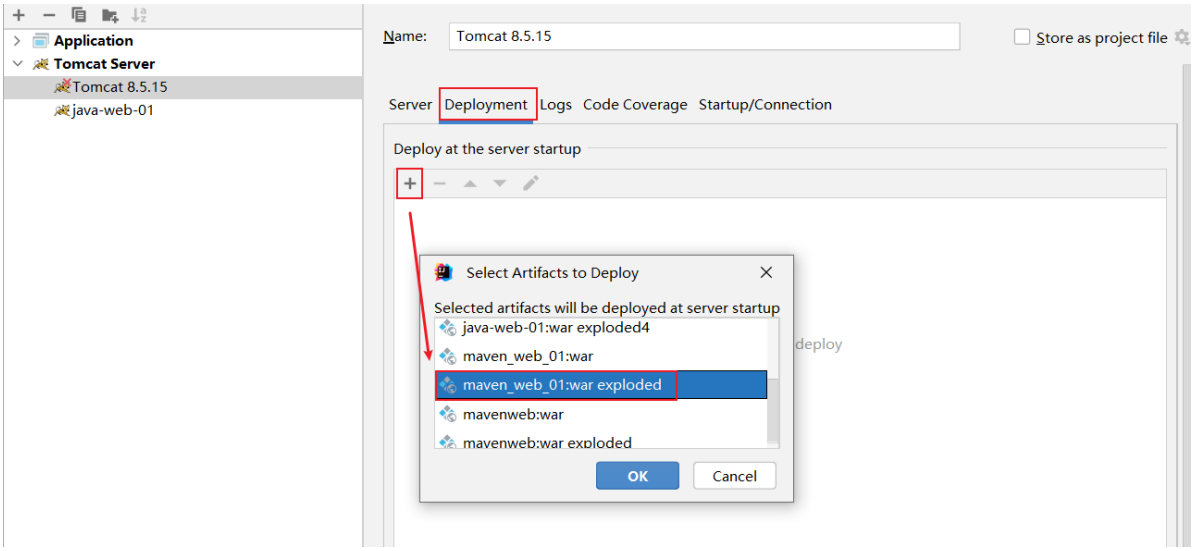
10.4.1 创建Maven的Web工程步骤



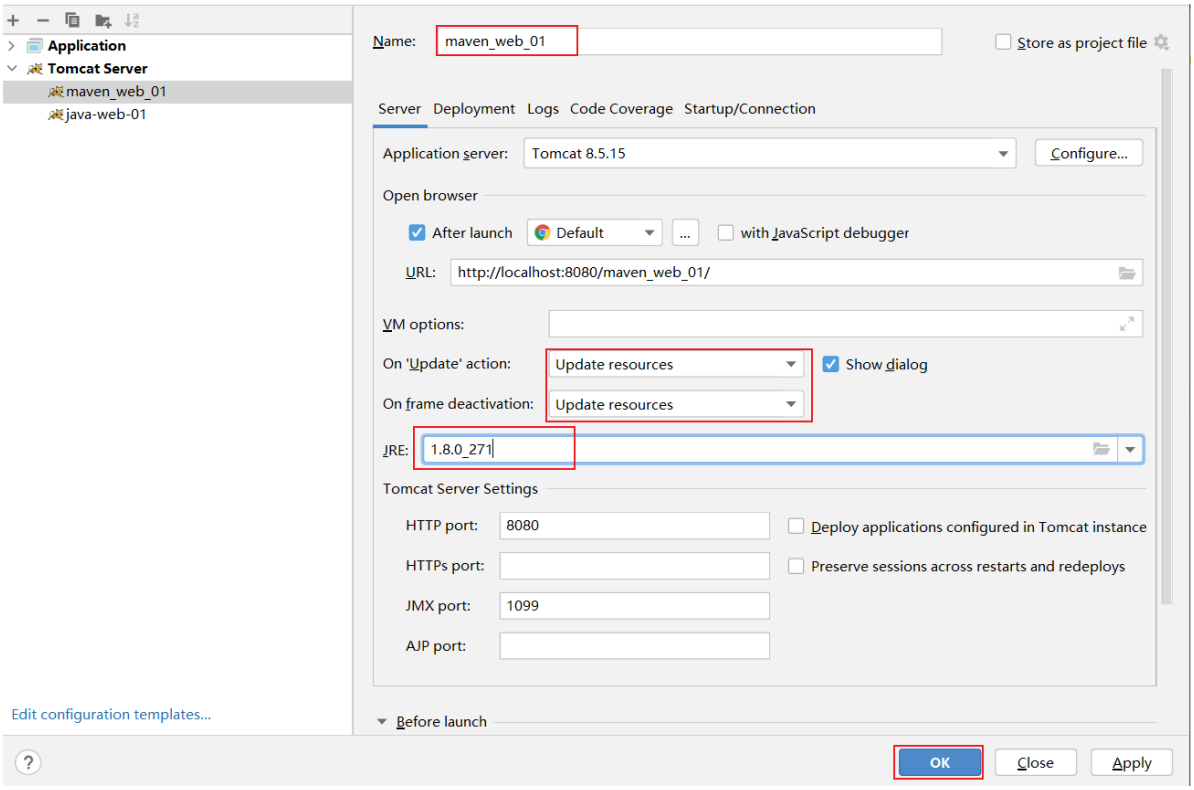
指明Maven的web工程的名称和模板。如下：



在Tomcat上进行部署：



配置部署的详细信息：



10.4.2 开发jsp依赖jar包

1、找不到HttpServlet错误

如果看到JSP报错：`The superclass "javax.servlet.http.HttpServlet" was not found on the Java Build Path`可以加入如下依赖解决。

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
</dependency>
```

2、EL表达式没有提示问题

${pageContext}这个EL表达式中通过pageContext对象访问reuqest属性时本身是应该有提示的，但如果没有的话加入下面依赖即可。

```
<dependency>
     <groupId>javax.servlet.jsp</groupId>
     <artifactId>jsp-api</artifactId>
     <version>2.1.3-b06</version>
     <scope>provided</scope>
</dependency>
```

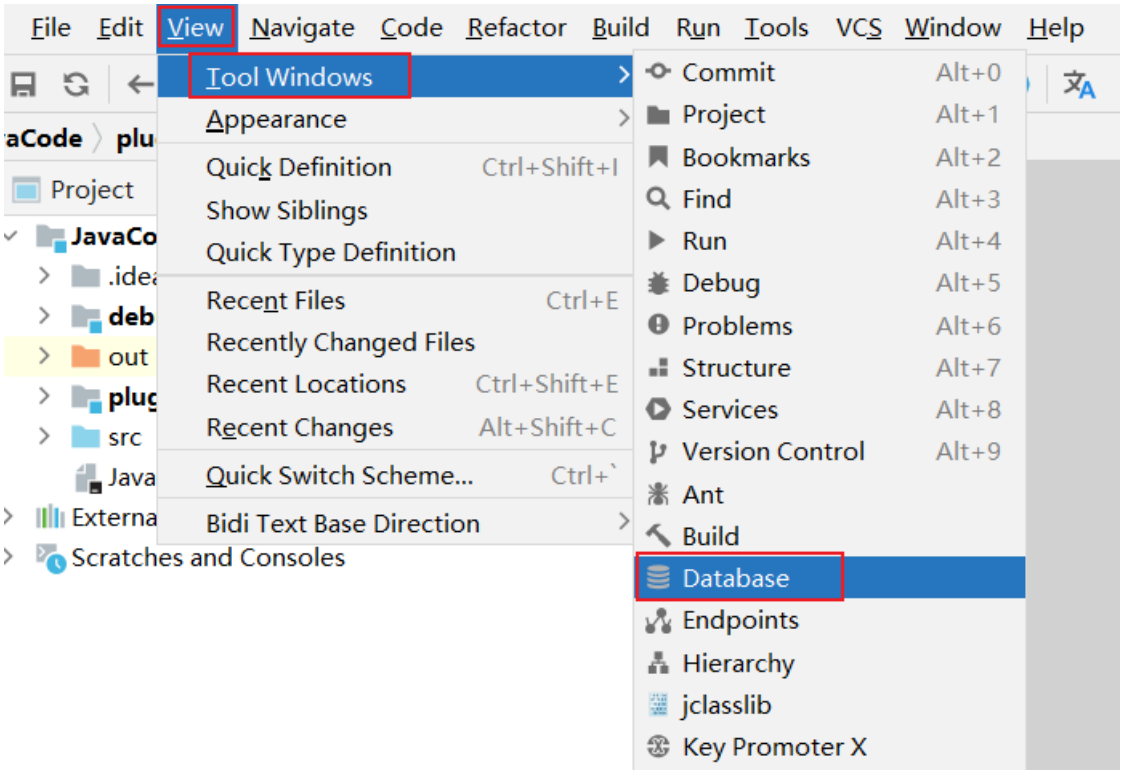
同时，针对index.jsp文件，修改一下文件头信息为：

```
<%@page language="java" pageEncoding="utf-8" contentType="text/html;UTF-8" %>
```

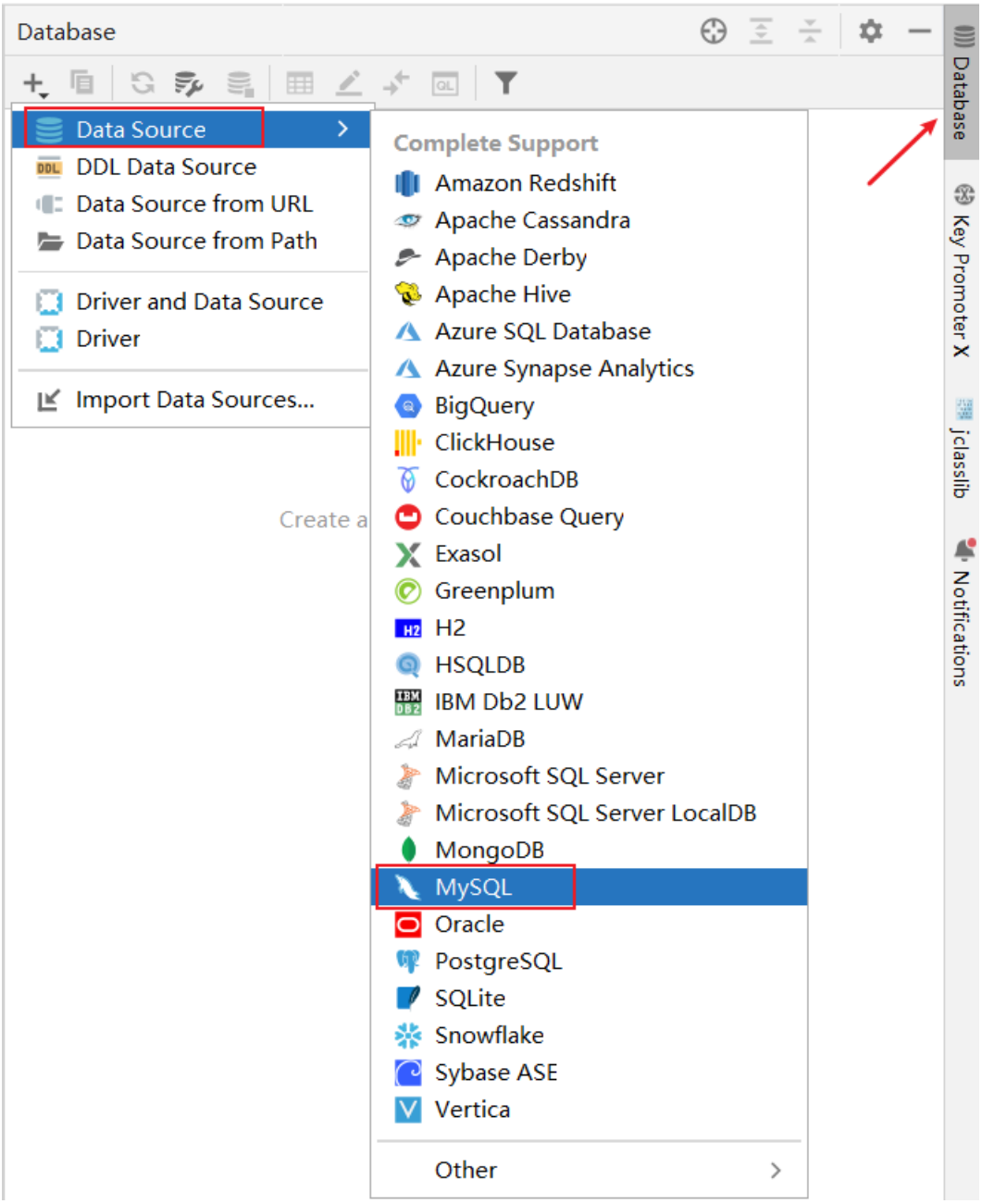
11. 关联数据库

11.1 关联方式

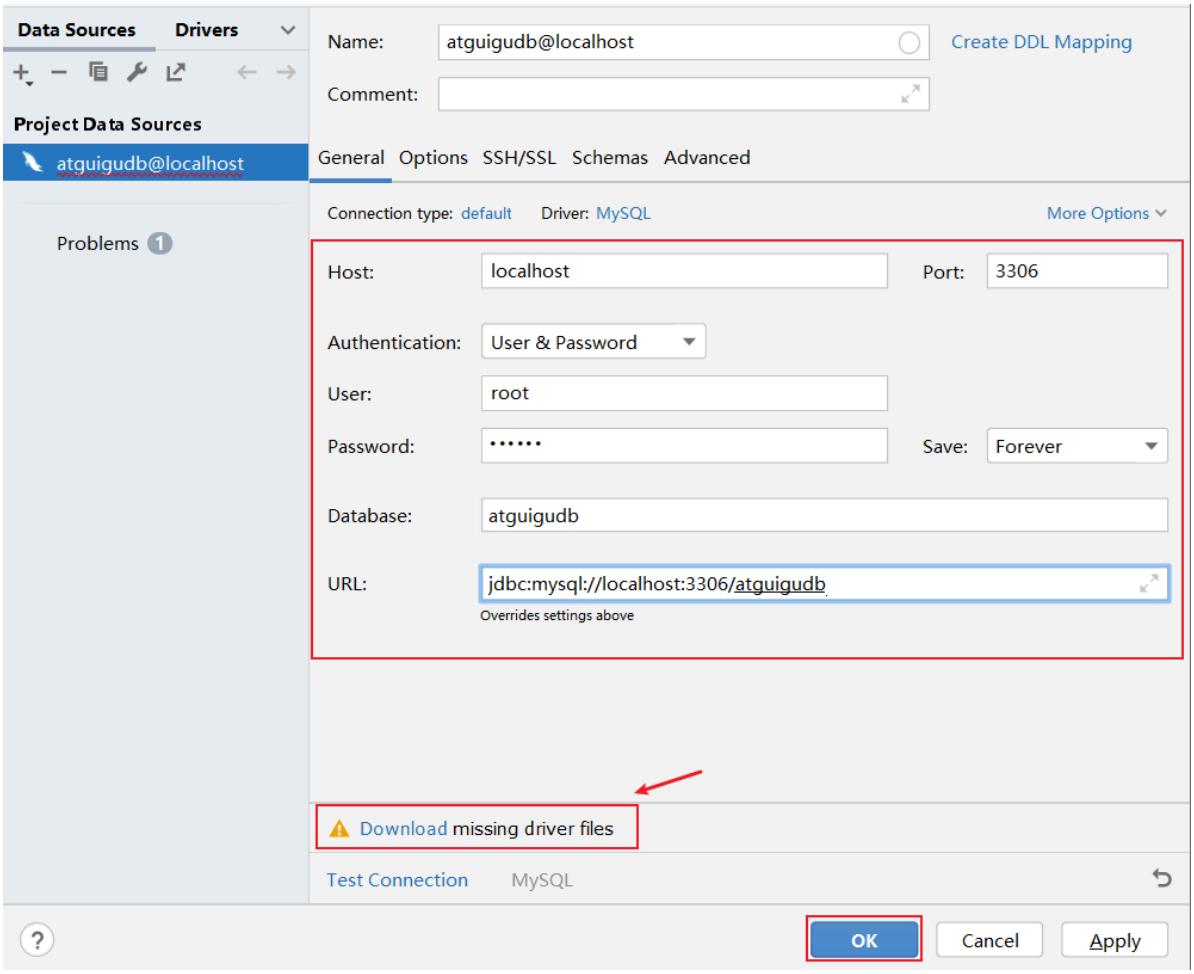
找到数据库选项：



添加指定数据库：



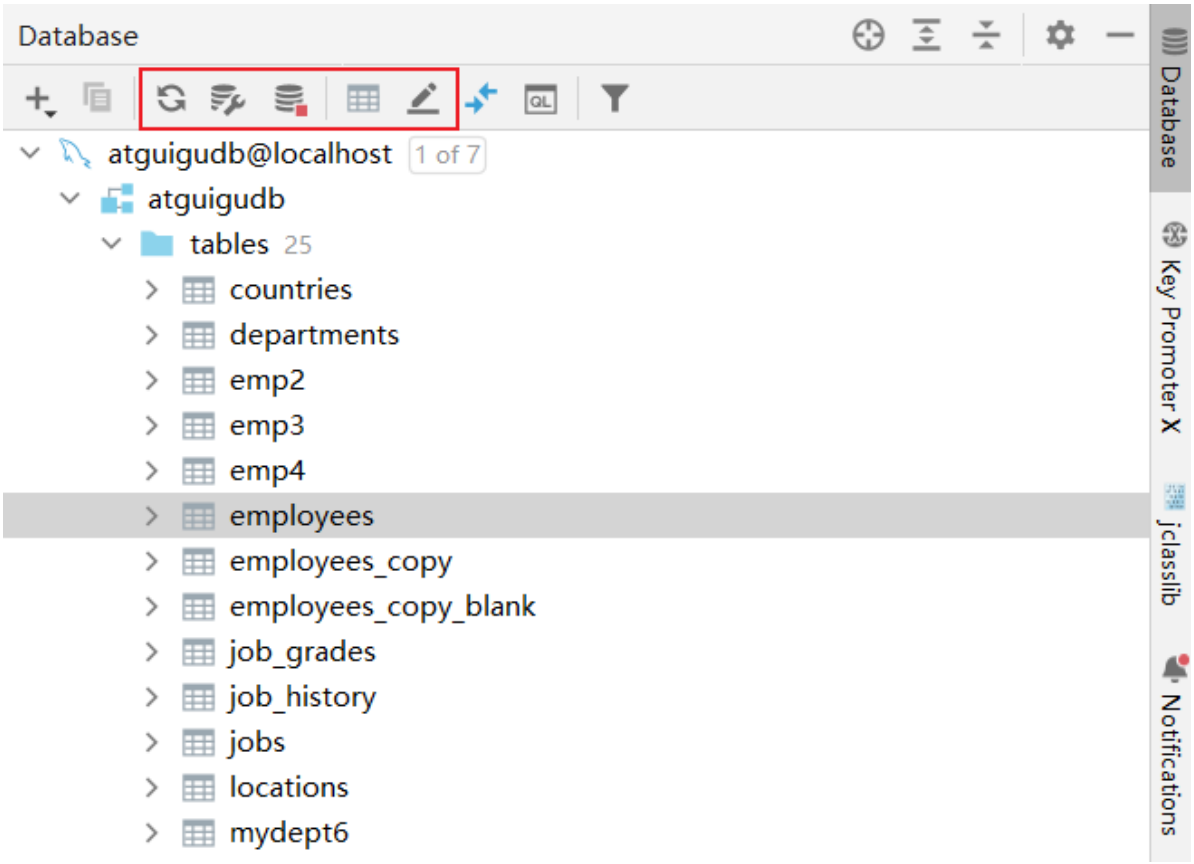
配置MySQL数据库的详细信息：



表面上很多人认为配置 Database 就是为了有一个 GUI 管理数据库功能，但是这并不是 IntelliJ IDEA 的 Database 最重要特性。数据库的 GUI 工具有很多，IntelliJ IDEA 的 Database 也没有太明显的优势。

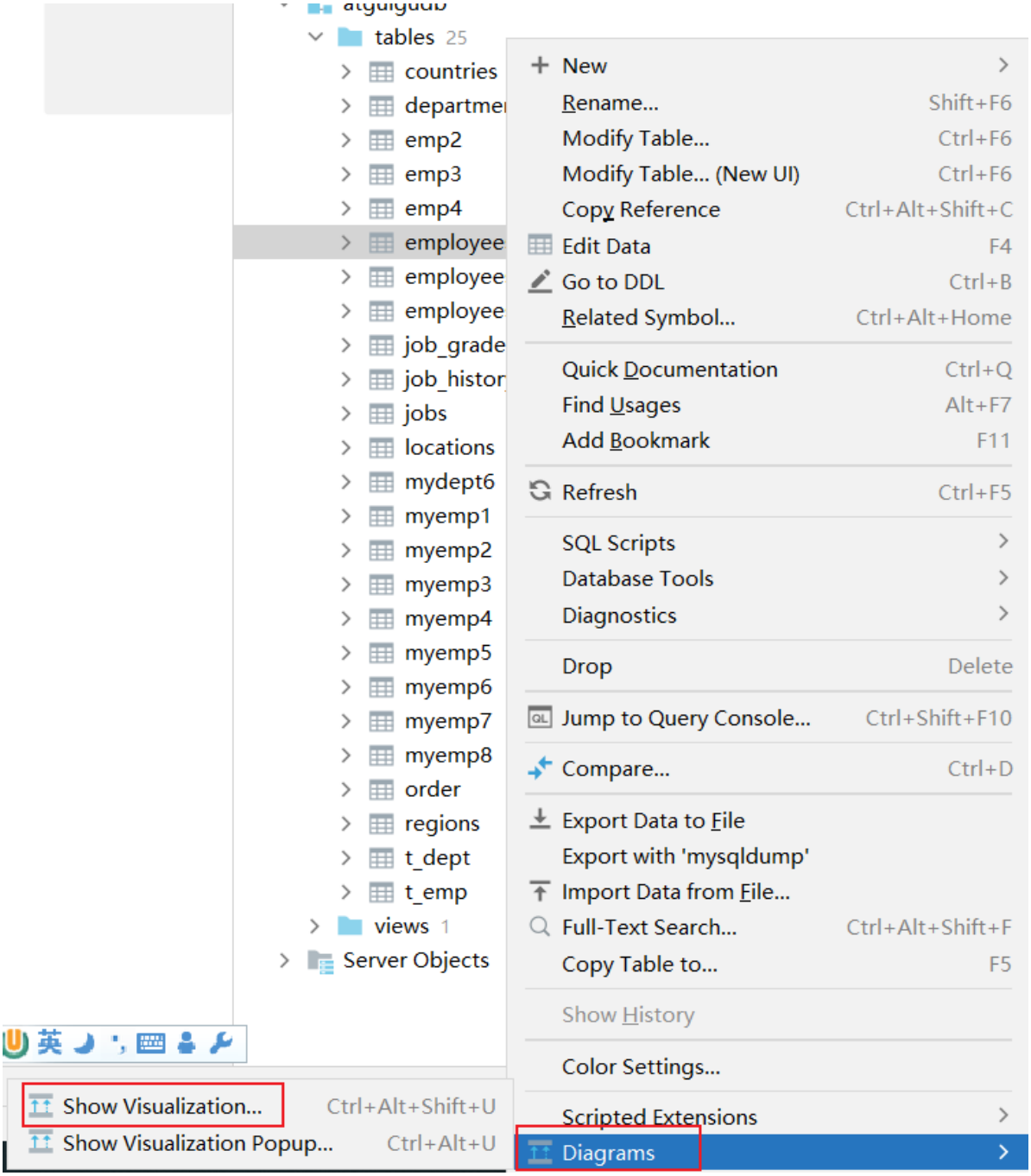
IntelliJ IDEA 的 Database 最大特性就是对于 Java Web 项目来讲，常使用的 ORM 框架，如 Hibernate、Mybatis 有很好的支持，比如配置好了 Database 之后，IntelliJ IDEA 会自动识别 domain 对象与数据表的关系，也可以通过 Database 的数据表直接生成 domain 对象等。

11.2 常用操作

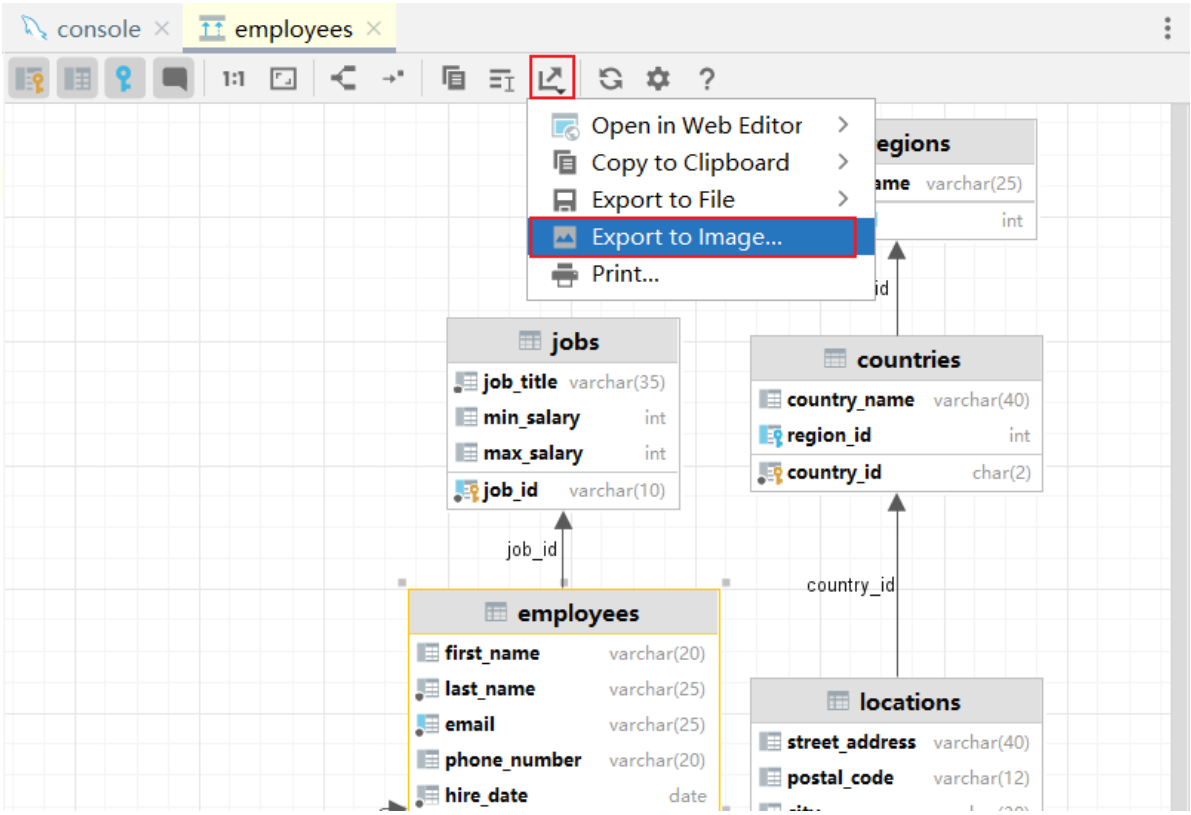


- 图标1：同步当前的数据库连接。这个是最重要的操作。配置好连接以后或通过其他工具操作数据库以后，需要及时同步
- 图标2：配置当前的连接
- 图标3：断开当前的连接
- 图标4：显示相应数据库对象的数据
- 图标5：编辑修改当前数据库对象

展示ER图：

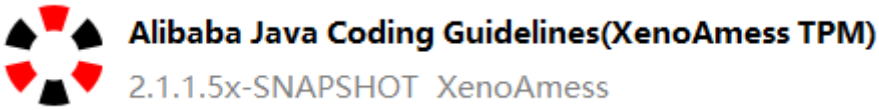

tables 25
countries
departmer
emp2
emp3
emp4
employee
employee
employee
job_grade
job_histor
jobs
locations
mydept6
myemp1
myemp2
myemp3
myemp4
myemp5
myemp6
myemp7
myemp8
order
regions
t_dept
t_emp
views 1
Server Objects
New
Rename... Shift+F6
Modify Table... Ctrl+F6
Modify Table... (New UI) Ctrl+F6
Copy Reference Ctrl+Alt+Shift+C
Edit Data F4
Go to DDL Ctrl+B
Related Symbol... Ctrl+Alt+Home
Quick Documentation Ctrl+Q
Find Usages Alt+F7
Add Bookmark F11
Refresh Ctrl+F5
SQL Scripts
Database Tools
Diagnostics
Drop Delete
Jump to Query Console... Ctrl+Shift+F10
Compare... Ctrl+D
Export Data to File
Export with 'mysqldump'
Import Data from File...
Full-Text Search... Ctrl+Alt+Shift+F
Copy Table to... F5
Show History
Color Settings...
Scripted Extensions
Diagrams
Show Visualization... Ctrl+Alt+Shift+U
Show Visualization Popup... Ctrl+Alt+U

可以导出文件：



12. IDEA常用插件

推荐1：Alibaba Java Coding Guidelines



阿里巴巴Java编码规范检查插件，检测代码是否存在问题，以及是否符合规范。

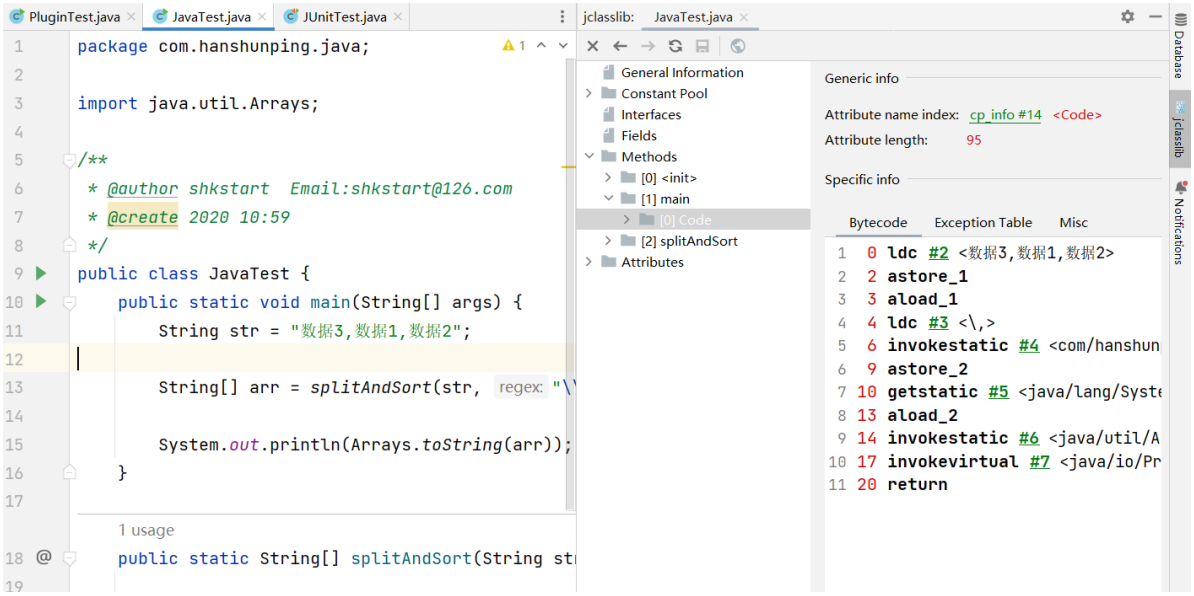
使用：在类中，右键，选择编码规约扫描，在下方显示扫描规约和提示。根据提示规范代码，提高代码质量。

推荐2：jclasslib bytecode viewer

可视化的字节码查看器。

使用：

1. 在 IDEA 打开想研究的类。
2. 编译该类或者直接编译整个项目（如果想研究的类在 jar 包中，此步可略过）。
3. 打开“view”菜单，选择“Show Bytecode With jclasslib”选项。
4. 选择上述菜单项后 IDEA 中会弹出 jclasslib 工具窗口。



英文设置：

在 Help -> Edit Custom VM Options ...，加上

```
-Duser.language=en
```

推荐3：Translation

Translation

3.3.5 Yii.Guxing

注册翻译服务（有道智云、百度翻译开放平台、阿里云机器翻译）帐号，开通翻译服务并获取其应用ID和密钥 绑定应用ID和密钥：偏好设置（设置） > 工具 > 翻译 > 常规 > 翻译引擎 > 配置...

使用：鼠标选中文本，点击右键即可自动翻译成多国语言。

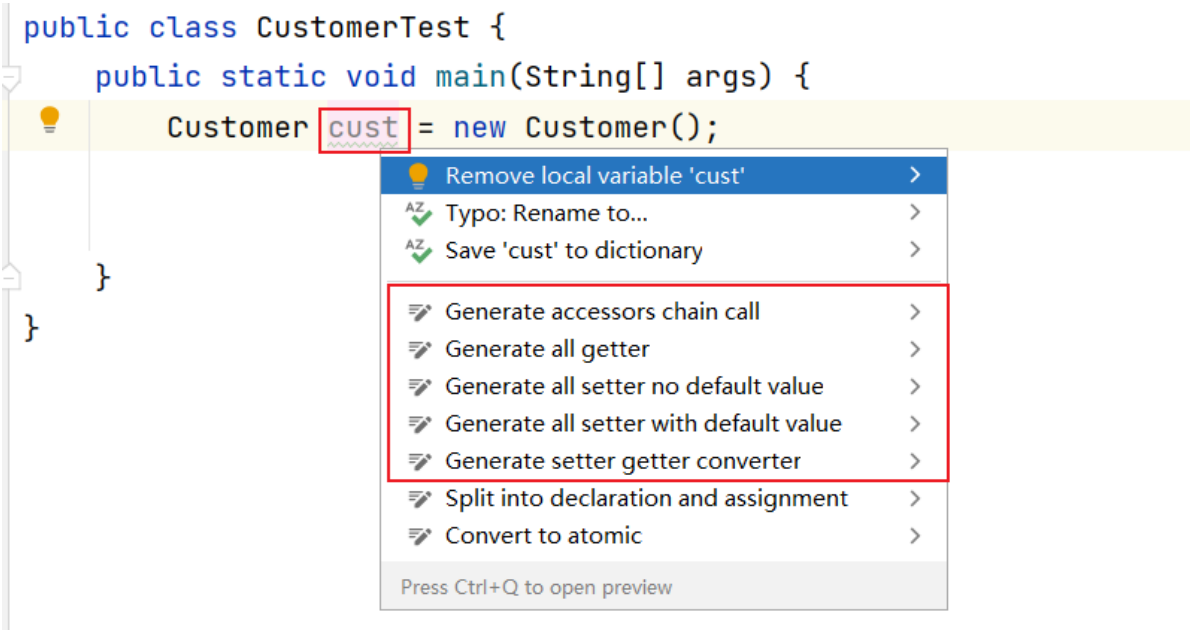
注：请注意保管好你的应用密钥，防止其泄露。

推荐4：GenerateAllSetter

GenerateAllSetter

2.8.1 bruceGe

实际开发中还有一个非常常见的场景： 我们创建一个对象后，想依次调用 Setter 函数对属性赋值，如果属性较多很容易遗漏或者重复。



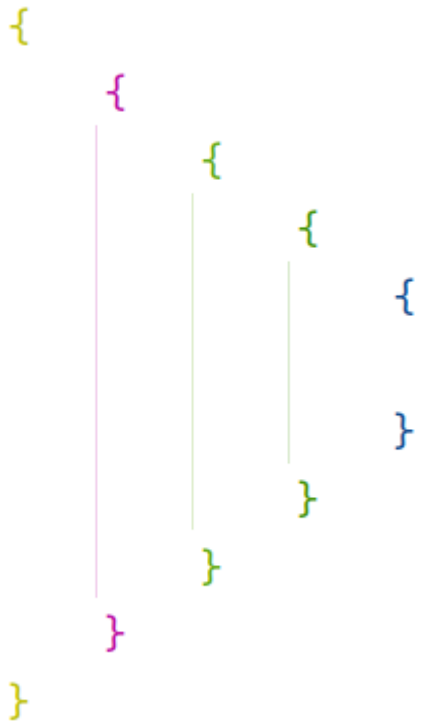
可以使用这 GenerateAllSetter 提供的功能，快速生成对象的所有 Setter 函数（可填充默认值），然后自己再跟进实际需求设置属性值。

插件5：Rainbow Brackets

Rainbow Brackets

6.26 izhangzhihao

给括号添加彩虹色，使开发者通过颜色区分括号嵌套层级，便于阅读

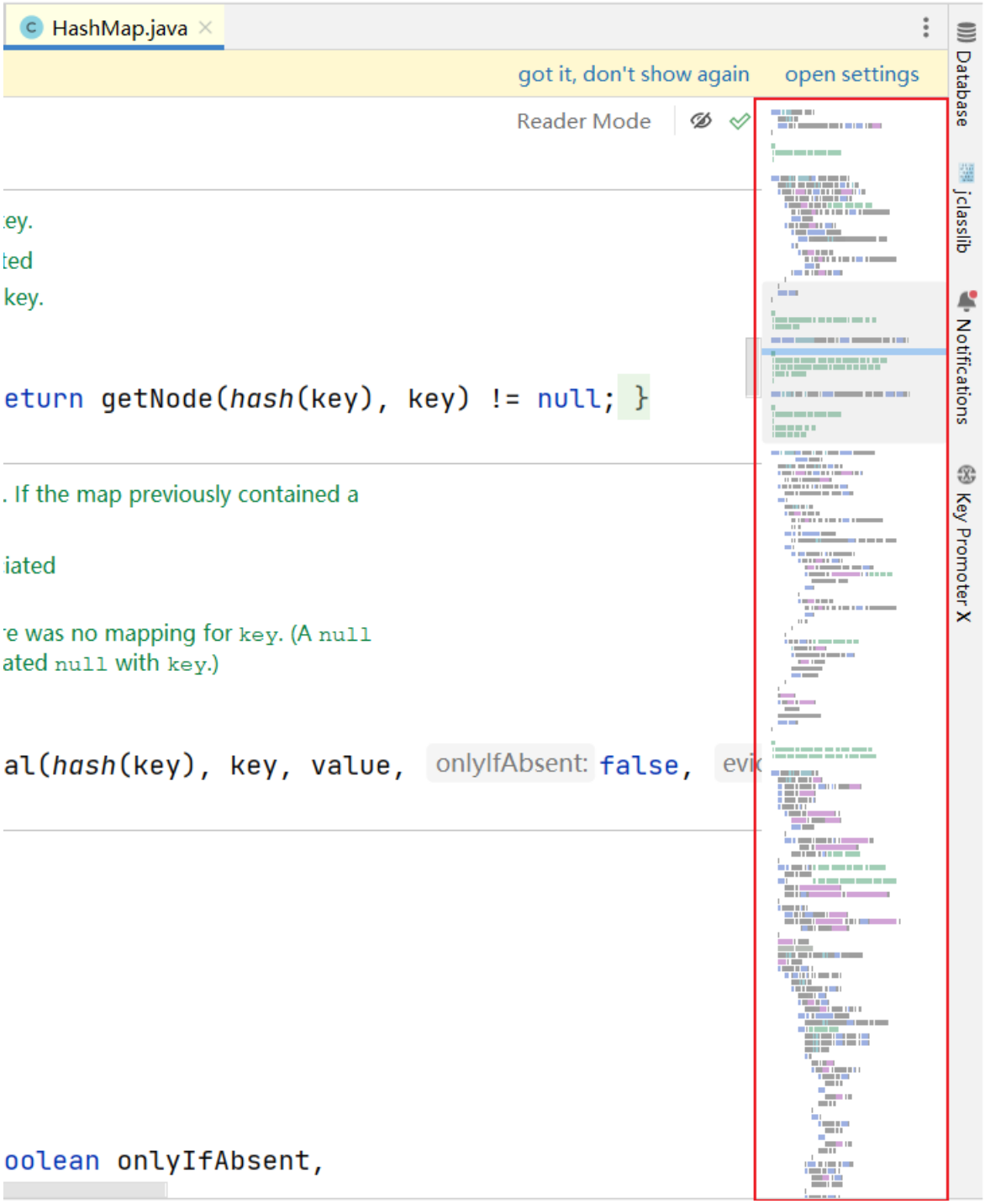


推荐6：CodeGlance Pro

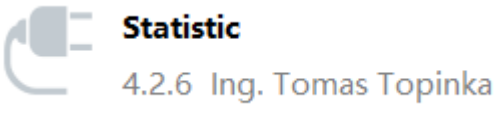
CodeGlance Pro

1.5.2 Nasller

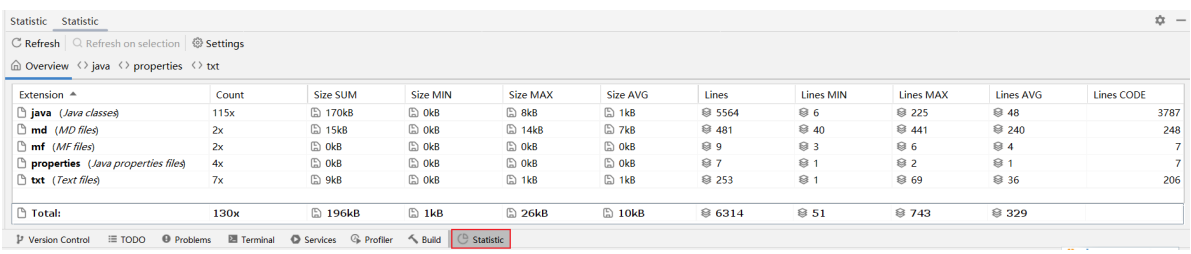
在编辑器右侧生成代码小地图，可以拖拽小地图光标快速定位代码，阅读行数很多的代码文件时非常实用。

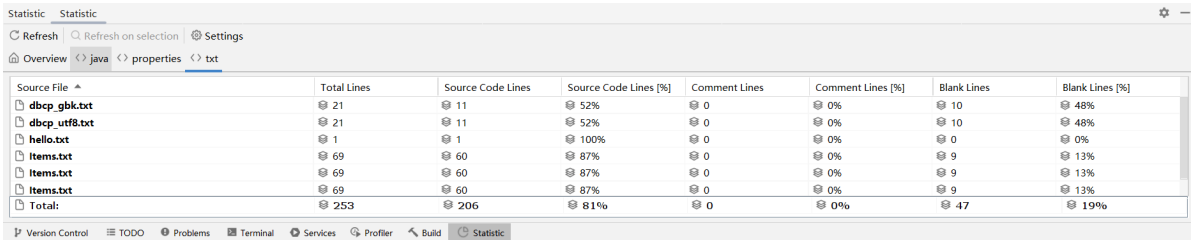


推荐7：Statistic

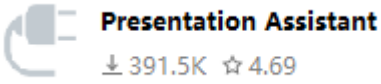


代码统计工具。



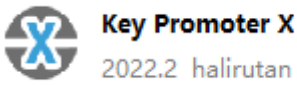


推荐8： Presentation Assistant



显示快捷键操作的按键

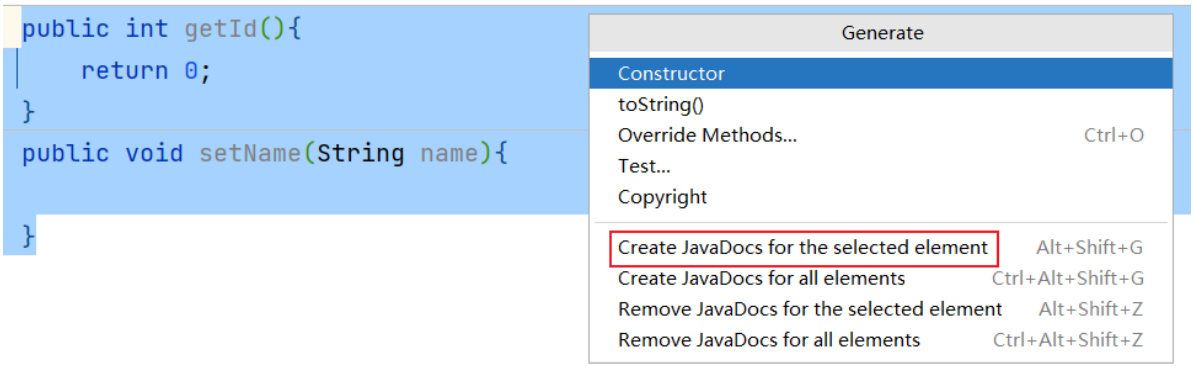
推荐9： Key Promoter X



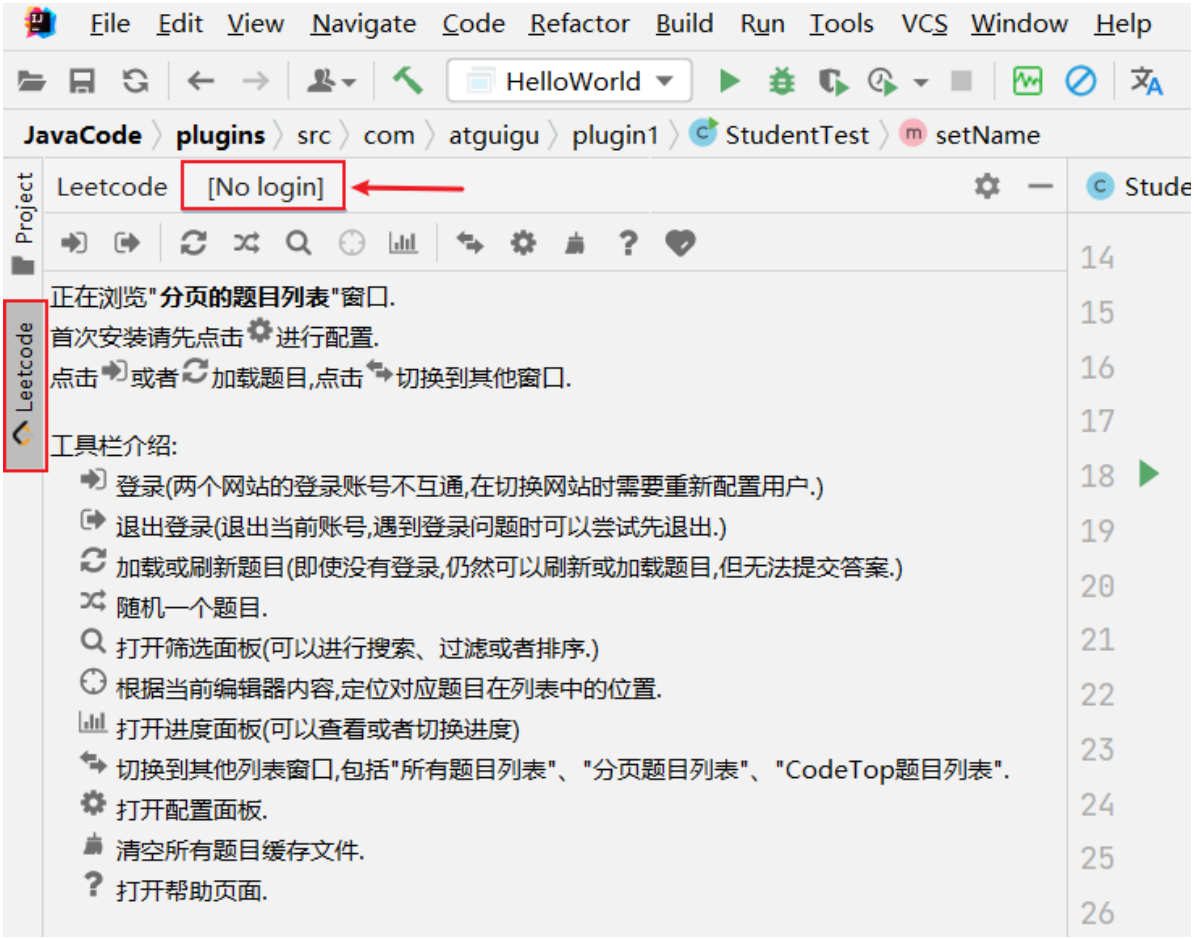
快捷键提示插件。当你执行鼠标操作时，如果该操作可被快捷键代替，会给出提示，帮助你自然形成使用快捷键的习惯，告别死记硬背。

推荐10： JavaDoc

按 `alt+insert` ，执行操作：

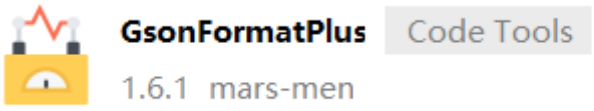


推荐11： LeetCode Editor



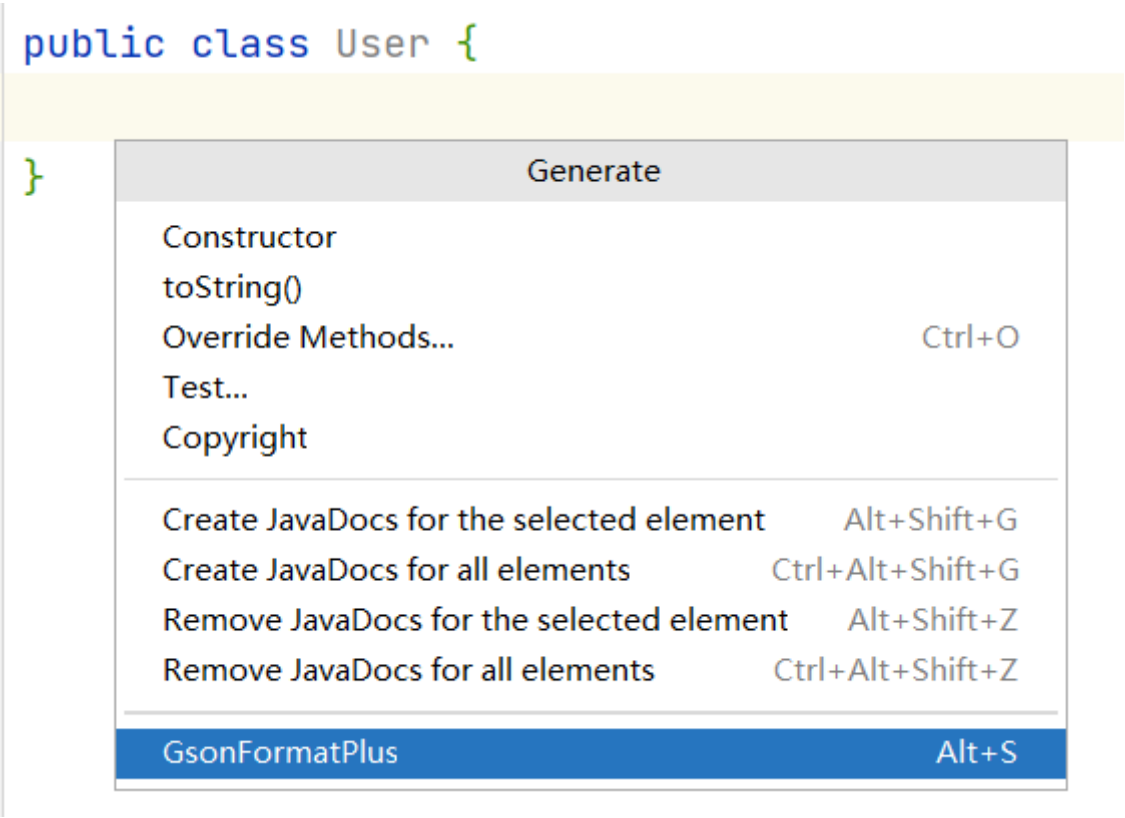
在 IDEA 里刷力扣算法题

推荐12：GsonFormatPlus



根据 json 生成对象。

使用：使用alt + s 或 alt + insert调取。



举例：

```
{
    "name": "tom",
    "age": "18",
    "gender": "man",
    "hometown": {
        "province": "河北省",
        "city": "石家庄市",
        "county": "正定县"
    }
}
```

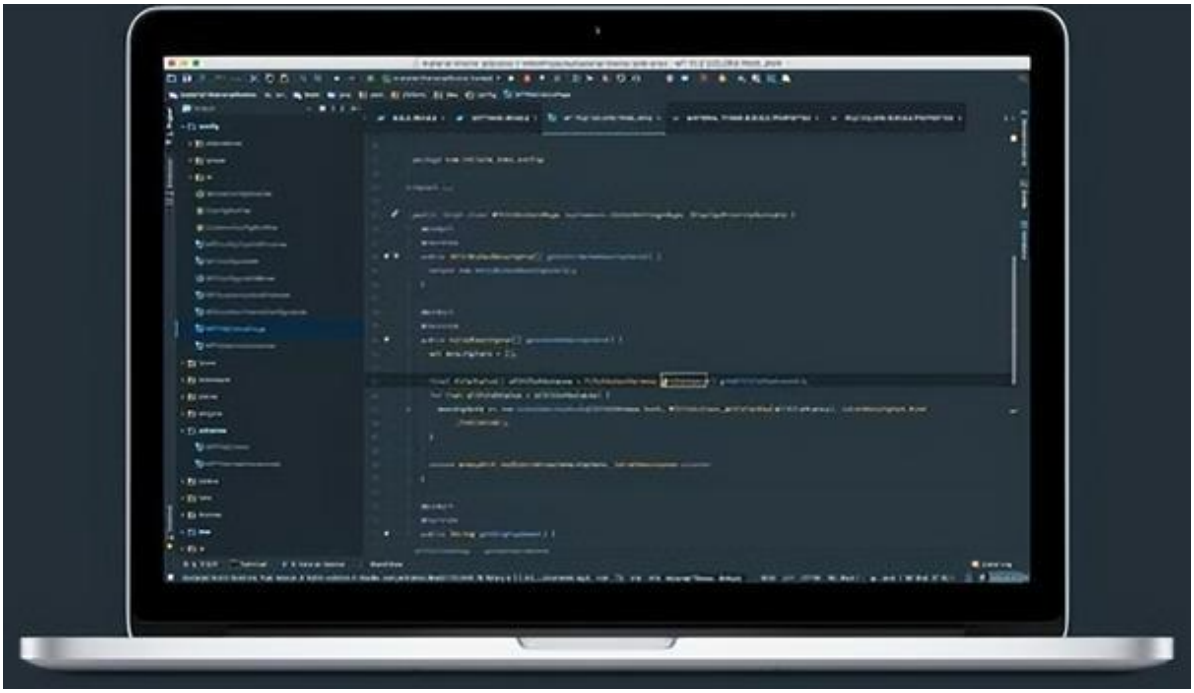
插件13： Material Theme UI

Material Theme UI

12.6M ☆2.85

对于很多人而言，写代码时略显枯燥的，如果能够安装自己喜欢的主题将为开发工作带来些许乐趣。

IDEA 支持各种主题插件，其中最出名的当属 Material Theme UI。



安装后，可以从该插件内置的各种风格个选择自己最喜欢的一种。