JavaWeb 开章序言

大家好，我是 willie，接下来一段时间我们将会一起来探究 JavaWeb ，开启大神之旅。

我曾经在企业中没日没夜的 “搬过砖”， “造过轮子”，也带过团队，开发过 web 也开发过 PC 端产品，深知 web 和

PC 端的区别以及 web 在行业中的重要性。

为什么要学习 JavaWeb?

我们在谈论任何一个新的知识点或新的方向时，都需要强调它所适应的使用场景，因为：**知识点之所以值的去**

学习，是因为它解决了一些咱们目前无法实现的需求难题。

目前互联网任何一款产品要让用户能够使用，方式一是用户下载软件进行安装（WPS）方式二是用户通过浏览器来访问（叩丁狼官网），第一种方式，产品维护成本相对比较高，因为每更新一个功能都需要所有用户来更新，而而方式二不需要，只需更新产品，用户即可随时操作最新的功能。所以第二种方式目前在互联网上的使用率是非常高的，所以作为程序员掌握第二种方式的开发是一项**基本技能。**而 JavaWeb 就可以帮咱们实现第二种方式。

但是，实现 WEB 的开发，我们需要解决的问题比较多，如：

1. 如何更好的设计一款软件让其更趋于合理，需要注意什么问题?
2. 如何让越来越多的代码在后期能很好的维护?
3. 通过浏览器访问 Java 项目的底层原理是怎样的?
4. 如何简单快速的将我们写好的 Java 项目发布出去，让用户可以通过浏览器来访问?
5. 如何开发可以供浏览器访问的 Java 项目?
6. 如何高效的保存用户从浏览器传递到 Java 项目的数据?

而接下来的 JavaWeb 课程将带着大家来学习软件的设计原则和思想，了解 JavaWeb 底层实现原理，掌握

JavaWeb 底层技术。目前市面上用 Java 实现的能够快速实现浏览器访问 Java 项目的工具框架其底层都是咱们接

下来要学习的 JavaWeb 知识点。学好该 JavaWeb 基础即可实现简单的个人网站的开发，可以对后期的web框架有

更深入的理解。

寄语

最后，我想跟大家说的是：**新阶段，新开始，沉迷于写代码，但不要只沉迷于代码。**

学习基础，学习底层技术的目的是为提升自身的语言功底，知其然并知其所以然，有了基础功底，可以更好的体会开源项目的设计理念，并将设计应用到实践中，开拓自己的编程思维。

接下来，让咱们开始一次紧张刺激的 JavaWeb 探索之旅，我也希望大家在学习过程中可以多分享你的成长心得和学习痛点，学习不是单向的输出，而是一次交流反馈的过程！加油，狼崽们。

学习目标 `*****`

- [ ] 了解开发环境统一的重要性 `*****`
- [ ] 了解软件的生命周期和设计原则 `**`
- [ ] 掌握并遵循基本命名规范 `*****`
- [ ] 了解 jar 的导出 `**`
- [ ] 掌握 jar 包的导入 `*****`
- [ ] 掌握单元(JUnit)测试的使用 `*****`
- [ ] 掌握基本的配置文件 `*****`

1 开发环境搭建

1.1 开发环境介绍

开发环境：支持软件或应用的开发、运行和维护的一组软件.

开发环境使用注意事项：

1.在团队开发中,需避免环境差异带来的一些问题,避免出现自己电脑上没问题,别人电脑运行报错的情况.
2.开发环境需要跟部署项目的环境要一致,避免部署的时候出现问题.

目标： 清楚统一环境在开发中的重要性，所以尽量将环境和老师的环境统一,否则后期可能出现各种幽灵问题,极可能就是环境的问题.

大神班使用的开发环境：

开发工具（IDE）： ideaIU-2019.3 版本，简称 idea

运行环境：JDK 1.11 环境,简称 JDK11

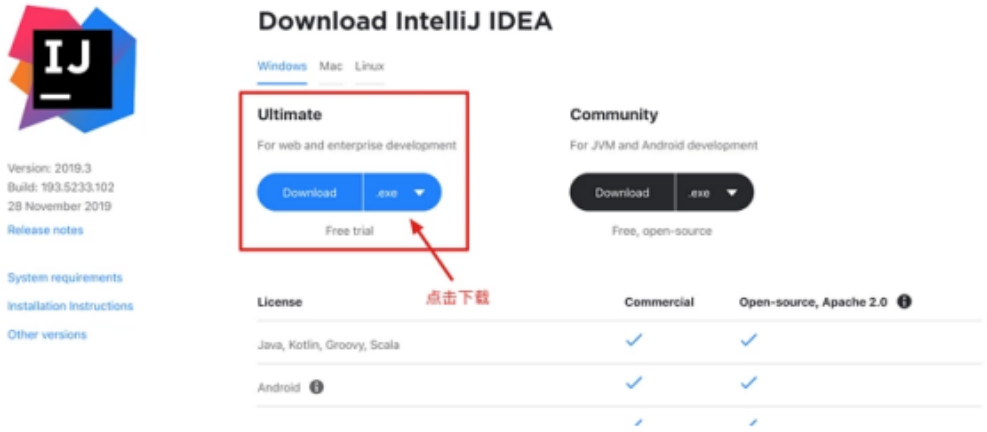
1.2 idea 安装

安装步骤：

下载 idea2019.3

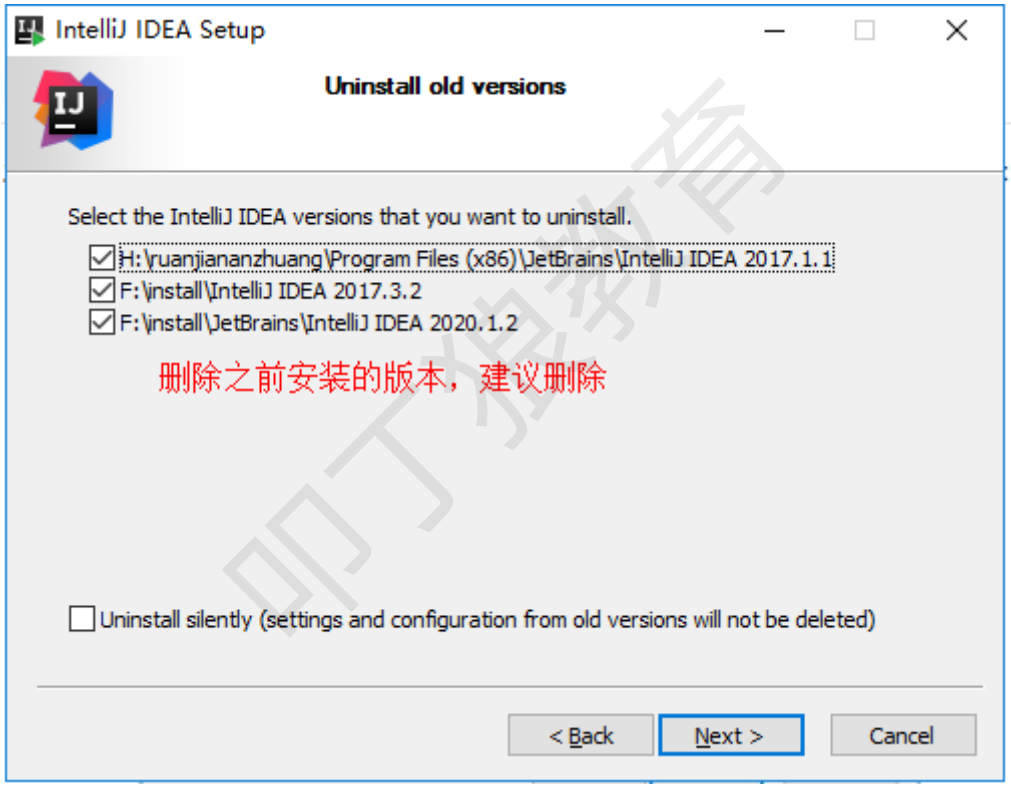
2019.3 相对比较稳定，对后期需要使用到的开发工具兼容性比较高

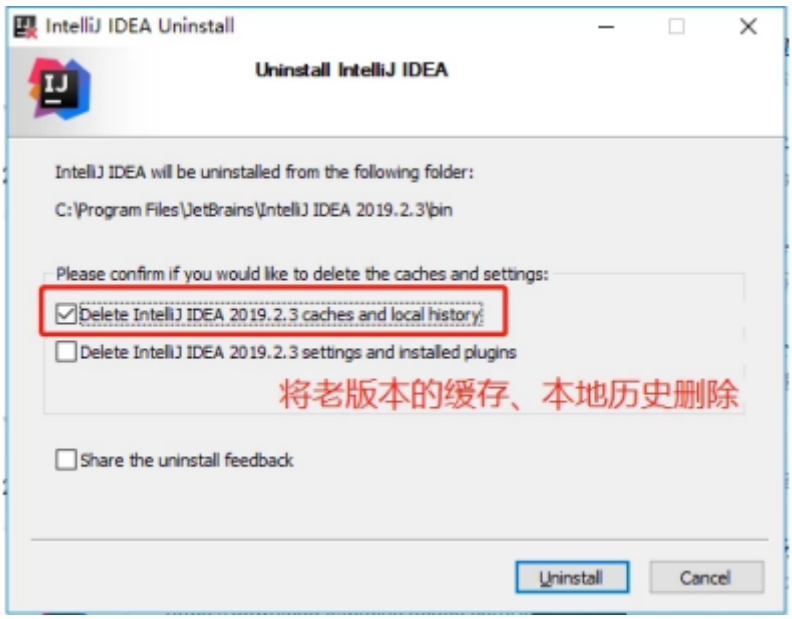
下载地址： https://www.jetbrains.com/idea/download/



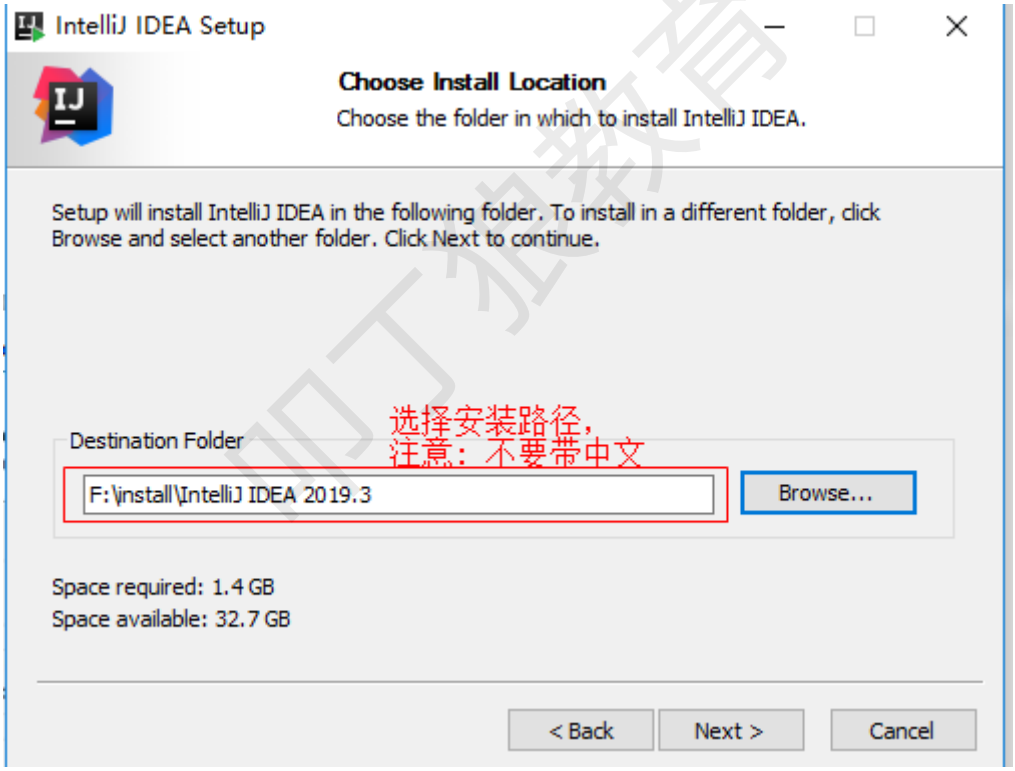
启动安装包

启动安装包，卸载已经安装的旧的 idea 版本并安装19.3 版本

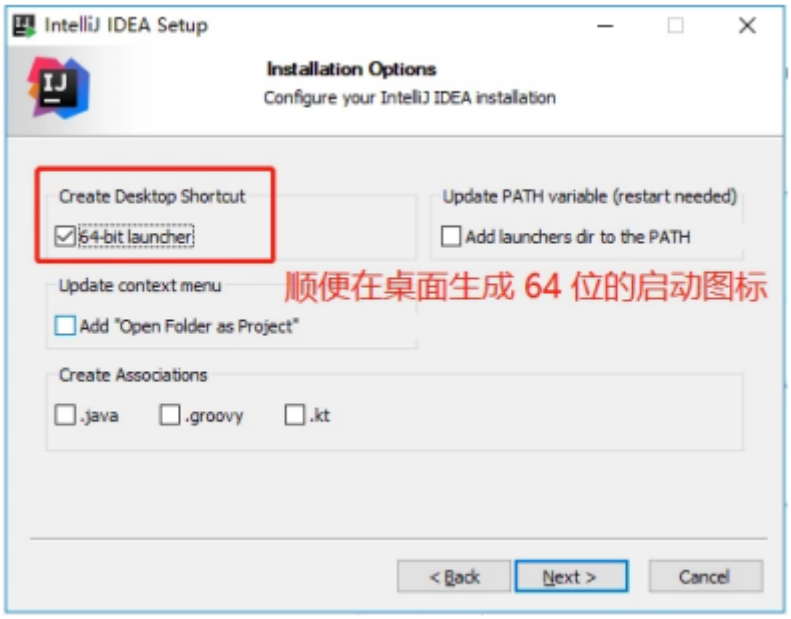




选择安装路径，注意**不能带中文,空格,特殊符号**，容易出幽灵问题



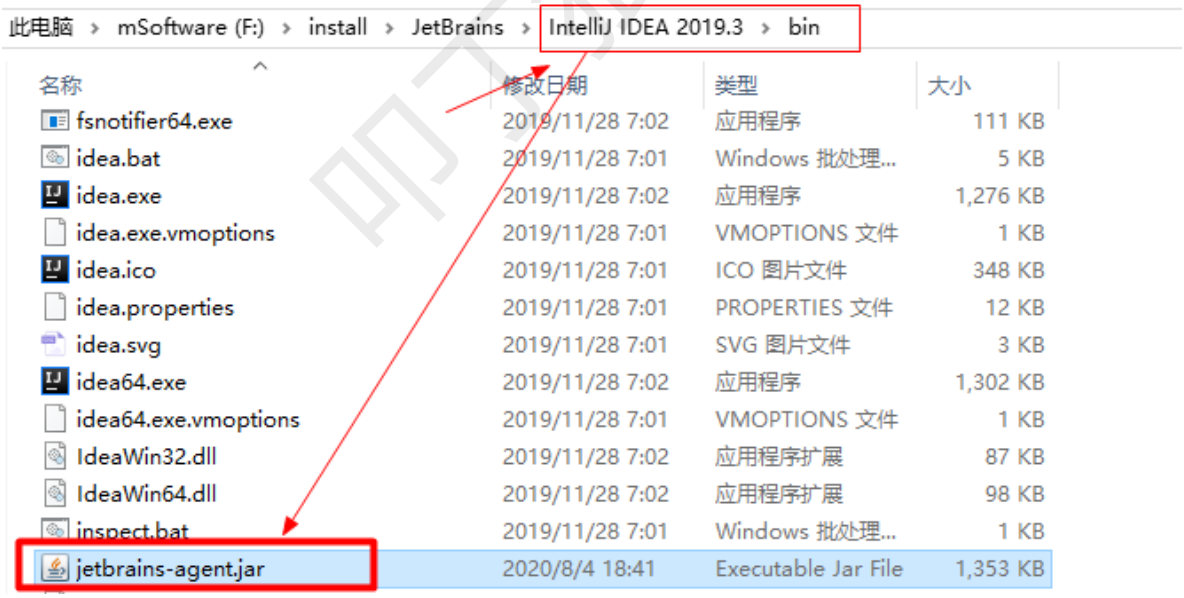
点击下一步，选择在桌面生成 64bit 的快捷键



安装完成，关闭 idea,激活 idea 供学习使用

导入激活包

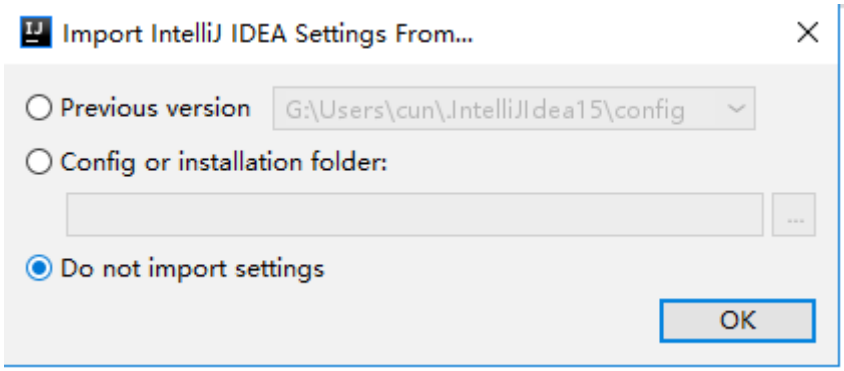
拷贝激活包到idea 的 bin目录下，原则上来说可放到任意地方，为了防止后期误删，将其存入bin 目录



拷贝激活包的路径，这里的路径为`F:\install\JetBrains\IntelliJ IDEA 2019.3\bin\jetbrains-agent.jar`

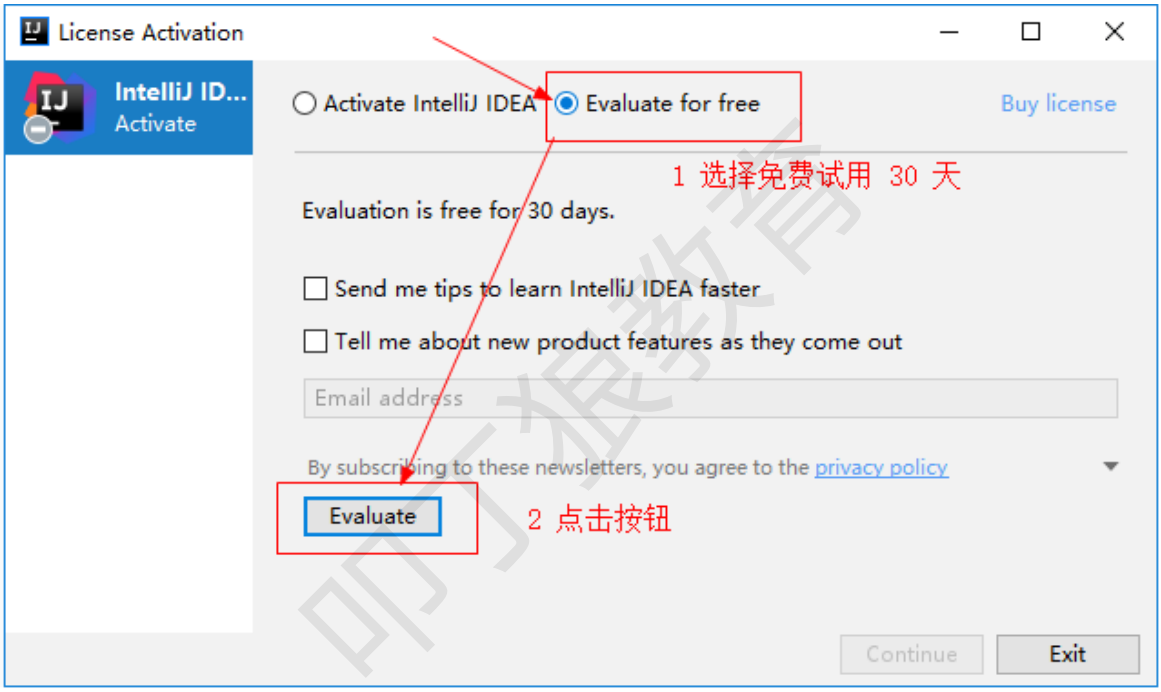
启动 IDEA

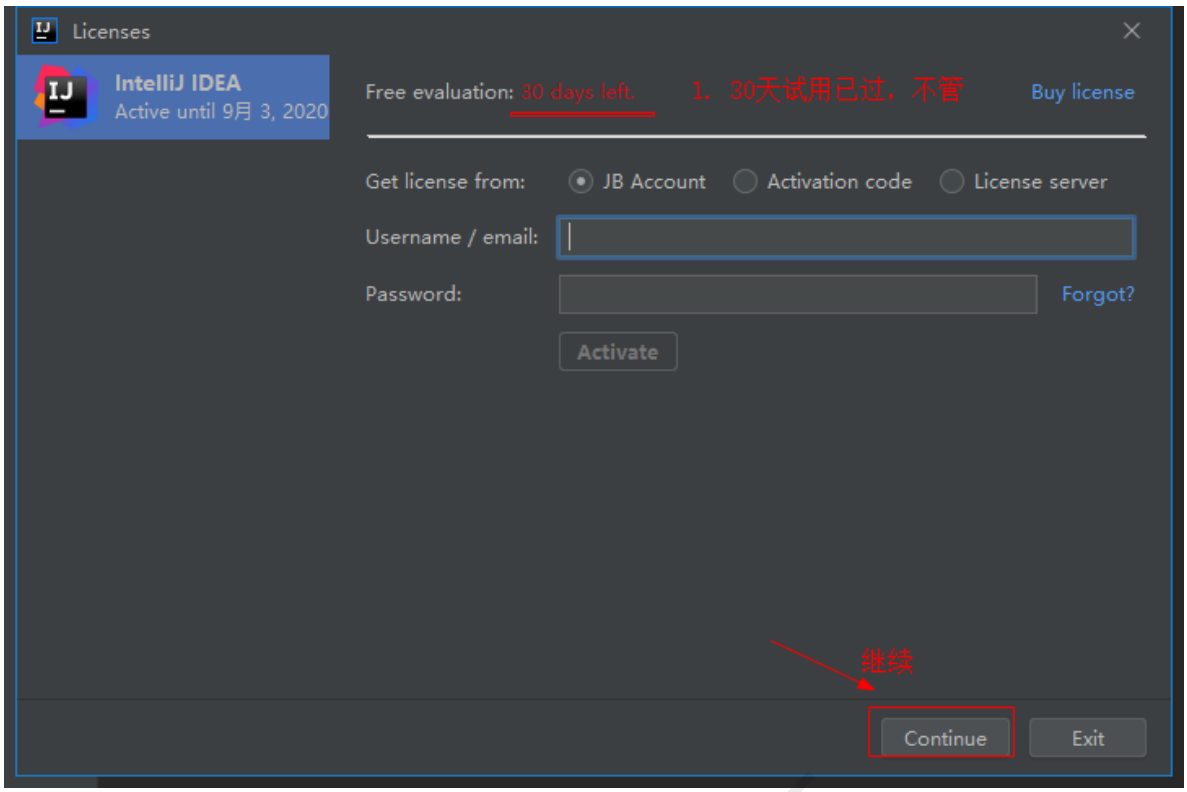
导入旧的 idea 的配置或者不导入，创建新的配置，看个人。



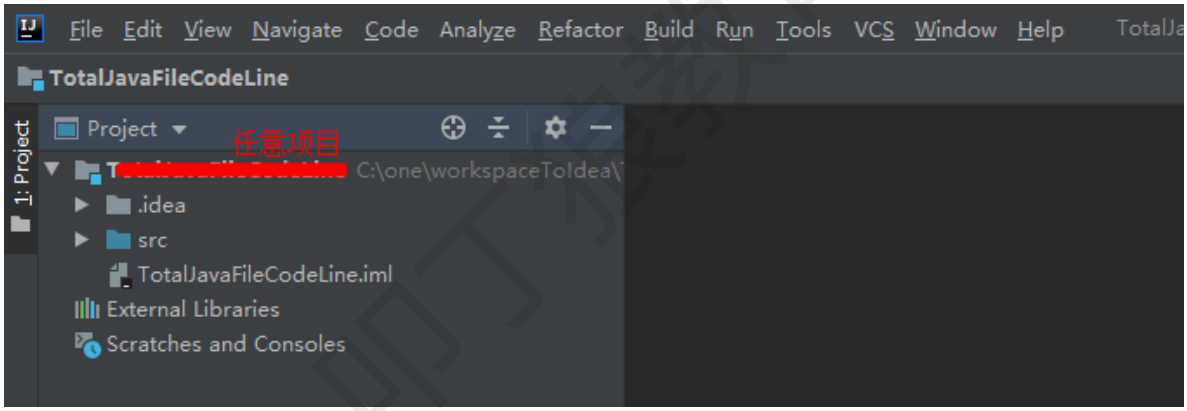
跳过，使用默认的插件

选择免费试用 30 天

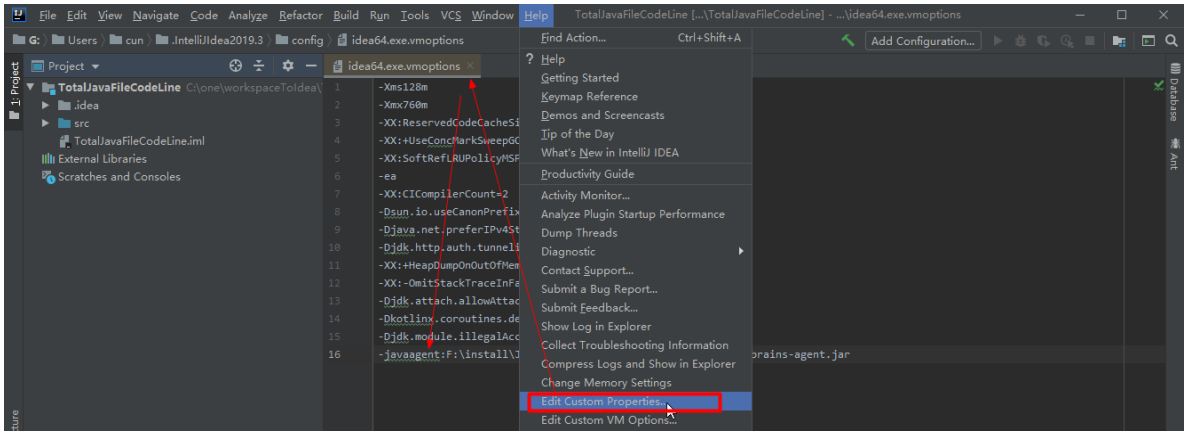




打开 idea，创建一个普通项目



Help -> Edit Custom VM Options



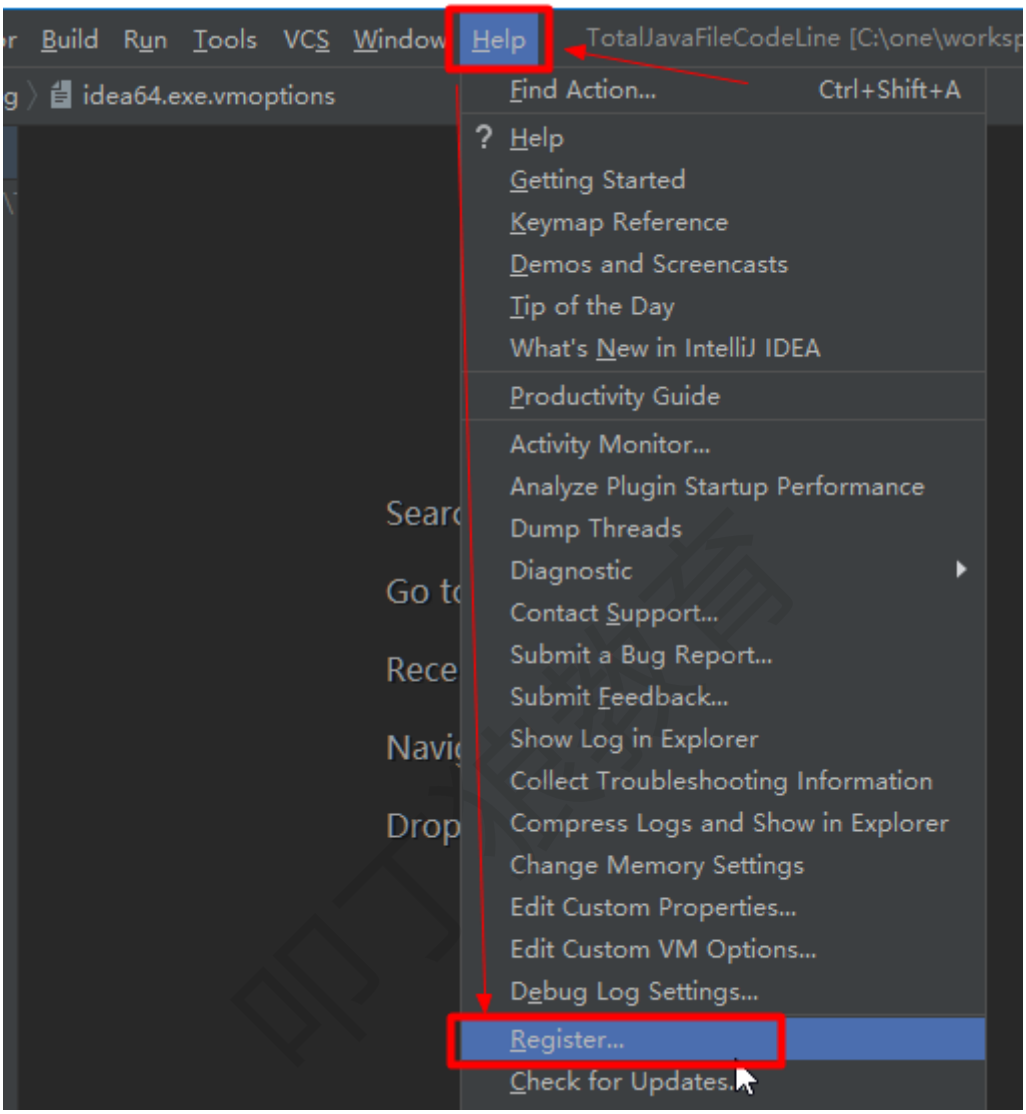
加入配置：

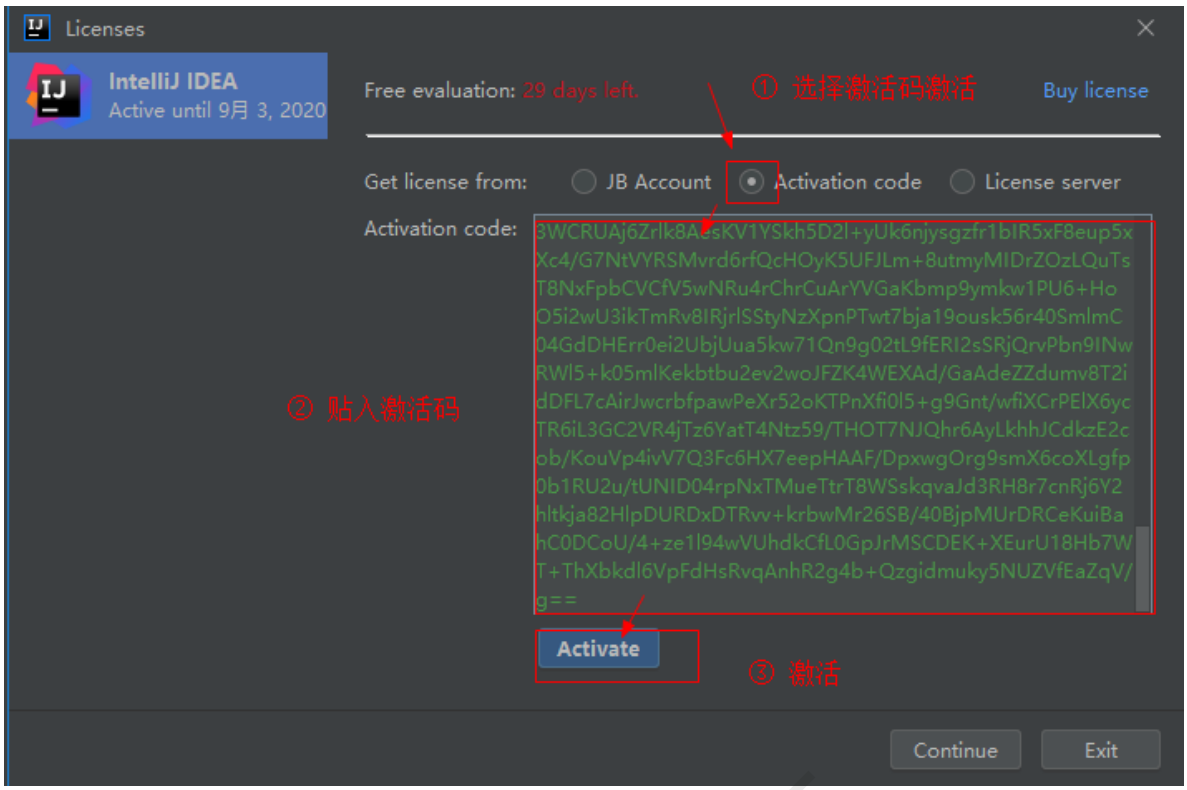
```
-javaagent:F:\install\JetBrains\IntelliJ IDEA 2019.3\bin\jetbrains-agent.jar
```

路径看个人电脑位置，修改成自己的路径即可。

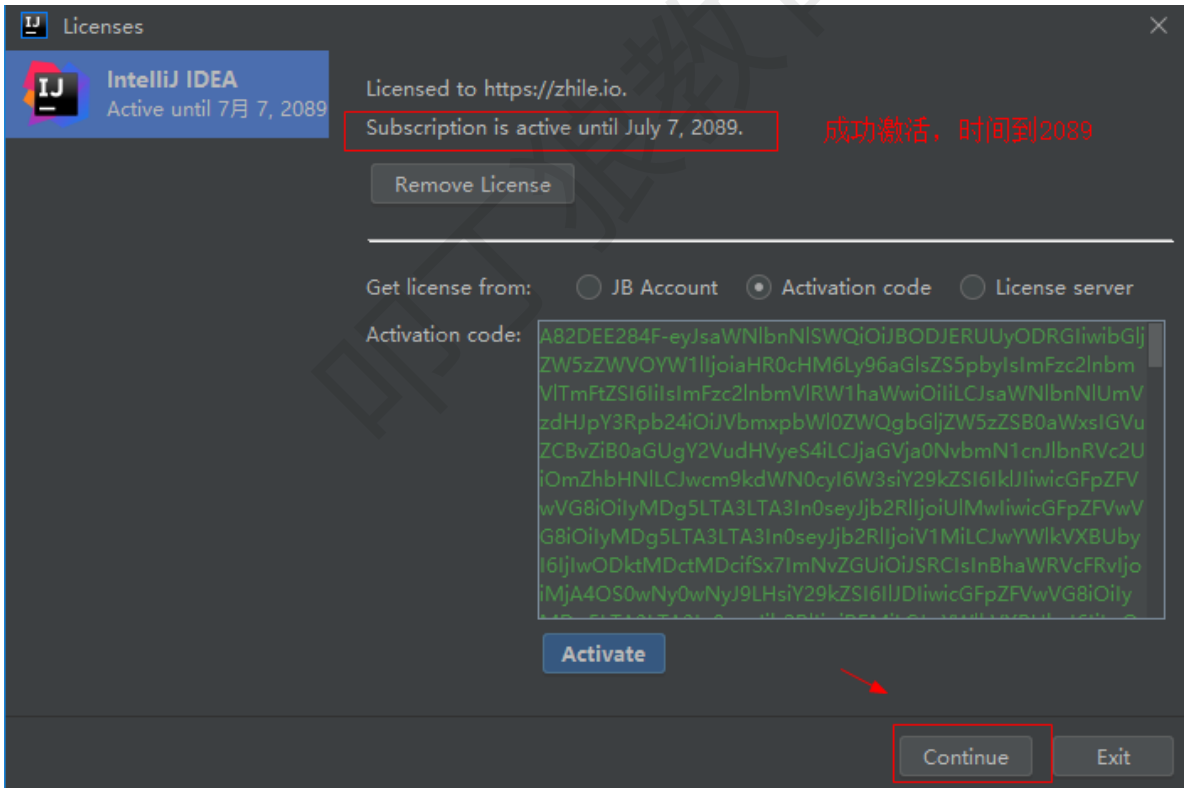
重启 idea 发现任然没有激活。

打开idea -> Help -> Register





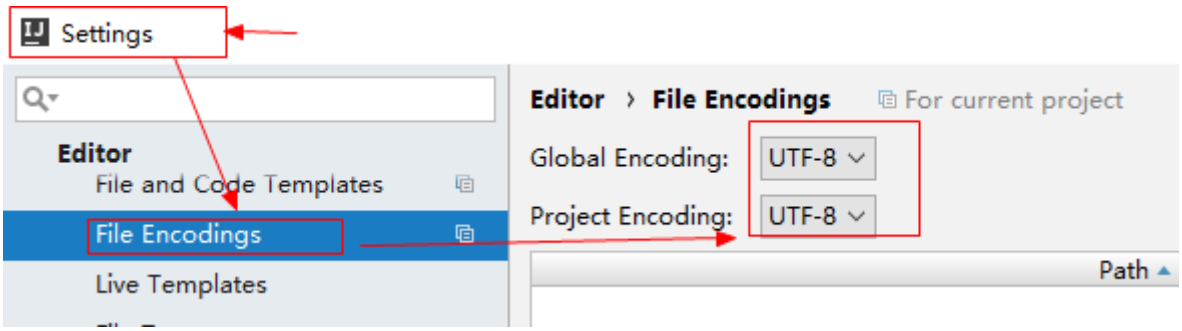
激活码：看资料 激活码.txt



1.2 idea 默认配置修改

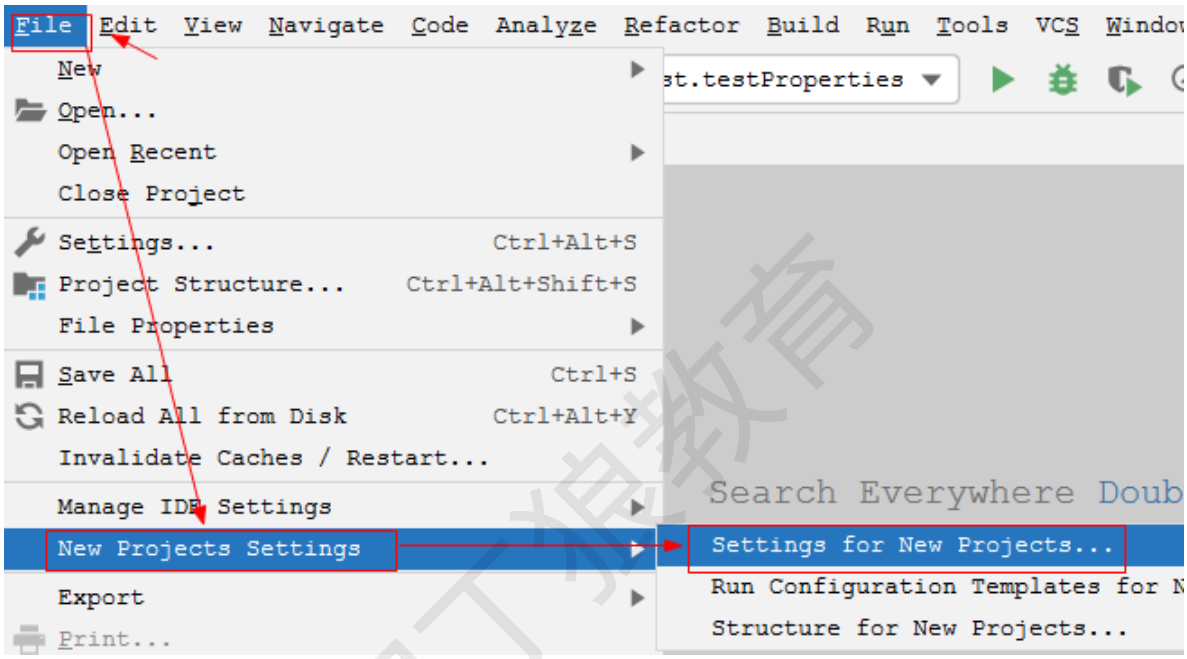
1.2.1 设置项目编码(掌握)

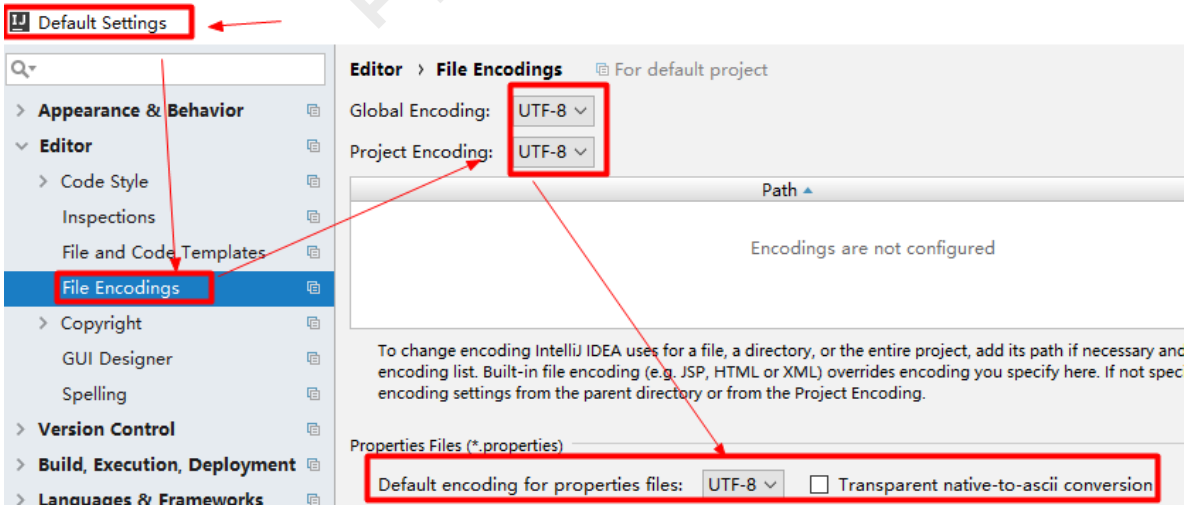
File -> setting -> Edior -> File Encodings



1.2.2 通用编码设置

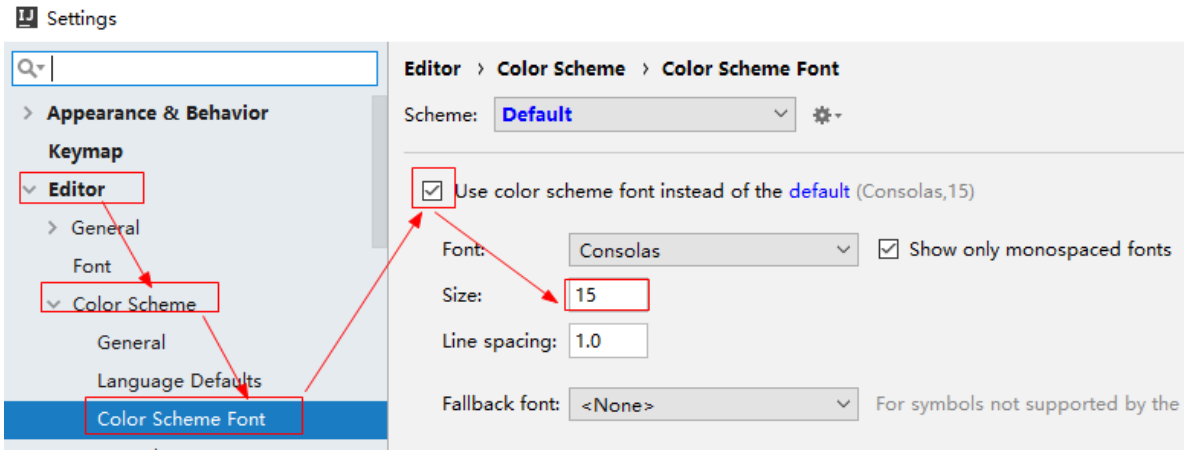
File -> Other Setting -> Settings for New Projects-> File Encoding





1.2.3 调整字体大小

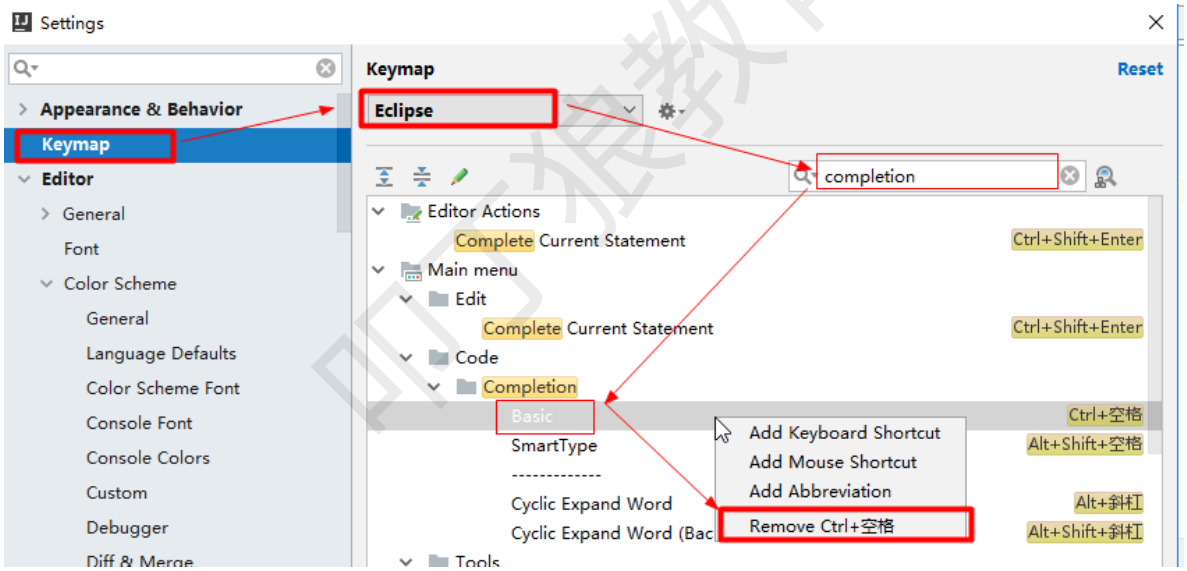
File -> Settings -> Editor -> Color Scheme -> Color Scheme Font-> Size



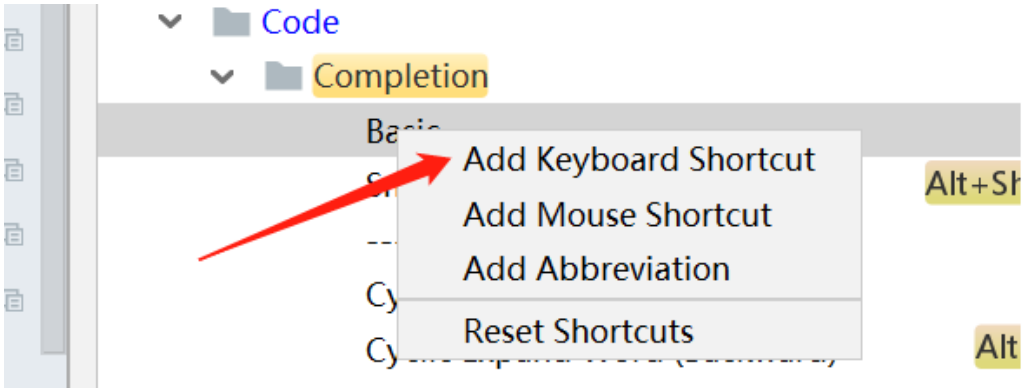
1.2.3 设置为 eclipse 快捷键(了解)

设置快捷键和 eclipse 相同

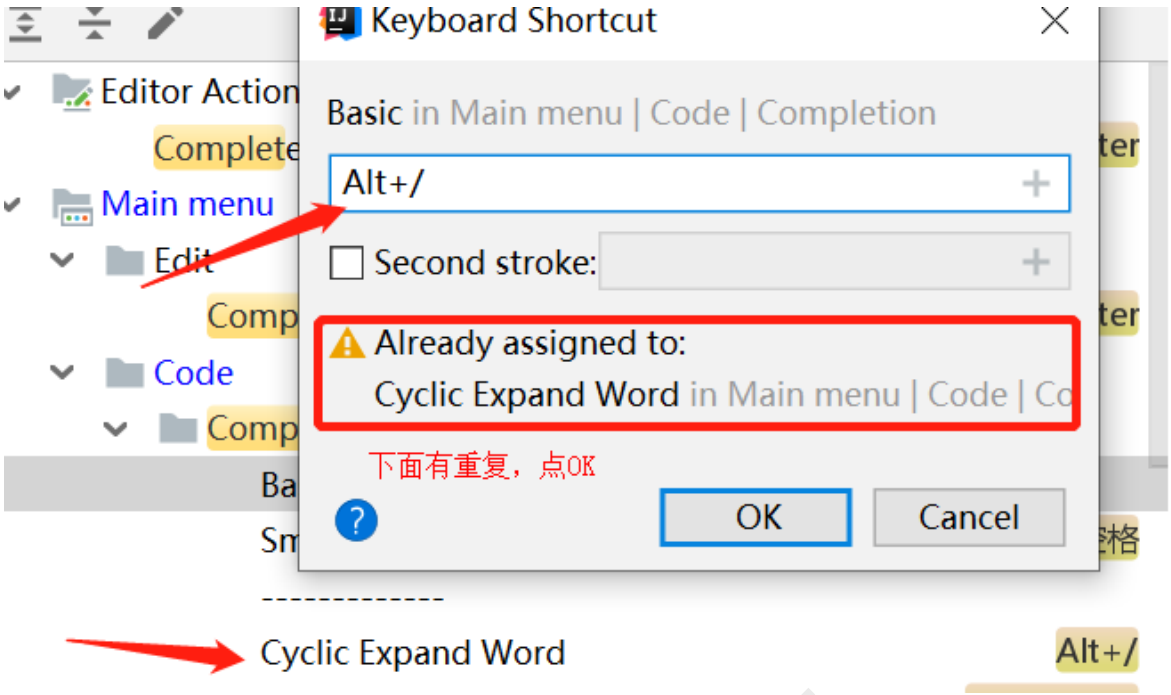
File -> Settings -> keymap 选择使用 eclipse 快捷键方式,修改代码提示快捷键（默认是 Ctrl + 空格）



Basic 右键，添加新快捷键



填入 Alt + / 提示存在，直接OK



移除其他 Alt +/ 的设置

2 软件开发概述（了解）

2.1 软件的生命周期

生命周期： 从立项到软件停用的过程

1. 问题的定义及规划: 此阶段是软件开发方与需求方共同讨论，主要确定软件的开发目标及其可行性
2. 需求分析: 在确定软件开发可行的情况下，对软件需要实现的各功能进行详细分析。需求分析阶段是一个很重要的阶段，这一阶段做得好，将为整个软件开发项目的成功打下良好的基础。
3. 软件设计: 此阶段主要根据需求分析的结果，把整个软件系统划分为大大小小的多个模块，设计出每一个模块的具体结构。如系统框架设计，

数据库设计等。软件设计一般分为总体设计和详细设计。
4. 程序编码: 此阶段是将软件设计的结果转换成计算机可运行的程序代码。在程序编码中必须要制定统一，符合标准的编写规范。以保证程序的可读性，易维护性，提高程序的运行效率。
5. 软件测试: 在软件设计完成后要经过严密的测试，以发现软件在整个设计过程中存在的问题并加以纠正。整个测试过程分单元测试（白盒）、集成测试（黑盒，功能测试、强度性能测试）以及系统测试三个阶段进行。测试的方法主要有白盒测试和黑盒测试两种。在测试过程中需要建立详细的测试计划并严格按照测试计划进行测试，以减少测试的随意性。
6. 运行维护: 安装部署软件系统，修复软件中存在的bug和升级系统。在软件开发完成并投入使后，由于多方面的原因，软件不能继续适应用户的要求。要延续软件的使用寿命，就必须对软件进行维护。软件的维护包括纠错性维护和改进性维护两个方面。

2.2 软件设计原则

为了提高软件的开发效率，降低软件开发成本，一个优良的软件系统应该具有以下特点：

1. **可重用性**：遵循 DRY 原则，减少软件中的重复代码。
2. **可拓展性**：当软件需要升级增加新的功能，能够在现有的系统架构上方便地创建新的模块，而不需要改变软件现有的结构，也不会影响已经存在的模块。
3. **可维护性**：当用户需求发生变化时，只需要修改局部的模块中的少量代码即可

如何让软件系统达到上述的特点，我们对模块的要求：

1. 结构稳定性：在软件设计阶段，把一个模块划分为更小的模块时，设计合理，使得系统结构健壮，以便适应用户的需求变化。
2. 可拓展性：当软件必须增加新的功能时，可在现有模块的基础上创建出新的模块，该模块继承了原有模块的一些特性，并且还具有一些新的特性，从而实现软件的可重用和可拓展性。
3. 可组合性：若干模块经过组合，形成大系统，模块的可组合性提高软件的可重用和可维护性，并且能简化软件开发过程。

4. **高内聚性**：内聚，强调一个系模块内的功能联系，每个模块只完成特定的功能，不同模块之间不会有功能的重叠，高内聚性可以提高软件的可重用性和可维护性。
5. **低耦合性**：耦合，强调的是多个模块之间的关系，模块之间相互独立，修改某一个模块，不会影响到其他的模块。低耦合性提高了软件的可维护性。

3 编码规范

3.1 为什么要有规范?

美国程序员枪击案　无法维护的代码

3.2 Alibaba 开发手册

阿里巴巴Java开发手册(终极版).pdf

本手册的愿景是 **码出高效，码出质量。**现代软件架构都需要协同开发完成，高效协作即降低协同成本，提升沟通效率，所谓无规矩不成方圆，无规范不能协作。众所周知，制订交通法规表面上是要限制行车权，实际上是保障公众的人身安全。试想如果没有限速，没有红绿灯，谁还敢上路行驶。对软件来说，适当的规范和标准绝不是消灭代码内容的创造性、优雅性，而是限制过度个性化，以一种普遍认可的统一方式一起做事，提升协作效率。代码的字里行间流淌的是软件生命中的血液，质量的提升是尽可能少踩坑，杜绝踩重复的坑，切实提升质量意识。

对于团队的协作开发。必须要有规范。**对于程序员来说，没有规范，就是灾难。**

3.3 SUN 规范要求

编码规范对于程序员而言尤为重要，有以下几个原因：

1. 一个软件的生命周期中，80%的花费在于维护.
2. 几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护。
3. 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码

为了执行规范，每个软件开发人员必须一致遵守编码规范。每个人

3.3 呕心沥血整理的规范

基本命名规范：使用有意义的英文单词，多个单词用驼峰表示法.

- **包名**：全小写 , 域名倒写.模块名.组件名
  cn.wolfcode.crm.util
- **接口名**：首字母大写，形容词，副词。习惯性的以 **I** 开头。此时的I表示 interface,见名知意.(不强制,要结合其他规范综合考虑)
  IUserService，IEmployeeService
- **接口实现类**: 习惯性使用 **Impl** 结尾.(不强制,要结合其他规范综合考虑)
  UserServiceImpl，EmployeeServiceImpl
- **类名**：首字母大写，名词。遵循驼峰表示法。
  User，Employee
- **方法名**：首字母小写。力求语义清晰，使用多个单词。遵循驼峰表示法。
  getUserInfoByName()，checkUsername();
  如果方法名清晰，完全可猜测出其实现的功能
- **变量名**：首字母小写。遵循驼峰表示法。
  userInfo，userId，password,username
- **常量名**：全大写，力求语义清晰，使用多个单词。使用下划线分割。
  MAX_STOCK_COUNT. 这种写法适用于 public static final 修饰的常量.

3.4 高薪秘诀

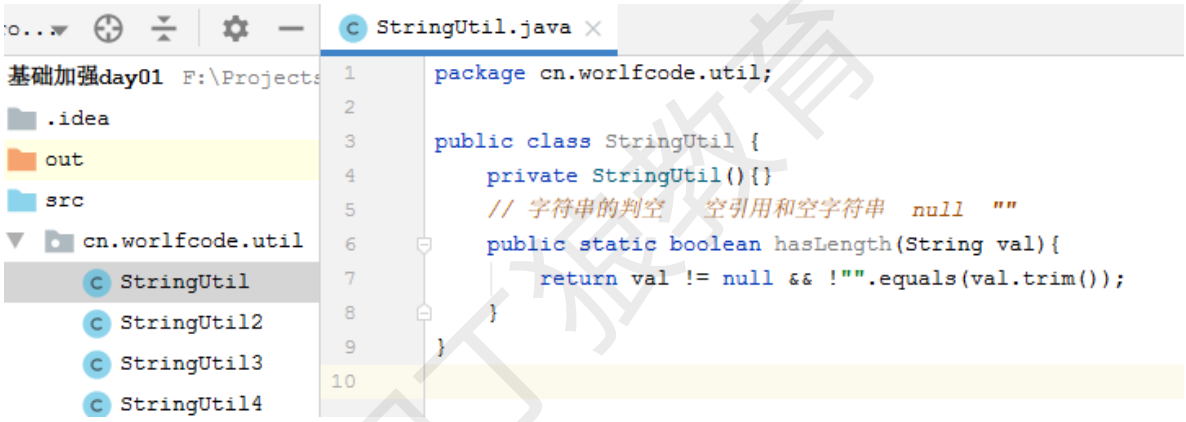
1. 从今天开始写代码必须遵循规范，无规范无高薪。
2. 跟这个存哥走，面包牛奶都会有。

jar 包引入

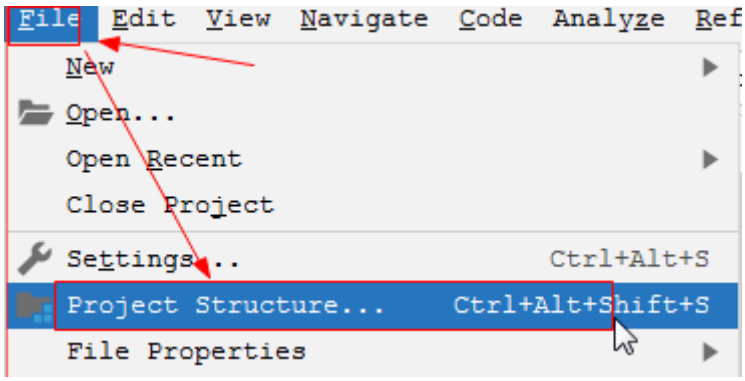
之前使用的是 jdk 内置的工具类,直接引用即可,如果现在自己想模仿 jdk 写一些工具类给别人是用,如何操作?

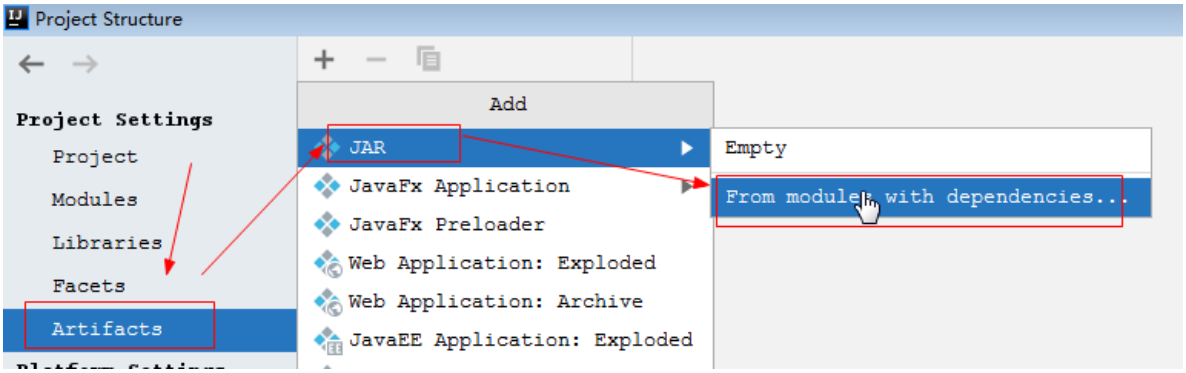
步骤:

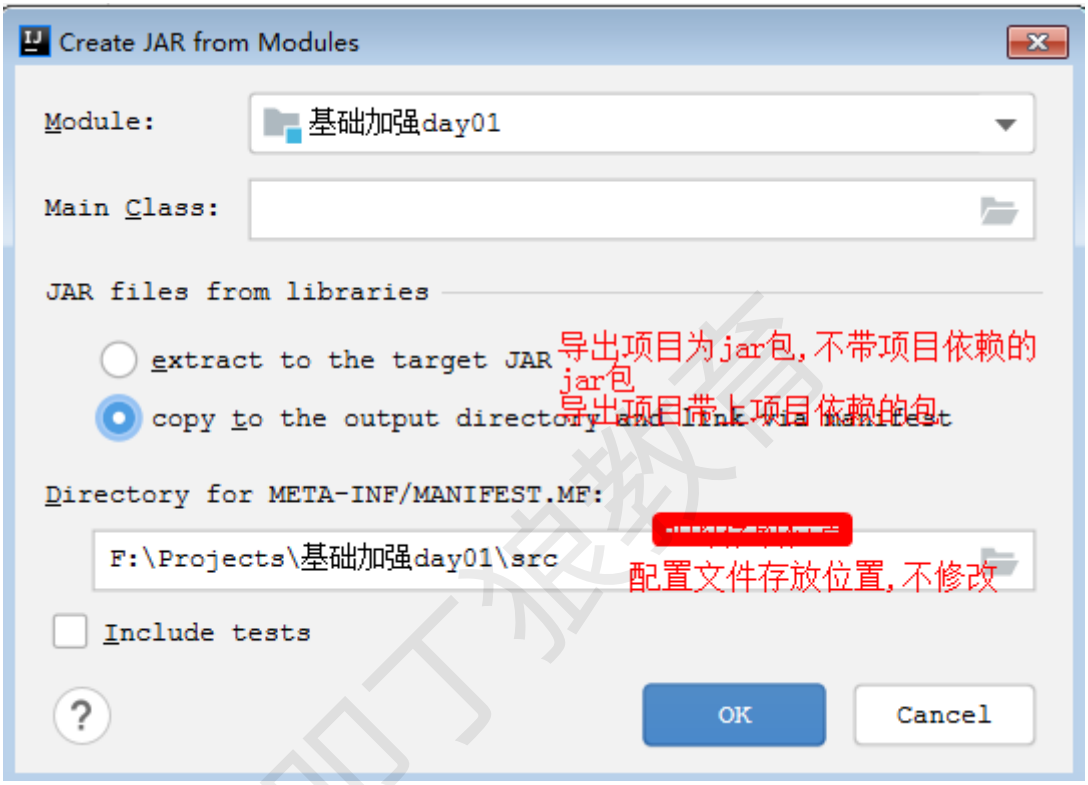
1 创建项目书写工具类和工具方法



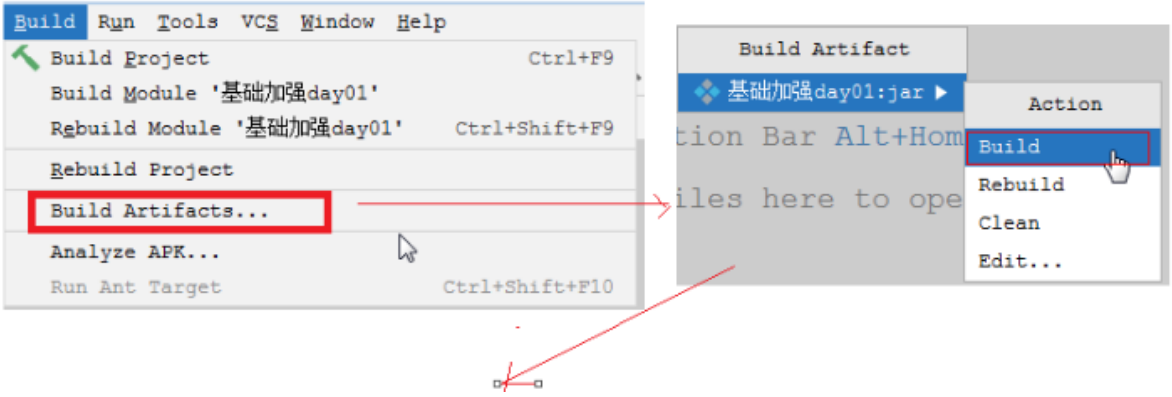
2 将项目导出为jar包,供其他项目使用



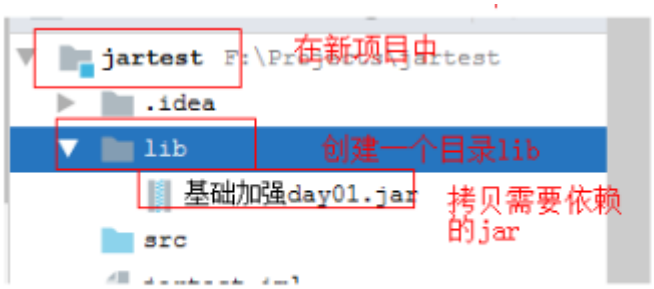

Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Add
JAR
JavaFx Application
JavaFx Preloader
Web Application: Exploded
Web Application: Archive
JavaEE Application: Exploded
Empty
From modules with dependencies...

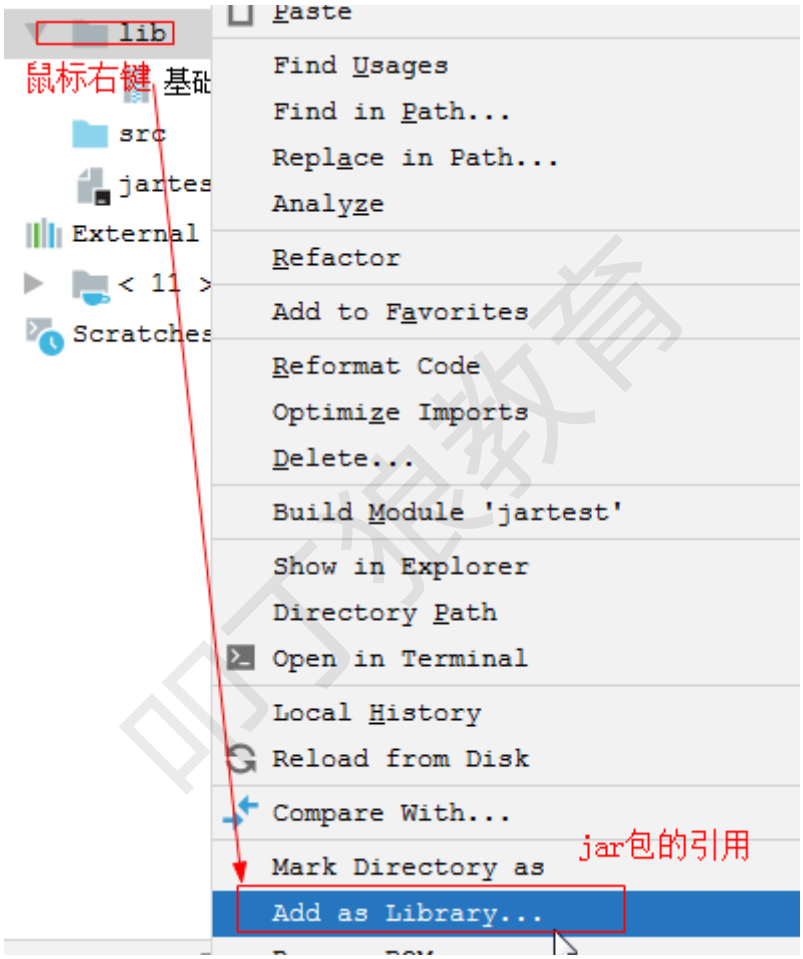

Create JAR from Modules
Module:
基础加强day01
Main Class:
JAR files from libraries
extract to the target JAR
copy to the output directory and link via manifest
导出项目为jar包,不带项目依赖的jar包
导出项目带上项目依赖的包
Directory for META-INF/MANIFEST.MF:
F:\Projects\基础加强day01\src
配置文件存放位置,不修改
Include tests
OK
Cancel


Name:
基础加强day01:jar
Type:
Other
Output directory:
E:\
指明jar导出的存放位置
Include in project build


Build
Run
Tools
VCS
Window
Help
Build Project
Ctrl+F9
Build Module '基础加强day01'
Rebuild Module '基础加强day01'
Ctrl+Shift+F9
Rebuild Project
Build Artifacts...
Analyze APK...
Run Ant Target
Ctrl+Shift+F10
Build Artifact
基础加强day01:jar
Action
Build
Rebuild
Clean
Edit...

3 在新项目中导入jar包,引用





4 使用 jar 包中的功能

```
import cn.worlfcode.util.StringUtil;

public class App {

    public static void main(String[] args) {
        String str = "ssss";
        System.out.println(StringUtil.hasLength(str));
    }
}
```

如果以后需要使用别人写好的功能,就只需要拷贝别人的 jar 包到自己的项目中,然后引用就可以只用使用 jar 包中的功能.

4 单元测试

4.1 软件测试介绍(了解)

在软件生命周期中,测试是一个重要的环节,它从编码开始,甚至可能早于编码就已经进入了软件生命周期了 .**优秀的软件,不是开发出来的,而是测试出来的**. 一个软件是否开发完毕,并不是开发工程师说了算,而是测试工程师说了算.只有经过精密测试后的软件,才能上线使用.

4.1.1 测试分类

站在是否能够看到代码的角度，是否需要写代码的角度来进行分类

黑盒测试

黑盒测试也称功能测试，是通过测试来检测每个功能是否能正常使用，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，在程序的接口上进行测试，检查程序功能是否 **按照需求规格说明书** 的规定正常使用。

结论： 手动测试，不用写代码

作用：主要试图发现下列几类错误
功能是否不正确或遗漏；
界面是否有错误；
输入和输出错误；
数据库访问错误；
性能是否有问题；
初始化和终止错误等

白盒测试

由开发人员来测试. 又称结构测试、透明盒测试、逻辑驱动测试或 **基于代码** 的测试。
它是按照程序内部的结构测试程序，通过测试来检测产品内部动作是否按照设计规格说明书的规定正常执行。测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。

结论：代码测试，需要写代码

作用：
检查程序内部结构是否存在遗漏
程序功能是否完成
程序功能是否有错
程序功能是否按规格说明书实现

而作为 Java 攻城狮，白盒测试是离不开的，接下来要学习的 JUnit 测试就是白盒测试

4.1.2 为什么要学习测试?

在你不知道如何测试代码前,就不要写代码去完成功能. -- **测试先行**
（marting flower）

除非测试成功,否则你不能认为你编写出了可以工作的程序.

4.2 JUnit 使用

JUnit 是由 Erich Gamma 和 Kent Beck 编写的一个回归测试框架（regression testing framework）。JUnit 测试是程序员测试，即所谓白盒测试，因为程序员知道被测试的软件如何（How）完成功能和完成什么样（What）的功能。
官网：http://www.junit.org/

Java 的单元测试:Junit,存在三个版本

1. junit3.x 针对于 Java5 之前的版本,android 中使用，不推荐.
2. junit4.x 针对于 Java5 以及之后的版本,使用注解,推荐.
3. junit5.x 针对于 Java8 以及以后的版本.

4.2.1 问题说明

目前我们实现完一个功能需要进行测试，比如现在要测试 Person 类中的 work 方法.测试如下。

```
class Person{
        public void work(){
                System.out.println( "Person.work()" );
        }
}
```

```
class PersonTest{
        public static void main(String[] args){
                Person p = new Person();
                p.work();
        }
}
```

问题：如果要测试100个方法就需要创建100个测试类，然后创建对象，调用方法，非常麻烦，

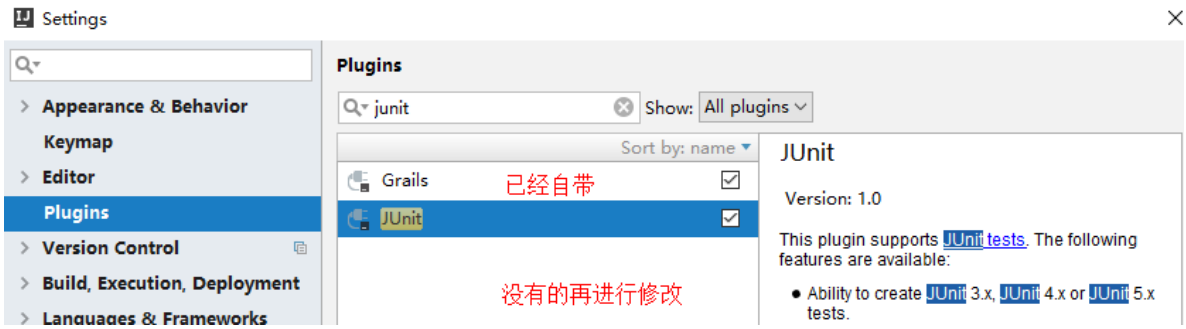
要保证每个功能的修改都能马上执行测试，目前实现太繁琐。怎么办？ 学习JUnit。

4.2.2 JUnit4 使用步骤

1 JUnit 4 依赖安装

由于 JUnit4 回归测试框架是三方提供的,不是 JDK 自带的，所有要使用需导入人家的 jar 包以及安装对应的插件

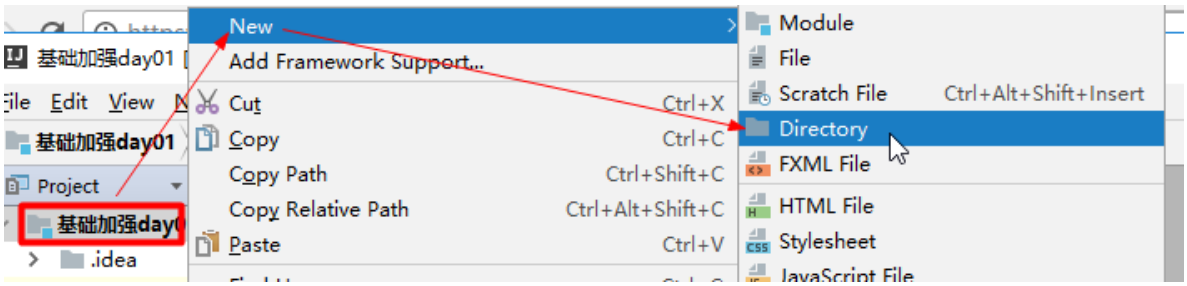
Idea 插件安装：File -> Settings -> Plugins -> 搜索 junit ，默认是自带的，没有的在此处安装。



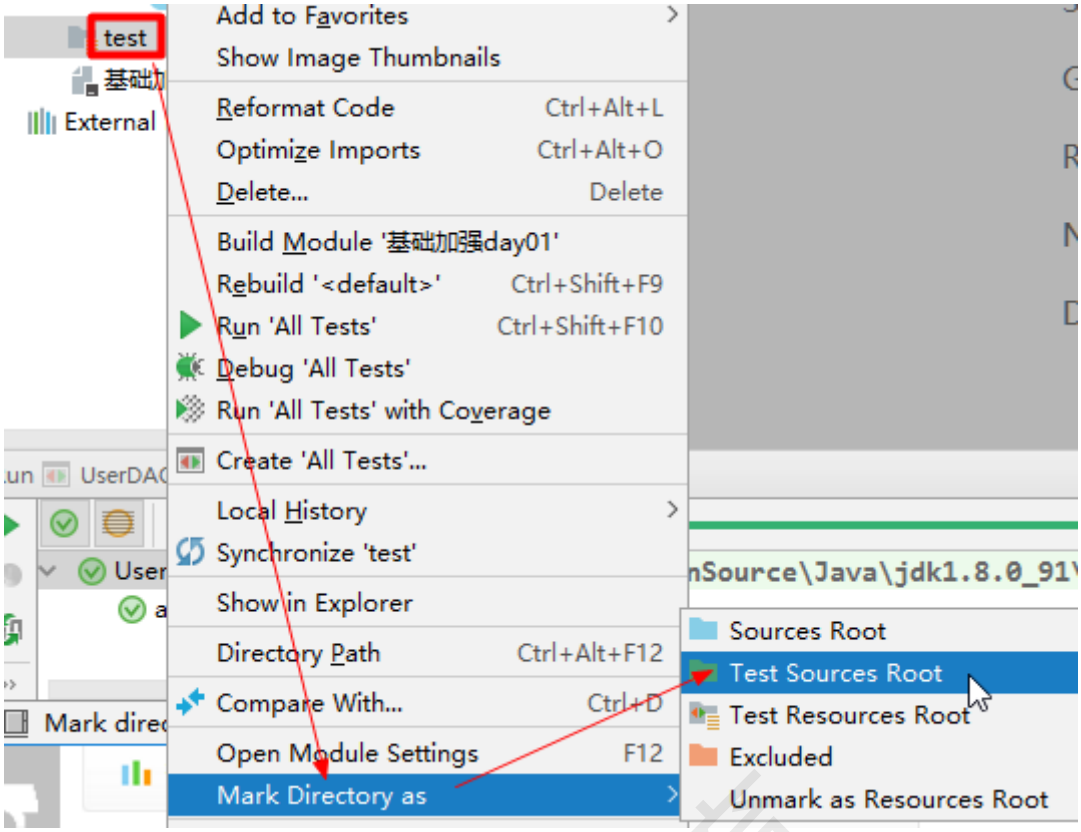
2 书写功能类，带有需要被测试的功能方法

```
/**
 * 功能类
 */
public class UserDAOImpl {
    public int add(int a, int b) {
        return a + b;
    }

    public void delete(int index){
        System.out.println("删除index");
    }
}
```

3 创建测试目录（创建一个普通文件夹test，make 为 测试文件夹 Test Soruces Root)



将普通目录改为测试目录



4 创建测试包和测试类

在 UserDAOTest 中编写测试方法:如

```
@Test
public void testXxx() throws Exception {
}
```

注意: 方法是 public 修饰的,无返回的无参数,该方法上必须贴有 @Test 标签, Xxx 表示测试的功能名字.

书写测试方法

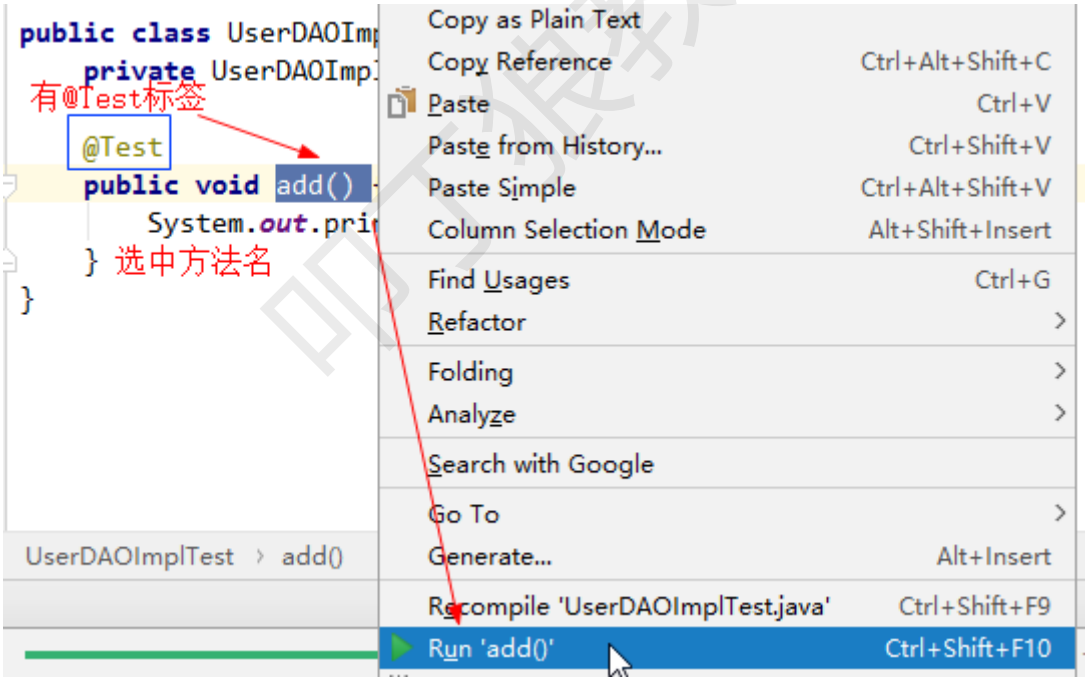
```java
public class UserDAOImplTest {
    // 引入要测试的类
    private UserDAOImpl userDAO = new UserDAOImpl();

    @Test
    public void add() {
        System.out.println(userDAO.add( a: 1,  b: 2));
    }                          书写测试代码

    @Test
    public void delete() {
        userDAO.delete( index: 1);        书写测试代码
    }
}
```

5 执行测试方法

选择某一个测试方法,鼠标右键选择 run testXxx(),或点击方法前的绿色三角形

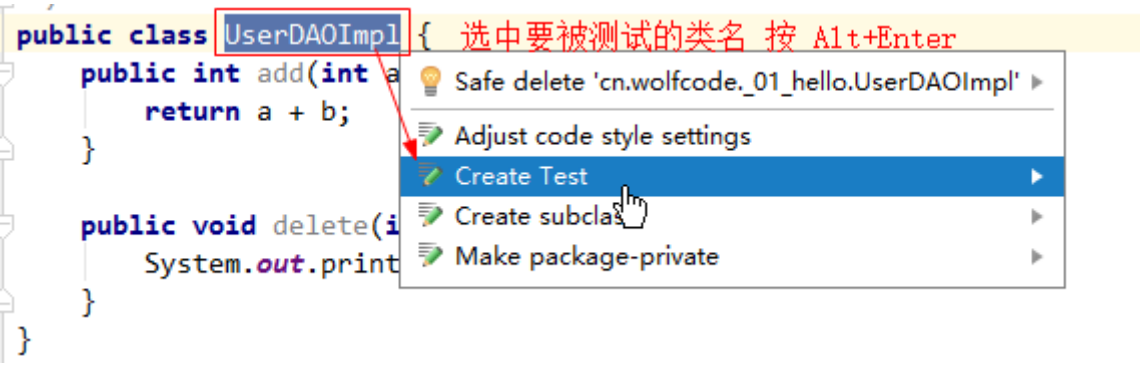
或者选中测试类,表示执行该类中所有的测试方法.

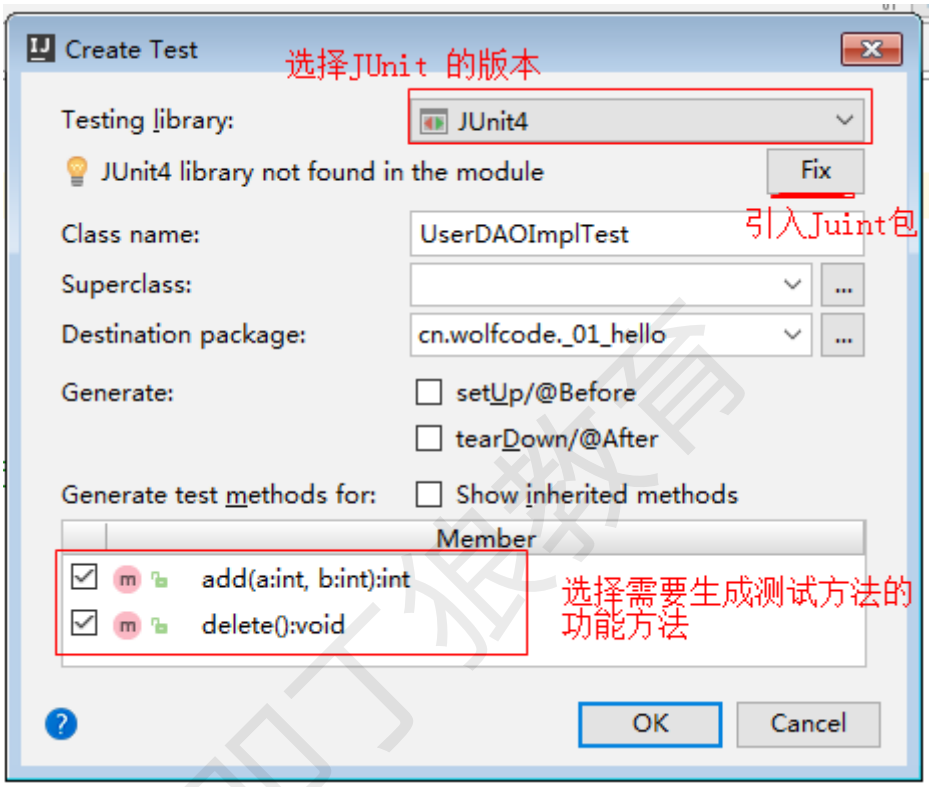
执行测试 (选中方法，右键运行)



4.3 生成测试类

选中要测试的类名 Alt + Enter-> Create Test





5 配置文件讲解

5.1 前言

在开发中，有时候存一些数据是直接在代码的变量中来实现的，但如果变量的值是需要偶尔改下的，这个时候就存在硬编码。

例子：用Java程序去连接其他程序，比如数据库，需要使用账号和密码，此时我们是使用两个变量来记录：如下

```
private String username = "root";

private String password = "admin";
```

想要使用账号和密码,直接重复使用两个变量即可.

问题： 账号和密码是直接写死在程序中的，如果数据库的账号密码改了，此时就需要修改代码，然后编译等才能使用，工作量大，

不利运维人员部署项目。这种显现成为 **硬编码**。

如何解决硬编码?

可以把这些需要修改的内容写在一个普通的文件中，然后程序去读取文件内容，以后修改只需修改普通文件即可。

例：user.txt 存入 username=root password=admin

而这种存了数据给程序使用的文件称为配置文件。

5.2 高效的配置文件

按理说只要能保存一些配置信息,供程序动态读取数据就OK，但是为了提高效率，在 IT 行业中，习惯使用两种具有特殊特点的文件来作为配置文件，分别是 **properties 文件 和 XML文件**

5.2.1 properties 文件

该文件称属性文件 / 资源文件 / 配置文件, 以 properties 作为文件后缀名

存数据特点： key=value 格式，多对数据使用换行分开。

user.properties

username=root

password=admin

使用注意事项:

1.配置文件需要跟随着字节码走.需要放在 Resource Root 中. 会直接编译到字节码输出路径

2 在配置文件中，所有的数据都是字符串，不需要使用引号

3 在配置文件中不需要使用空格

创建步骤:

1 创建普通文件夹 resource

2 将文件夹改为 Resources Root 目录

3 在目录中创建配置文件

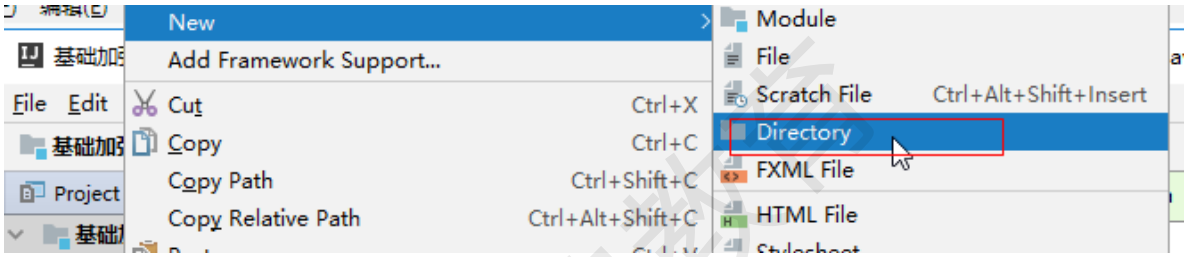


image-20200701144136837

有了配置文件，接下来就是读取配置文件中的数据，那要如何读取呢?

意识:如果一个操作非常复杂,自己实现非常麻烦.而很多人都需要，这个时候一般有人提供好的工具来实现该功能

比如,想要读取 properties 中的数据,我们使用 IO 操作,一行一行的读取,再通过"="来切分字符串也可以完成.但是还是比较麻烦的,如果有注释更麻烦.

此时就要有意识, SUN公司已经提供好了读取 properties 的工具方法.这就是 Properties.

5.2.2 解析 properties 文件

JDK API

Properties 是 Map 的实现类.可以继承过来map的常见的操作方法(get,put....),map 中的方法,我们一般都不用.因为 Properties 文件比较特殊,使用 Properties 类的新增的方法.

常用的 API

- public void load(InputStream inStream); // 通过输入流加载配置文件中的内容.
- public String getProperty(String key); // 通过属性名获取属性值

```
@Test
public void testGetUser() throws Exception {
    Properties ps = new Properties();
    //1.通过输入流,加载配置文件(使用绝对路径)
    InputStream in = new FileInputStream(
            new File("E:/sts-projects/day03-配置文件/resources/user.properties"));
    ps.load(in);// 数据通过load之后,就已经存储到map中
    //2.使用Properties对象去获取属性值
    String username = ps.getProperty("username");
    String password = ps.getProperty("password");
    System.out.println(username + "," + password);
    // 从配置文件中获取出来了账号和密码.调用登录功能.
}
```

上面的代码,使用了绝对路径,在开发中绝对不能使用.可以使用相对路径(相对于字节码输出路径).

使用相对路径来加载配文件

使用类的加载器，直接从字节码的输出路径去读取配置文件

类的加载器是哪个？ClassLoader

如何获取类的加载器?

如何通过类的加载器去获取字节码输出路径下的文件?

```java
public class PropertiesDemo2 {
    public static void main(String[] args) throws Exception {
        // 读取配置文件中的数据
        Properties p = new Properties();

        // 解决配置文件路径写死的问题,使用相对路径
        // 相对于字节码的输出根路径,使用ClassLoader对象来获取 字节码输出路径下的文件
        // 为了获取ClassLoader 对象,跟 Thread 没有关系
        ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
        // 去字节码的根路径下去读取配置文件
        InputStream is = classLoader.getResourceAsStream( name: "user.properties");

        // 加载配置文件
        p.load(is);

        // 获取已经加载到 Properties 对象中的数据
        System.out.println(p.getProperty("username"));
        System.out.println(p.getProperty("password"));
    }
}
```

小结

1 环境的搭建

```
+ 使用idea 去创建项目（掌握)
+ 能够修改默认配置 -> 看着文档修改即可
```

2 今天开始写代码需要遵循编码规范-> 命名规范

3 掌握单元测试 JUnit 的使用 -> 创建测试目录，生成测试类，测试对应的方法

4 掌握配置文件

```
+ 配置文件的书写要求
+ 配置文件 properties 的创建使用 -> 创建 Resource Root 目录，创建 properties文件
+ 使用相对于字节码跟路径的路劲去解析配置文件 properties
```