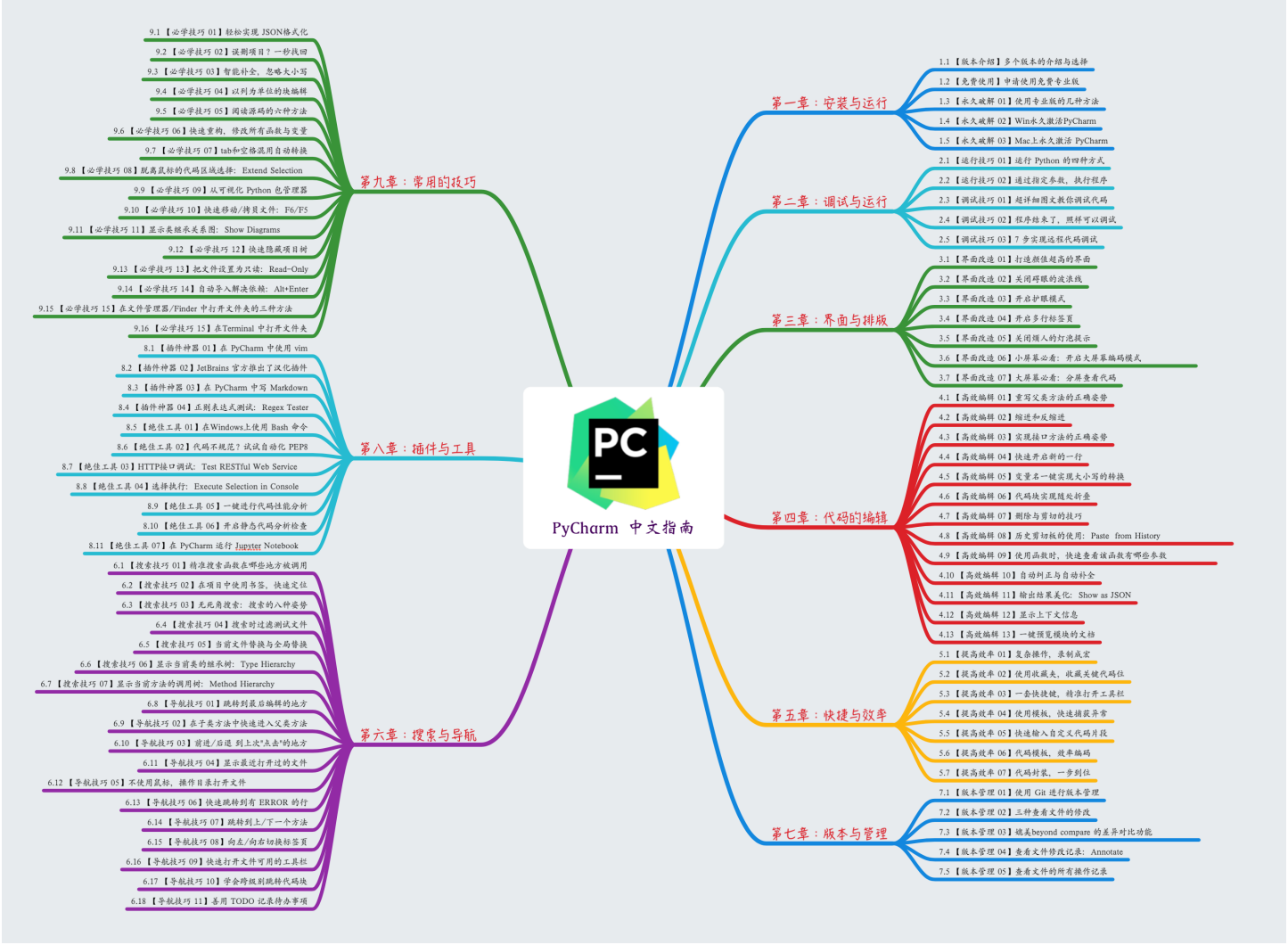
PyCharm 中文指南

作者：王炳明
版本：v1.0
发布时间：2020年08月30日
微信公众号：Python编程时光
联系邮箱：wongbingming@163.com
项目主页：http://pycharm.iswbm.com
Github：https://github.com/iswbm/pycharm-guide

回复 "pycharm"，获取最新版 PDF

目录大纲



前言

本书是一本面向 Python 开发者的 PyCharm 详细使用指南。

为了更好的展示使用技巧，原博客里采用了大量的 GIF 动态图解。而在导出为 PDF 电子版后，动态图会变成静态图片。

因此当你看到一张图片下面有如下提示，说明这张图是 GIF 动态图，如想查看可以前往我的项目主页（http://pycharm.iswbm.com/）。

> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

另外写本书时，大部分都是基于 PyCharm 2020.2 版本的 Mac 电脑写的，若截图的界面和快捷键有与环境上的不一致，请切换到同一版本下再次尝试。

由于 Mac 和 Windows 的键盘布局不同，有些快捷键我只给出了 Mac 版本的，不过不要紧，你只要掌握下面这个规律，就能轻松切换。

Mac	Windows
⌘ :Command	ctrl
∧: ^	ctrl
⇧ :Shift	Shift
⌥ :Option	Alt

如果转换还有误的，可以到我的公众号（Python编程时光）后台回复 "**pycharm快捷键**"，获取两张 Jetbrains 公开的 PyCharm 所有快捷键表格，有 Windows 和 Mac 版的。

第一章：下载与安装

1.1 【版本介绍】多个版本的介绍与选择

Jetbrain 公司是一家专业的 IDE 生产商，只要是市面上主流的编程语言，Jetbrain 都有相应的产品。

比如：Python 对应 PyCharm ， Golang 对应 Goland，Java 对应 IntelliJ IDEA，C 语言对应 Clion 等等。

在这些众多的 IDE 中，有一些提供了多种版本：`教育版`、`社区版` 和 `专业版`。

> *PyCharm Edu is based on PyCharm Community Edition and comprises all its features, making it just perfectly suitable for writing professional projects with Python.*

教育版：教育版是免费的，具备社区版的所有功能，除此之外，还提供有一个教学功能，因此它更适合学生。老师可以用它创建教学，学生可以通过他完成教学任务。

社区版：就是阉割版的专业版，它也是免费的，如果你并不需要使用专业版才有那些功能，可以选择社区版。

专业版：提供所有 PyCharm 的功能，虽然是收费的，但是可以试用一个月。

社区版和专业版在功能上有哪些区别呢？你可以看下面这个表格。

可以看出专业版比社区版多了 `科学工具`、`WEB 开发`、`Python Web 框架`、`Python 代码分析`、`远程开发调试`、`数据库支持`。

	PyCharm Professional Edition	PyCharm Community Edition
Intelligent Python editor	✓	✓
Graphical debugger and test runner	✓	✓
Navigation and Refactorings	✓	✓
Code inspections	✓	✓
VCS support	✓	✓
Scientific tools	✓	
Web development	✓	
Python web frameworks	✓	
Python Profiler	✓	
Remote development capabilities	✓	
Database & SQL support	✓	

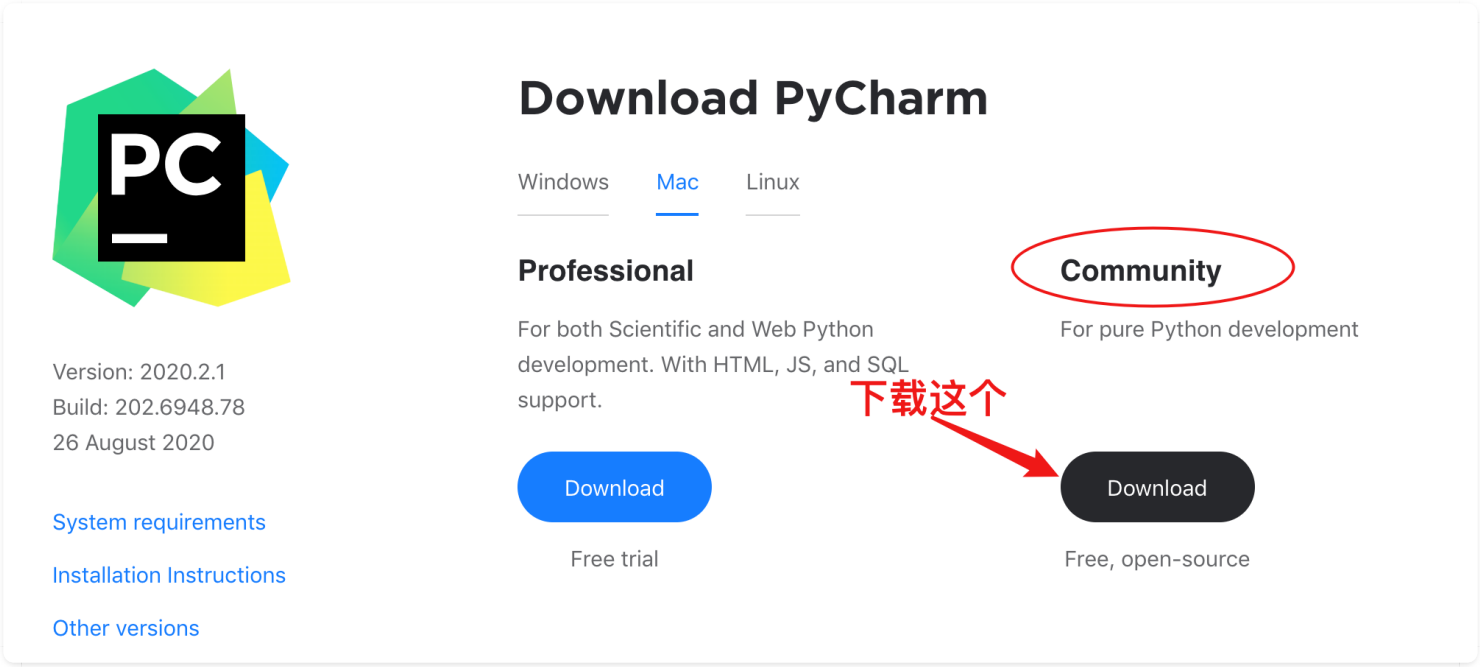
1.2 【安装使用 01】下载使用社区版

1. 下载链接

PyCharm for Windows： https://www.jetbrains.com/pycharm/download/#section=windows

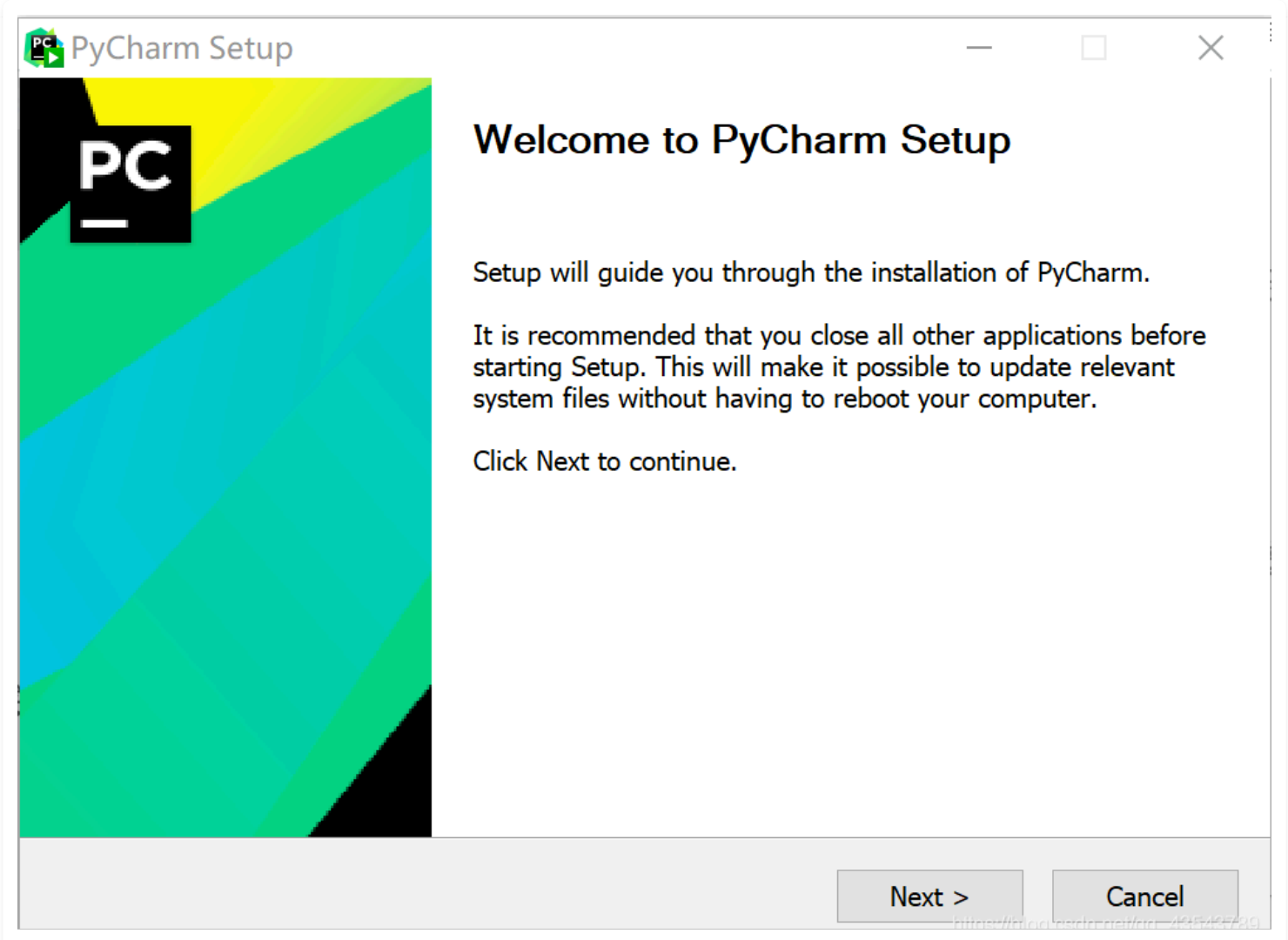
PyCharm for Mac： https://www.jetbrains.com/pycharm/download/#section=mac

PyCharm for Linux： https://www.jetbrains.com/pycharm/download/#section=linux

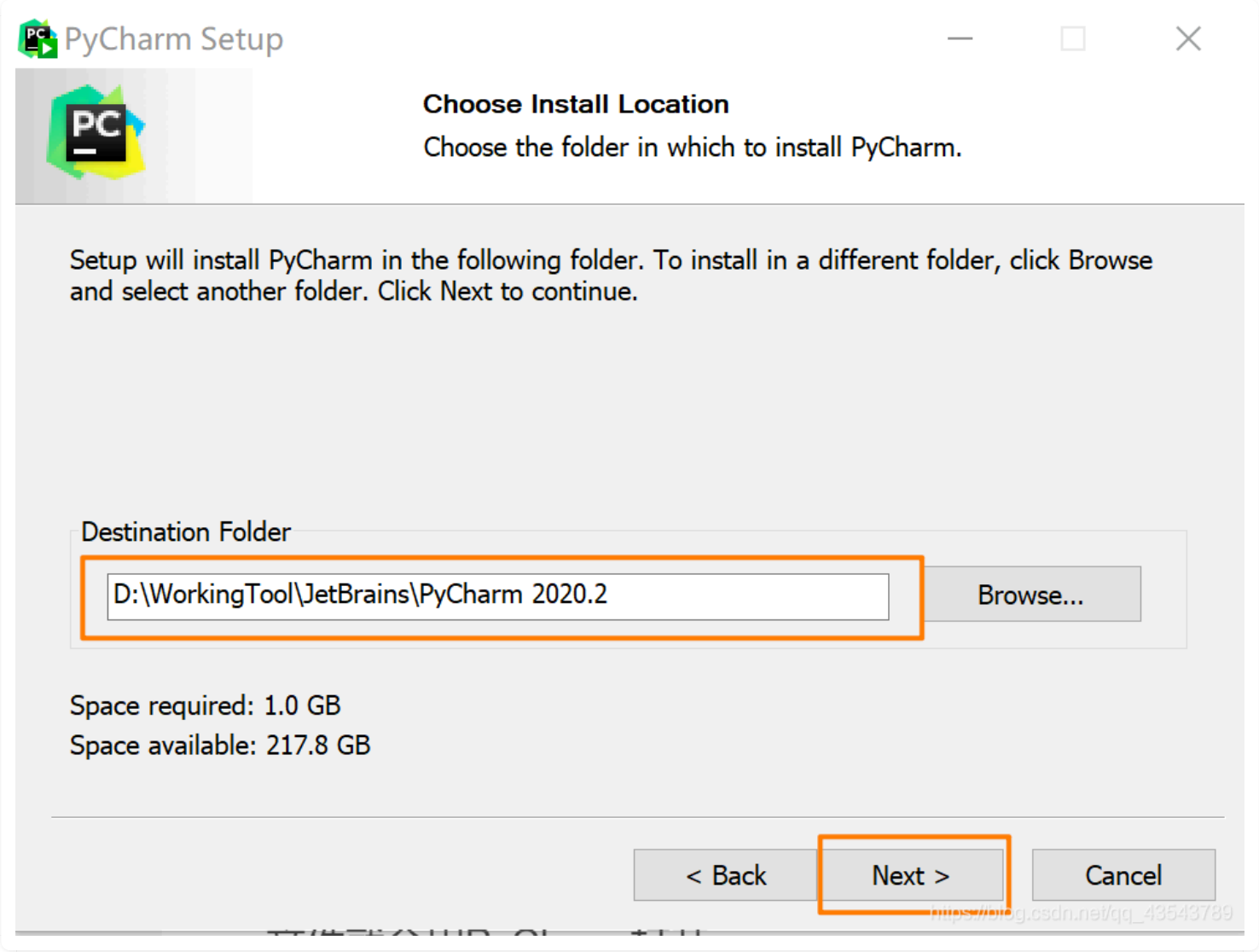


2. 安装步骤

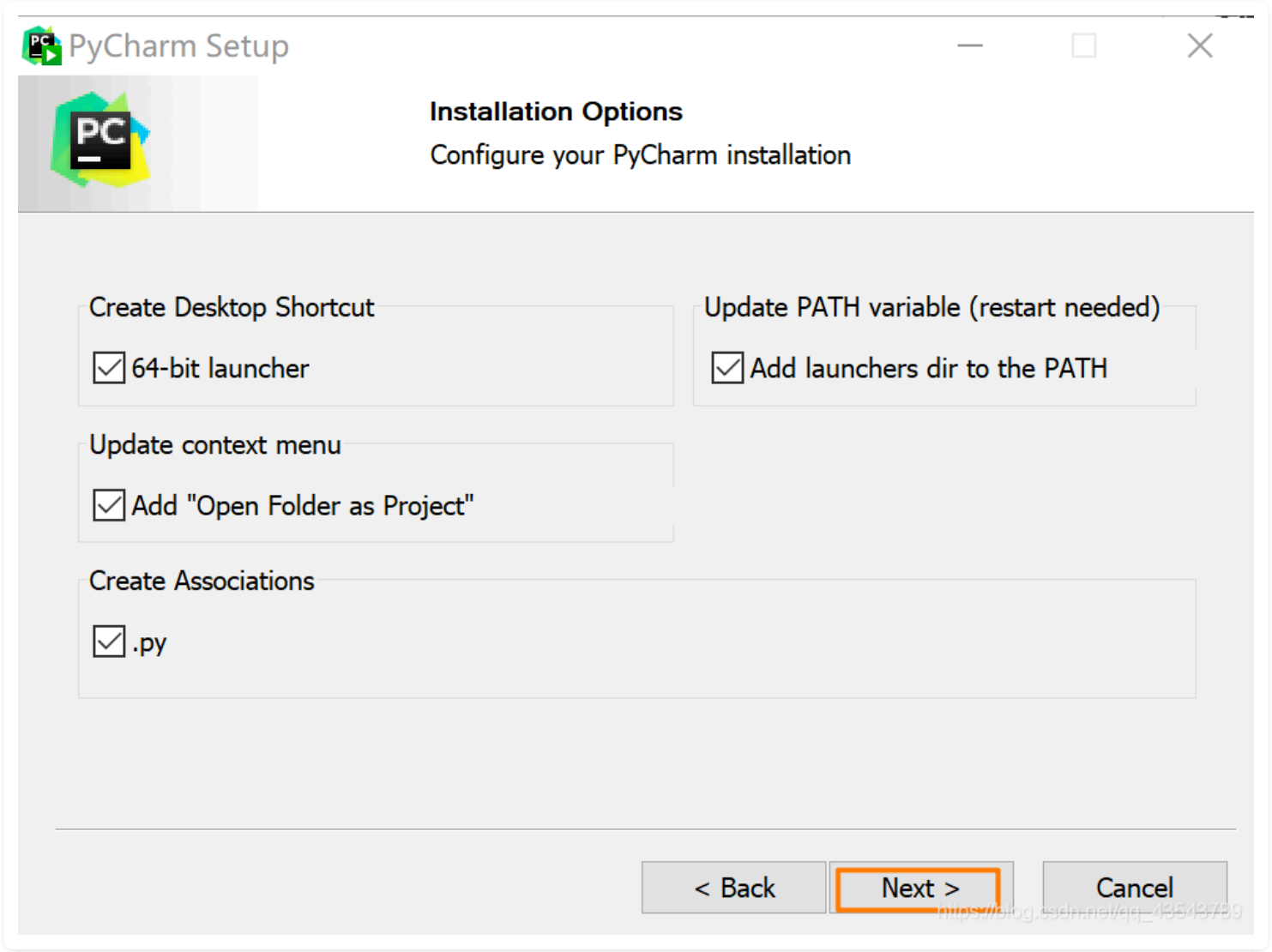
下载完成后，双击 exe 文件



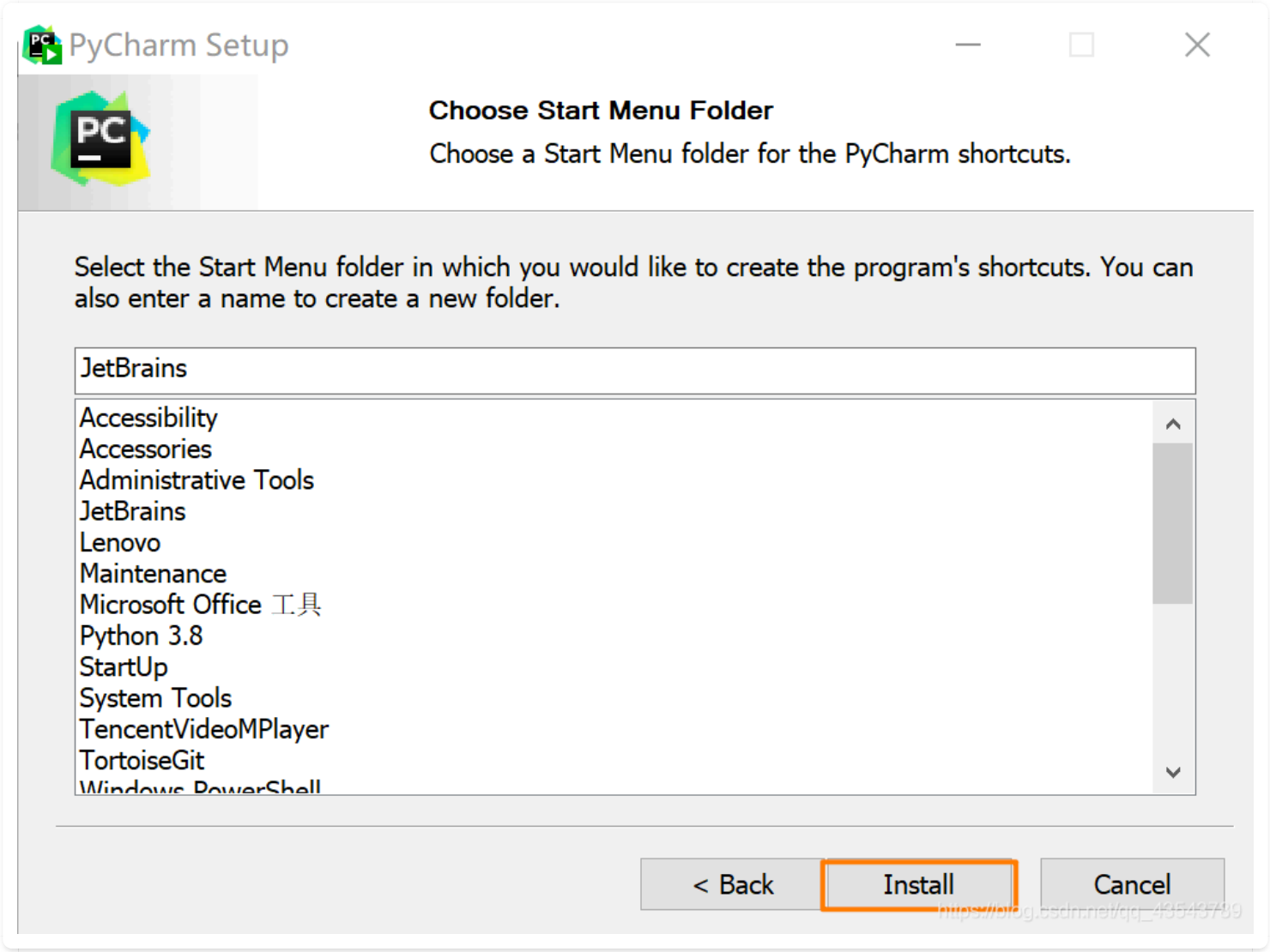
选择安装目录，Pycharm需要的内存较多，建议将其安装在D盘或者E盘，不建议放在系统盘C盘：



选好路径后，点击 Next ， 创建桌面快捷方式等一系列选项参照下图勾选！

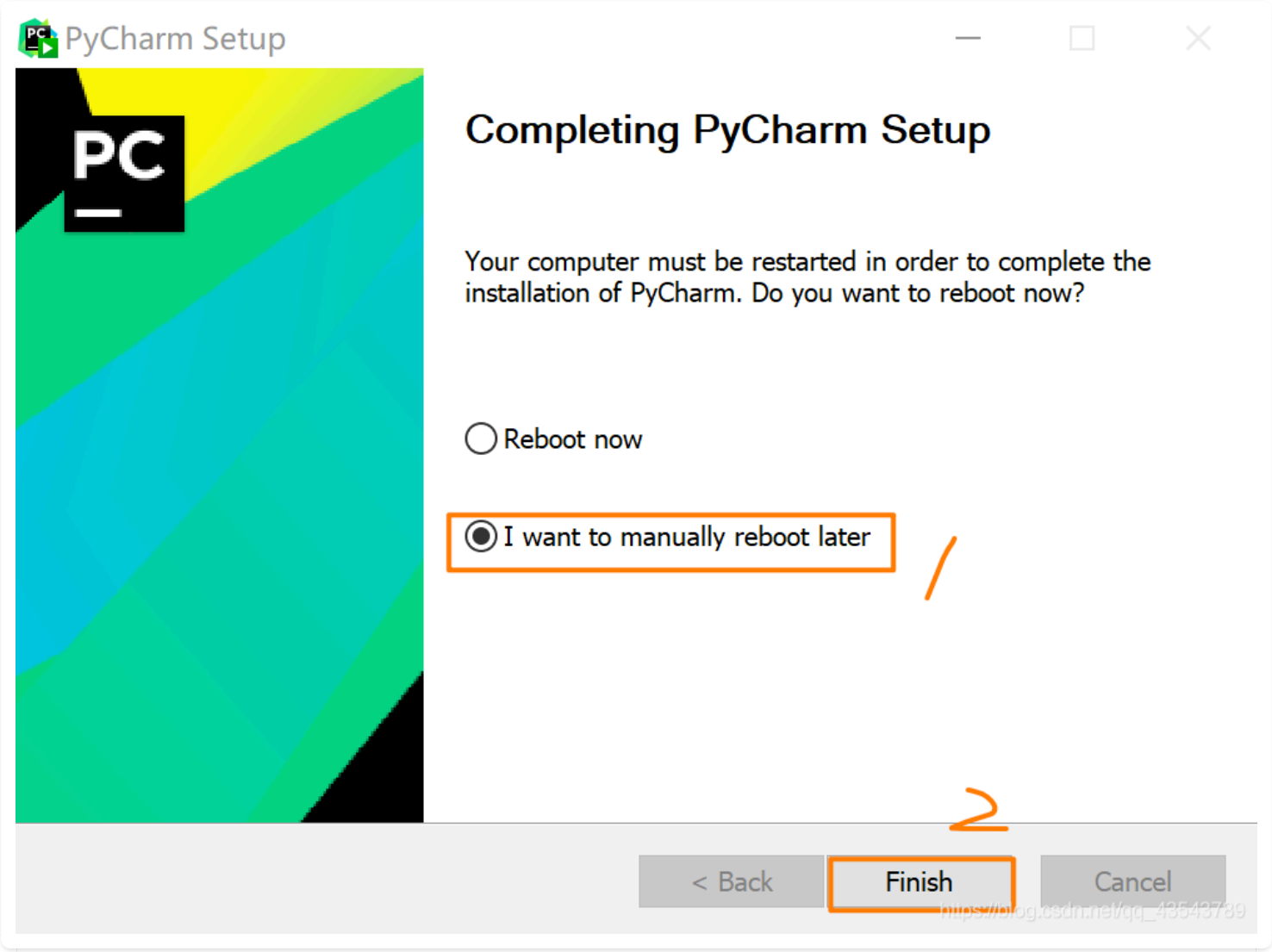


最后默认安装即可，直接点击Install。



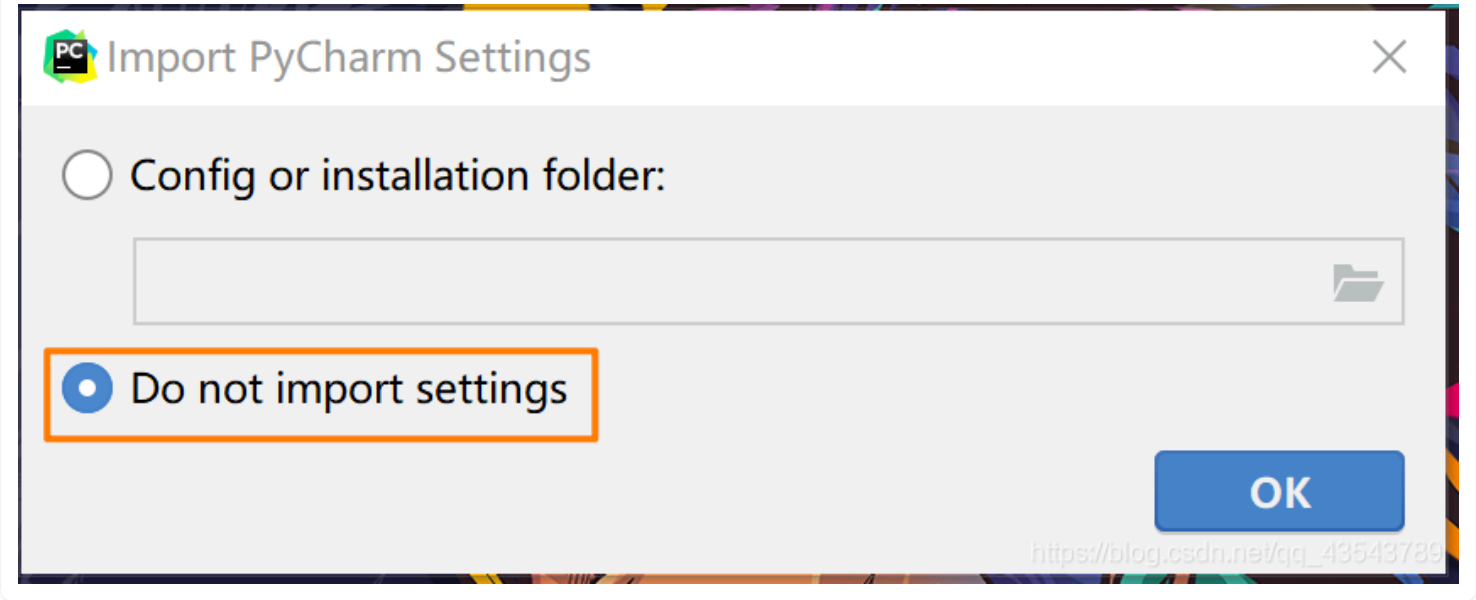
7、耐心的等待两分钟左右。

之后就会得到下面的安装完成的界面



点击Finish，Pycharm安装完成。

接下来对Pycharm进行配置，双击运行桌面上的Pycharm图标，进入下图界面：

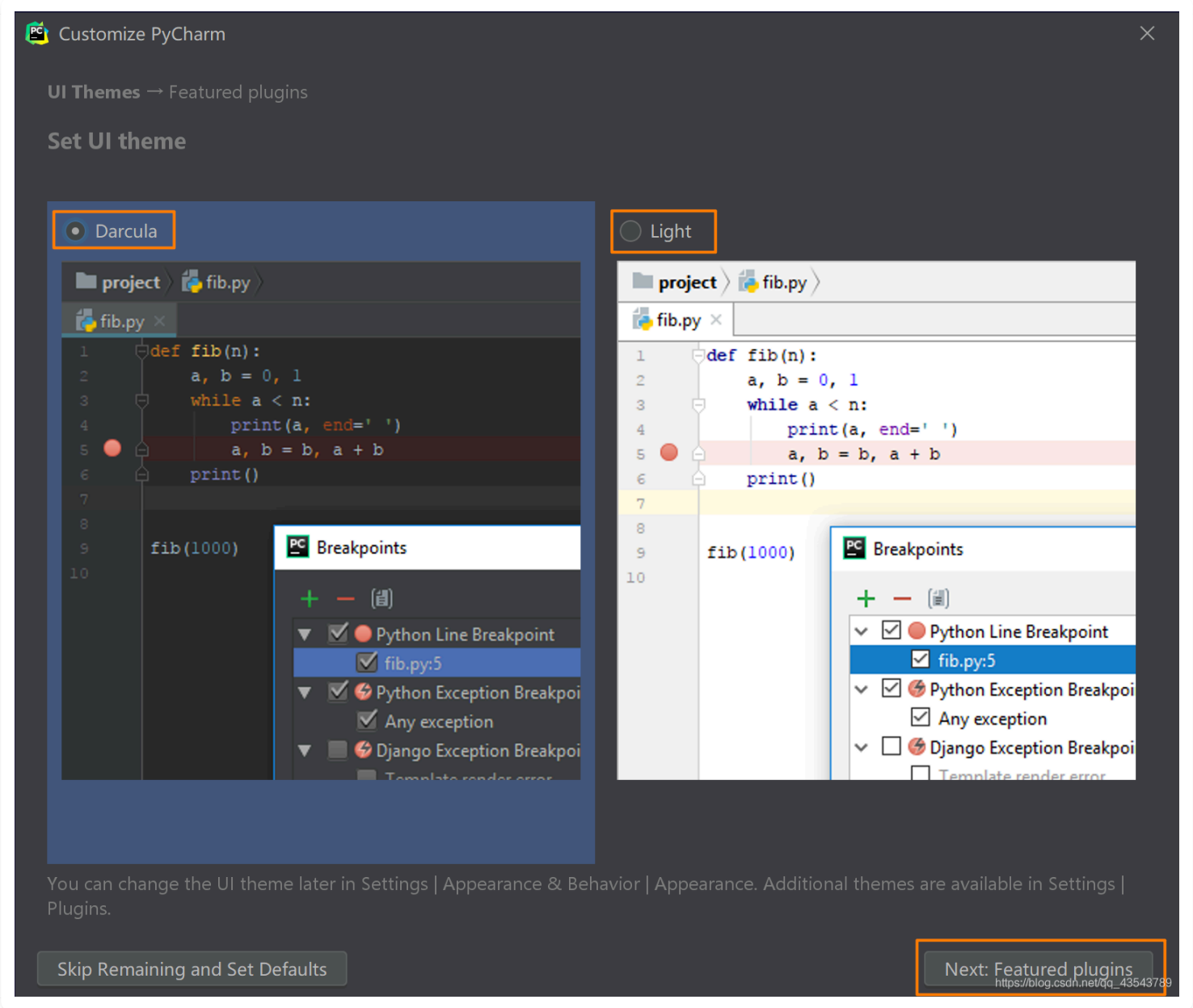


选择Do not import settings，之后选择OK，进入下一步。

下面是选择主题

-> 这里默认选择黑色(左边黑色,右边白色)

-> 点击Next:Featured plugins



建议选择Darcula主题，该主题更有利于保护眼睛。

至此，PyCharm 就安装完成。

1.3 【安装使用 02】使用专业版的五种方法

社区版的功能有限，有些非常好用的功能只有专业版才有，比如 远程调试。

如果你想使用到专业版，那有什么办法呢？

1. 有钱的就是大爷，付费购买。

2. 穷人自有穷活法，每次试用一个月，试用期到，卸载干净，再来一次。
3. 利用学生与教师的特权，可申请免费使用（下一节说到）
4. 若你有开源项目，也可以申请免费使用
5. 用一些 `非寻常手段` （也就是破解）来实现。

破解的方法，其实还分很多种：

1. 可以使用注册服务器的方式，优点是非常方便，缺点是过一段时间就有可能失效，不稳定。
2. 还可以使用破解补丁的方式，优点是永久破解（使用期限到 2099 或者 2100年，某种意义上算是永久了），缺点是对于最新版的 PyCharm 你可能找不到相应的破解补丁。如果要使用这种方法，就意味着你得使用旧版的 PyCharm 。
3. 使用绿色免安装的 PyCharm 安装包，其实原理和第二种一样，这一种只是别人帮你破解好，你直接用而已。只适用于 Windows 。

1.4 【免费使用 01】学生和老师可申请免费专业版

有一种邮箱，叫做教育邮箱，这东西在这个互联网的世界有很大的优惠及特权，在 Jetbrain 这里，如果你有教育邮箱(`.edu.cn` 后缀的邮箱)但很多学生、甚至老师都未必有。

没有教育邮箱怎么办?

你只要能提供能证明你的学生或者老师身份的证明，比如学生证、教师证等，Jetbrain 也可以让你免费一定期限（申请地址：https://www.jetbrains.com/student/），学生证有效期是一年，每年都要复审一次，老师的话就是长期的了。

1.5 【免费使用 02】利用开源项目申请免费专业版

Jetbrains 鼓励开源，只要你拥有一个符合条件的开源项目，你或者你的团队就可以免费使用 `JetBrains` 公司旗下所有的 `Ultimate` 版本的 `IDE` 开发工具，即全家桶的使用权 `1` 年，如果到期了还可以继续申请。

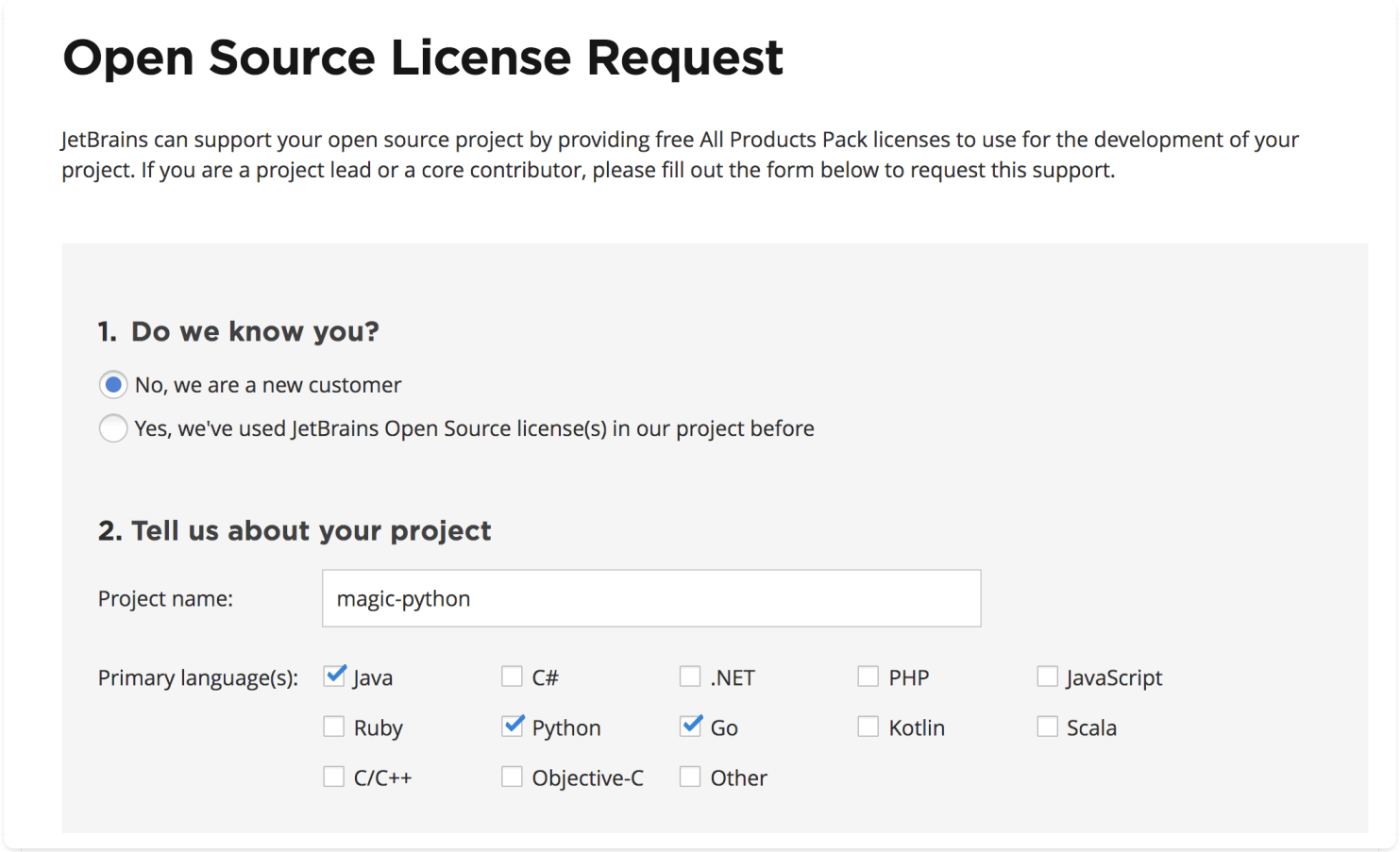
大致的开源项目要求是这个样子的：

- 你必须是项目的发起人或是活跃的 `commiter`
- 你的项目需要积极开发 `3` 个月以上
- 定期发布版本
- 符合开源的定义，不能包含有关商业性质的内容

如何申请

申请链接: https://www.jetbrains.com/shop/eform/opensource

按表单要求填写即可。



Open Source License Request

JetBrains can support your open source project by providing free All Products Pack licenses 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: magic-python

Primary language(s):

[x] Java	[ ] C#	[ ] .NET	[ ] PHP	[ ] JavaScript
[ ] Ruby	[x] Python	[x] Go	[ ] Kotlin	[ ] Scala
[ ] C/C++	[ ] Objective-C	[ ] Other		

等待个一天左右，你的邮箱估计就能收到激活码了。

第二章：调试与运行

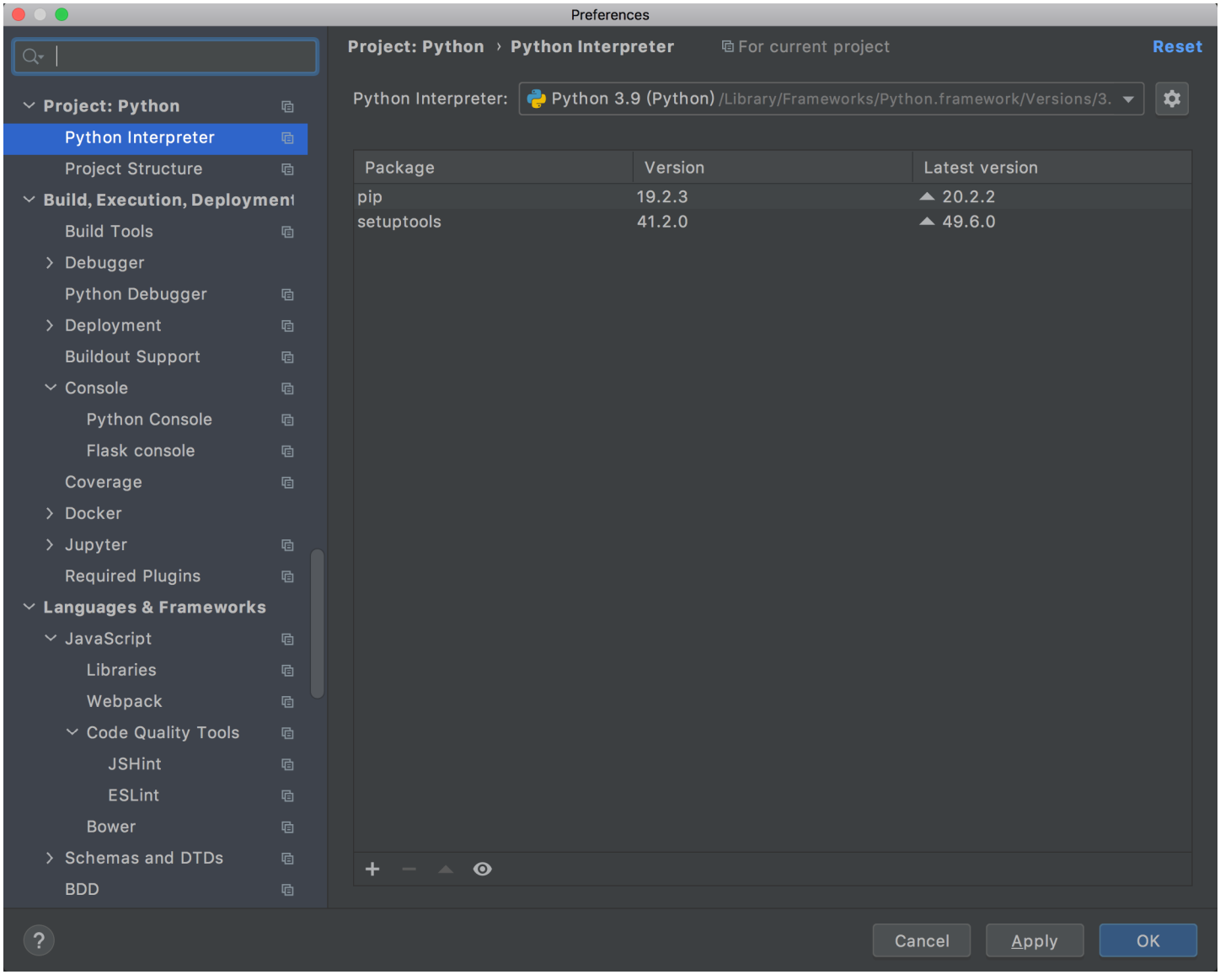
2.1 【运行技巧 01】运行 Python 的四种方式

1. 设置 Python 解释器

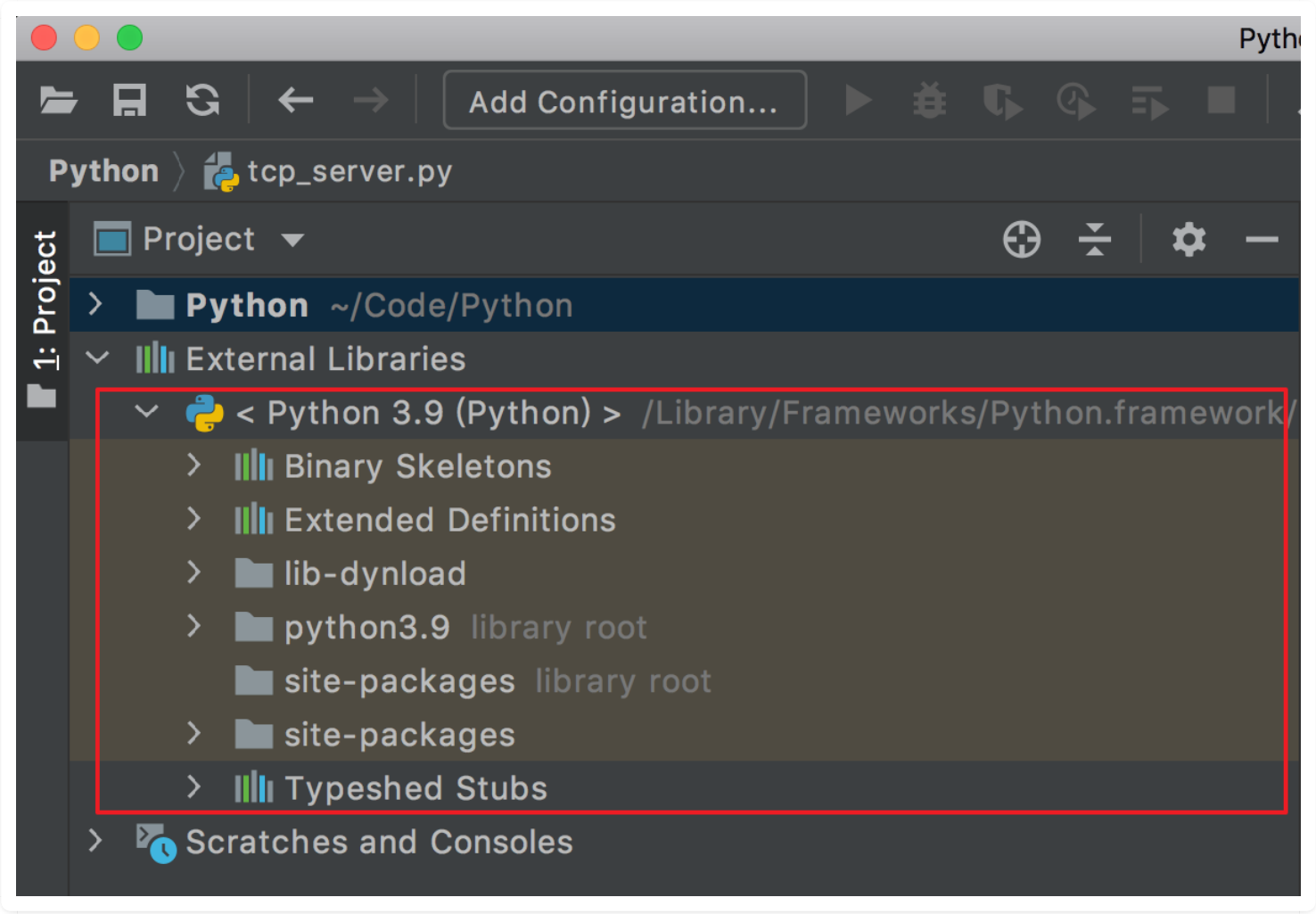
PyCharm 只是提供一个集成开发环境，你在执行 Python 程序时，还是得依赖 Python 解释器。

在一台电脑上，可以存在多个版本的 Python 解释器，所以你在执行 Python 程序前，你首先得告诉 PyCharm 你想用哪个 Python 解释器去执行程序。

打开设置，搜索 Interpreter （如下图），就可以添加你的 Python 解释器了。



设置完成后，在主界面就可以看到这里多了这么块内容。你以后想读一些内置模块的代码，可以直接从这里点进去。



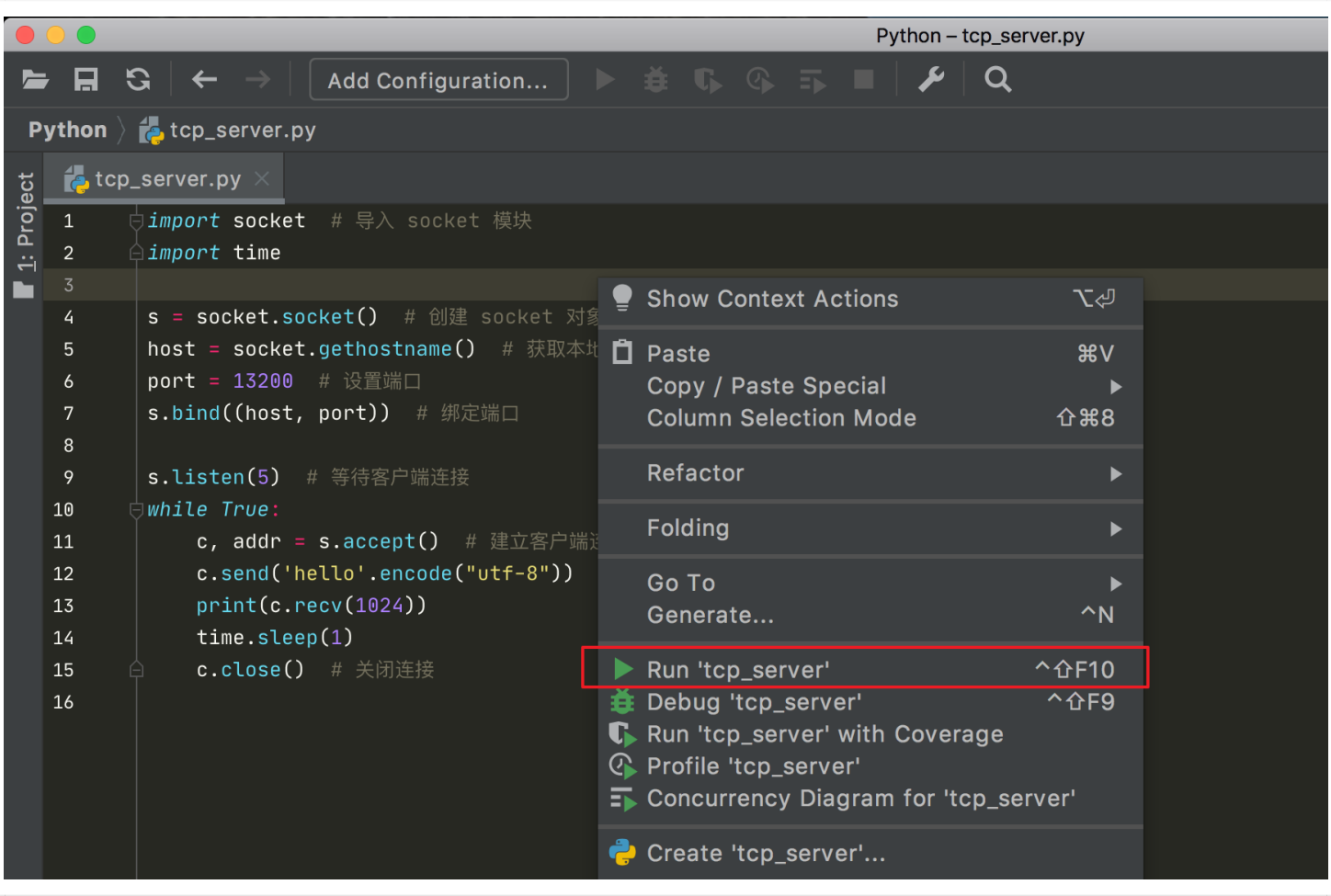
2. 运行 Python 程序

设置好解释器后，就可以直接运行Python 程序了。

方法有三种:

第一种

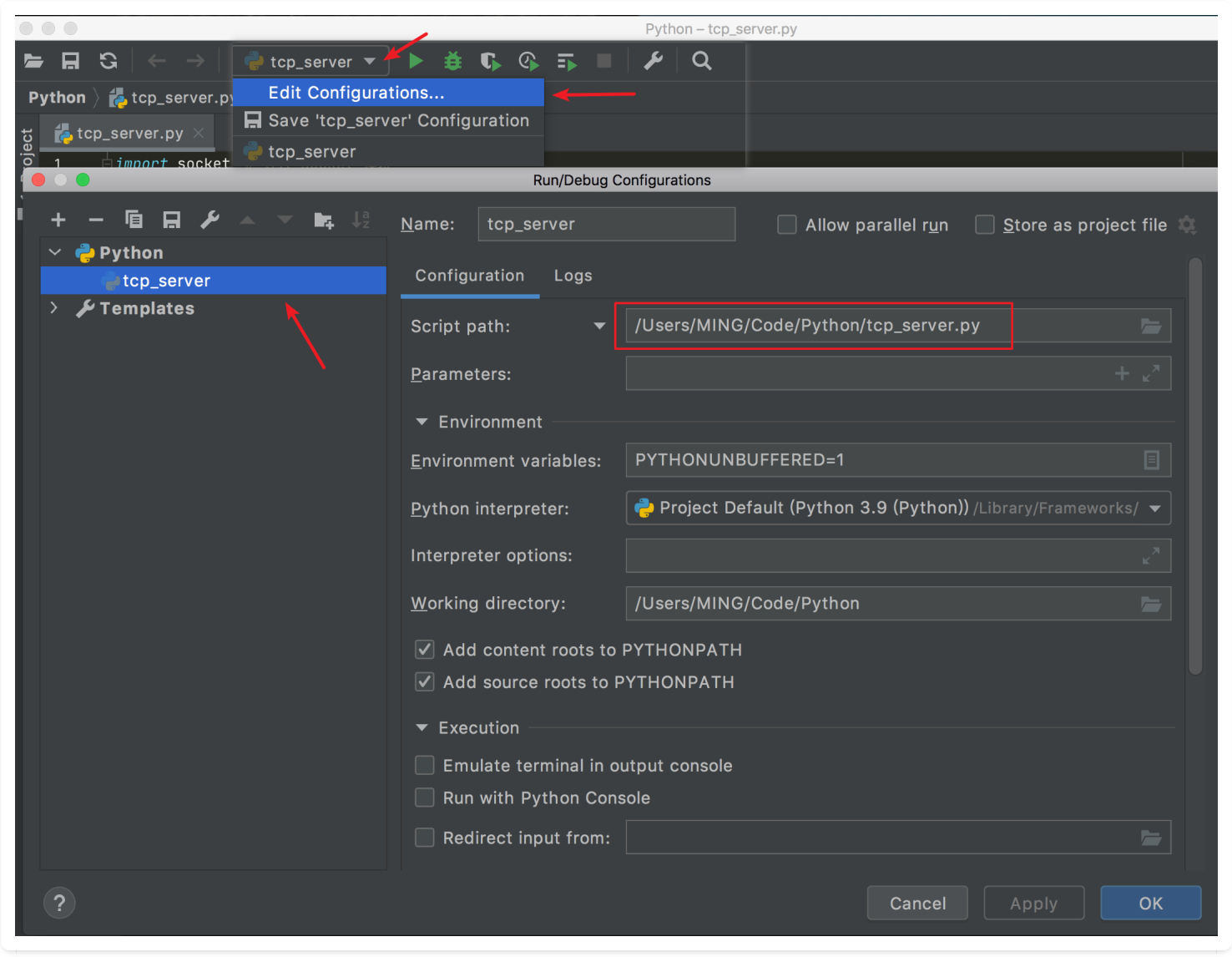
右键 - 点击 Run 就可以运行该程序



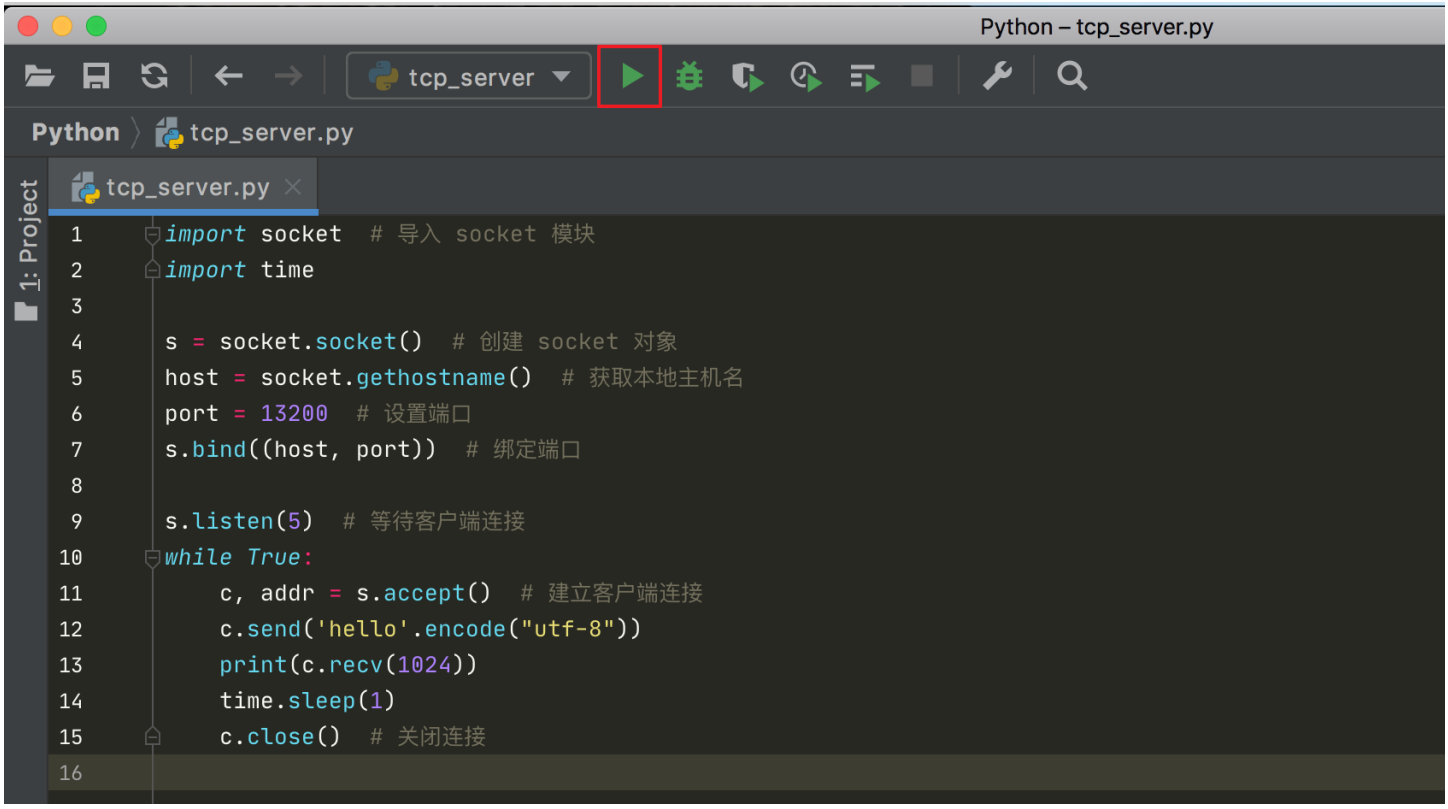
从右键可以看到 Run 是有快捷键的，你只要使用 ctrl+shift+F10 就可以运行该程序.

第二种

在你运行过一次该脚本后，PyCharm 就会自动为你记录一次运行记录



因此你以后直接点击这里，就可以直接该程序了。

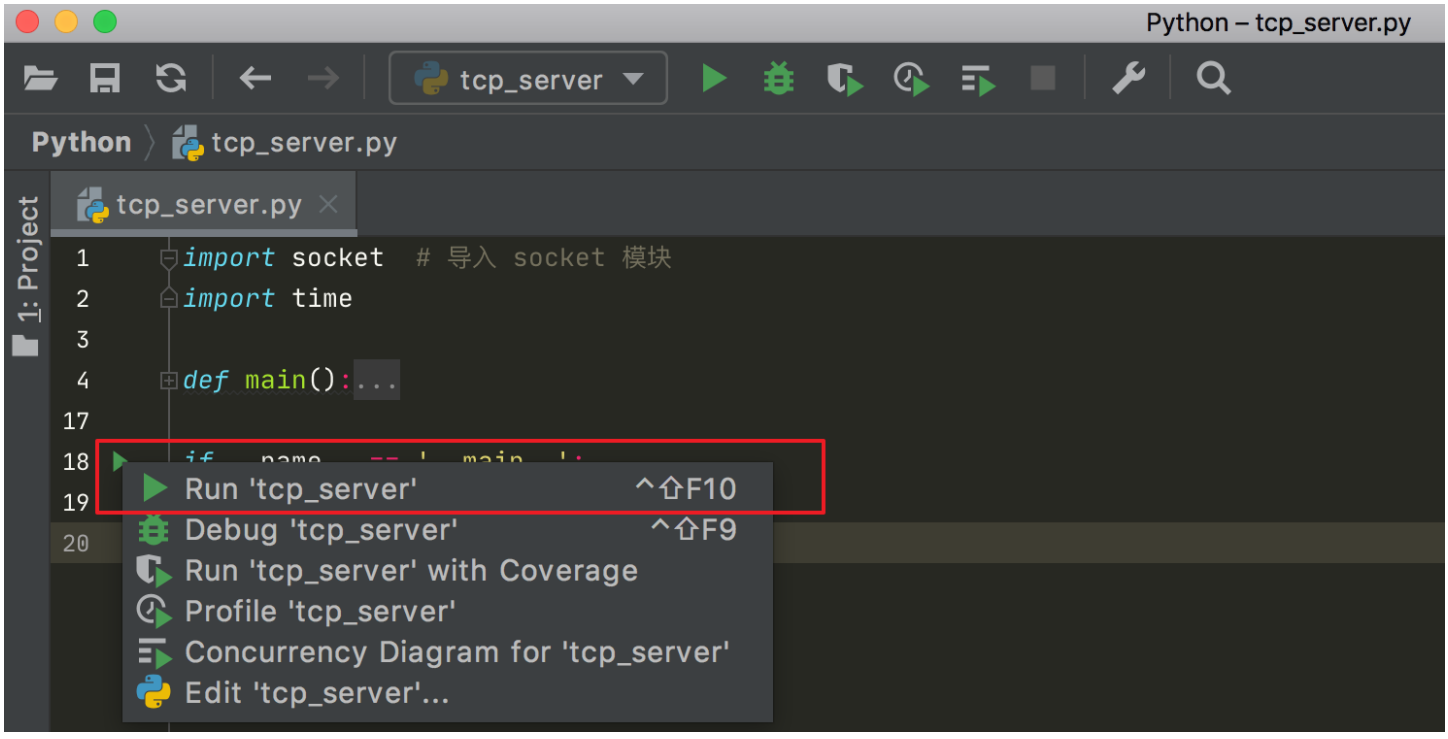


第三种

如果你在程序里有如下代码

```
if __name__ == '__main__':
    main()
```

就会出现如下的运行按钮，点击第一个就是 Run 。



第四种

这一种方法，可以让你在任意地方编写小段的测试代码，而不用新开一个文件。

具体方法我在8.8【绝佳工具 04】选择执行：Execute Selection in Console 有详细的介绍使用方法，可以点击前往。

3. 运行相关的快捷键

- ⇧ + F10：运行当前文件
- ⌥ + ⇧ + F10：弹出菜单，让你选择运行哪一个文件

2.2 【运行技巧 02】通过指定参数，执行程序

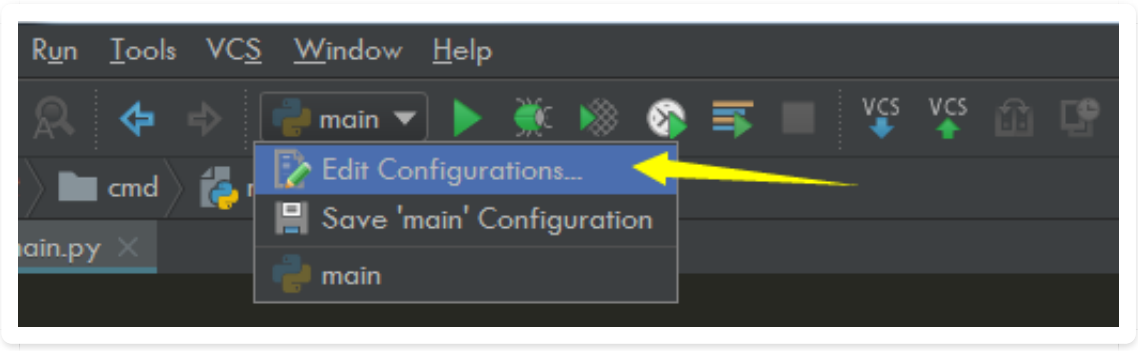
你在 Pycharm 运行你的项目，通常是怎么执行的？我的做法是，右键，然后点击 `Run`，或者使用快捷键 `Shift + F10` 。

有时候，在运行/调试脚本的时候，我们需要指定一些参数，这在命令行中，直接指定即可。

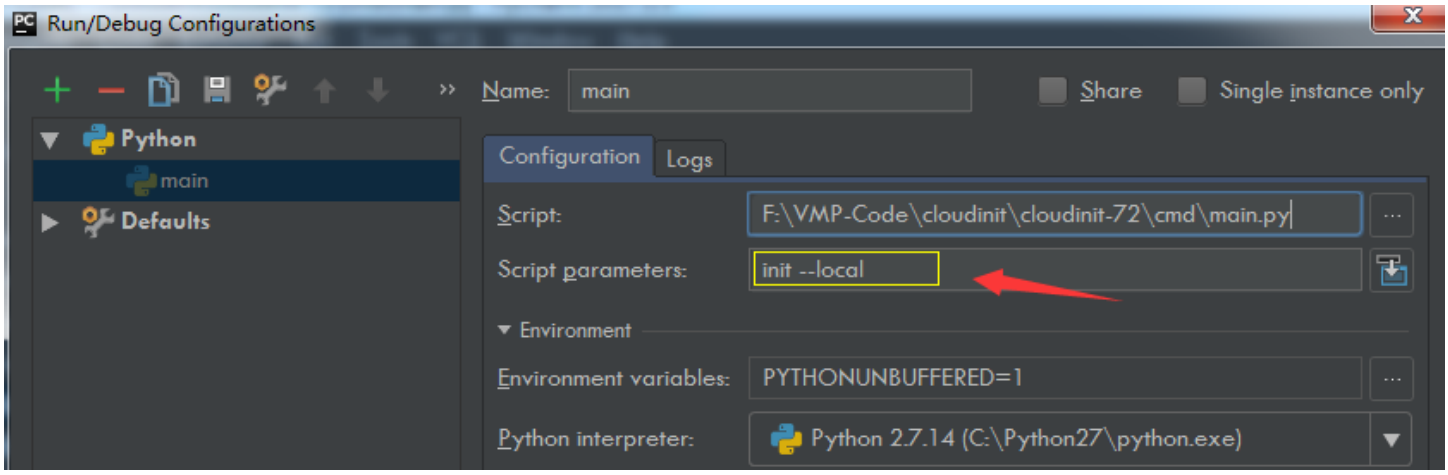
假设在命令行中，运行脚本的命令是这样

```
python main.py init --local
```

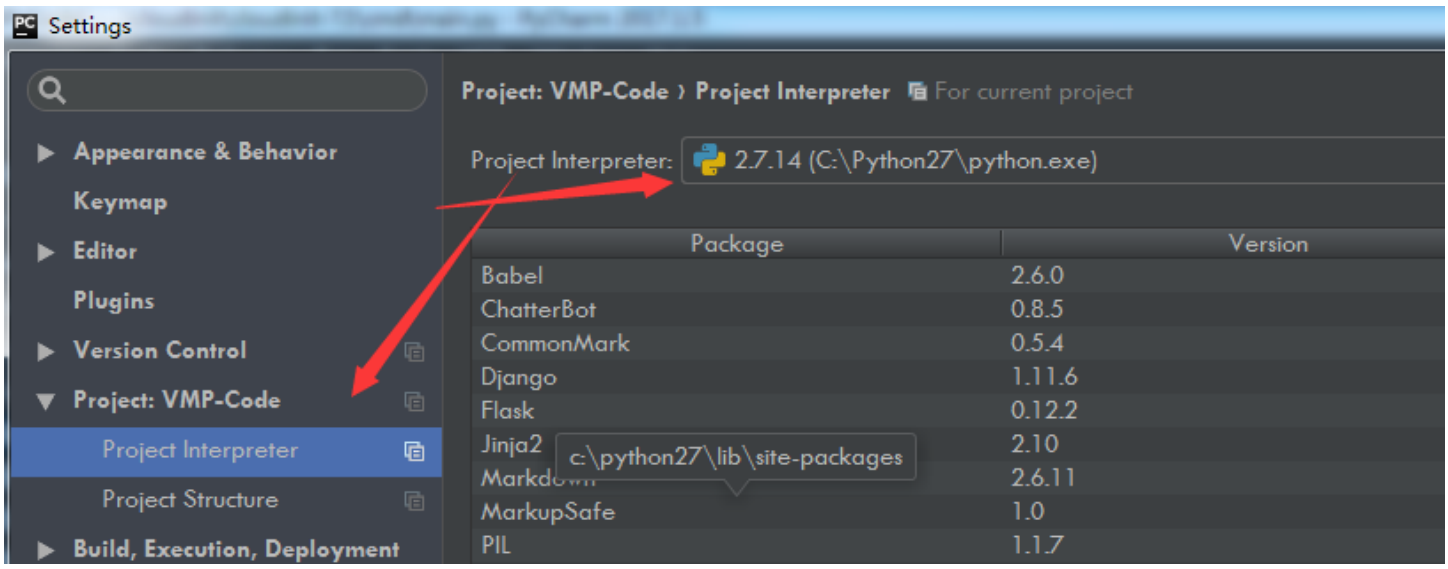
对于刚使用 Pycharm 的同学，可能并不知道 Pycharm 也是可以指定参数的。点击下图位置



进入设置面板，在 `Script parameters` 中填入参数即可。



同时在上图的底部，你可以看到，这里可以很方便的切换 解释器，比你跑到这边来要容易得多吧



2.3 【调试技巧 01】超详细图文教你调试代码

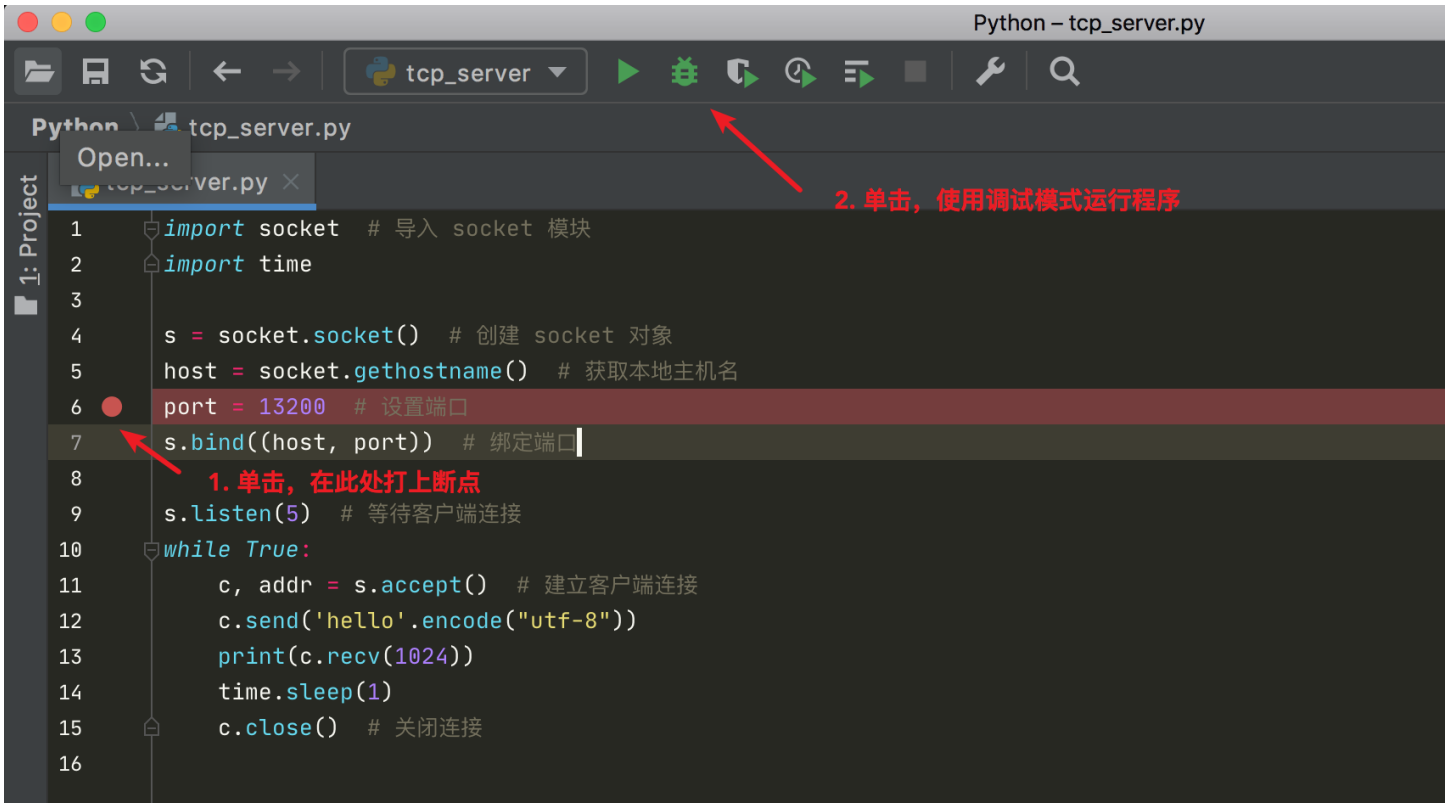
1. 调试的过程

调试可以说是每个开发人员都必备一项技能，在日常开发和排查 bug 都非常有用。

调试的过程分为三步：

1. 第一步：在你想要调试的地方，打上断点
2. 第二步：使用调试模式来运行这个 python 程序
3. 第三步：使用各种手段开始代码调试

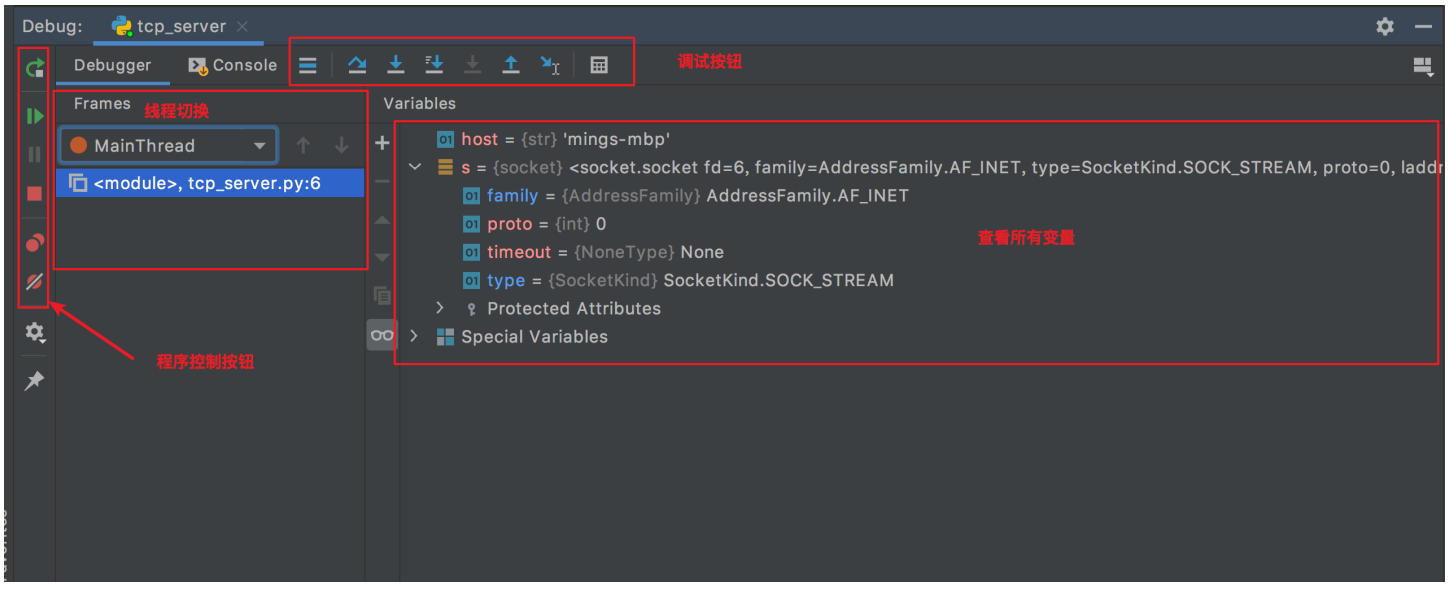
首先第一步和第二步，我用下面这张图表示



点击上图中的小蜘蛛，开启调试模式后，在 PyCharm 下方会弹出一个选项卡。

这个选项卡的按键非常多，包括

1. 变量查看窗口
2. 调试控制窗口
3. 线程控制窗口
4. 程序控制窗口



在变量查看窗口，你可以查看当前程序进行到该断点处，所有的普通变量和特殊变量，你每往下执行一行代码，这些变量都有可能跟着改变。

如果你的程序是多线程的，你可以通过线程控制窗口的下拉框来切换线程。

以上两个窗口，都相对比较简单，我一笔带过，下面主要重点讲下调试控制按钮和程序控制按钮。

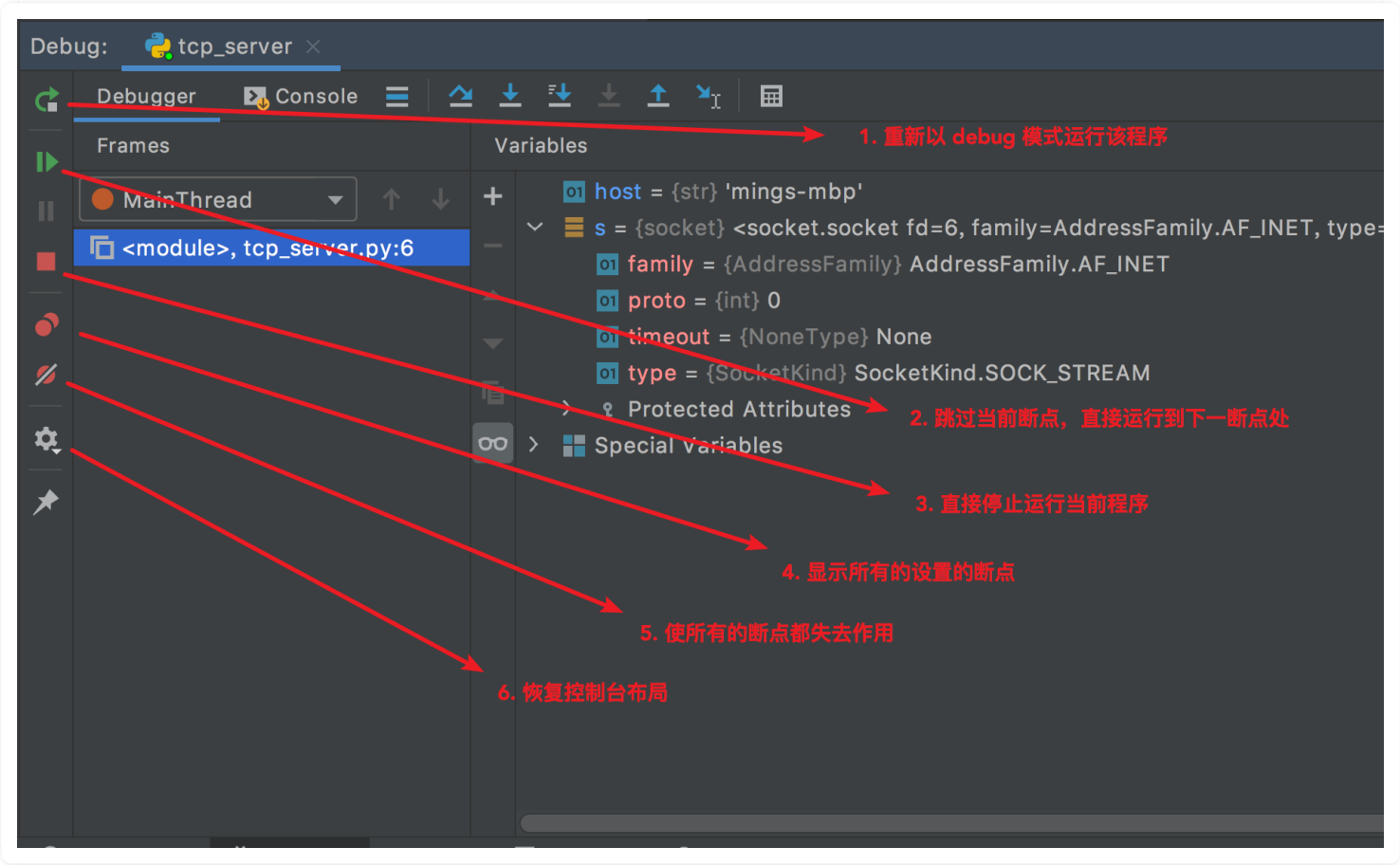
在调试控制窗口，共有 8 个按钮，他们的作用分别是什么呢？

1. Show Execution Point：无论你的代码编辑 窗口的光标在何处，只要点下该按钮，都会自动跳转到程序运行的地方。
2. Step Over：在单步执行时，在函数内遇到子函数时不会进入子函数内单步执行，而是将子函数整个执行完再停止，也就是把子函数整个作为一步。在不存在子函数的情况下是和step into效果一样的。简单的说就是，程序代码越过子函数，但子函数会执行，且不进入。
3. Step Into：在单步执行时，遇到子函数就进入并且继续单步执行，有的会跳到源代码里面去执行。
4. Step Into My Code：在单步执行时，遇到子函数就进入并且继续单步执行，不会进入到源码中。
5. Step Out：假如进入了一个函数体中，你看了两行代码，不想看了，跳出当前函数体内，返回到调用此函数的地方，即使用此功能即可。
6. Run To Cursor：运行到光标处，省得每次都要打一个断点。
7. Evaluate Expression：计算表达式，在里面可以自己执行一些代码。

以上七个功能，就是最常用的功能，一般操作步骤就是，**设置好断点，debug运行，然后 F8 单步调试，遇到想进入的函数 F7 进去，想出来在 shift + F8，跳过不想看的地方，直接设置下一个断点，然后 F9 过去。**

看这张图就行了（下面第6点有误，应该是运行到光标处，而不是下一断点处）

在程序控制窗口，共有 6 个按钮，他们的作用分别又是什么呢？同时看下面这张图就行了。



2. 调试相关的快捷键

- ⇧ + F9：调试当前文件
- ⌥ + ⇧ + F9：弹出菜单，让你选择调试哪一个文件
- F8：单步执行，不进入函数
- F7：单步执行，进入函数
- ⌥ + ⇧ +F7：单步执行，只进入自己写的函数
- ⇧ + F8：跳出函数体
- F9：运行到下一断点
- ⌥ + F9：运行到光标处
- ⇧ + ⌘ + F8：查看所有设置的断点
- ⌘ + F8：切换断点（有断点则取消断点，没有则加上断点）
- ⌥ + F5：重新以调试模式运行
- ⌥ + F8 计算表达式（可以更改变量值使其生效）

2.4 【调试技巧 02】程序结束了，照样可以调试

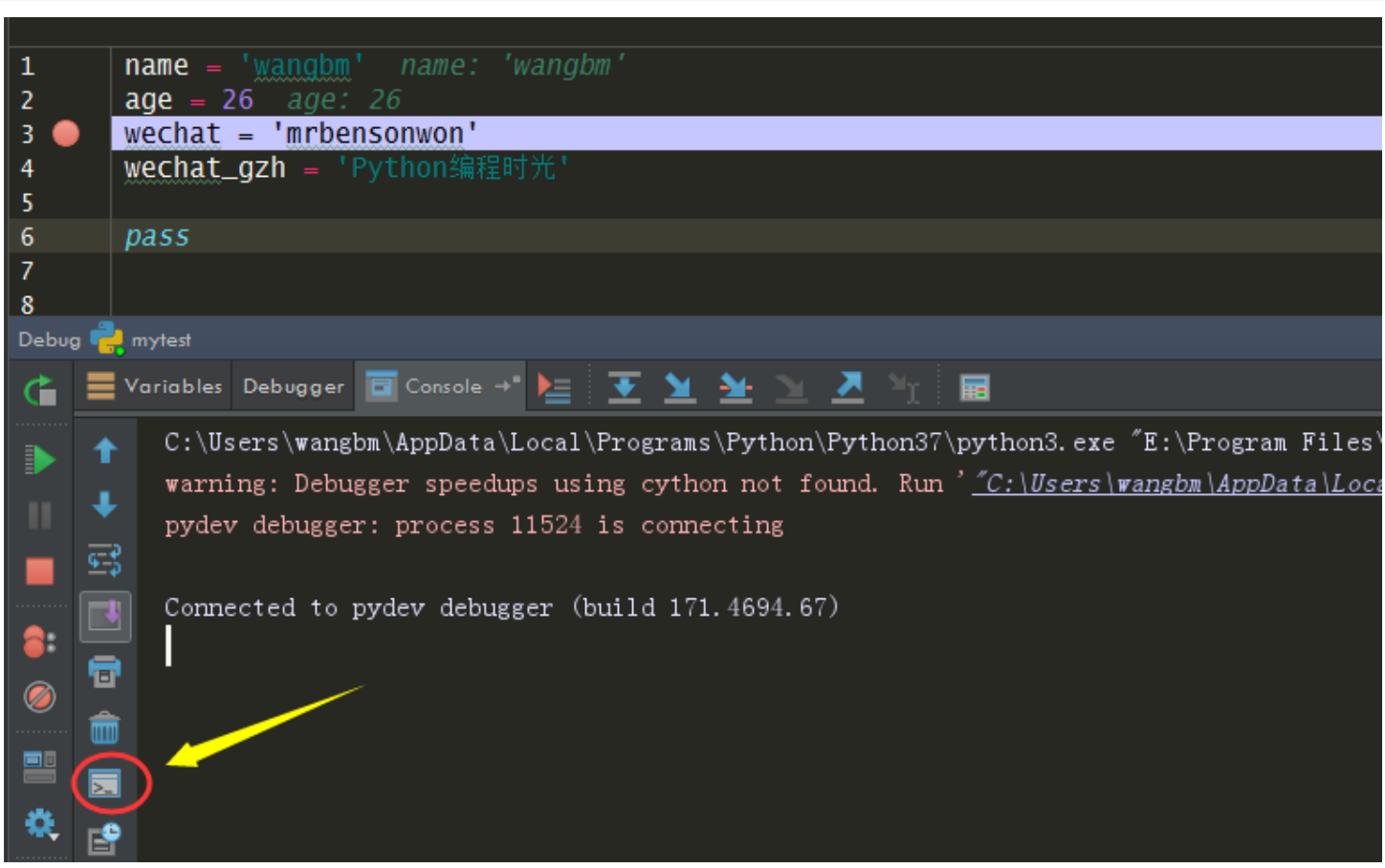
假如我们在一个爬虫的项目中，会使用到 正则表达式 来匹配我们想要抓取的内容。正则这种东西，有几个人能够一步到位的呢，通常都需要经过很多次的调试才能按预期匹配。在我们改了一次正则后，运行了下，需要重新向网站抓取请求，才能发现没有匹配上，然后又改了一版，再次运行同样需要发起请求，结果还是发现还是没有匹配上，往往复复，正则不好的同学可能要进行几十次的尝试。

（上面这个例子可能不太贴切，毕竟是有很多种方法实现不用重新发请求，只是列举了一种很笨拙且低效的调试过程，你看看就好了）

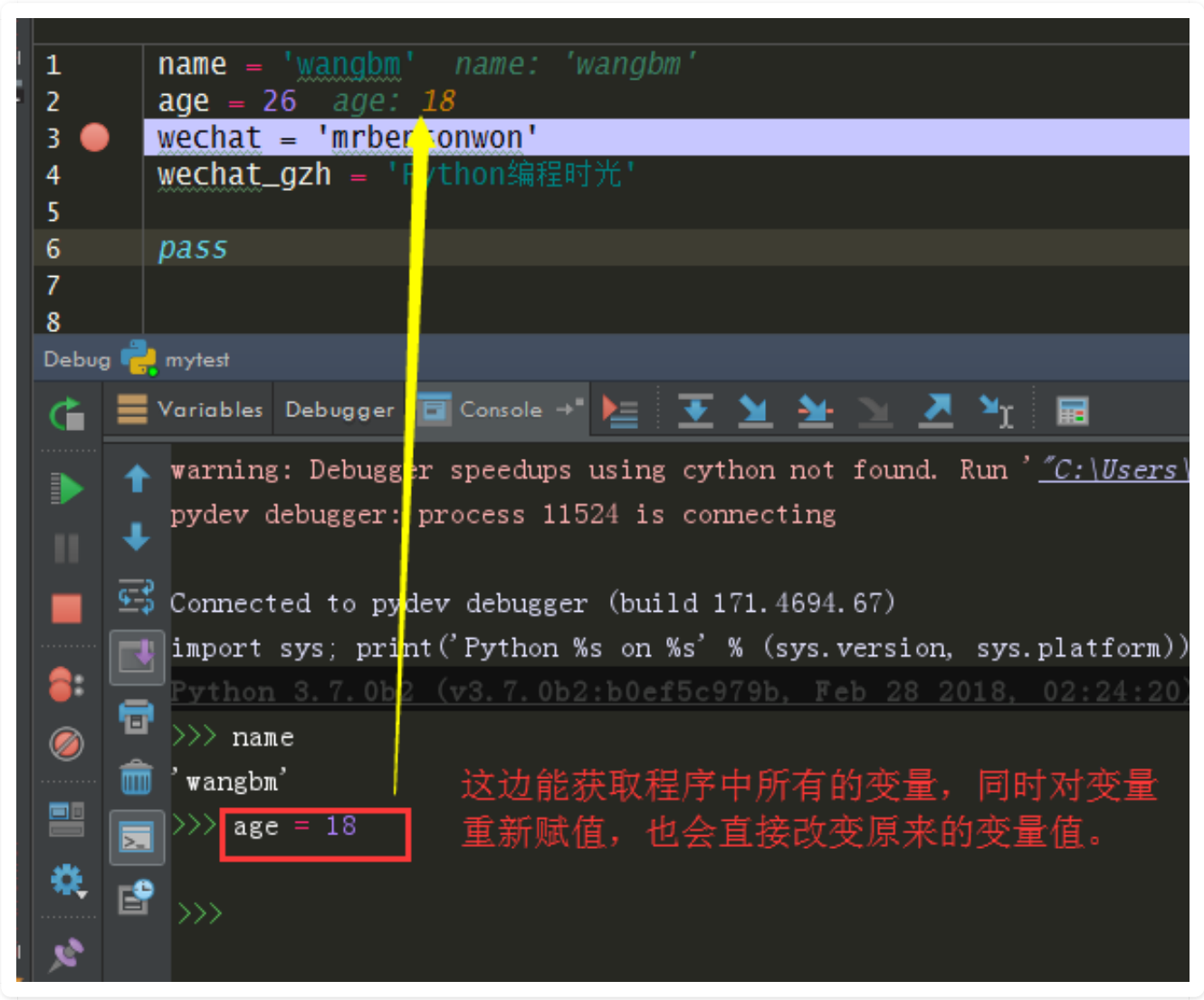
而我们在这几十次的调试中，向同一网站发起请求都是没有意义的重复工作。如果在 Pycharm 中可以像 IPython Shell 和 Jupyter Notebook 那样，可以记住运行后所有的变量信息，可以在不需要重新运行项目或脚本，就可以通过执行命令表达式，来调整我们的代码，进行我们的正则调试。

答案当然是有。

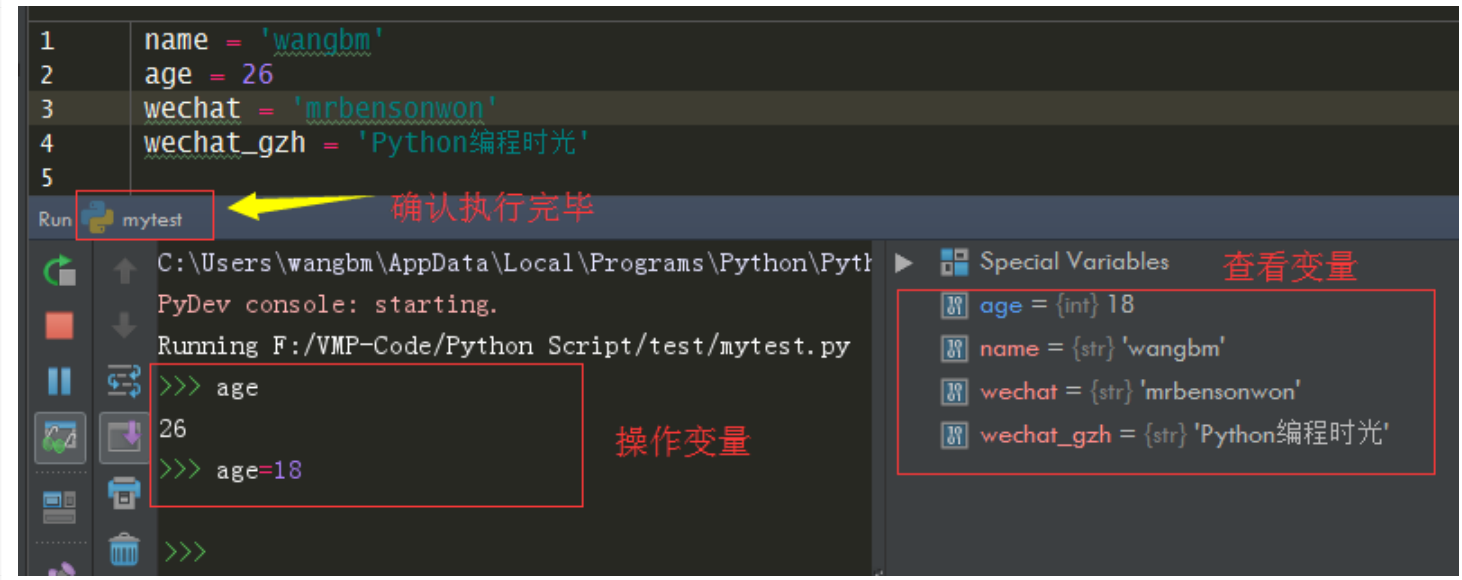
假如我在调试如下几行简单的代码。在第 3 行处打了个断点。然后点击图示位置 `Show Python Prompt` 按钮。



就进入了 `Python Shell` 的界面，这个Shell 环境和我们当前运行的程序环境是打通的，变量之间可以互相访问，这下你可以轻松地进行调试了。

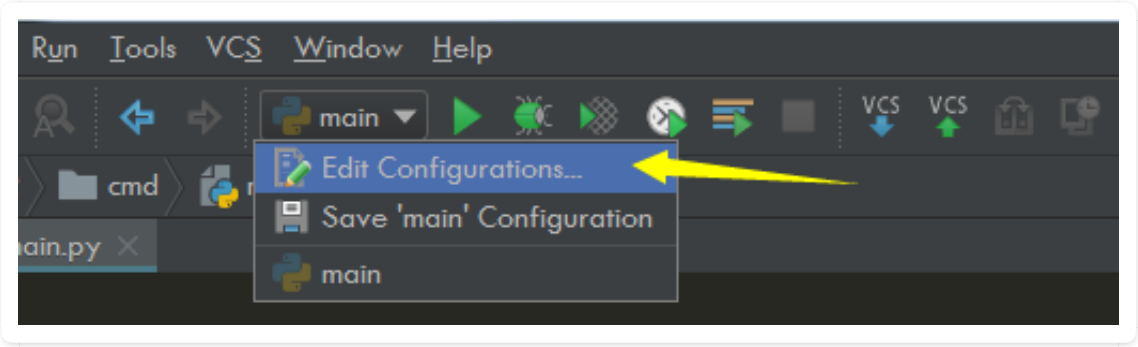


上面我们打了个断点，是为了方便说明这个效果。并不是说一定要打断点。如果不打断点，在脚本执行完成后，也仍然可以在这个界面查看并操作所有变量。

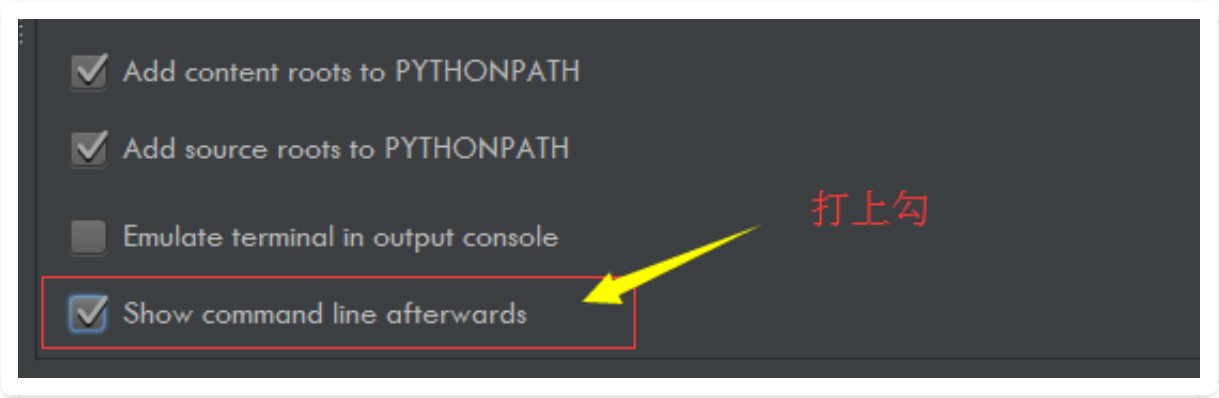


现在我们已经可以满足我们的调试的需求，但是每次运行脚本，都要手动点击 `Show Python Prompt`，有点麻烦。嗯？其实这个有地方可以设置默认打开的。这个开关还比较隐秘，一般人还真发现不了。

你需要点击图示位置 `Edit Configurations` 处。



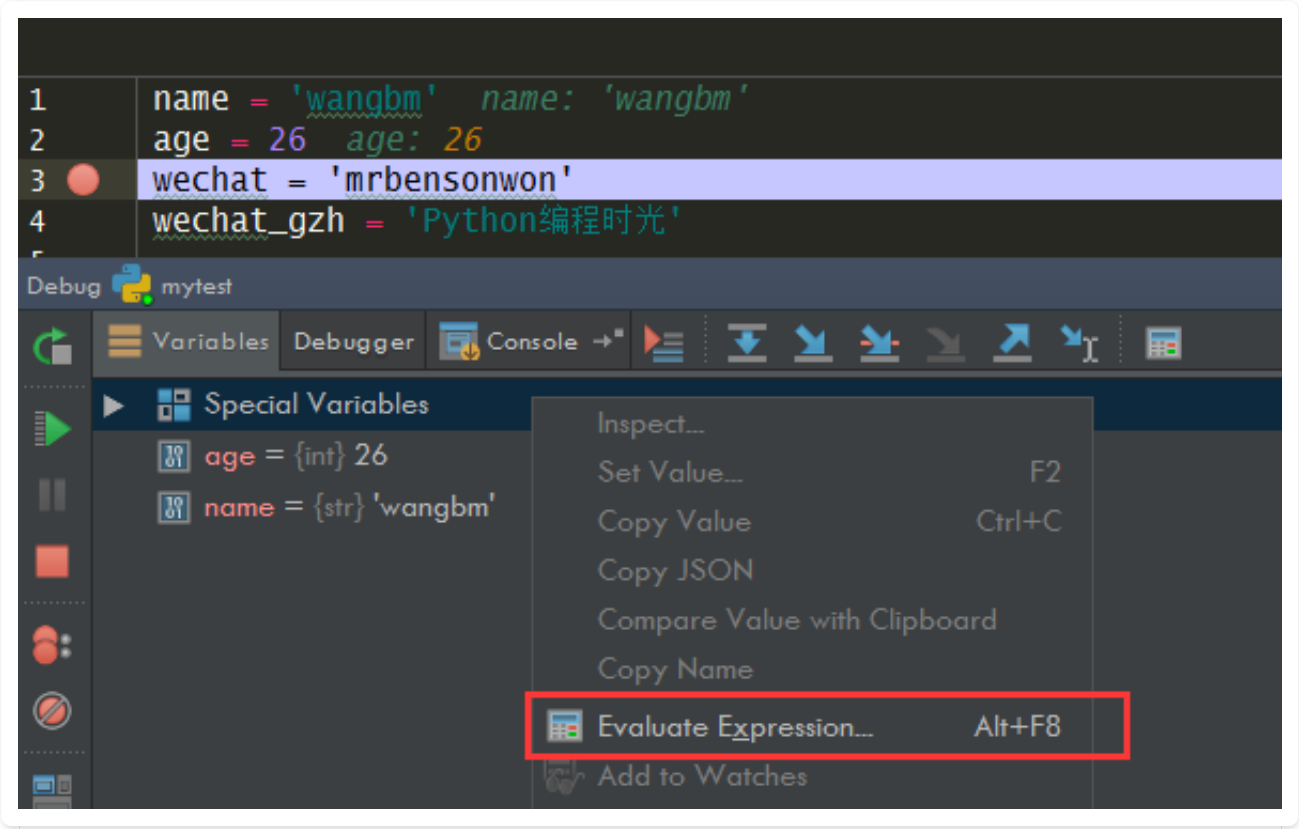
然后在这里打勾选中。



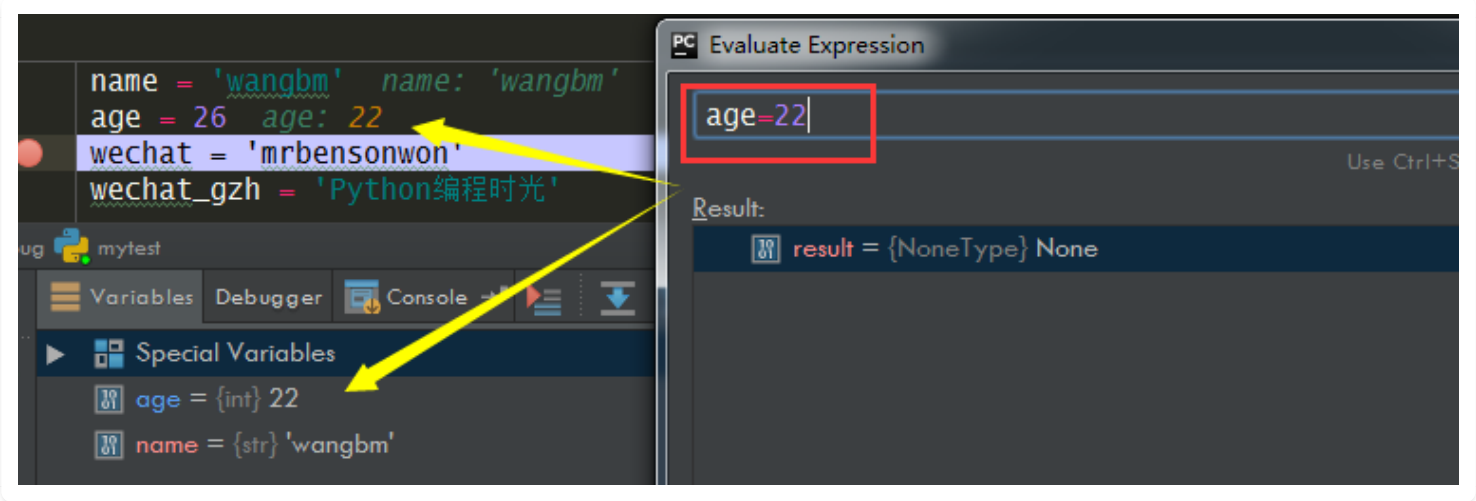
设置上之后，之后你每次运行后脚本后，都会默认为你存储所有变量的值，并为你打开 console 命令行调试界面。

除了上面这种方法，其实还有一种方法可以在调试过程中，执行命令表达式，而这种大家可能比较熟悉了，这边也提一下，就当是汇总一下。但是从功能上来说，是没有上面这种方法来得方便易用的。因为这种方法，必须要求你使用 debug 模式运行项目，并打断点。

使用方法就是，在你打了断点后，在图示位置处，点击右键使用 `Evaluate Expression`



就弹出了一个 `Evaluate Expression` 窗口，这里 可以运行命令表达式，直接操作变量。



2.5 【调试技巧 03】7 步实现远程代码调试

一般情况下，我们开发调试都是在个人PC上完成，遇到问题，开一下 `Pycharm` 的调试器，很快就能找到问题所在。

可有些时候，项目代码的运行会对运行环境有依赖，必须在部署了相关依赖组件的服务器上才可以运行，这就直接导致了我们不能在本地进行调试。

对于这种特殊的场景，就我所知，有如下两种解决方案：

- pdb

- 远程调试

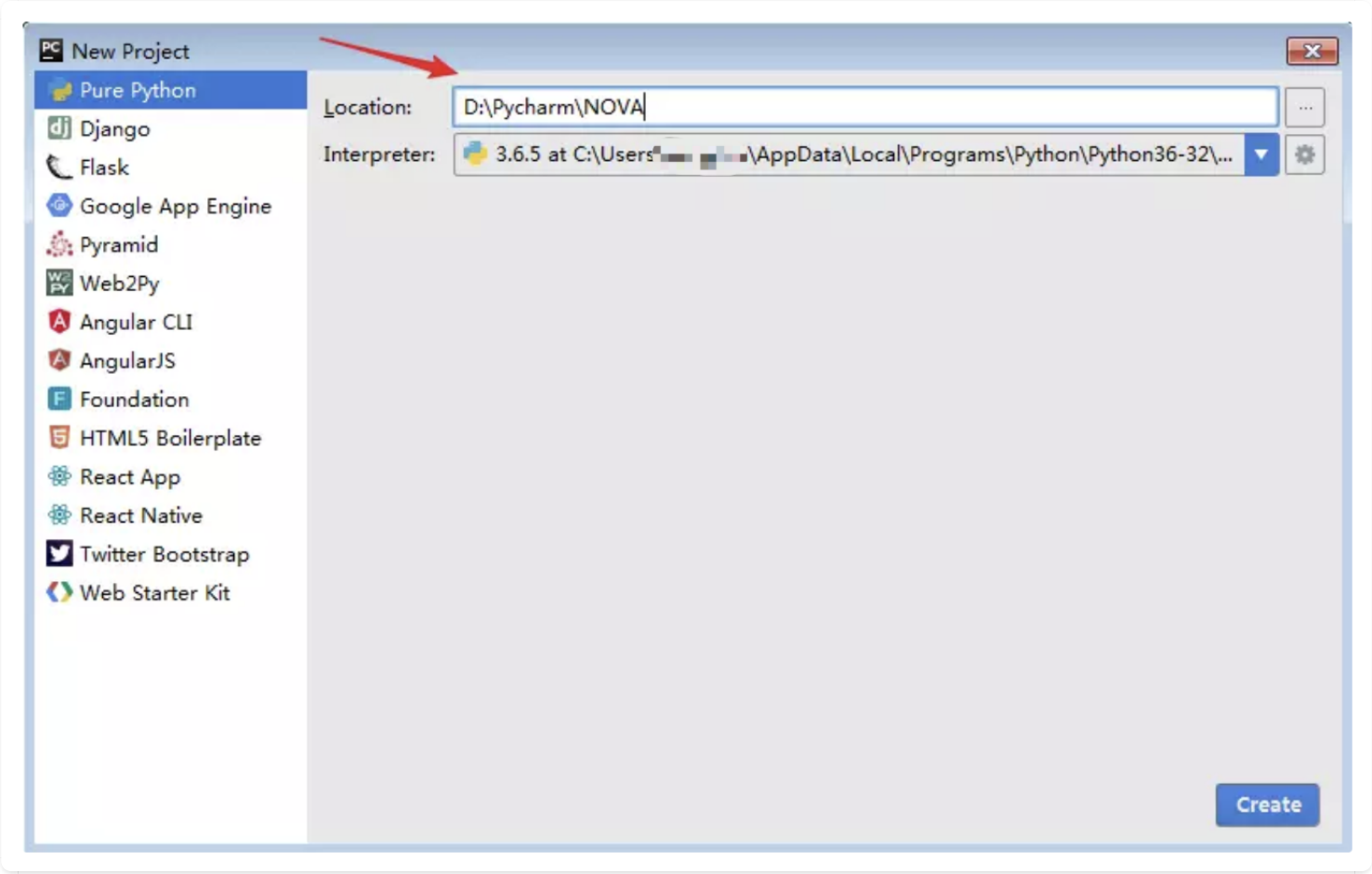
关于 pdb，之前也写过专门的文章介绍使用方法，你可以点此查看：[无图形界面的代码调试方法 - pdb]

而远程调试呢，是让我们可以在我们在 PC 上用 PyCharm 的图形化界面来进行调试远方服务器上代码，它和本地调试没有太大的区别，原来怎么调试的现在还是怎么调试。

区别就在于，本地调试不需要事前配置，只要你的代码准备好了，随时可以开始 Debug，而远程调试呢，需要不少前置步骤，这也正是本篇文章的内容，教你如何配置远程调试环境。

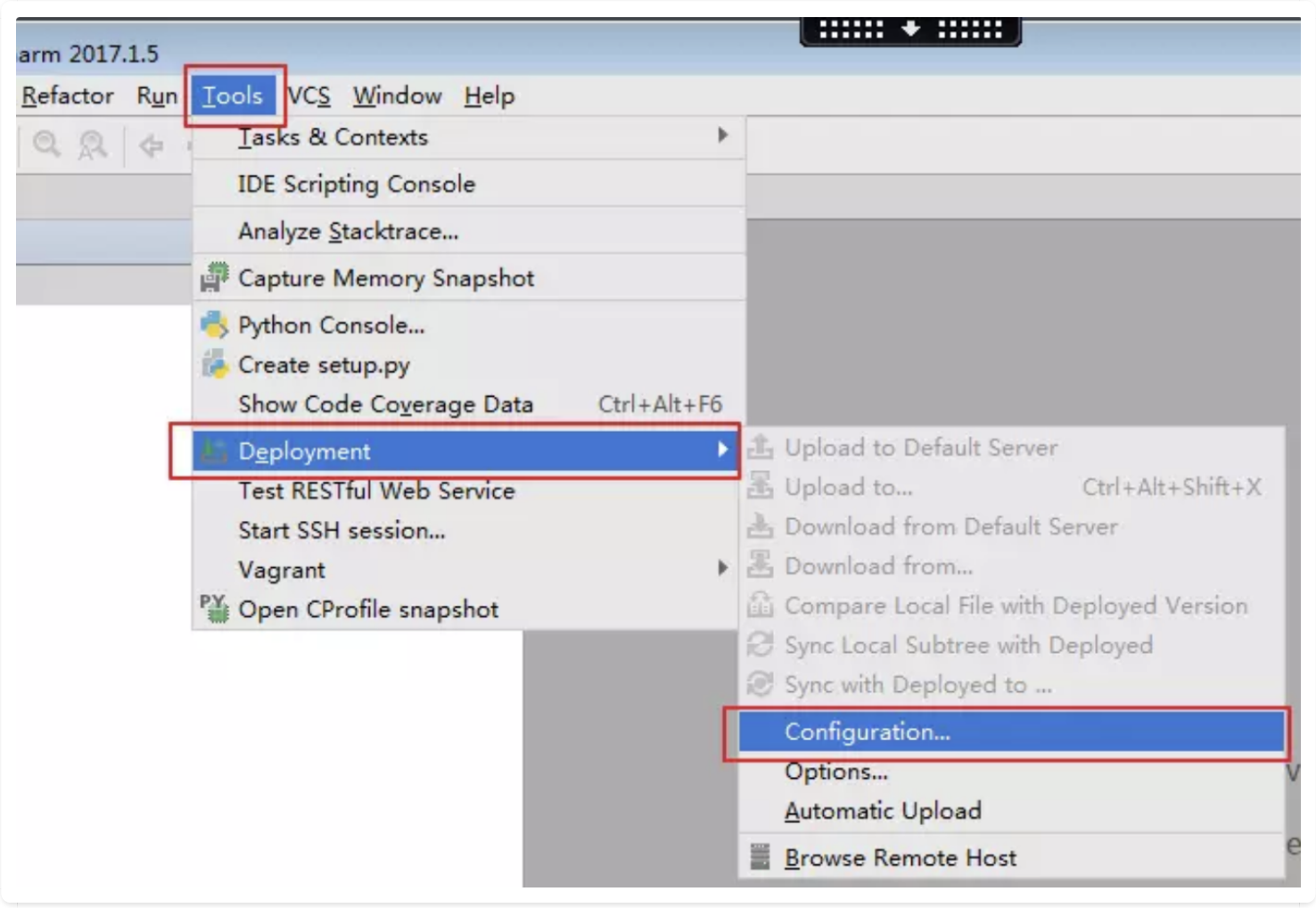
1. 新建一个项目

首先，要在Pycharm中新建一个空的项目，后面我们拉服务器上的项目代码就会放置在这个项目目录下。我这边的名字是 NOVA，你可以自己定义。



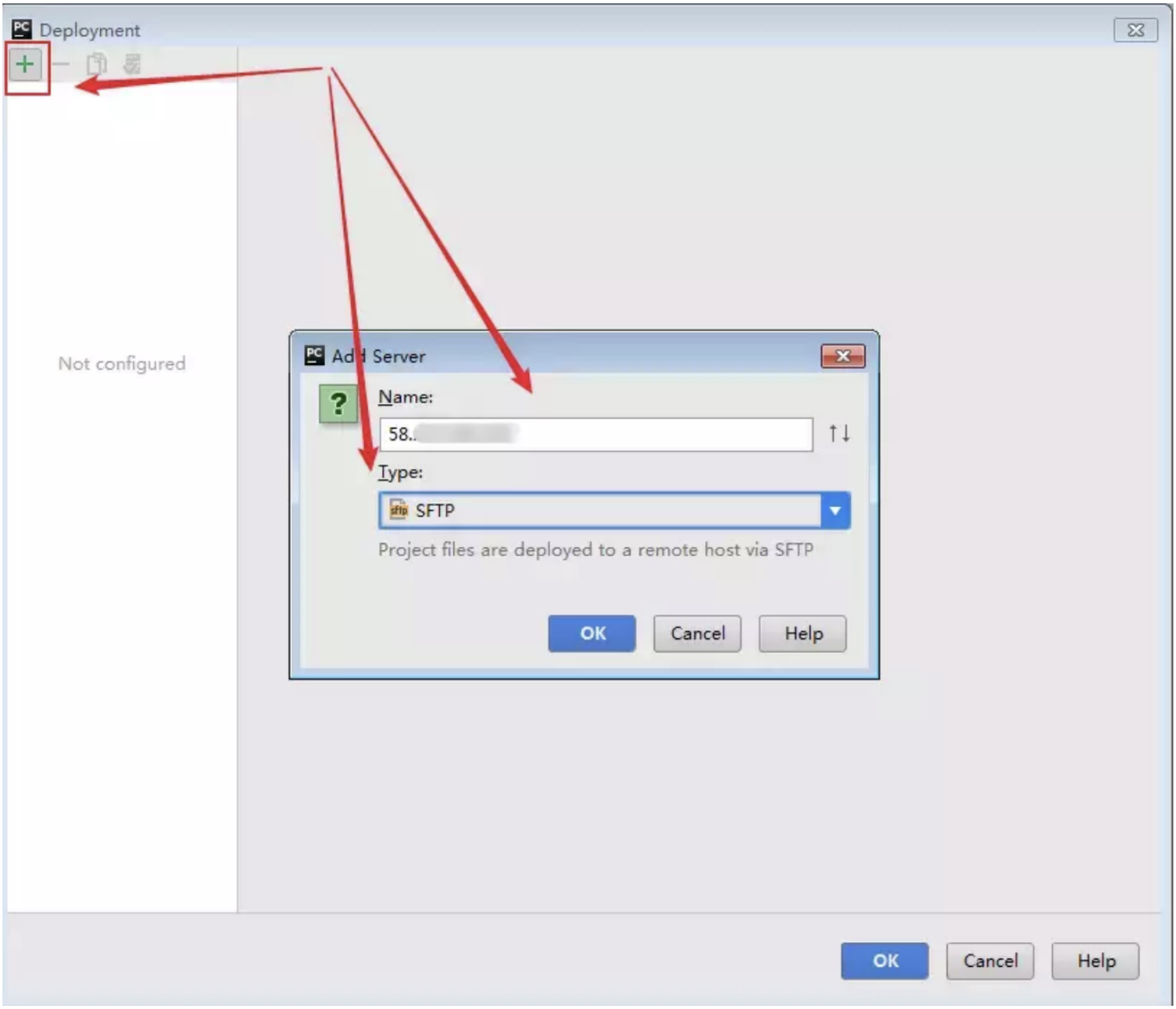
2. 配置连接服务器

Tools -> Deployment -> configuration



添加一个`Server`

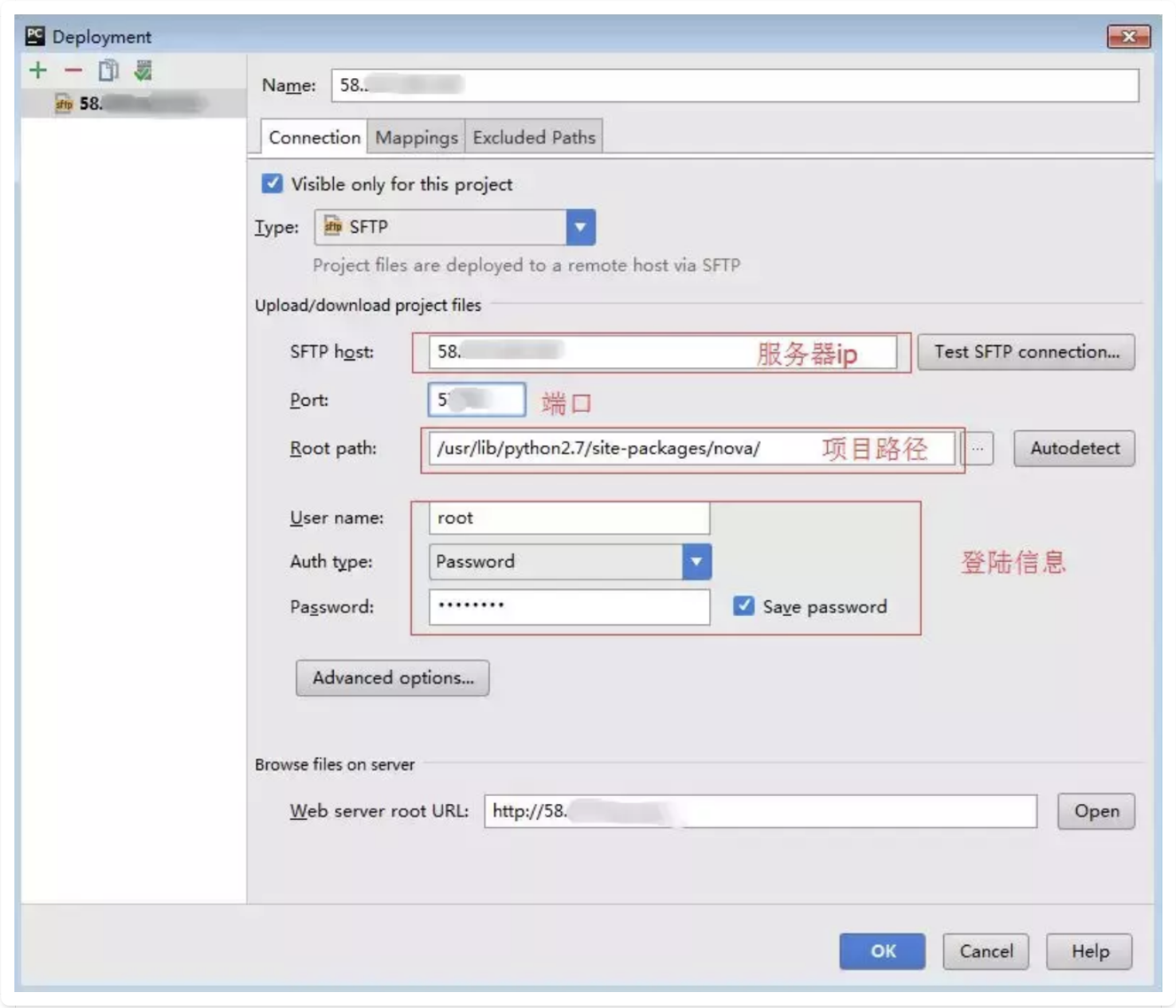
- Name：填你的服务器的IP
- Type：设定为SFTP



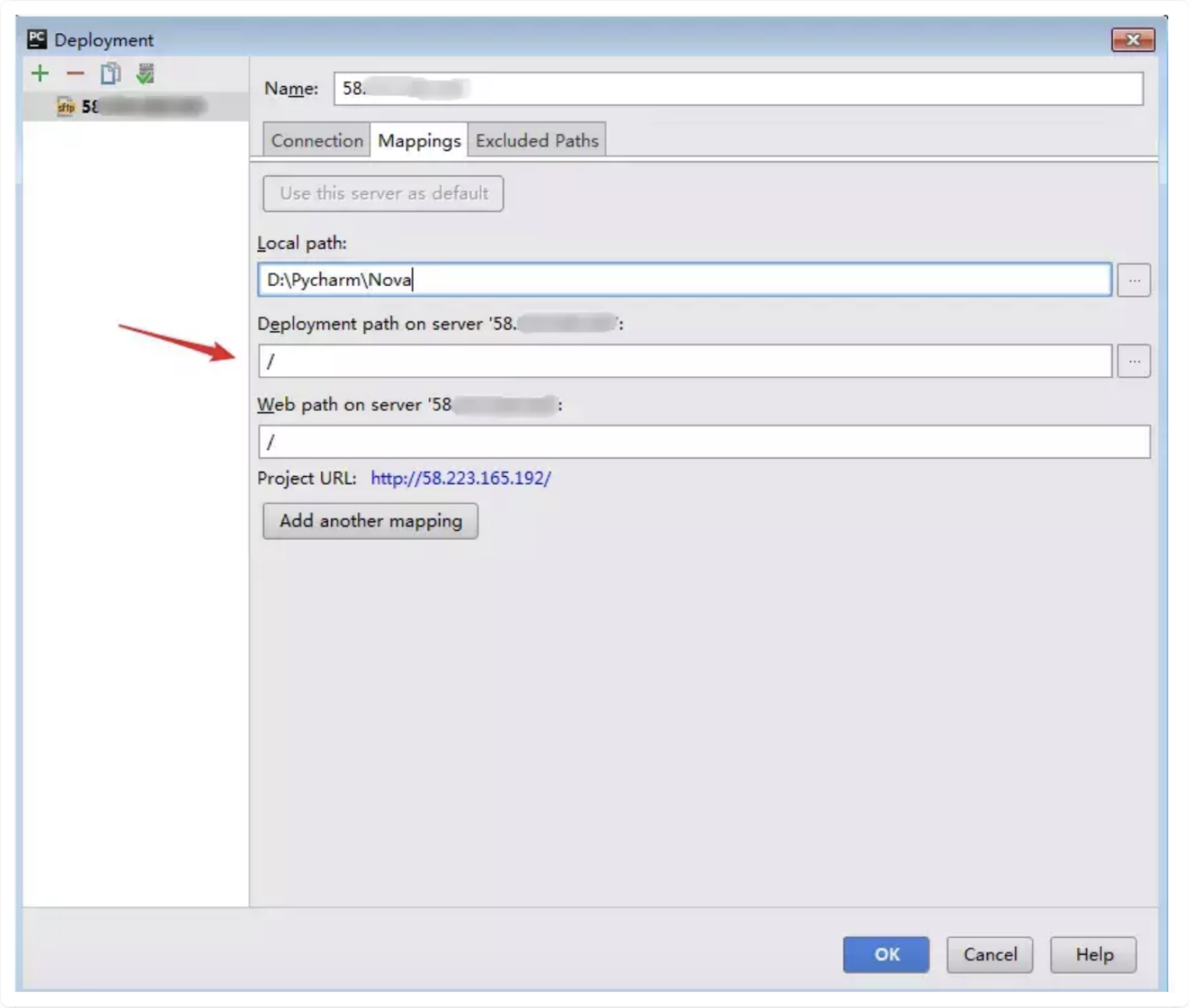
点击`OK`后，进入如下界面，你可以按我的备注，填写信息：

- SFTP host：公网ip
- Port：服务器开放的ssh端口
- Root path：你要调试的项目代码目录
- Username：你登陆服务器所用的用户
- Auth type：登陆类型，若用密码登陆的就是Password
- Password：选密码登陆后，这边输入你的登陆密码，可以选择保存密码。

这里请注意，要确保你的电脑可以ssh连接到你的服务器，不管是密钥登陆还是密码登陆，如果开启了白名单限制要先解除。

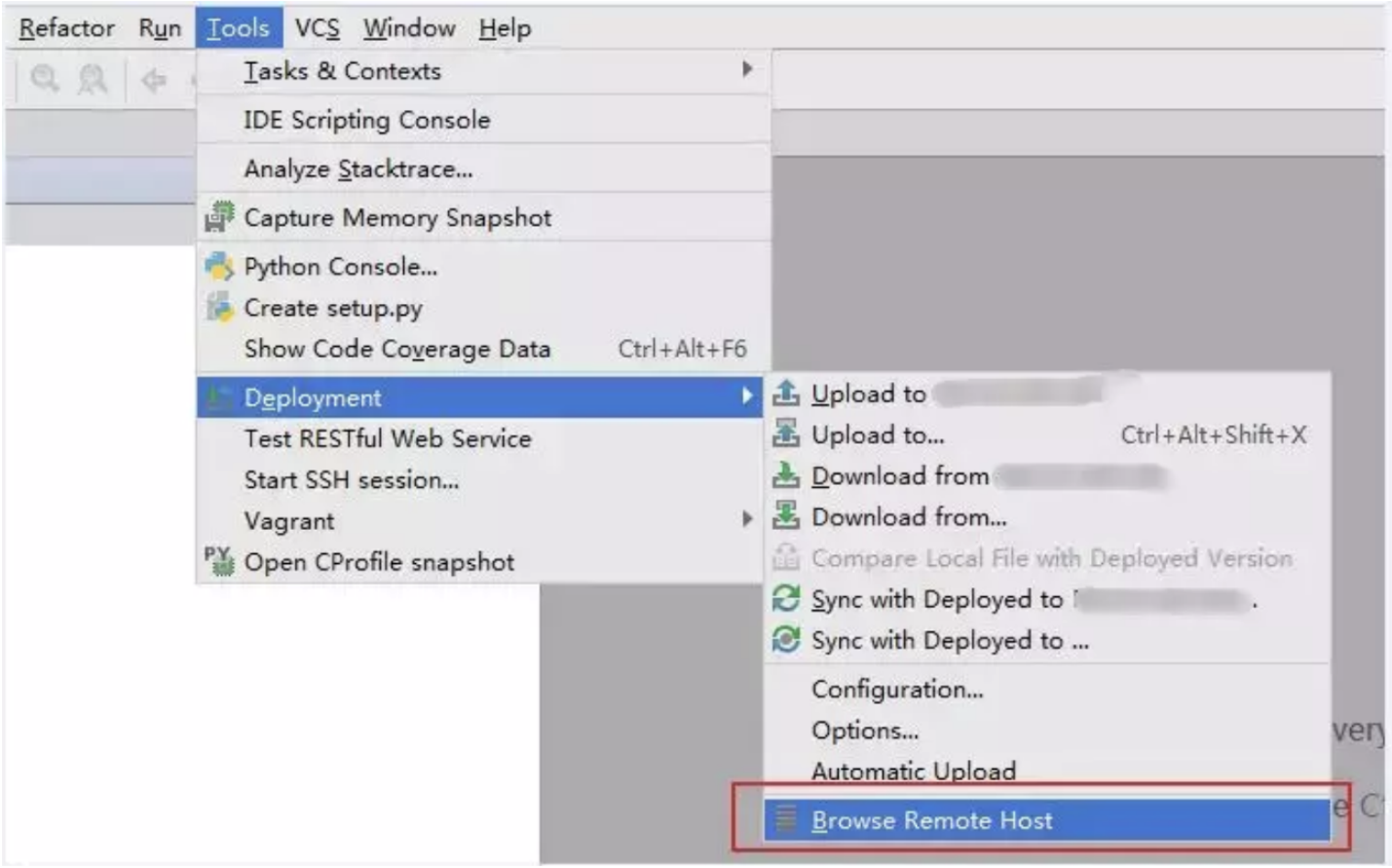


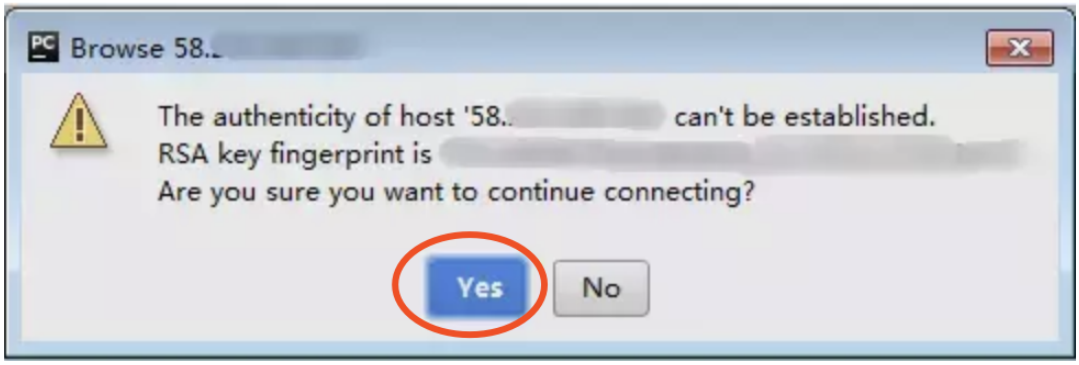
填写完成后，切换到`Mappings`选项卡，在箭头位置，填写`\`



以上服务器信息配置，全部正确填写完成后，点击`OK`

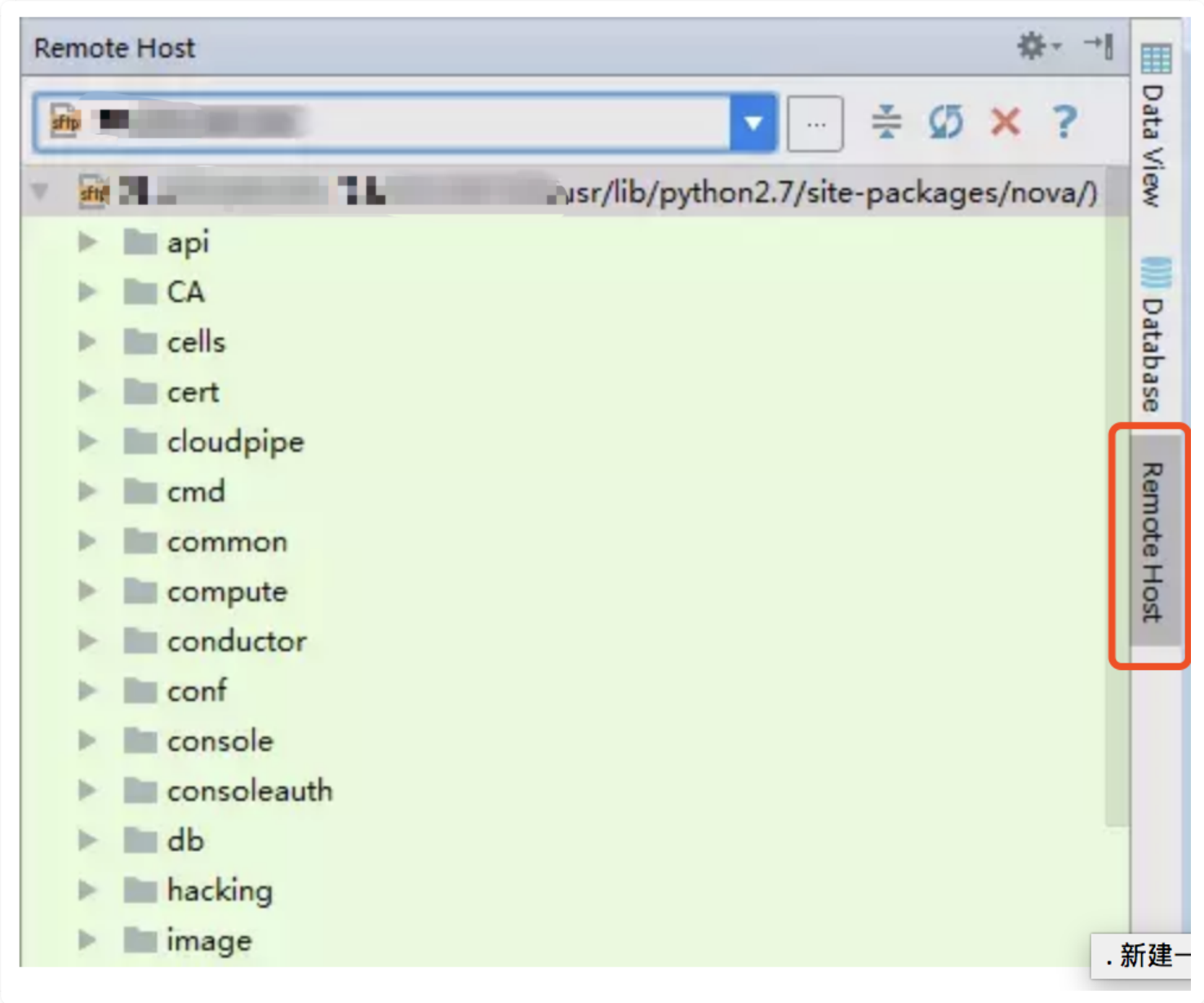
接下来，我们要连接远程服务器了。
Tools -> Deployment -> Browse Remote Host



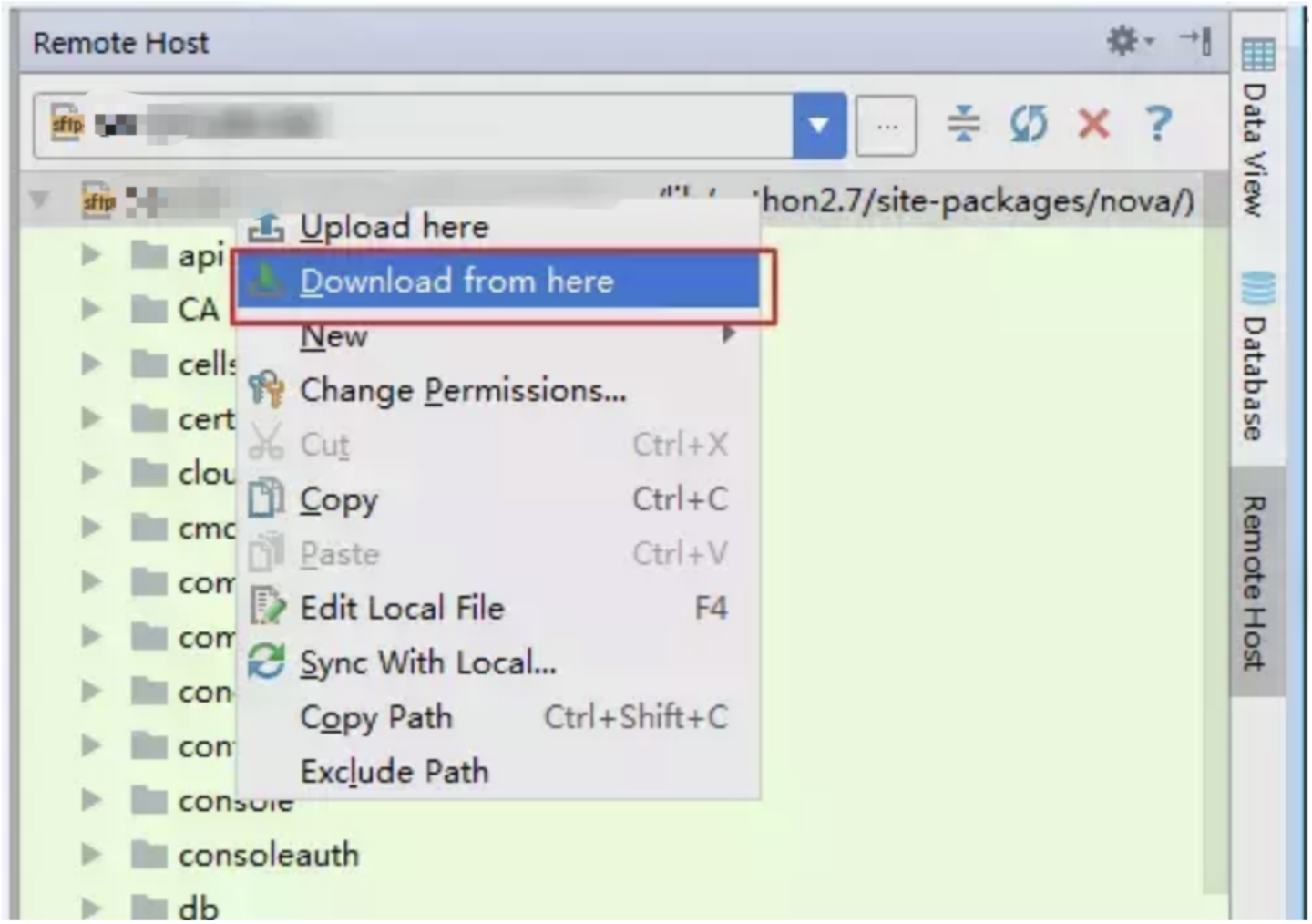


3. 下载项目代码

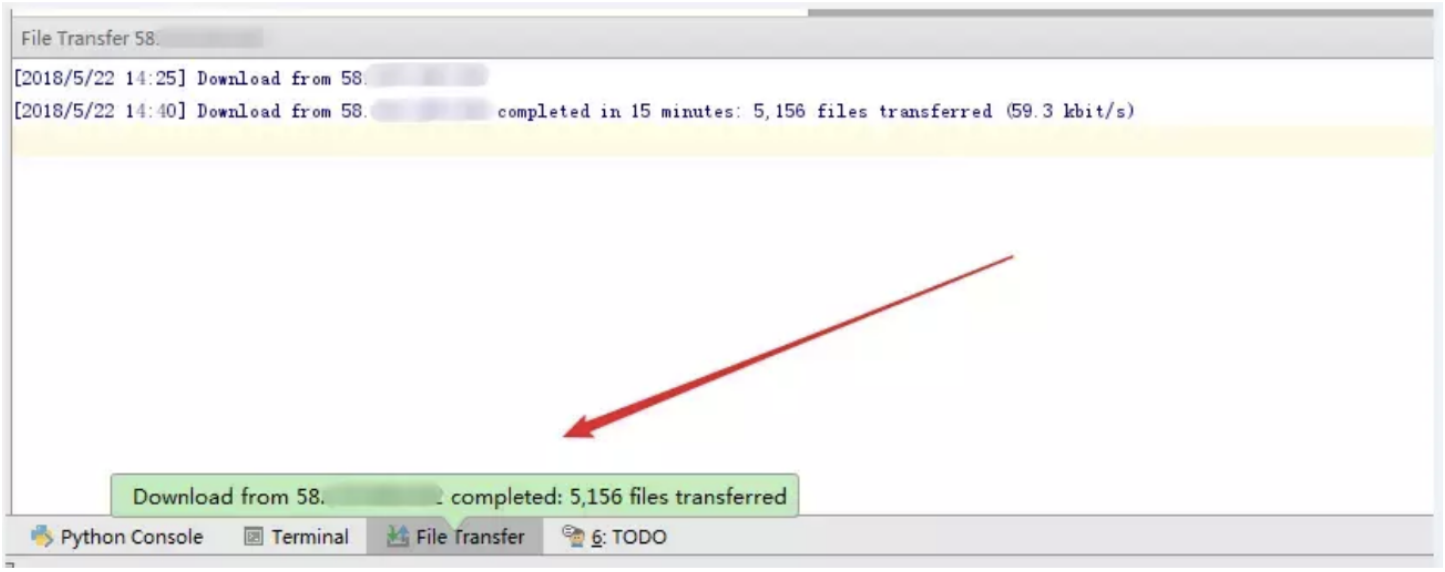
如果之前填写的服务器登陆信息准确无误的话，现在就可以看到远程的项目代码。



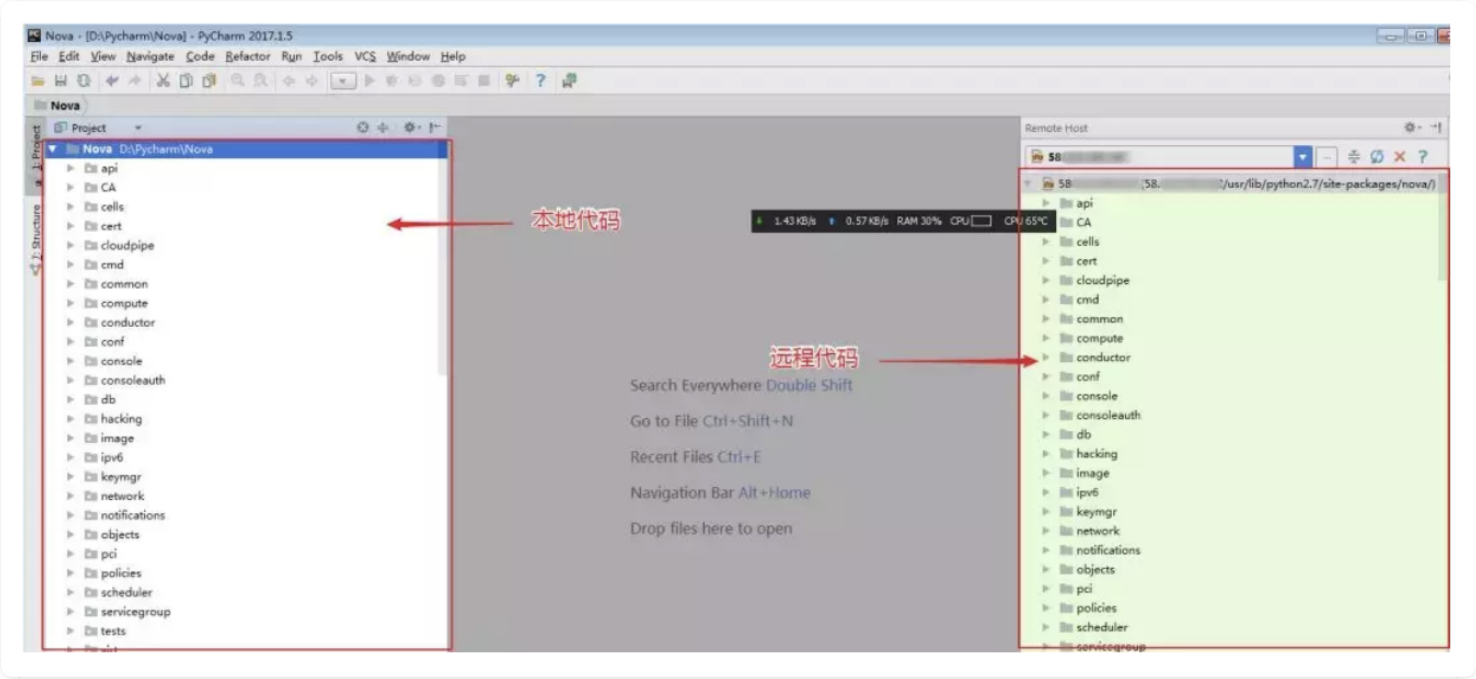
选择下载远程代码要本地。



下载完成提示。



现在的IDE界面应该是这样子的。

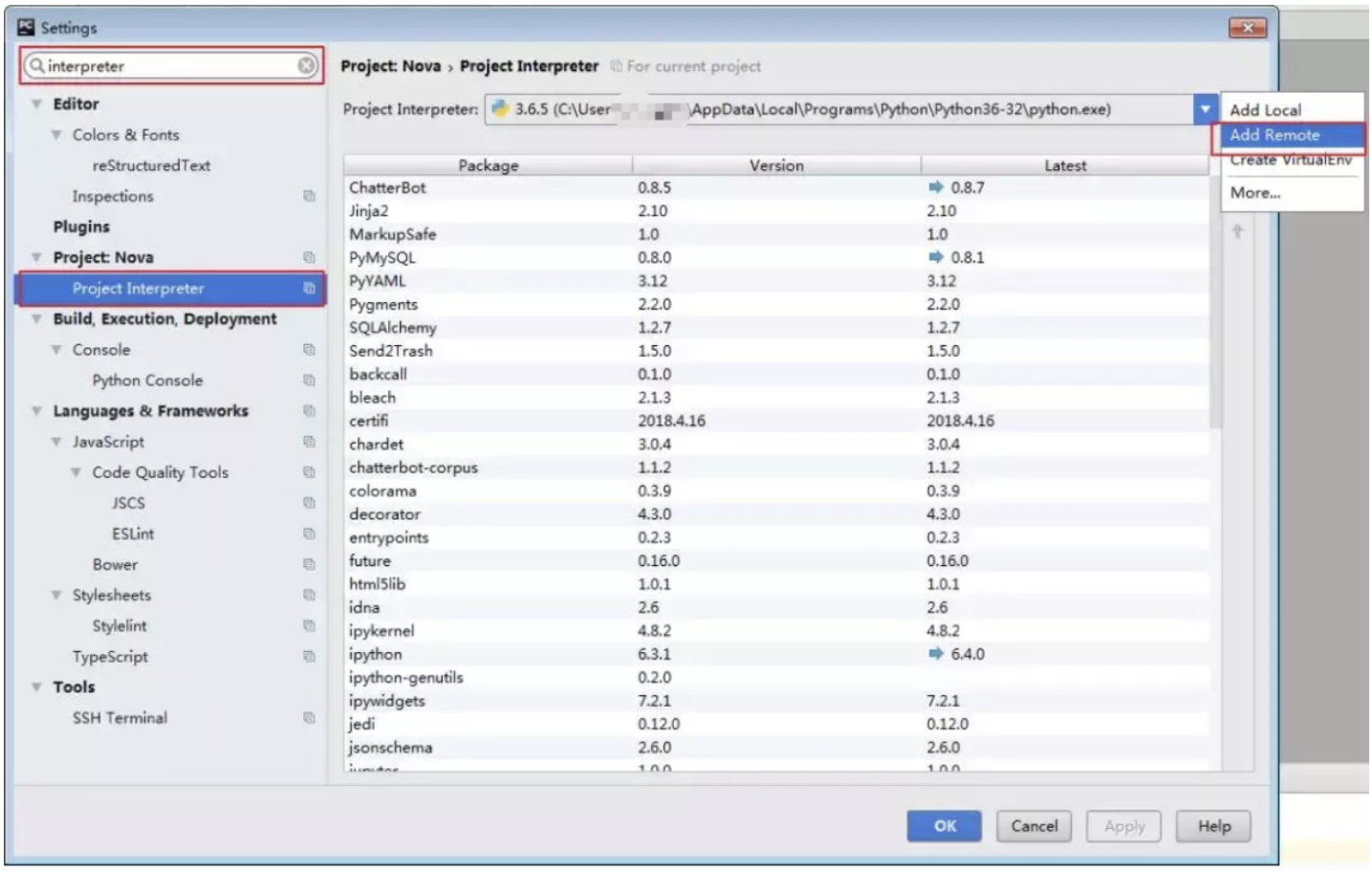


4. 下载远程解释器

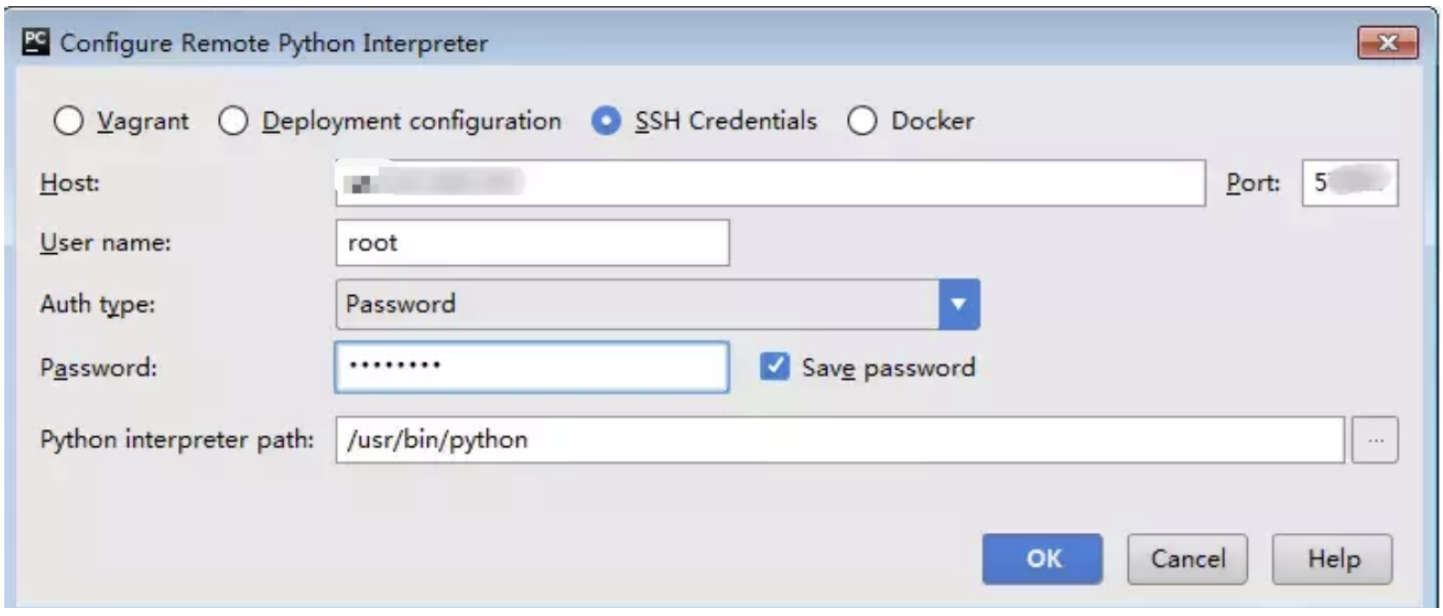
为什么需要这步呢?

远程调试是在远端的服务器上运行的，它除了依赖其他组件之外，还会有一些很多Python依赖包我们本地并没有。

进入 File -> Settings
按图示，添加远程解释器。



填写远程服务器信息，跟之前的一样，不再赘述。



点击`OK`后，会自动下载远程解释器。如果你的项目比较大，这个时间可能会比较久，请耐心等待。

5. 添加程序入口

因为我们要在本地DEBUG，所以你一定要知道你的项目的入口程序。如果这个入口程序已经包含在

你的项目代码中，那么请略过这一步。

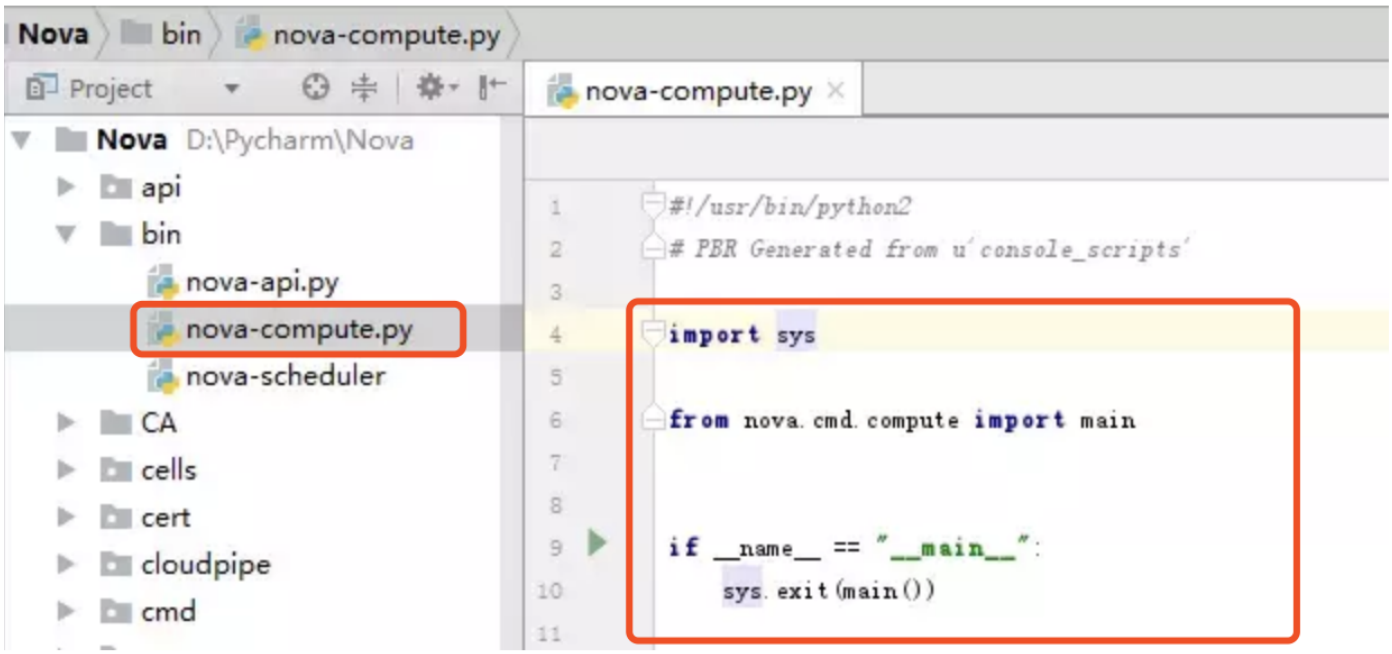
如果没有，就请自己生成入口程序。

比如，我这边的项目，在服务器上是以一个服务运行的。而我们都知道服务的入口是`Service文件`。

`cat /usr/lib/systemd/system/openstack-nova-compute.service`

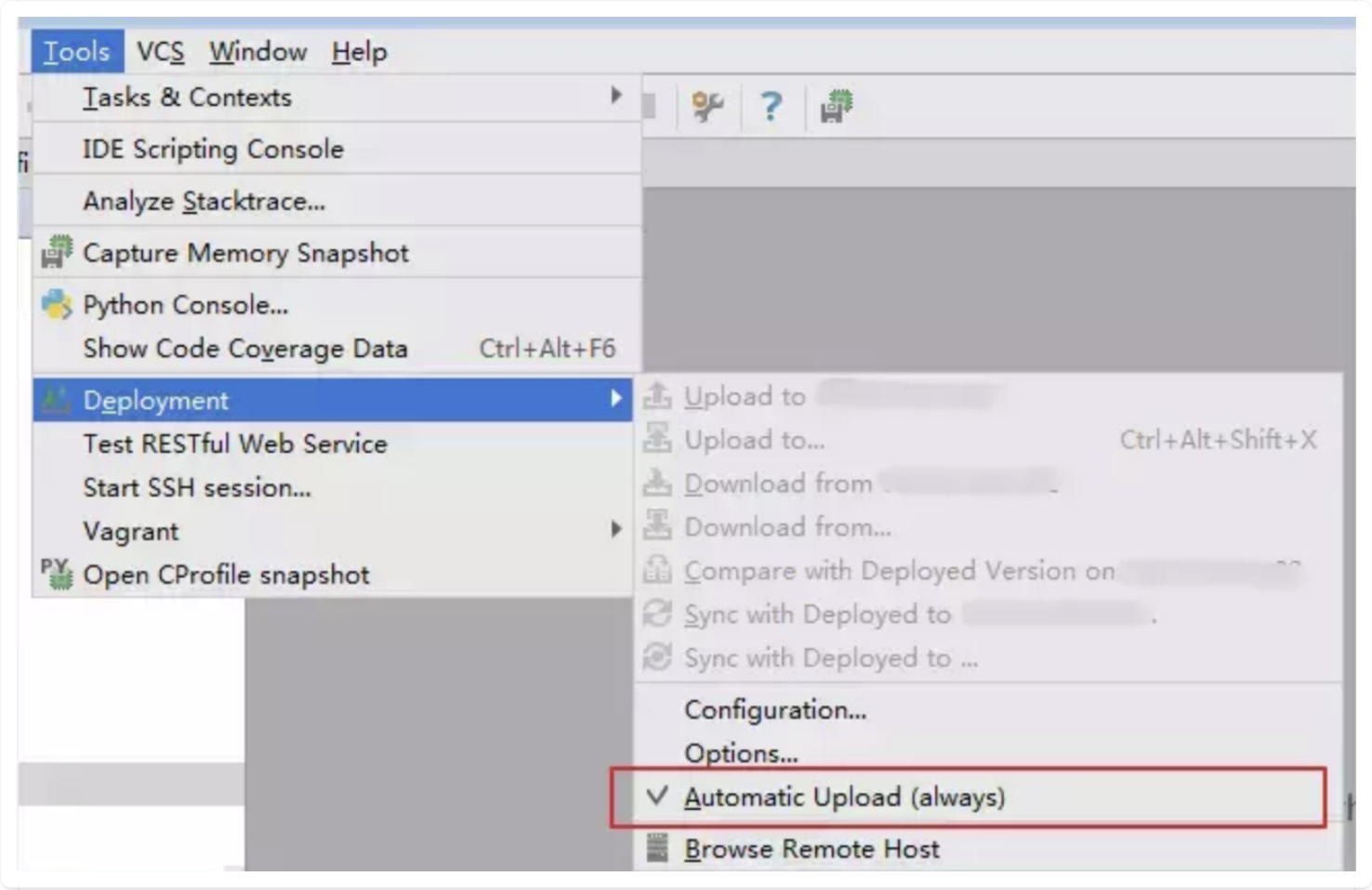
```
[Unit]
Description=OpenStack Nova Compute Server
After=syslog.target network.target libvirtd.service

[Service]
Environment=LIBGUESTFS_ATTACH_METHOD=appliance
Type=notify
NotifyAccess=all
TimeoutStartSec=0
Restart=always
User=nova
ExecStart=/usr/bin/nova-compute

[Install]
WantedBy=multi-user.target
```

看到那个`ExecStart`没有？那个就是我们程序的入口。
我们只要将其拷贝至我们的Pycharm中，并向远程同步该文件。

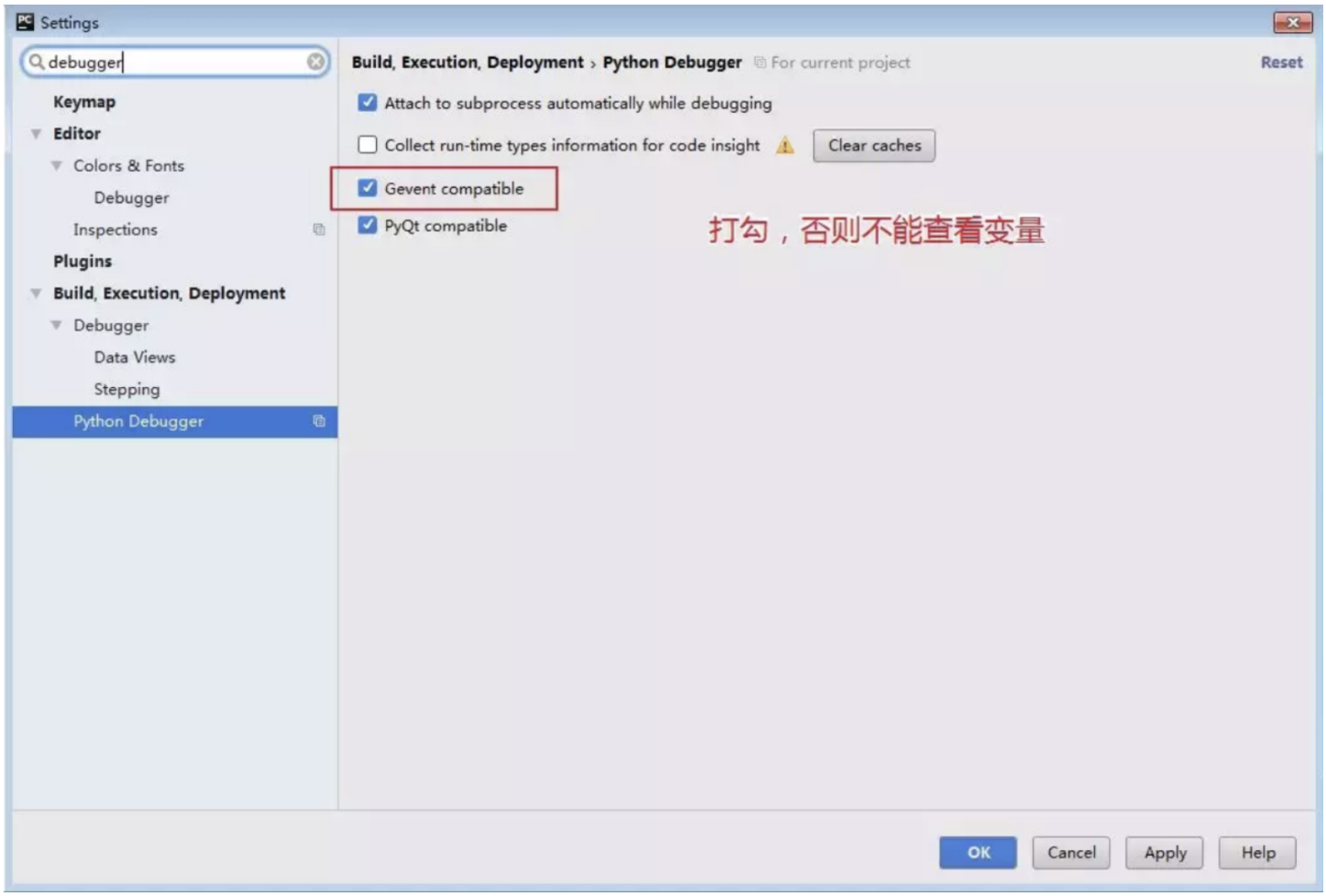


6. 调试前设置

开启代码自动同步，这样，我们对代码的修改Pycharm都能识别，并且为我们提交到远程服务器。



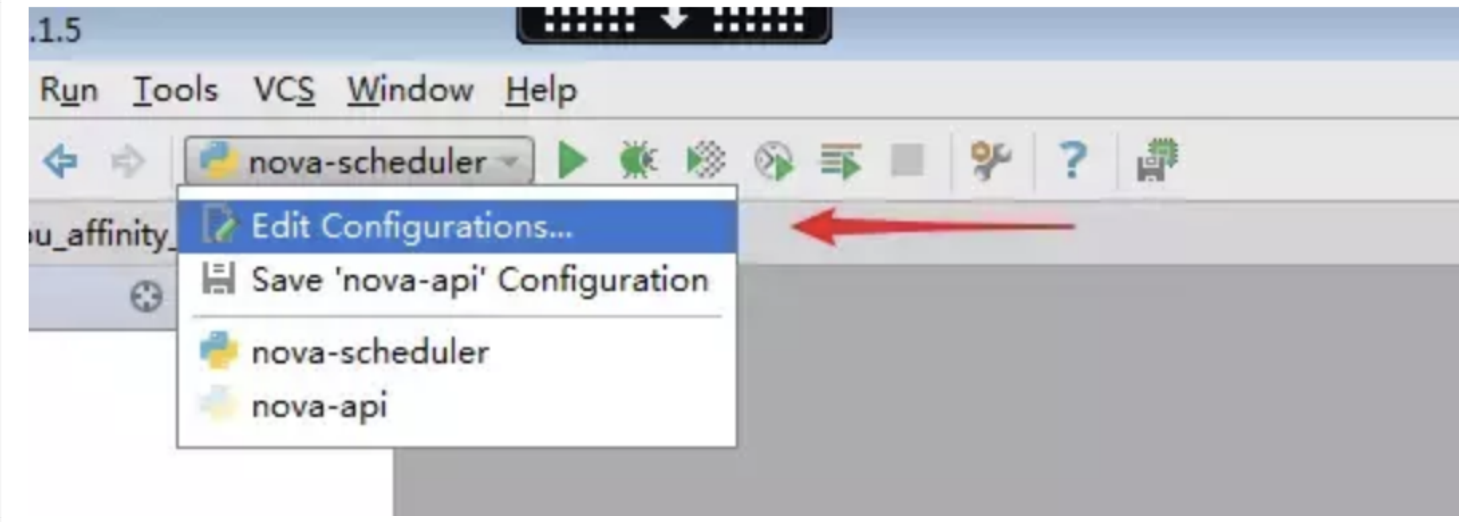
开启 `Gevent compatible`，如果不开启，在调试过程中，很可能出现无法调试，或者无法追踪/查看变量等问题。



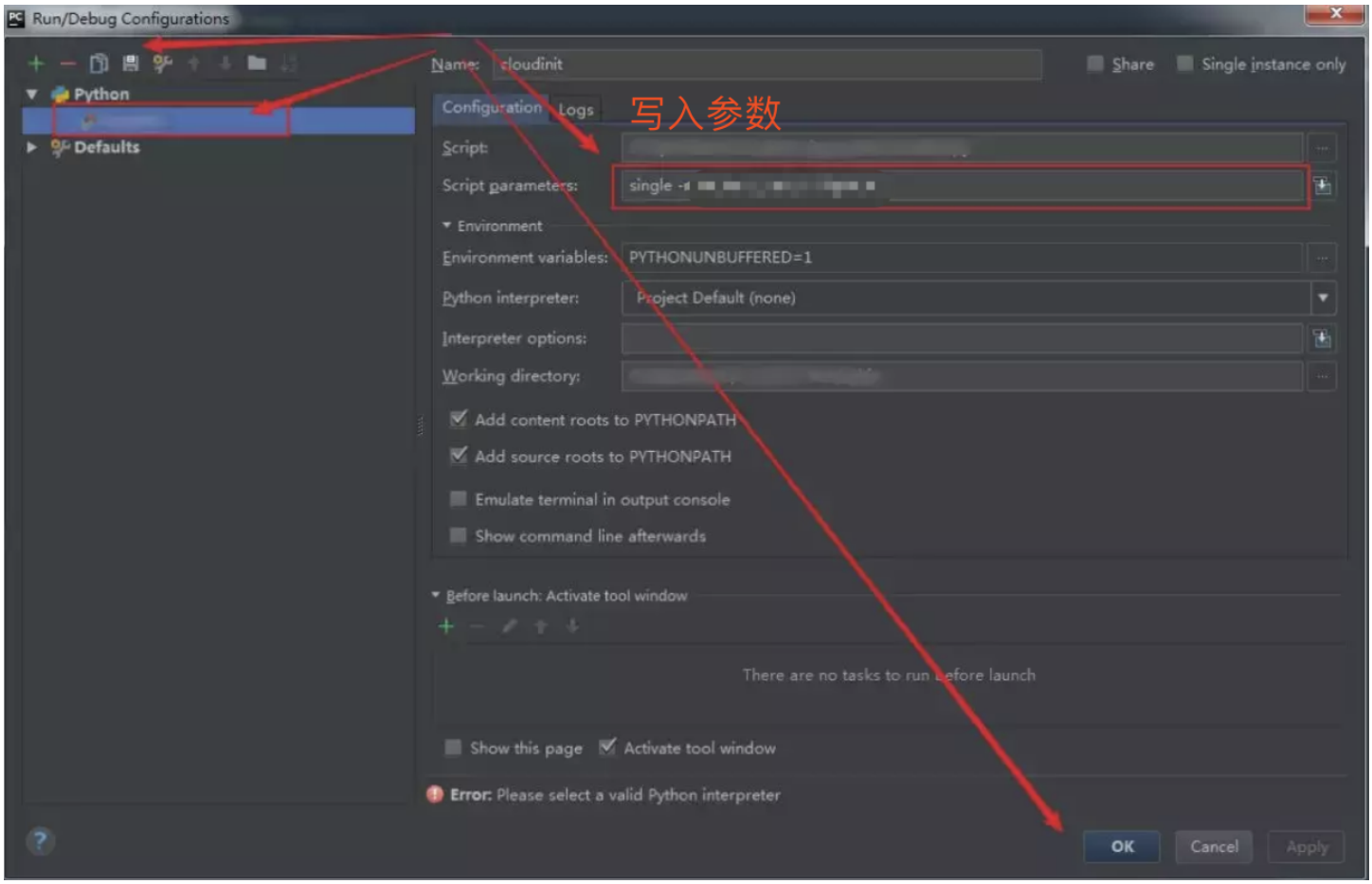
7. 开始调试代码

在你的程序入口文件处，点击右键，选择Debug即可。

如果你的程序入口，需要引入参数，这是经常有的事，可以的这里配置。



配置完点击保存即可。



8. 友情提醒

按照文章的试调试代码，会自动同步代码至远端，千万不要在生产环境使用，一定要在开发环境中使用，否则后果自负。

调试工具给了程序员提供了很大的便利，但还是希望你不要过度依赖。尽量在每次写代码的时候，都追求一次成型，提高自己的编码能力。

作者：王炳明
版本：v1.0
发布时间：2020年08月30日
微信公众号：Python编程时光
联系邮箱：wongbingming@163.com
项目主页：http://pycharm.iswbm.com
Github：https://github.com/iswbm/pycharm-guide

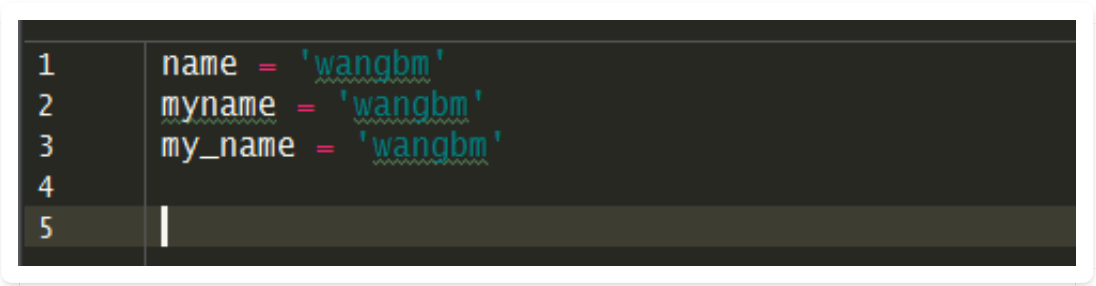
👆👆👆

回复 "**pycharm**"，获取最新版 PDF

第三章：界面与排版

3.2 【界面改造 02】关闭碍眼的波浪线

下面我先给出了一小段代码示例，思考一下，为什么name，my_name 不会有波浪线，而 myname 和 wangbm 会有波浪线呢？

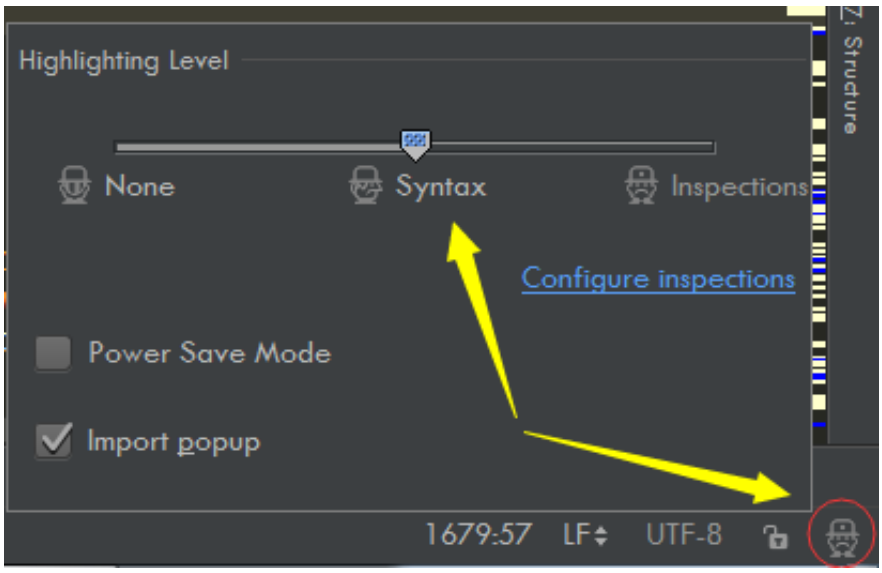


Pycharm 本身会实时地对变量名进行检查，如果变量名不是一个已存在的英文单词，就会出现一条波浪线，当一个变量里有多个单词时，Python 推荐的写法是用下划线来分隔（其他语言可能会习惯使用驼峰式命名法，但 Python 是使用下划线），所以在 Pycharm 看来 my_name 是规范的，而 myname 会被当成是一个单词对待，由于它在单词库里并没有它，所以 myname 是不规范的。

每个人的变量命名习惯不一样，如何你在项目里大量使用了 myname 这种风格的变量命名方法，像下面这样（随便找了一段 cloudinit 的代码），是让人挺不舒服的，总有一种代码有 bug 的错觉。

```
    if name in ("modules", "init"):
        functor = status_wrapper

    report_on = True
    if name == "init":
        if args.local:
            rname, rdesc = ("init-local", "searching for local datasources")
        else:
            rname, rdesc = ("init-network",
                            "searching for network datasources")
    elif name == "modules":
        rname, rdesc = ("modules-%s" % args.mode,
                        "running modules for %s" % args.mode)
    elif name == "single":
        rname, rdesc = ("single/%s" % args.name,
                        "running single module %s" % args.name)
        report_on = args.report

    elif name == 'dhclient_hook':
        rname, rdesc = ("dhclient-hook",
                        "running dhclient-hook module")

    args.reporter = events.ReportEventStack(
        rname, rdesc, reporting_enabled=report_on)
```

那么如何关闭这个非语法级别的波浪线呢？很简单，它的开关就在你的右下角那个像 人头像 一样的按钮



然后选择 `Syntax` 级别的即可。同样一段代码，效果如下，干净了很多。

```
    report_on = True
    if name == "init":
        if args.local:
            rname, rdesc = ("init-local", "searching for local datasources")
        else:
            rname, rdesc = ("init-network",
                            "searching for network datasources")
    elif name == "modules":
        rname, rdesc = ("modules-%s" % args.mode,
                        "running modules for %s" % args.mode)
    elif name == "single":
        rname, rdesc = ("single/%s" % args.name,
                        "running single module %s" % args.name)
        report_on = args.report

    elif name == 'dhclient_hook':
        rname, rdesc = ("dhclient-hook",
                        "running dhclient-hook module")

    args.reporter = events.ReportEventStack(
        rname, rdesc, reporting_enabled=report_on)
```

3.3 【界面改造 03】开启护眼模式

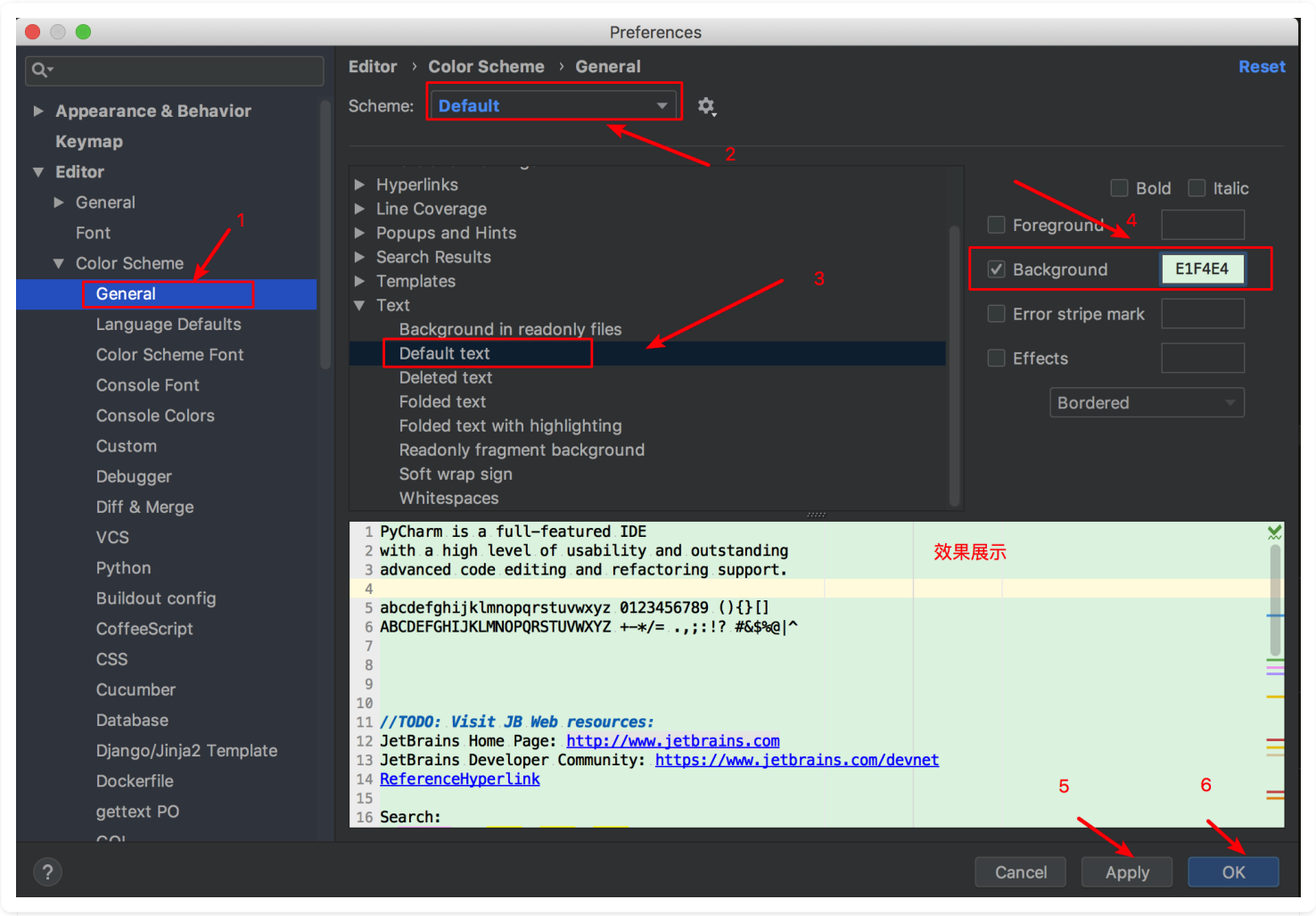
眼睛对着电子产品，容易干涩，对于程序员这种长期以电脑为伍的人群，应该更加注重眼睛的保护。

有些小白领为了保护自己的眼睛，他们通常会将一些办公软件（比如 word、excel，还有资源管理器）的背景色都设置为 **护眼色**，跟我们所说的原谅色差不多 hhh。

那在 PyCharm 中有没有办法也这样子设置呢？

当然有啦

我用一张图，就能给你说清楚，设置方法如下：

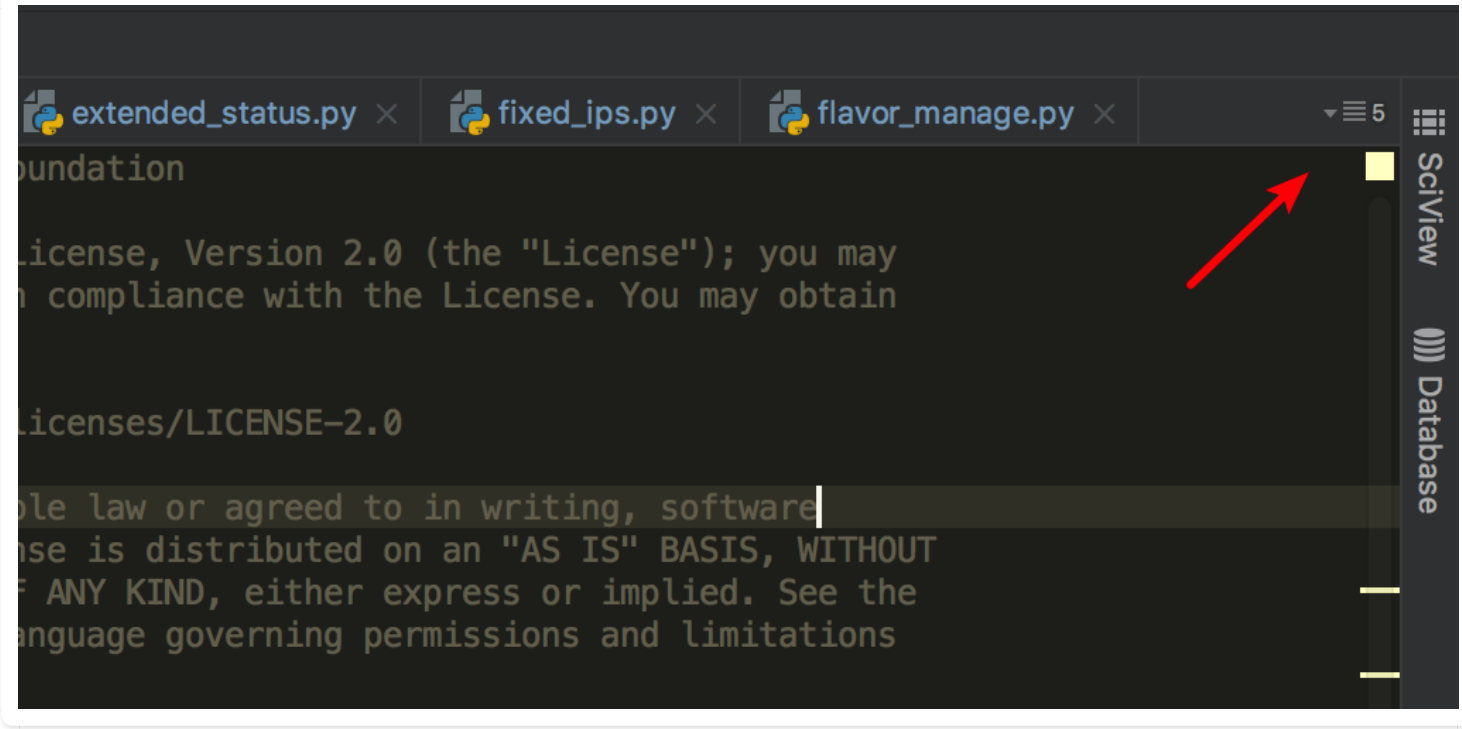


设置了护眼色，会牺牲 PyCharm 的颜值，不再是那个酷炫的极客风了，这就需要你从中取一个取舍。

3.4 【界面改造 04】开启多行标签页

PyCharm 打开一个文件，就占用一个标签面。

你有没有发现，不知不觉地，打开的文件越来越多，多到一行标签都装不下，装不下的标签页 PyCharm 会将其隐藏起来，并以数字的形式告诉你隐藏了几个文件。

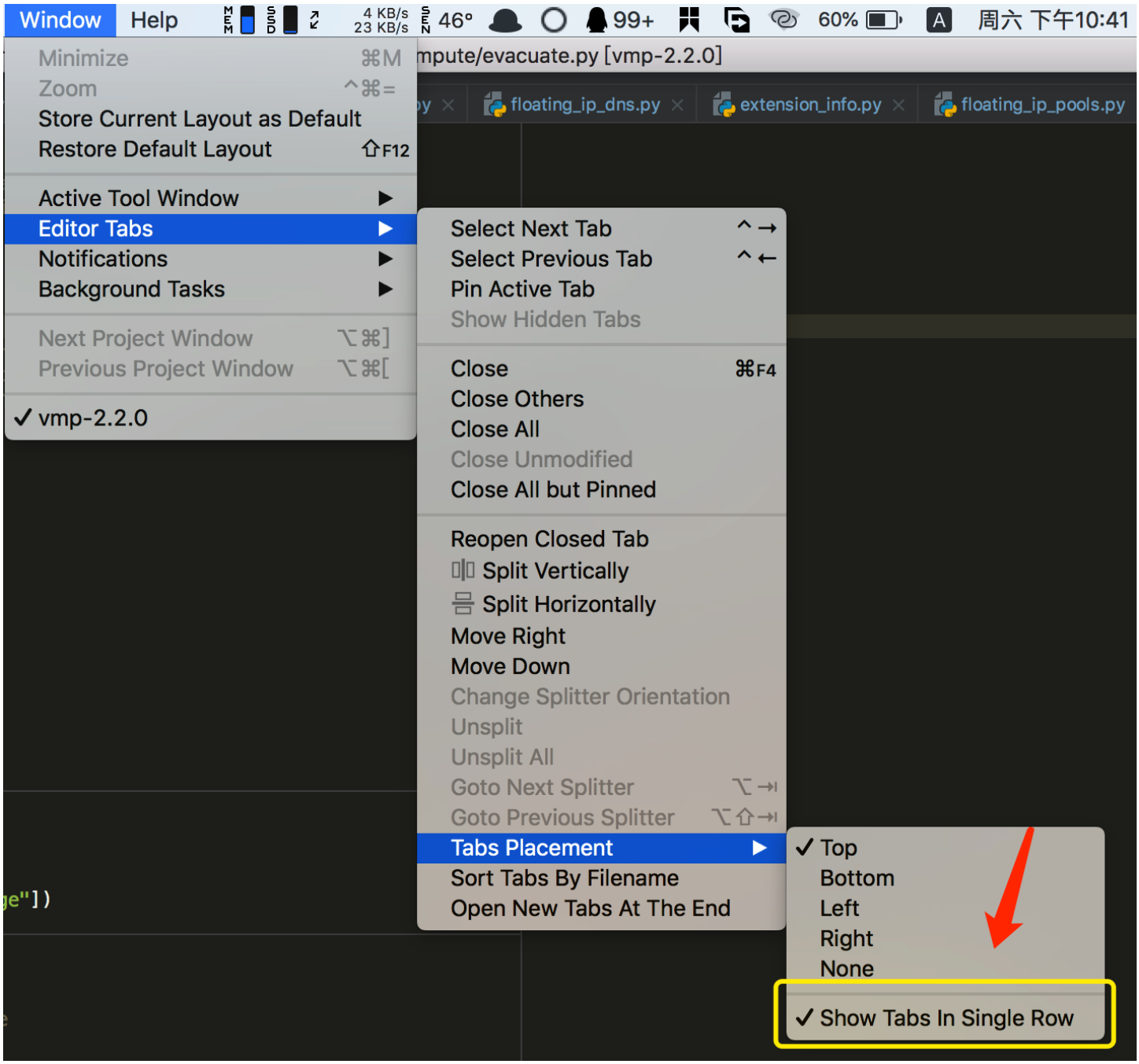


点击数字5，你才可以查看隐藏了哪些文件。

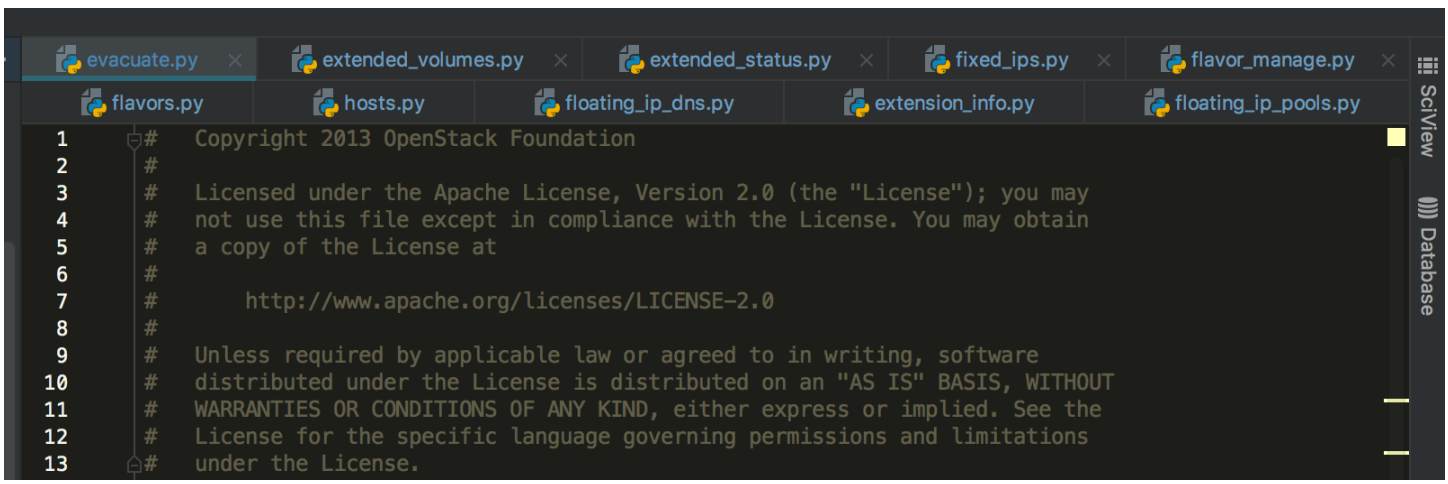
这时你肯定会说，一行装不下 PyCharm 为什么不能多行显示呢？

答案是，不是不能，而是需要你设置。

如下图，将单行显示取消勾选即可。



设置完后，有哪些文件就非常清晰了。



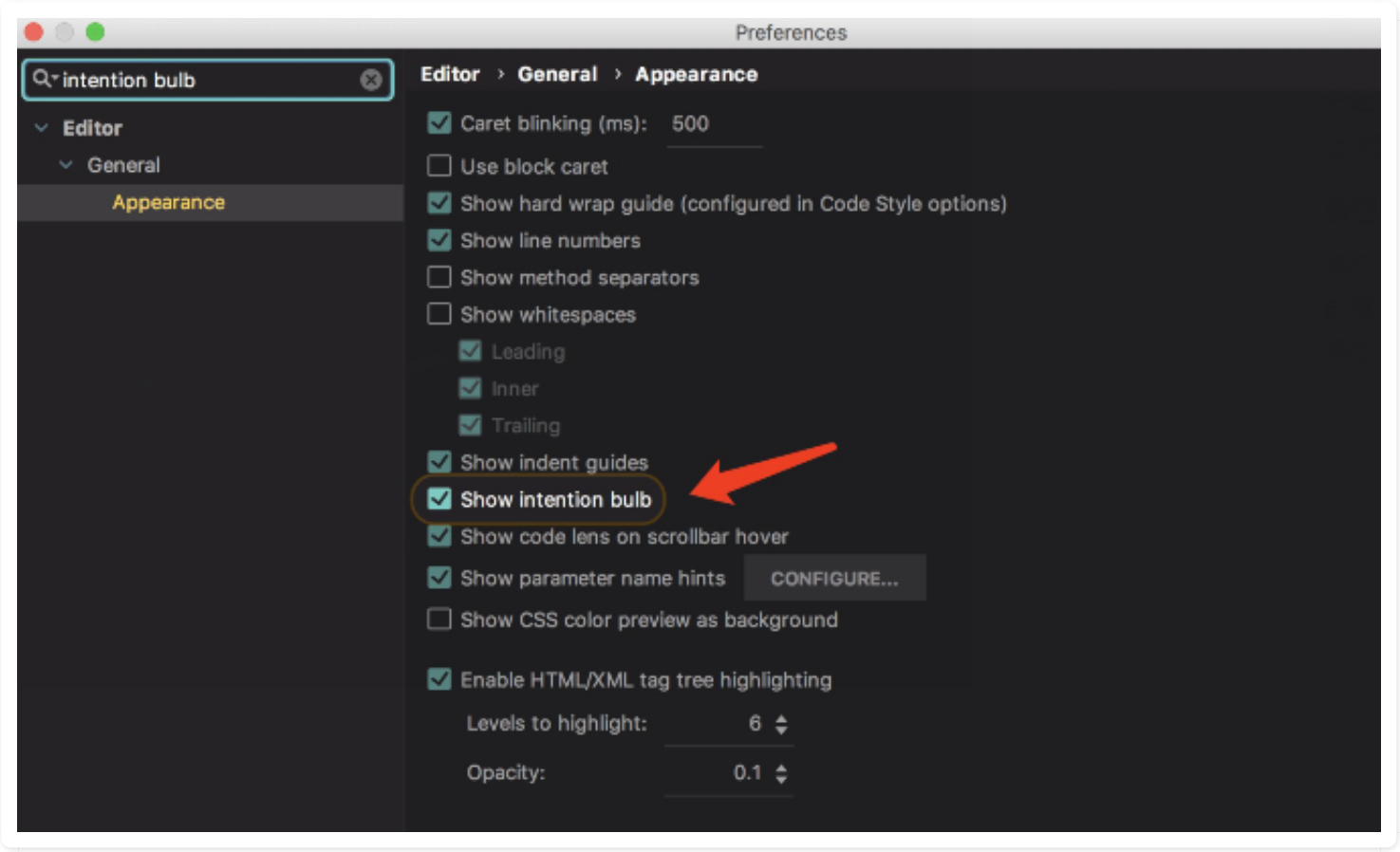
3.5 【界面改造 05】关闭烦人的灯泡提示

当我们在代码里面有语法错误，或者代码编写不符合 pep8 代码规范时，鼠标选择有问题的代码，就会自动弹出小灯泡，这个灯泡是有颜色之分的，如果是红灯泡，一般都是语法问题，如果不处理会影响代码运行。而如果是黄灯泡，就只是一个提示，提示你代码不规范等，并不会影响程序的运行。

虽然这个灯泡，是出于善意之举，但我认为它确实有点多余（可能是我个人没有使用它的习惯），要是语法错误会有红色波浪线提示。你可能会说灯泡不仅起到提示的作用，它还可以自动纠正代码，我个人感觉并没有人工校正来得效率，来得精准。

基于有时还会像知乎上这个朋友说的这样，会挡住我们的代码，会经常误点，这确实也是一个烦恼。

我研究了下，Pycharm （2018版本）里是有开关按钮的，将下图中的这个选项（`Show intention bulb`）取消勾选，就可以关闭这个功能。



3.6 【界面改造 06】小屏幕必看：开启大屏幕编码模式

如果你是使用笔记本来写代码的，那你一定知道，小屏幕写代码的体验可真是太糟糕了。

为此在这里介绍两个小技巧，让你在小屏幕下也能轻松。

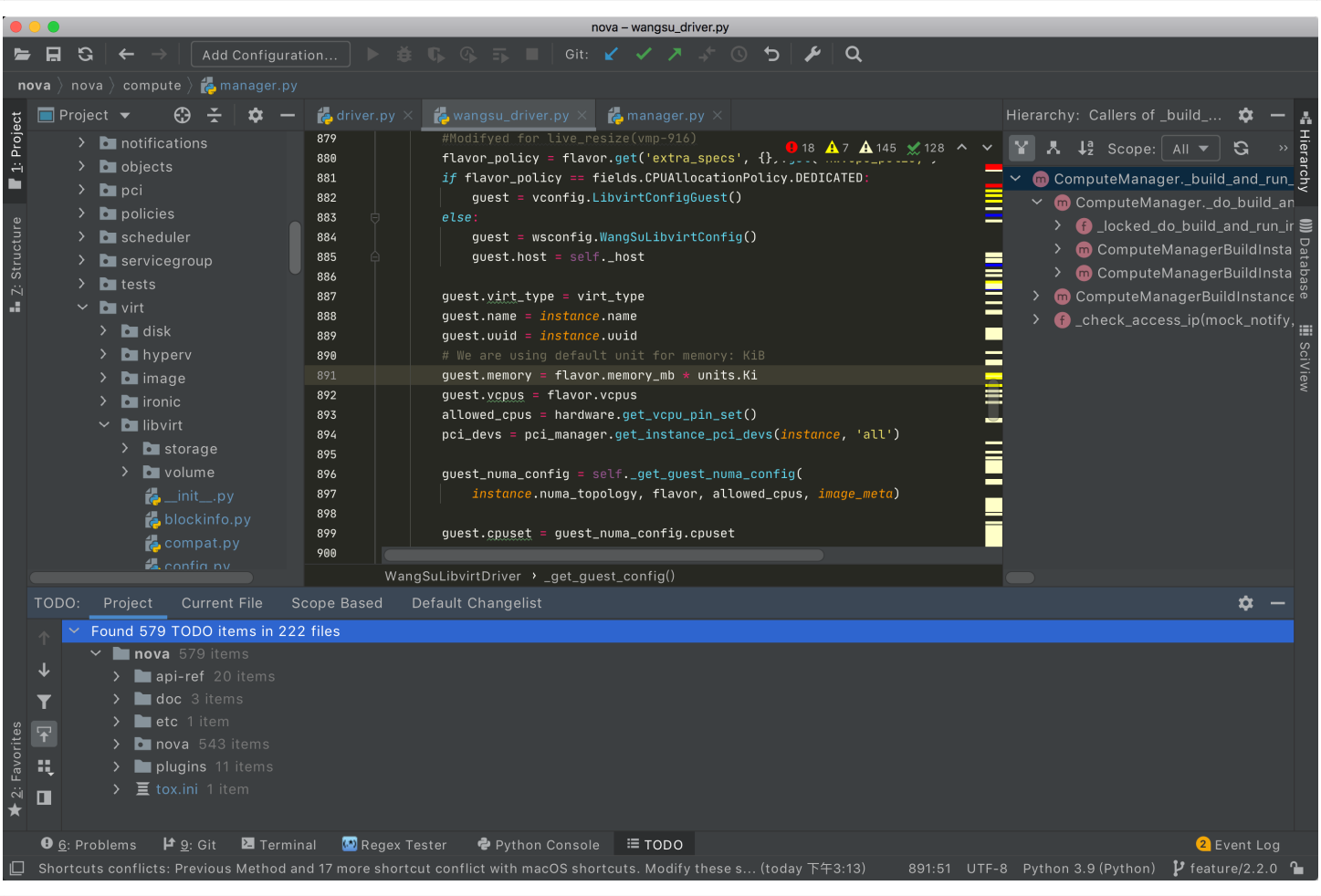
第一个技巧

使用快捷键：⌃ + ⌘ + F， 就可以开启全屏模式。

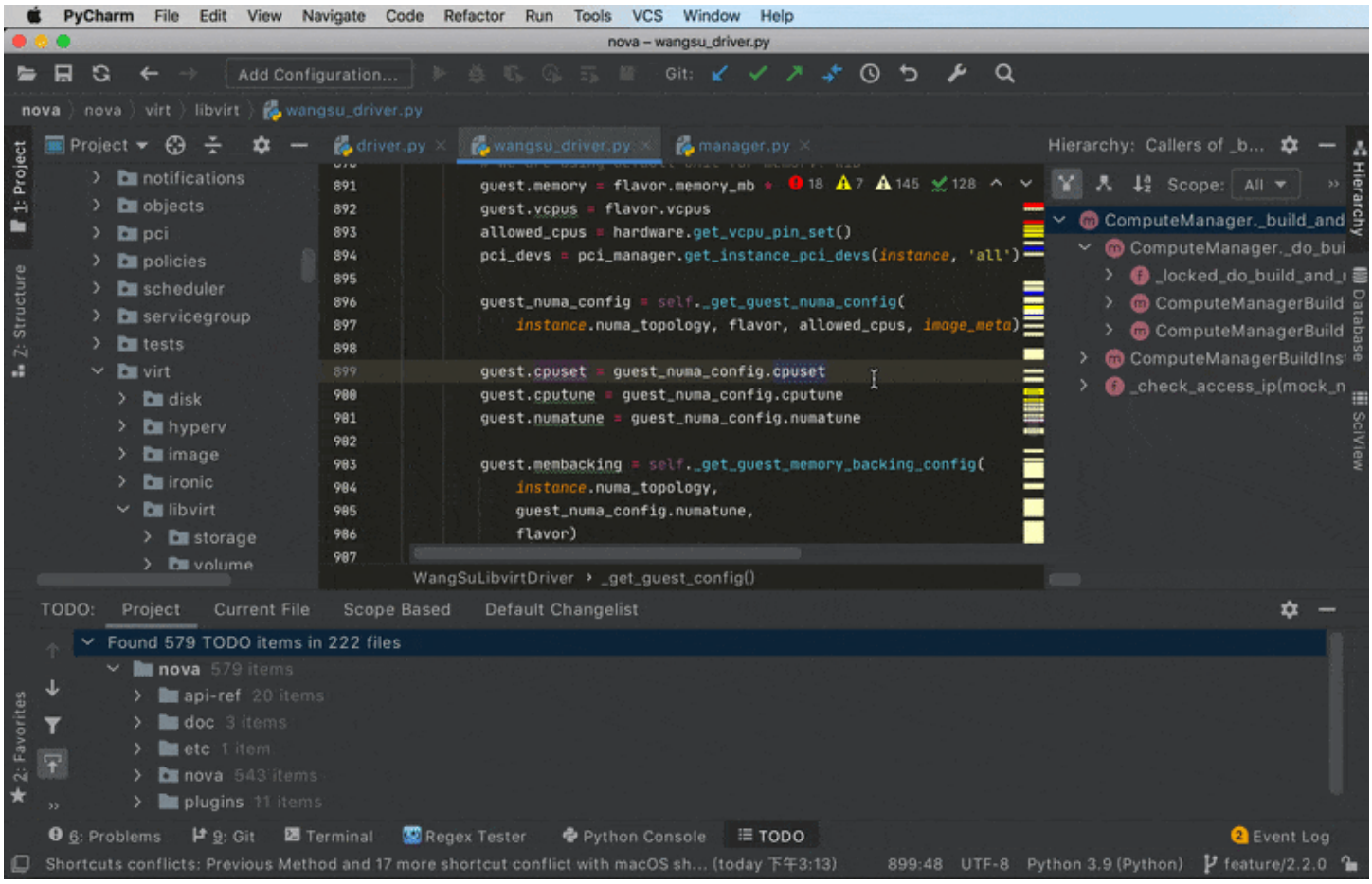
再按一次快捷键，又可以切换回普通 模式（总之就是可以来回切换）。

第二个技巧

如果你的 PyCharm 开启了太多的工具栏，左边，右边，下边，大部分的空间都被工具栏占用了。（如下图）



此时你可以使用快捷键：⌘ + ⇧ + F12，将这些工具栏全部隐藏掉，隐藏掉后再按一次，原来的界面就又回来了。



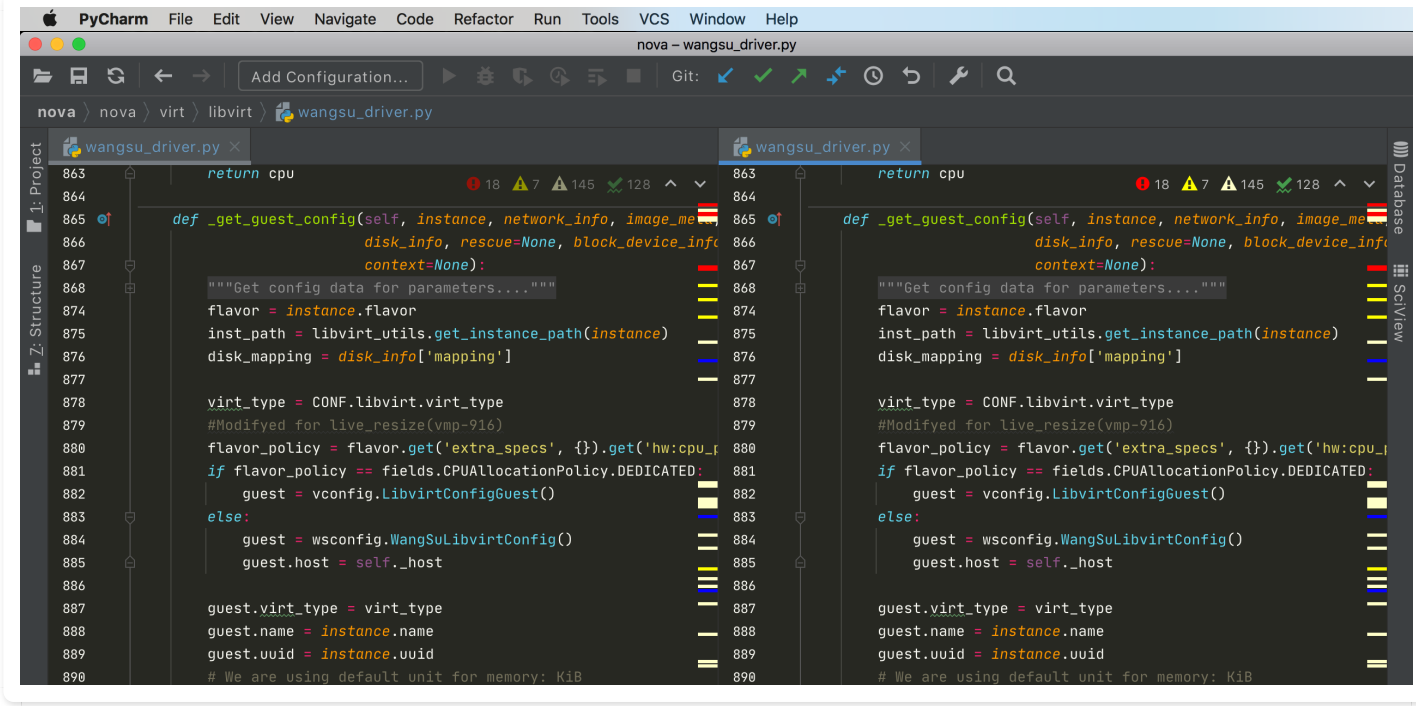
> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

3.7 【界面改造 07】大屏幕必看：分屏查看代码

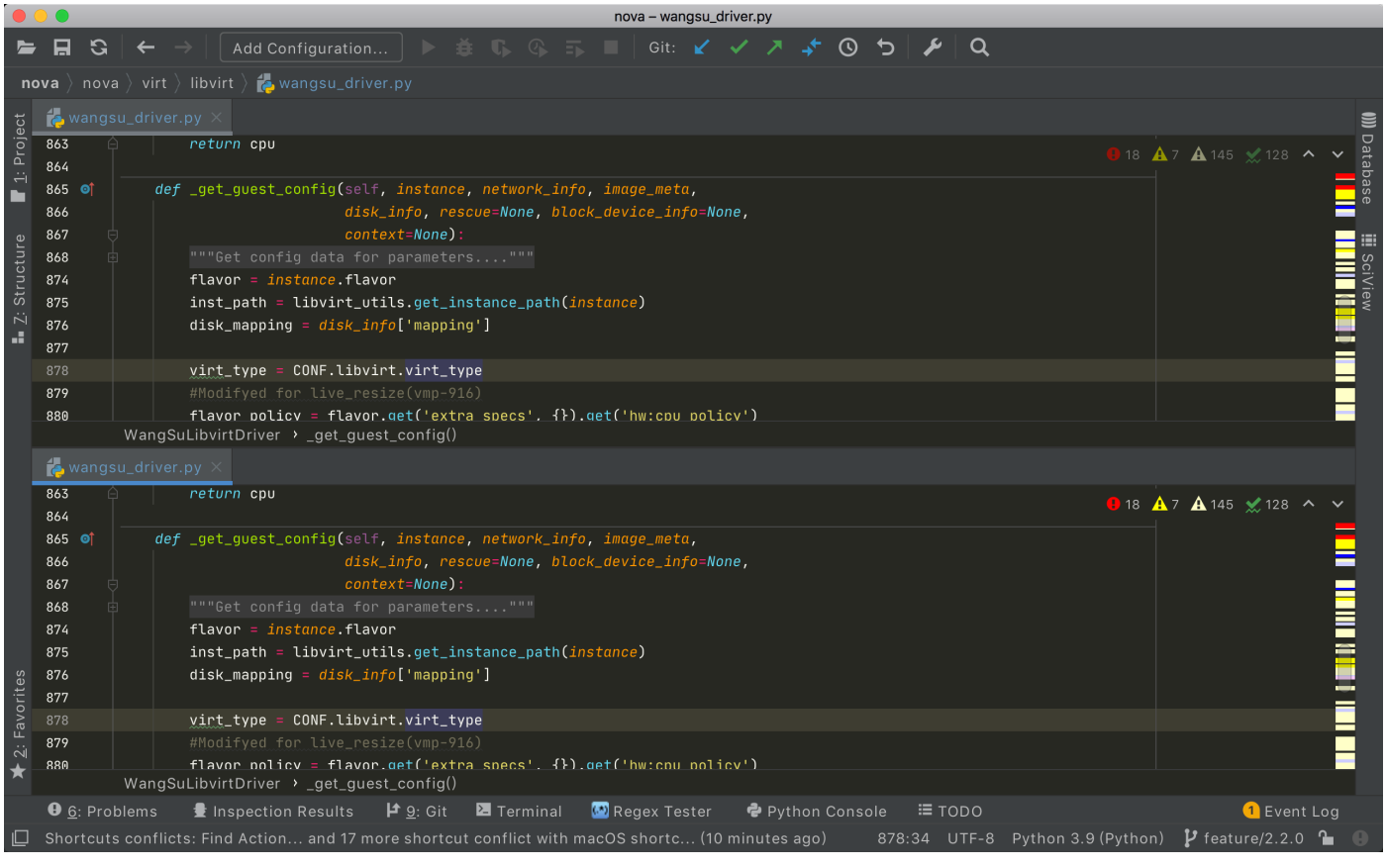
如果需要在一个文件中编写两处代码，而这两处代码又相隔比较远。那么你可以使用对该文件开启分屏模式。

分屏分为两种：

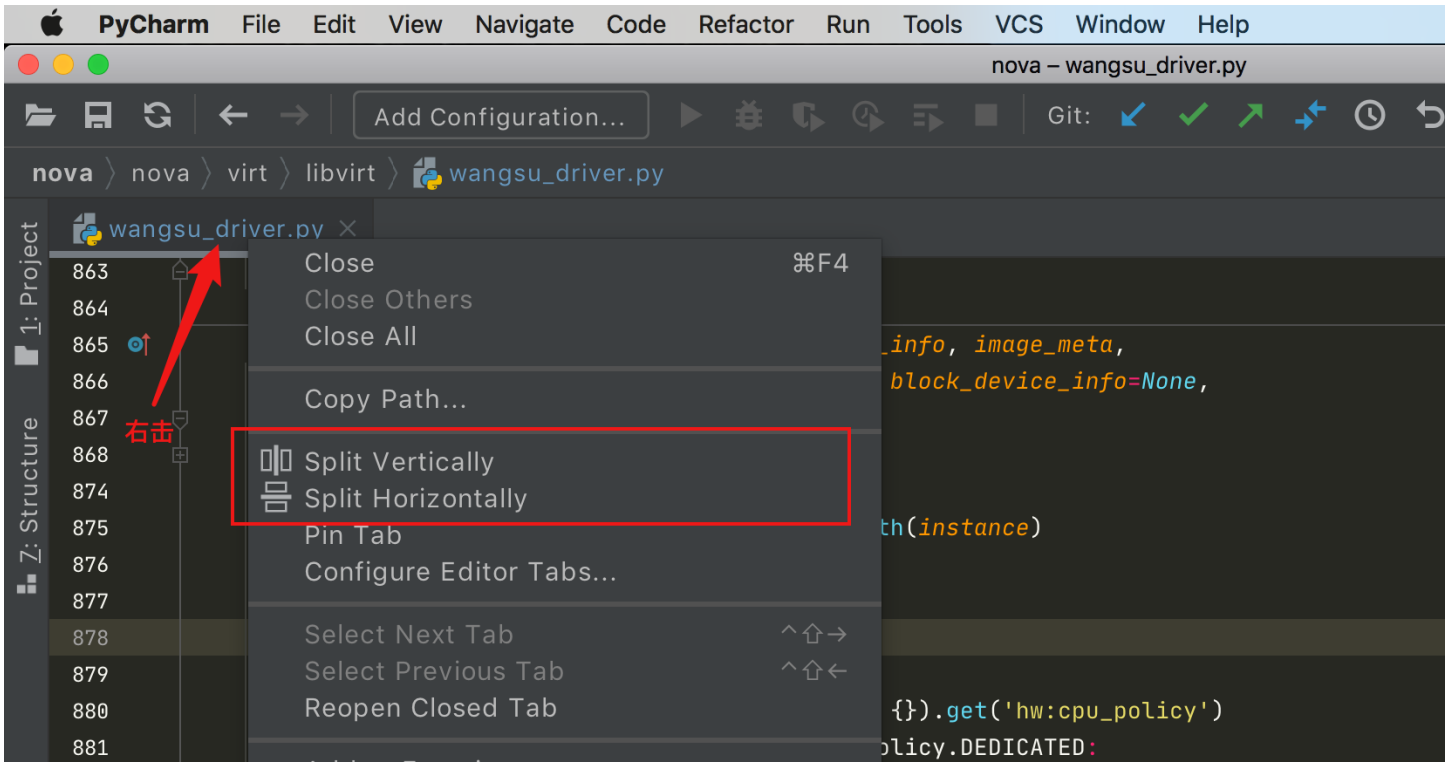
- 竖屏



- 横屏



那怎么开启呢？右击标签页，会有如下两个选项，点击即可。



作者：王炳明
版本：v1.0
发布时间：2020年08月30日
微信公众号：Python编程时光
联系邮箱：wongbingming@163.com
项目主页：http://pycharm.iswbm.com
Github：https://github.com/iswbm/pycharm-guide

回复 "pycharm"，获取最新版 PDF

第四章：代码的编辑

4.1 【高效编辑 01】重写父类方法的正确姿势

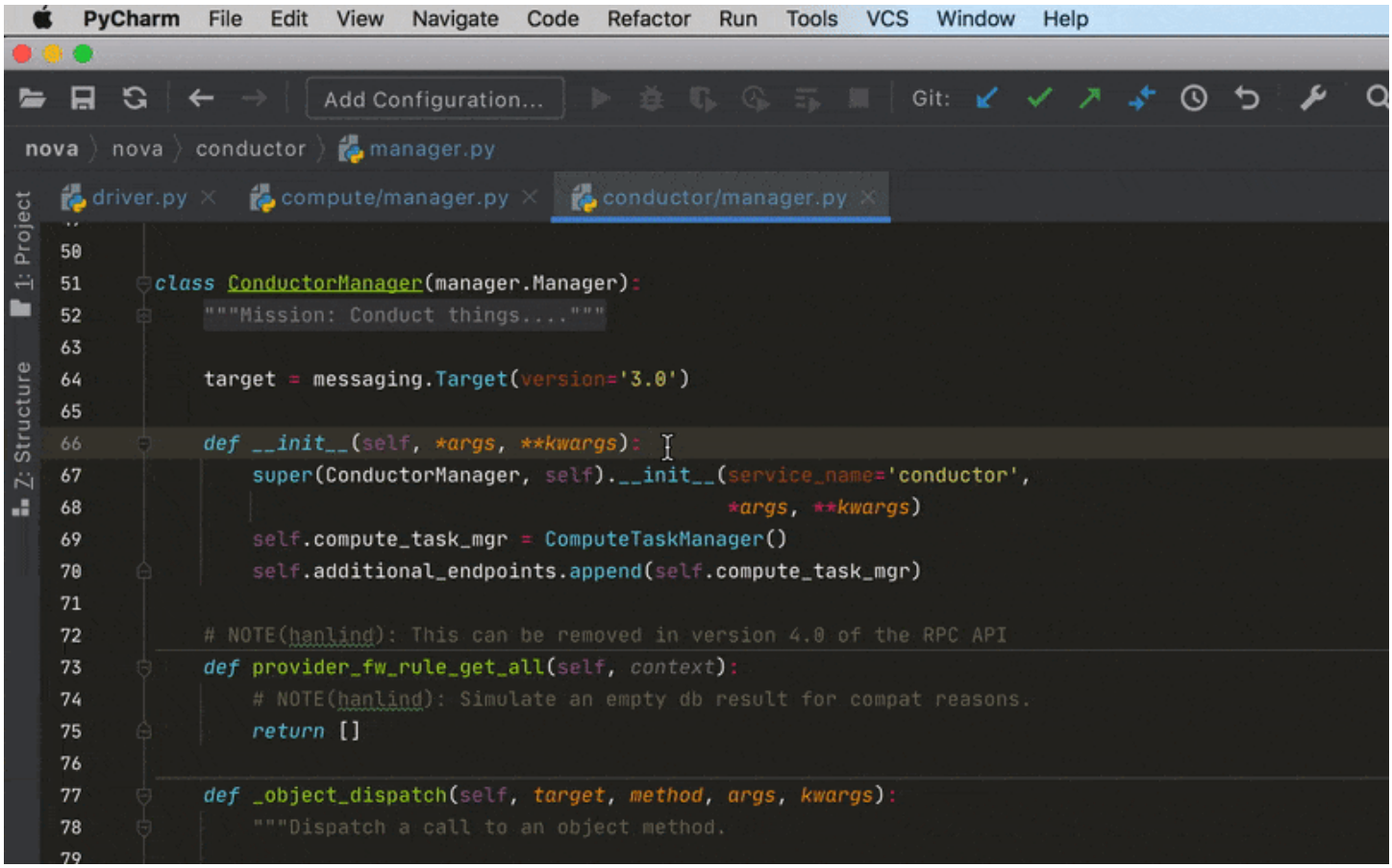
当你想要在子类里，复写父类的一个方法时。

通常都是人工定义一个函数，然后再写 super 表达式。

如果你使用了 PyCharm 还要如此笨拙的方式，那真是埋没了这么好的软件了。

在 PyCharm 中正确复写父类方法的姿势 是使用快捷键：⌘ + O （注意是字母 O，不是数字 0）。

效果如下：



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

4.2 【高效编辑 02】缩进和反缩进

缩进的快捷键是：tab

反缩进的快捷键是：⇧ + tab

4.3 【高效编辑 03】实现接口方法的正确姿势

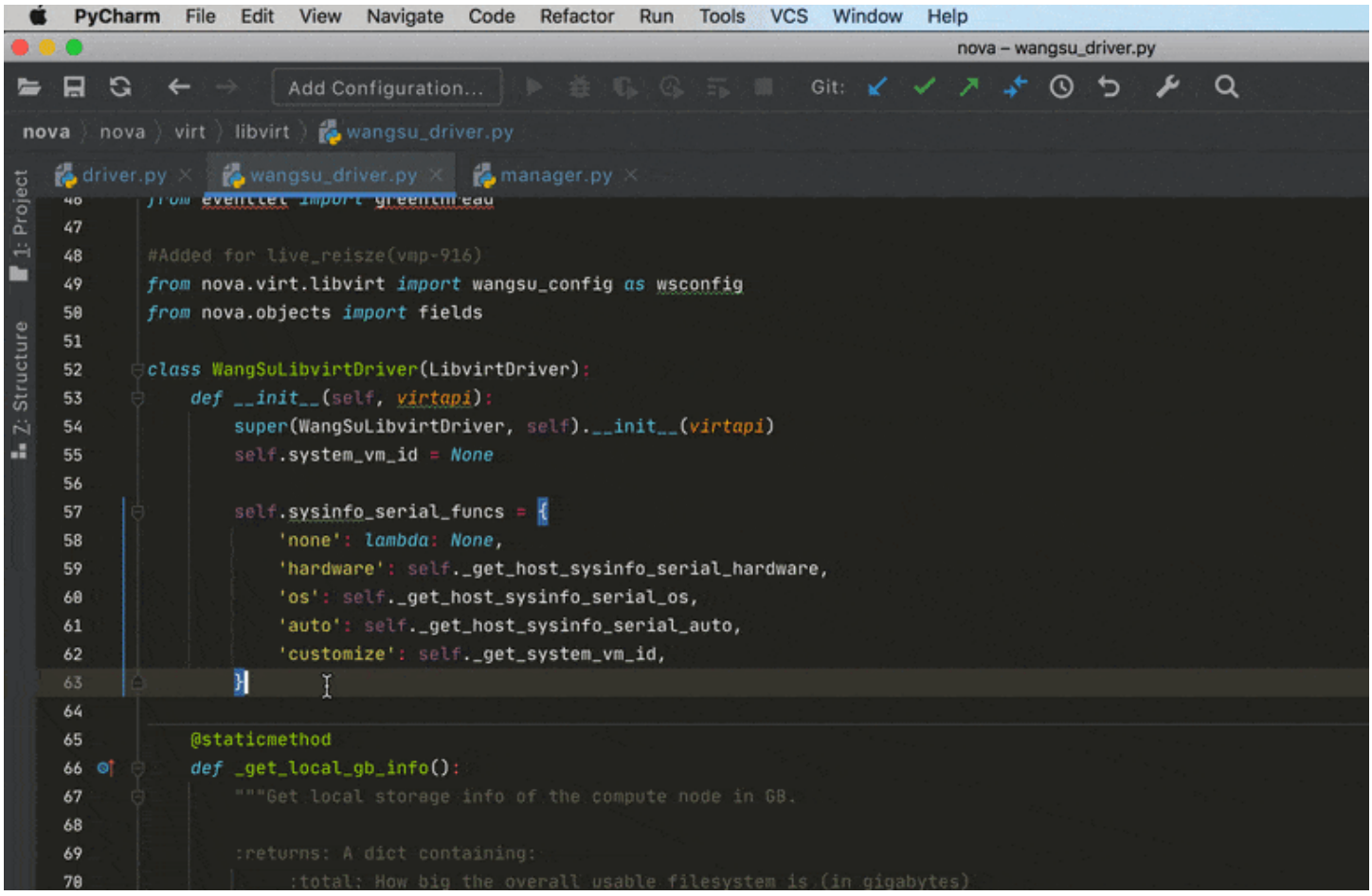
当你想要在类里，实现基类的一个方法时。

通常都是人工定义一个函数，然后写具体的逻辑。

如果你使用了 PyCharm 还要如此笨拙的方式，那真是埋没了这么好的软件了。

在 PyCharm 中正确复写父类方法的姿势 是使用快捷键：⌘ + I （注意是字母 I，不是数字 1）。

效果如下：



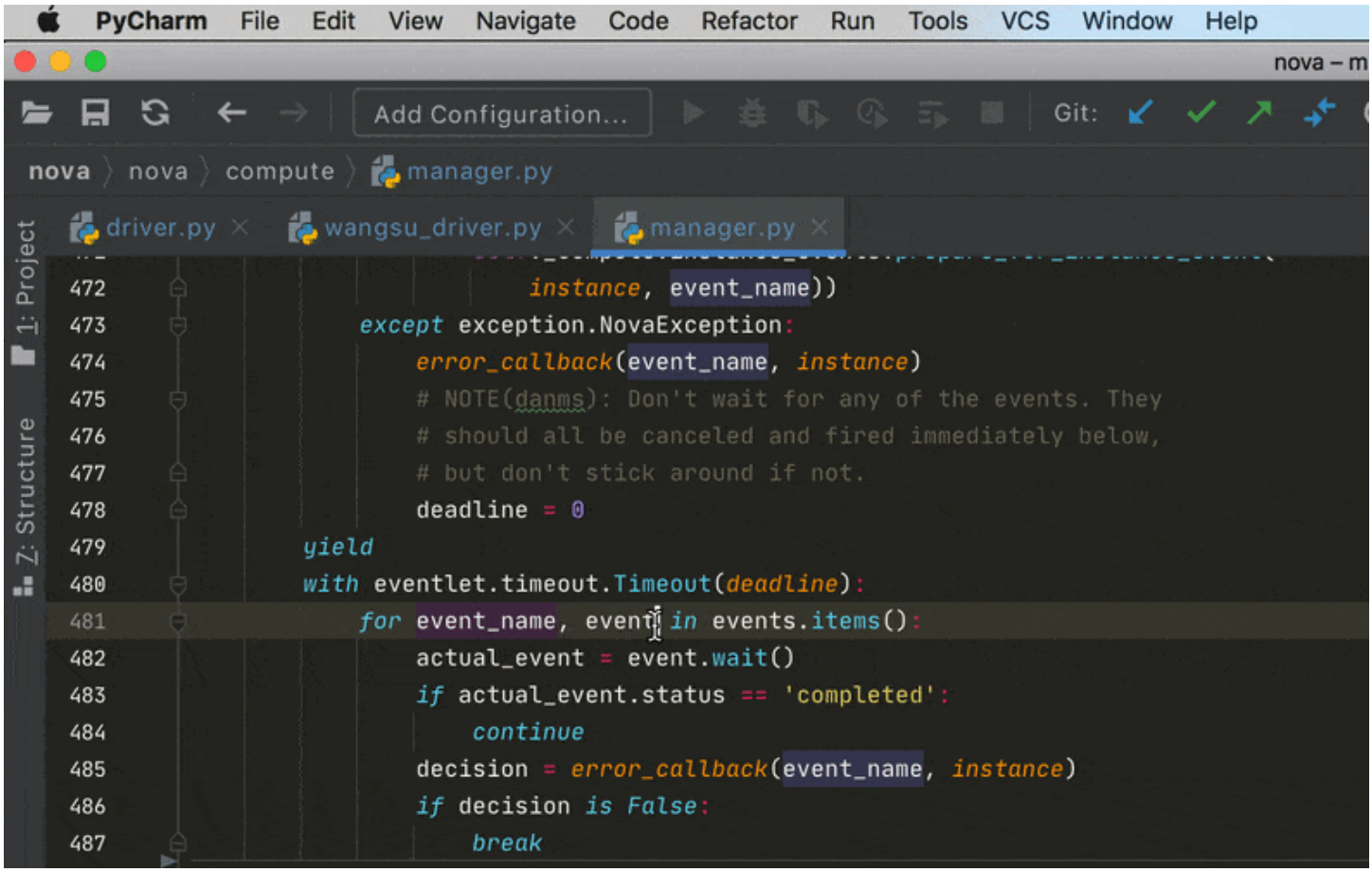
> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

4.4 【高效编辑 04】快速开启新的一行

当你的光标不是处在一行代码的结尾，此时你想换行，一般都是先切换到行尾，再按回车。

其实这两个操作，都可以合并成一个快捷键来完成

那就是：⇧ + Enter，无论你的光标在哪里，都会另起一行，效果如下：

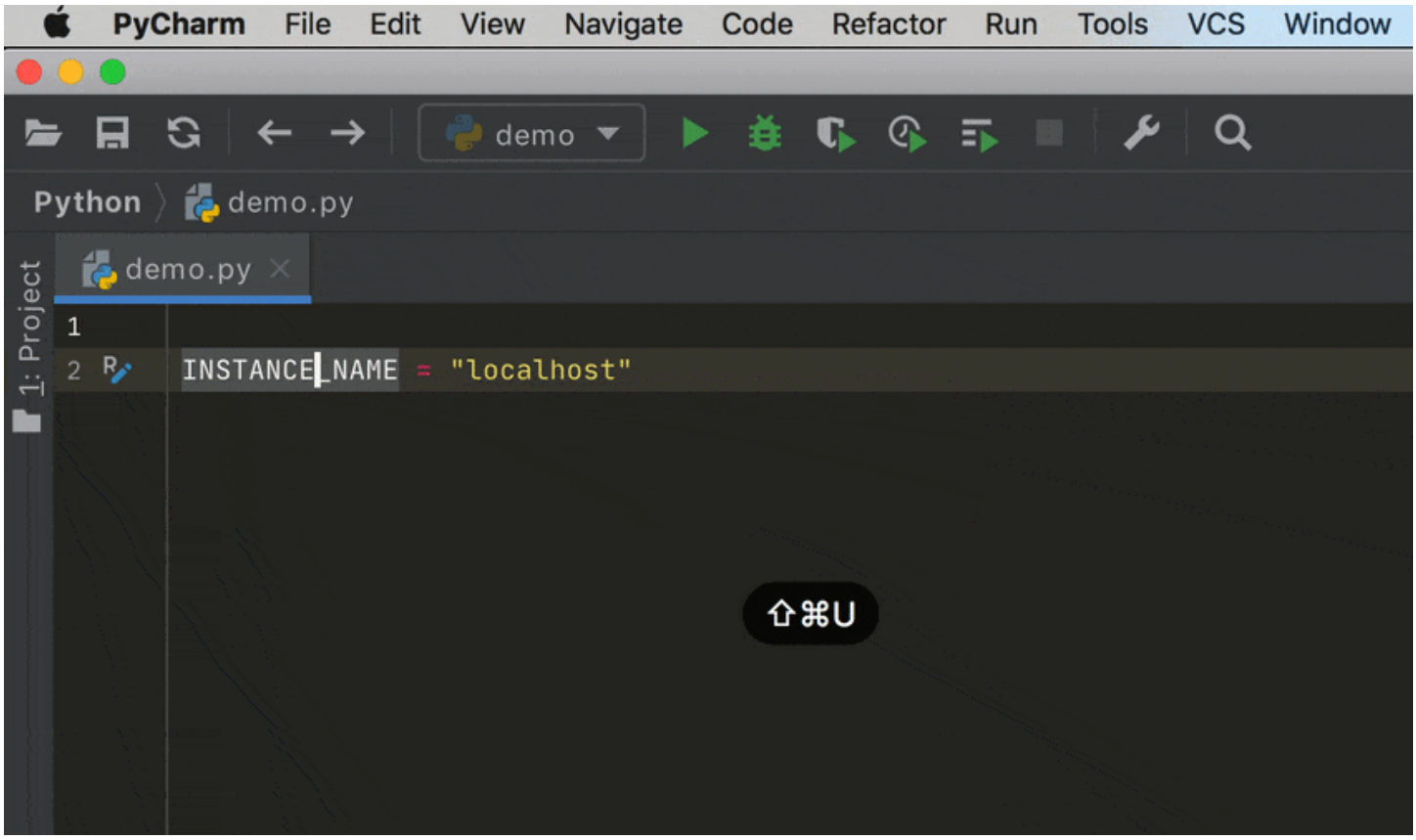


> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

4.5 【高效编辑 05】变量名一键实现大小写的转换

常量通常都是以大写的形式存在的，若你不小心写成了小写，也可以用 ⇧ + ⌘ + U 这组快捷键进行转换。

效果如下：



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

4.6 【高效编辑 06】代码块实现随处折叠

阅读一个新项目的源码，应该先理解代码的整体逻辑，这时候对那些比较细节的通常我们会将其折叠。

但是 PyCharm 中，默认只有整体的代码块，比如一个类，一个函数，一个 if 代码块，一个 for 循环代码块，才会有折叠的按钮。

对于这种原本就有折叠/反折叠按钮的，可以使用下面两组快捷键：

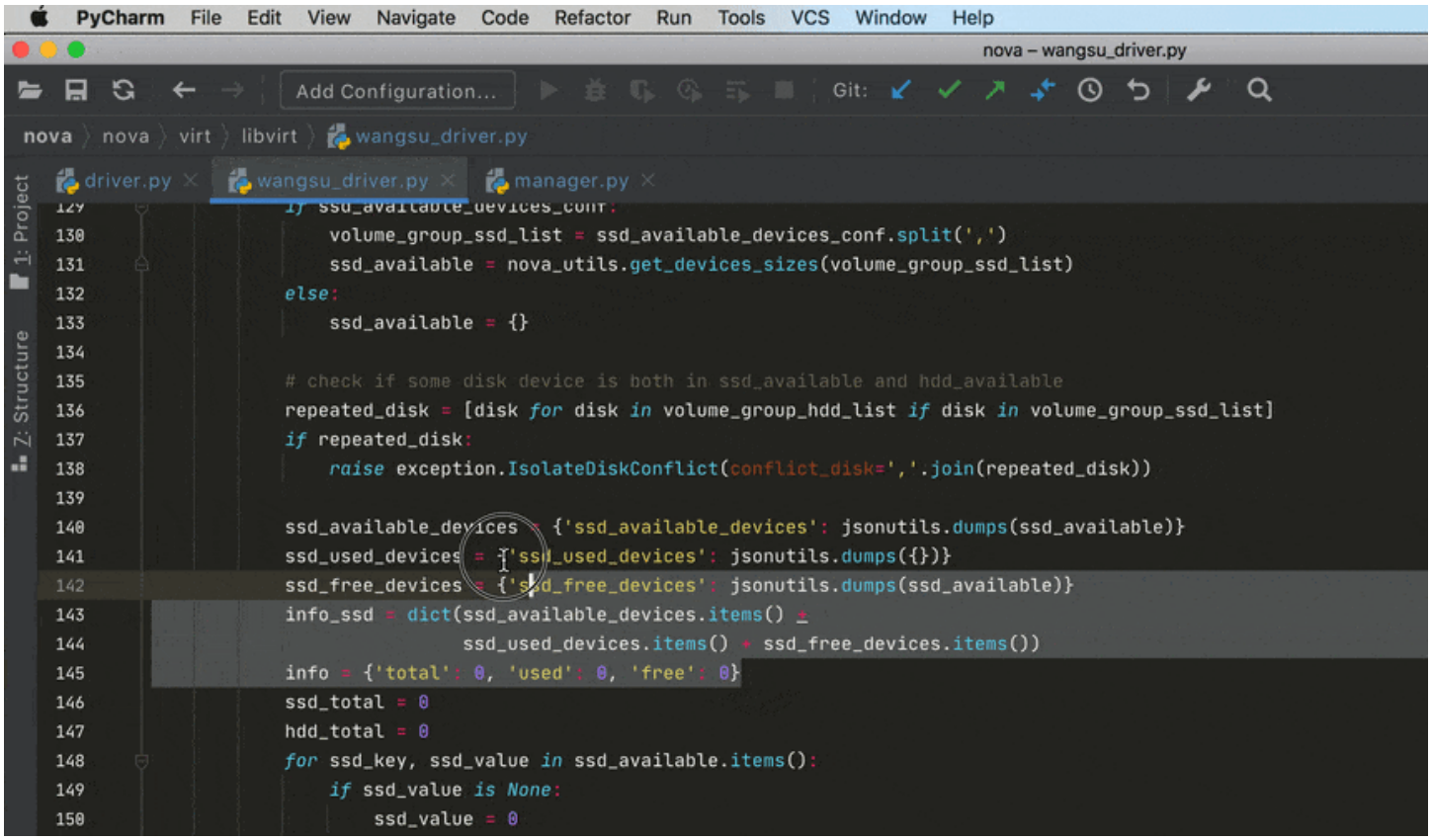
- 折叠：⌘ -
- 反折叠：⌘ +

但有时候，我们并不希望整块代码进行折叠，而只想对其他一大段暂时对我们无用的代码进行折叠。那能做到吗？

答案是可以的。

只要你先选中你想折叠的代码，再按住 ⌘ 紧接着按住 `.` 就可以了。

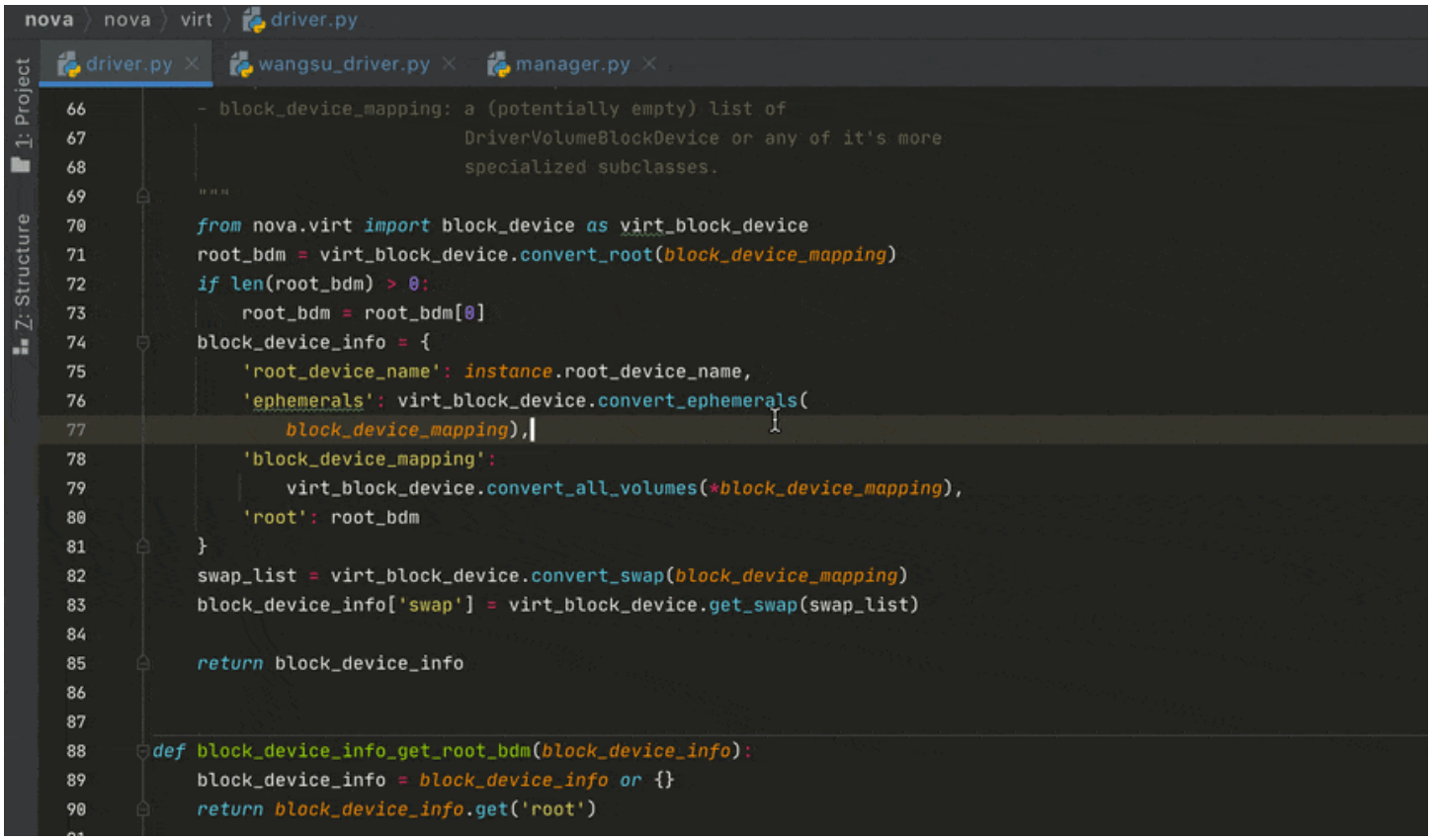
效果如下：



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

另外，还有一组快捷键也要说下

- ⌘ ⇧ +： 展开所以代码块
- ⌘ ⇧ -：折叠所有代码块



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

4.7 【高效编辑 07】删除与剪切的技巧

删除单词到开头

快捷键： ⌥ + ⌫ ， 删除到单词的开头

删除单词到尾部

快捷键： ⌥ + ⌦ ， 删除到单词的末尾（⌦键为Fn+Delete）

删除单行

快捷键： ⌘ + X

复制当前行到剪切板

快捷键： ⌘ + C

剪切当前切并粘贴

快捷键： ⌘ + D

4.8 【高效编辑 08】历史剪切板的使用：Paste from History

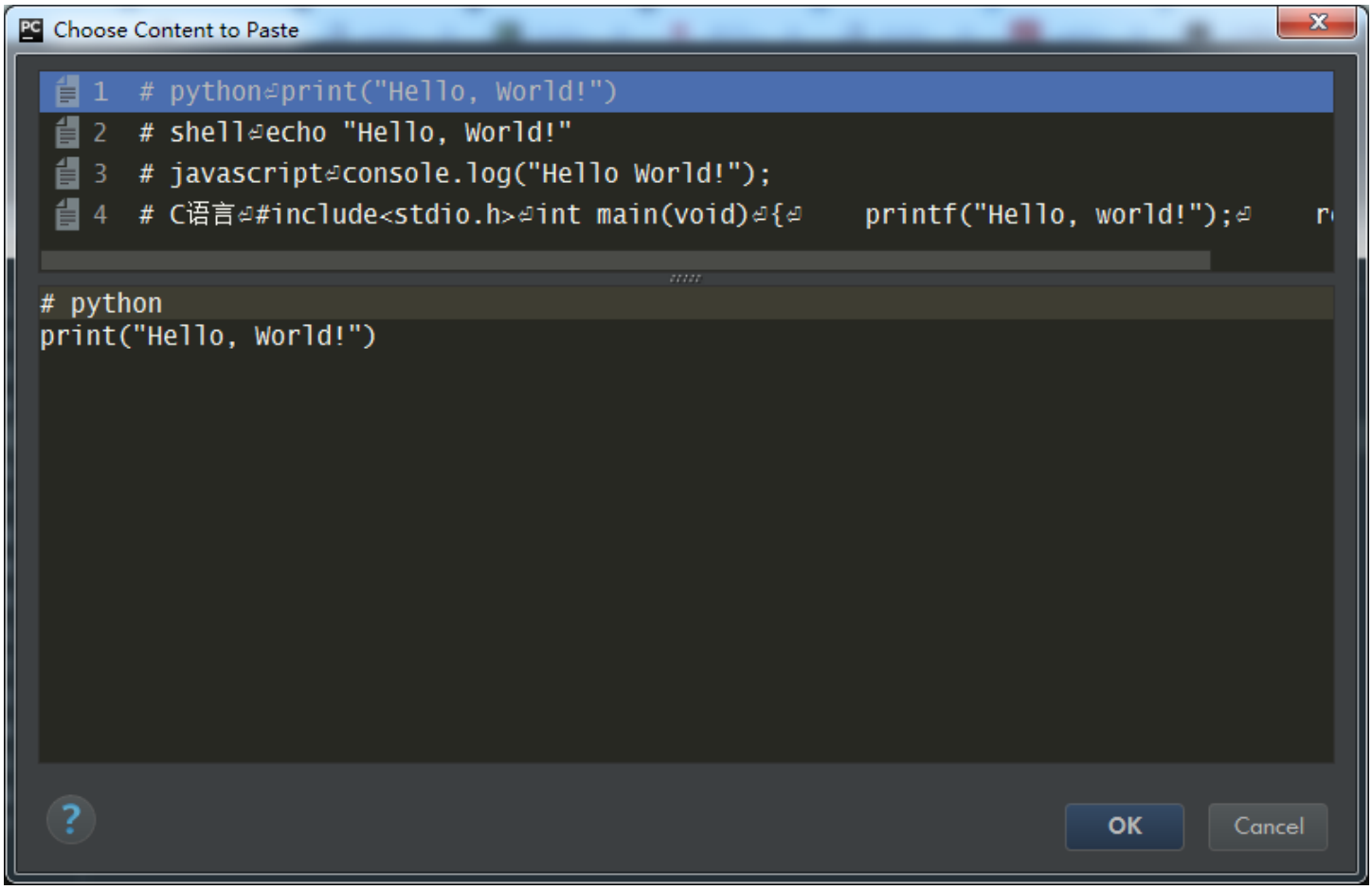
在 Windows 上有一个剪切板神器 - `Ditto`， 它可以将你间经复制粘贴过的内容都保存下来， 以便你重复使用。

在 Mac 上有一个神器， 叫 Alfred， 它也有类似的功能。

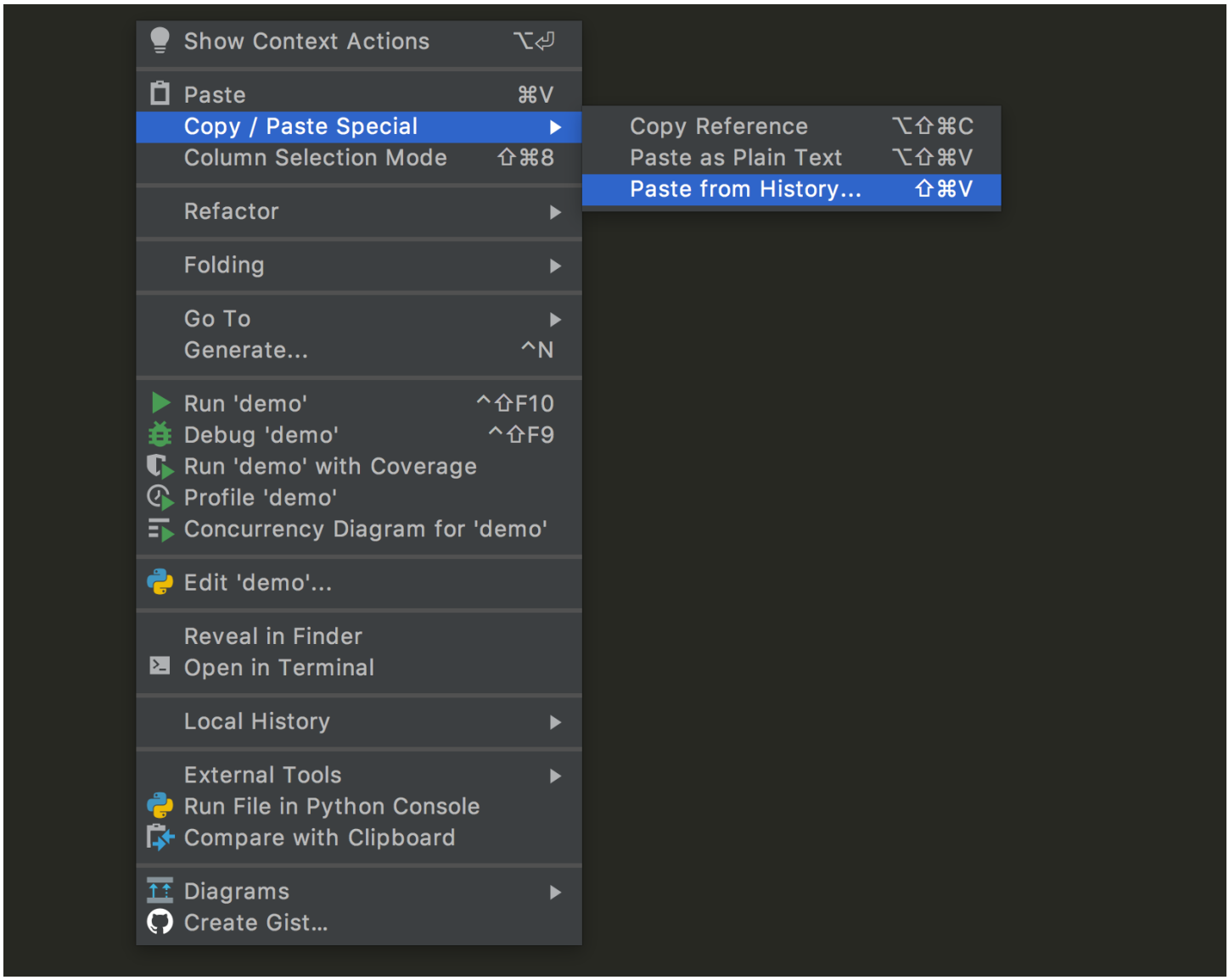
如果你没有使用过 Ditto 和 Alfred， 不要紧，其实 PyCharm 也有这样的功能。

只要你按住 `Command + Shift + V` （Windows 上是 `Ctrl + Shift + V`）就可以调出像下面这样的剪切板。

这里我提前准备了几种编程语言的 Hello World， 效果如下：



你可以通过右键调出此窗口



4.9 【高效编辑 09】使用函数时，快速查看该函数有哪些参数

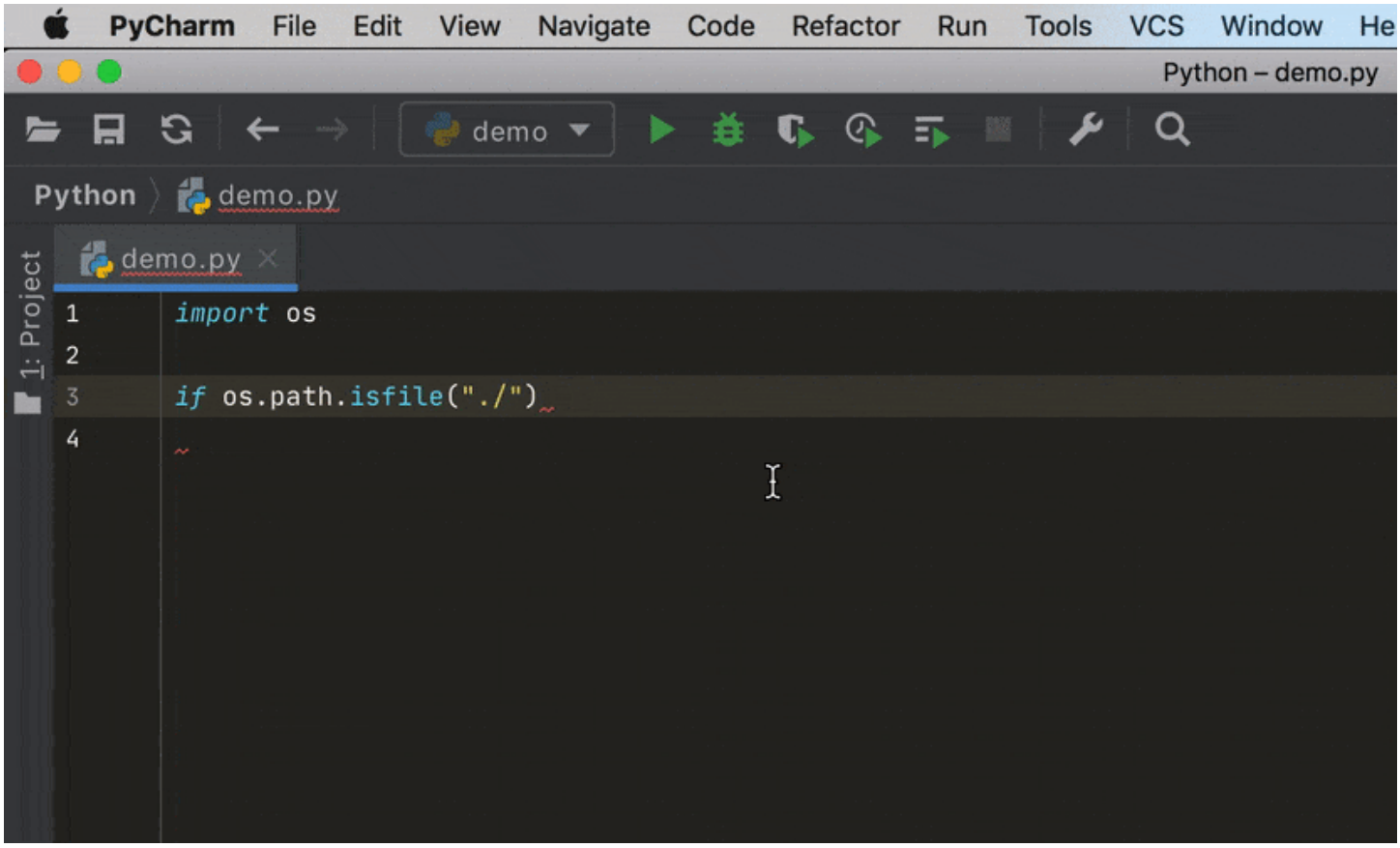
快捷键：⌘ + P

```
38  def driver_dict_from_config(named_driver_config, *args, **kwargs):
39      driver_registry = dict()
40
41      for driver_str in named_driver_config:
42          driver_type, _sep, driver = driver_str.partition('=')
43          driver_class = importutils.import_class(driver)
44          try:
45              driver_registry[driver_type] = driver_class(*args, **kwargs)
46          except ValueError:
47              # NOTE(arne_r):
48              # stable/newton can not enforce os_brick versions that include
49              # the InvalidConnectorProtocol exception. Since it inherits from
50              # ValueError, this fix is still compatible with it.
51              LOG.debug('Unable to load volume driver %s. It is not '
52                        'supported on this host.', driver_type)
53
54
55  driver_dict_from_config()
56
```

named_driver_config, *args, **kwargs

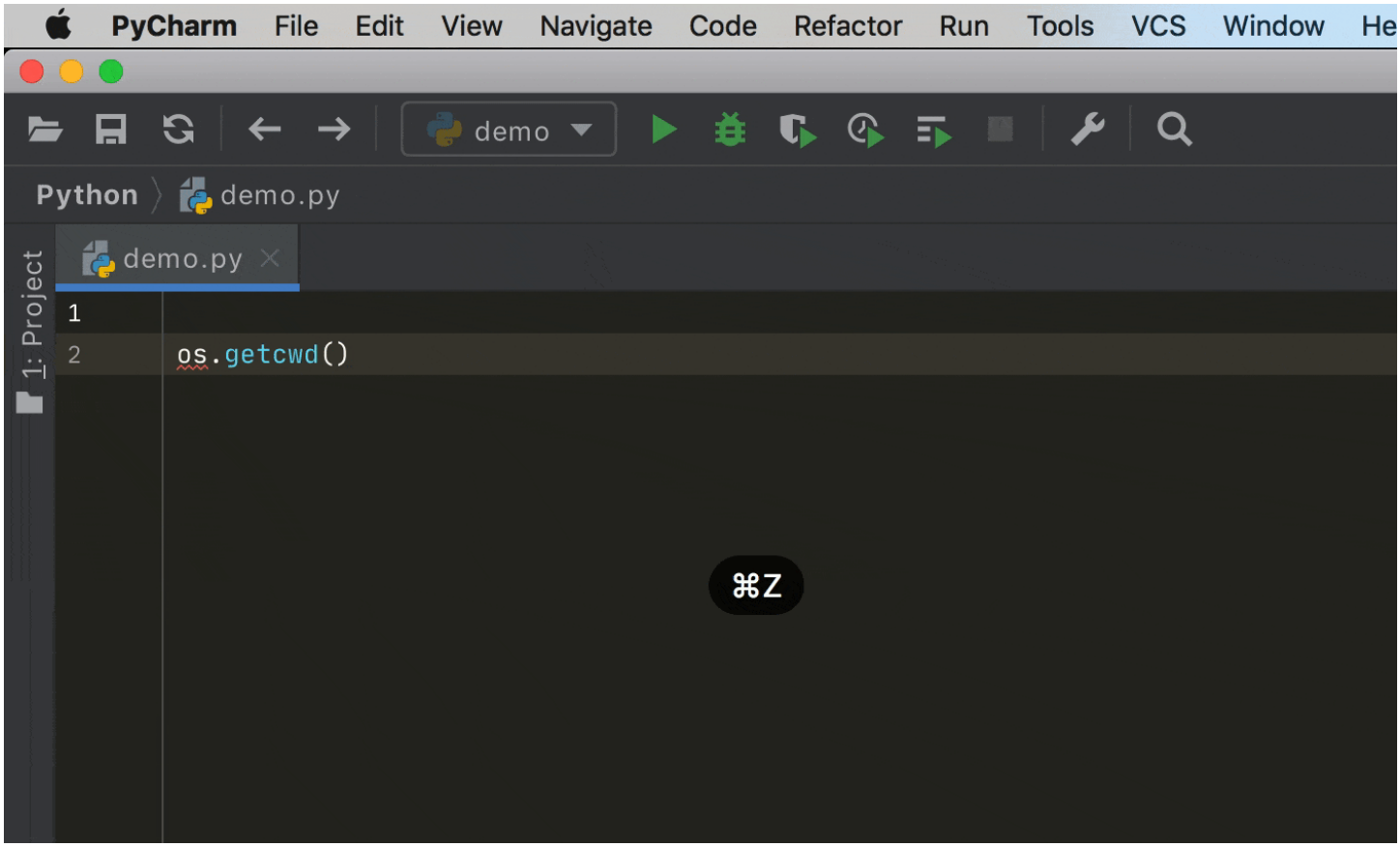
4.10 【高效编辑 10】自动纠正与自动补全

快捷键：⌘ + ⇧ + ↩，自动结束代码，行末自动添加分号



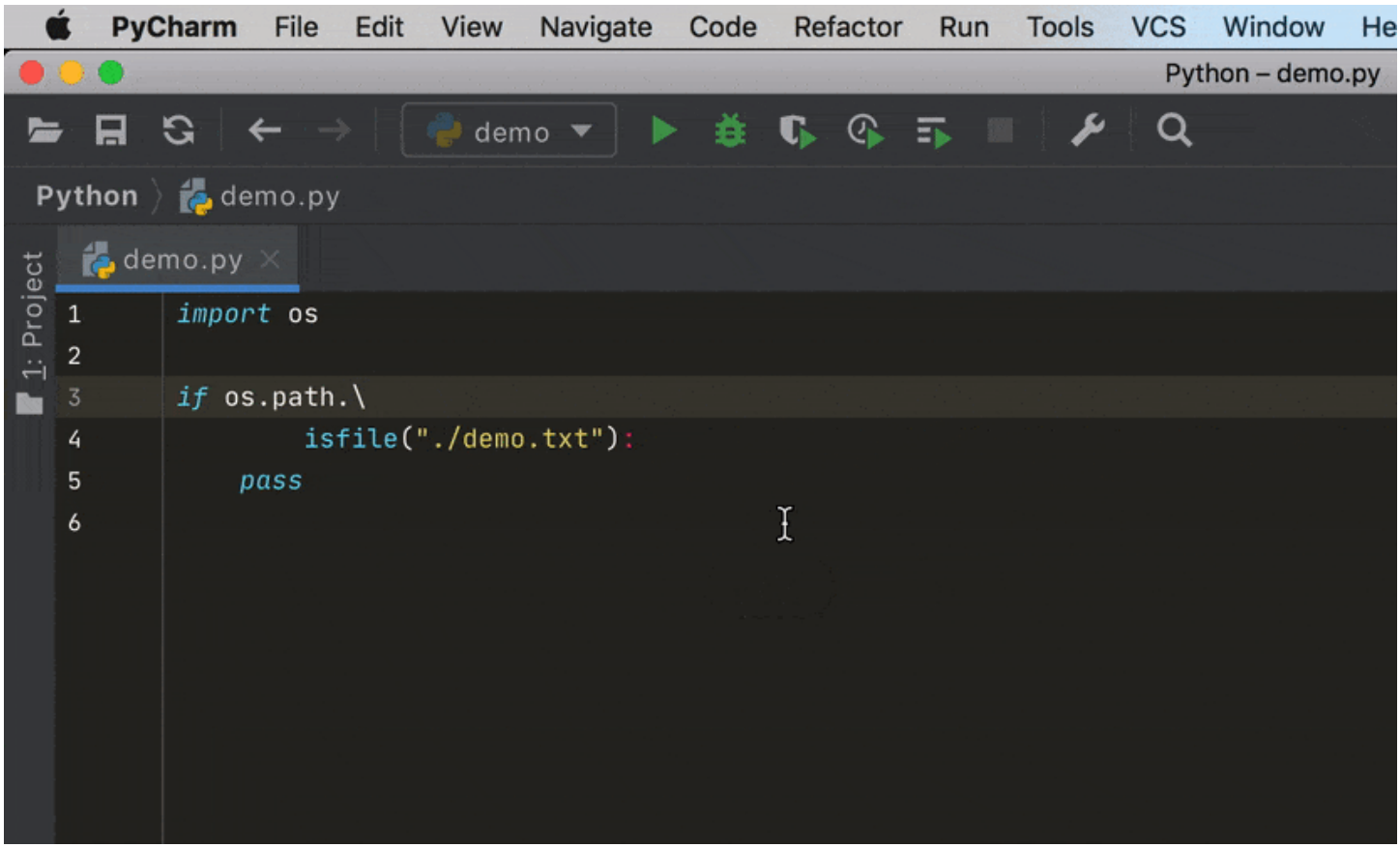
> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

快捷键：⌥ + ↩，也称万能键，显示意向动作和快速修复代码



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

快捷键：^ + ⇧ + J， 智能的将代码拼接成一行



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

快捷键：⌘ + ↩，智能的拆分拼接的行(未实践)

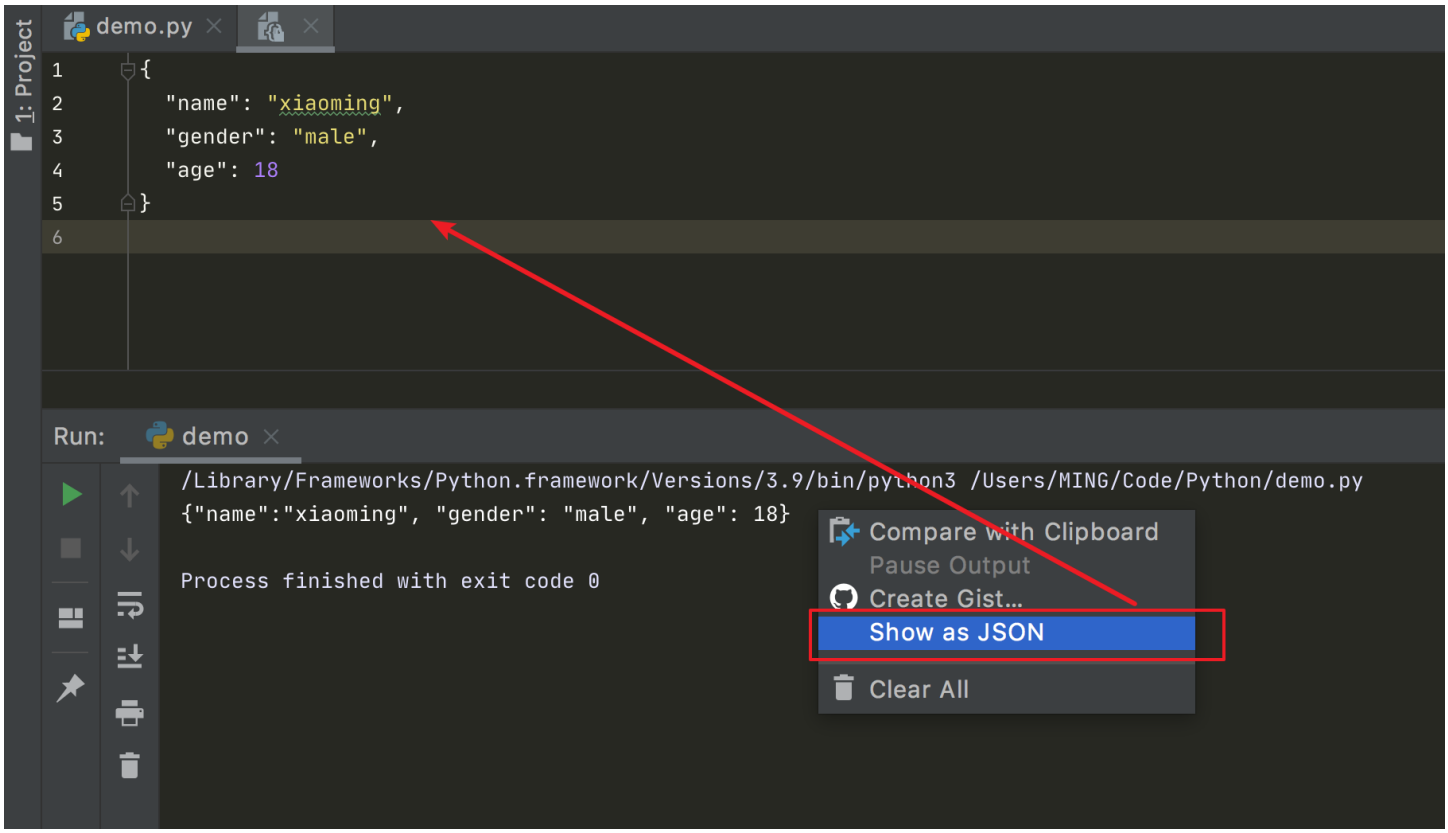
4.11 【高效编辑 11】输出结果美化：Show as JSON

当你使用 PyCharm 运行程序后，如果打印了 JSON 字符串，对于人的肉眼来说是很不友好的。

比如这样一段代码

```
member = '{"name":"xiaoming", "gender": "male", "age": 18}'
print(member)
```

这时候可以在输出窗口点击右键，选择 `Show as JSON`， PyCharm 就会新开一个临时文件显示被格式化过的 JSON 字符串



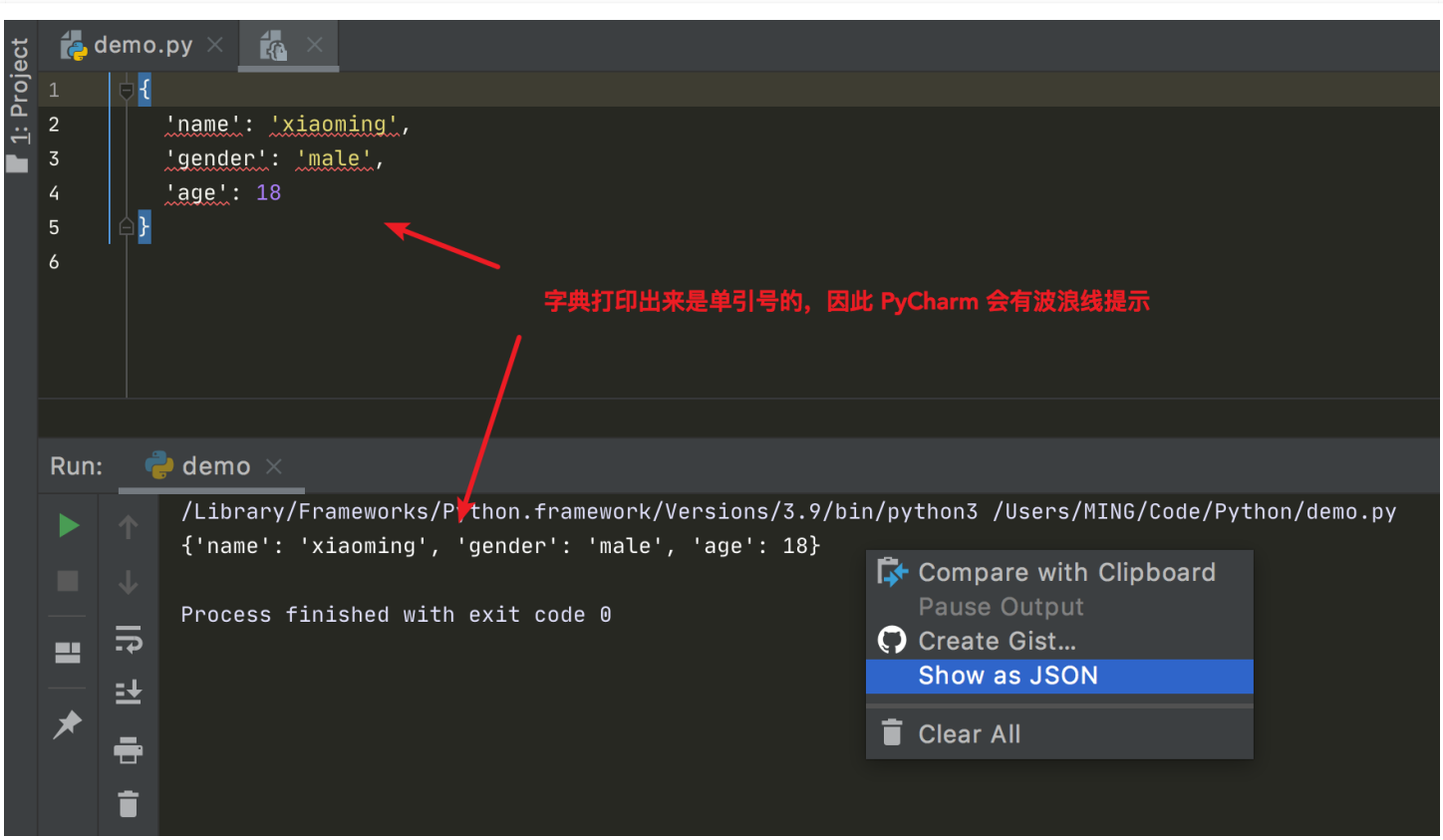
这个功能在较低版本的 PyCharm 中是没有的。

同时，我发现这个功能有两个缺点

1、只能美化打印的 JSON 字符串，在美化打印的字典时，会有一点问题。

比如这样一段代码

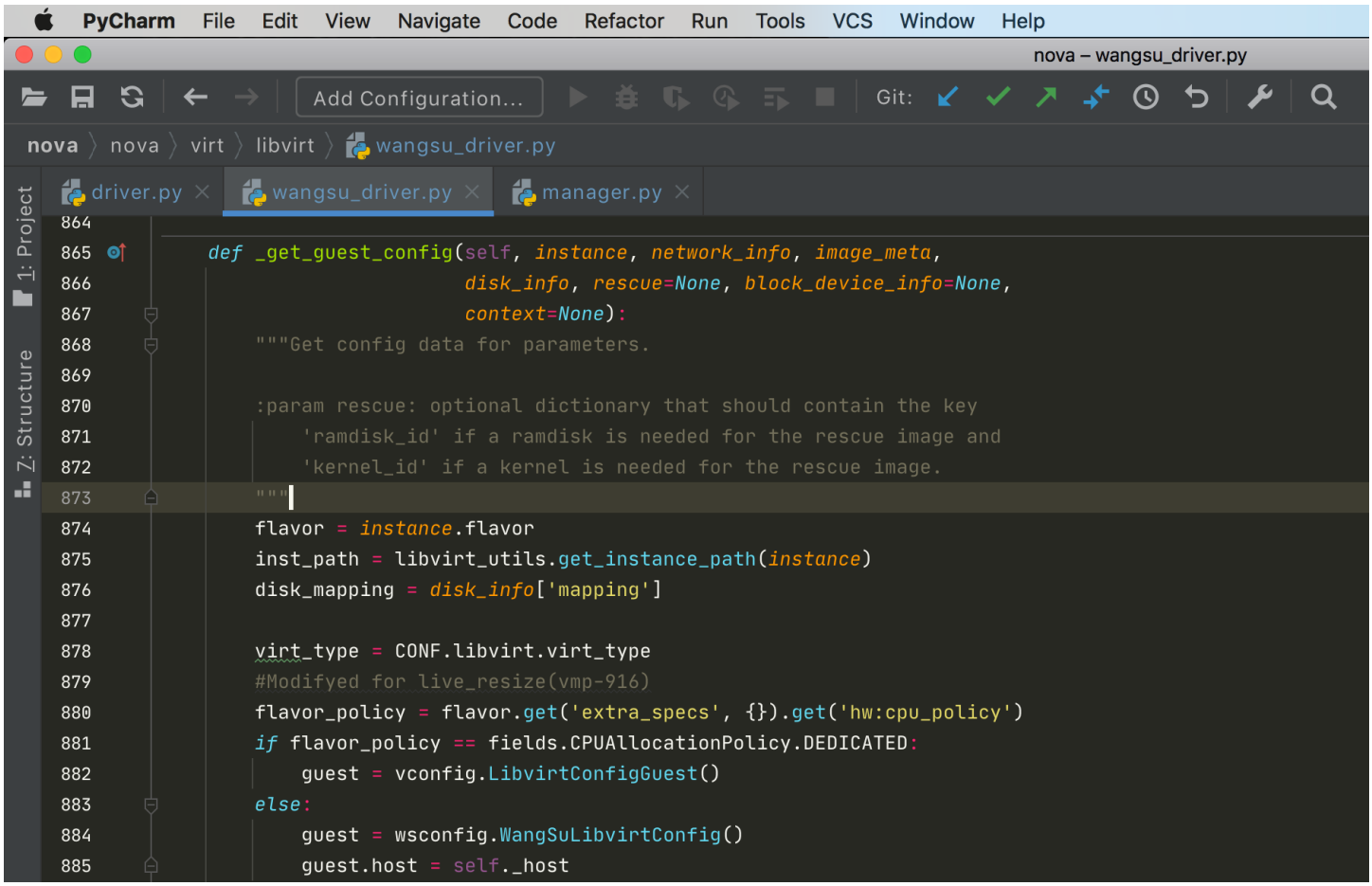
```
member = {"name":"xiaoming", "gender": "male", "age": 18}
print(member)
```



2、打印的 JSON 字符串里不能有 `[]`，不然解析会有问题

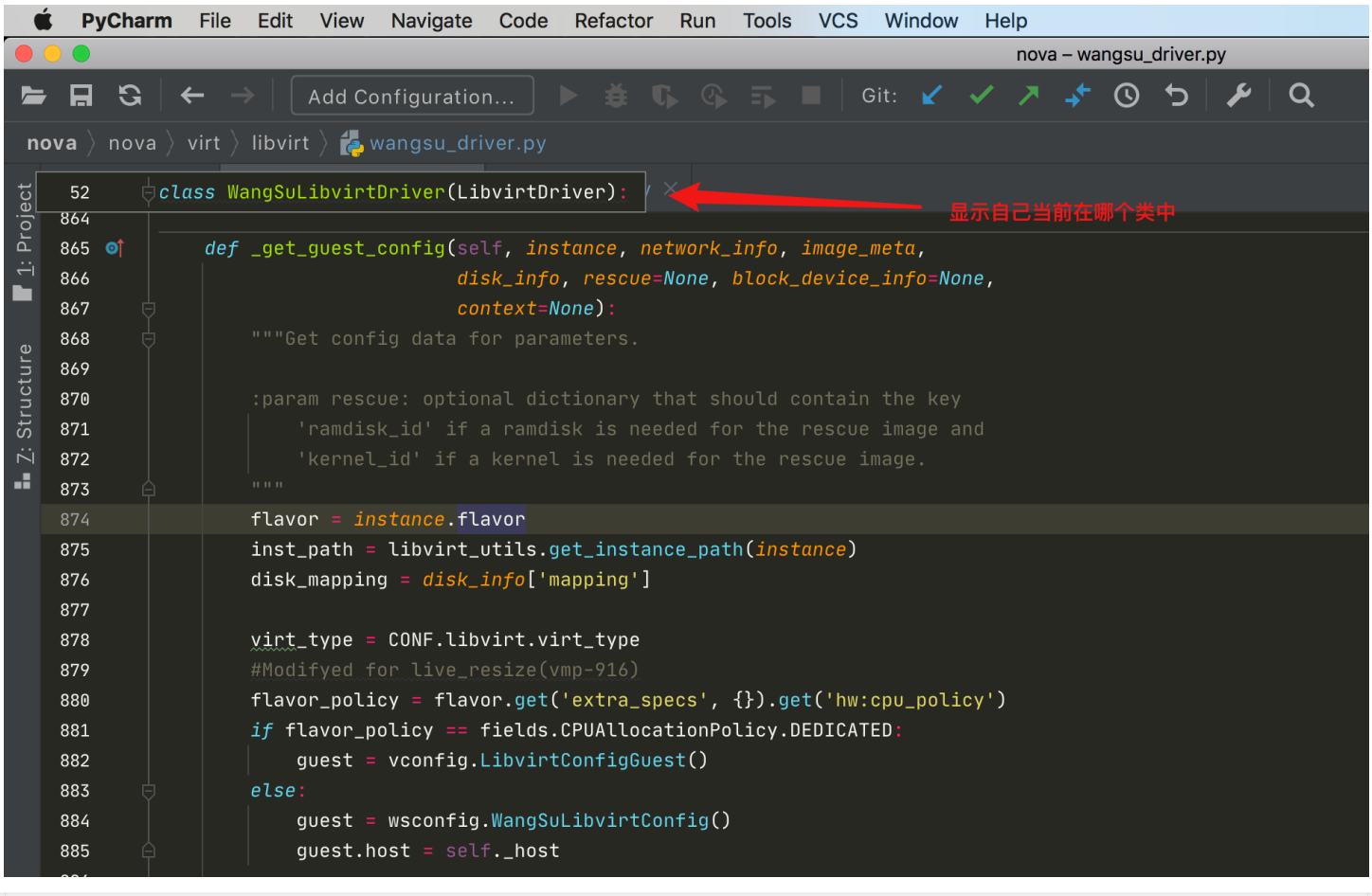
4.12 【高效编辑 12】显示上下文信息

如果一个类的定义写得非常的长，就像下面这样子。



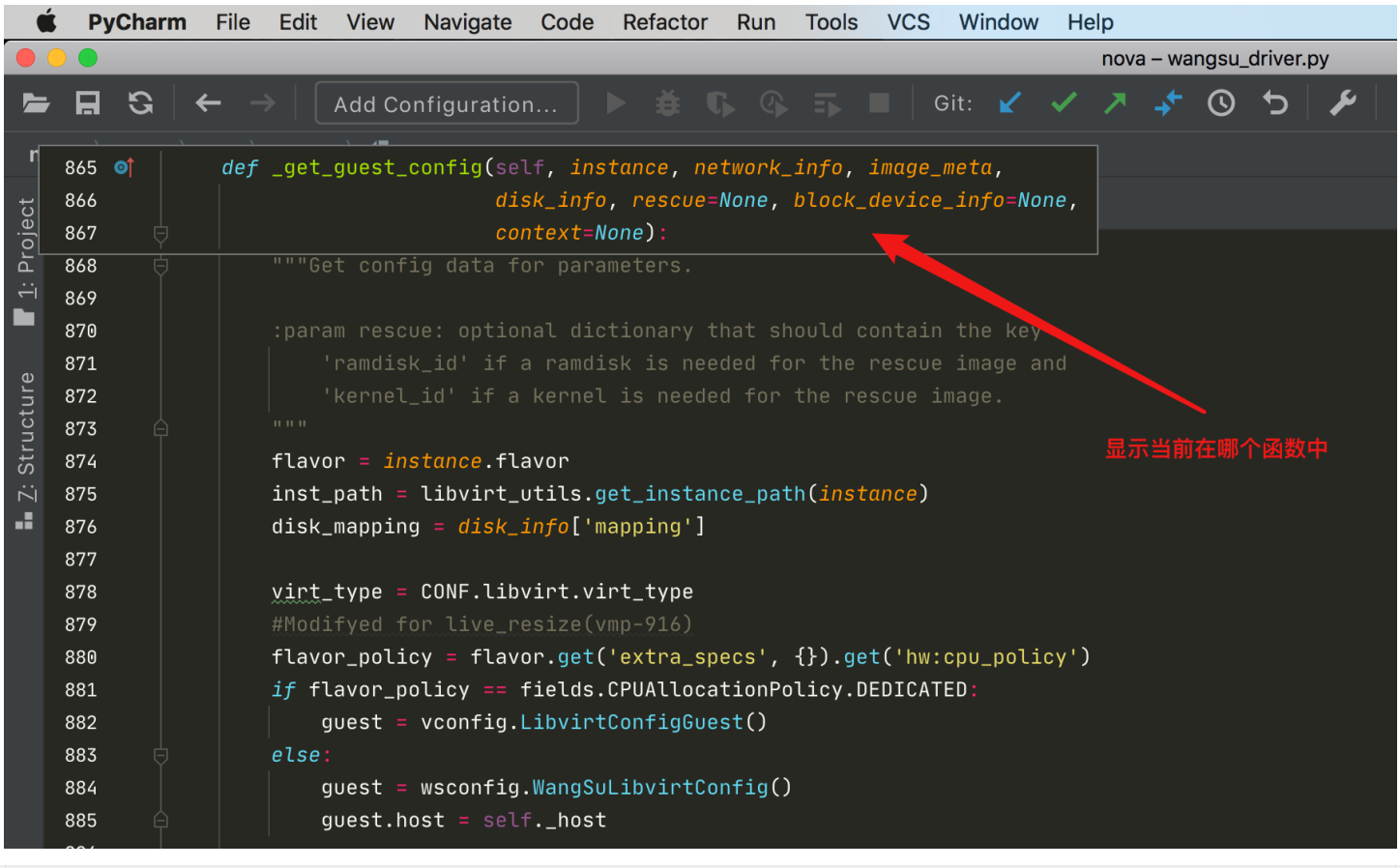
你处在这样一个位置，对框架代码不熟悉的人，根本不知道自己目前处在哪个类中。

PyCharm 提供了一个快捷键：^ + ⇧ + Q，用来显示上下文信息。



如果视野再往下来一点，你连在哪个函数都不知道呢?

按下这组快捷键：^ + ⇧ + Q，就会显示当前处在哪个函数里。



4.13 【高效编辑 13】一键预览模块的文档

Ctrl + 鼠标左键 （Mac 上是：⌘ + 鼠标左键），可以实现函数跳转查看源码，这几乎是每一个 PyCharmer 都会的一个惯用技巧。

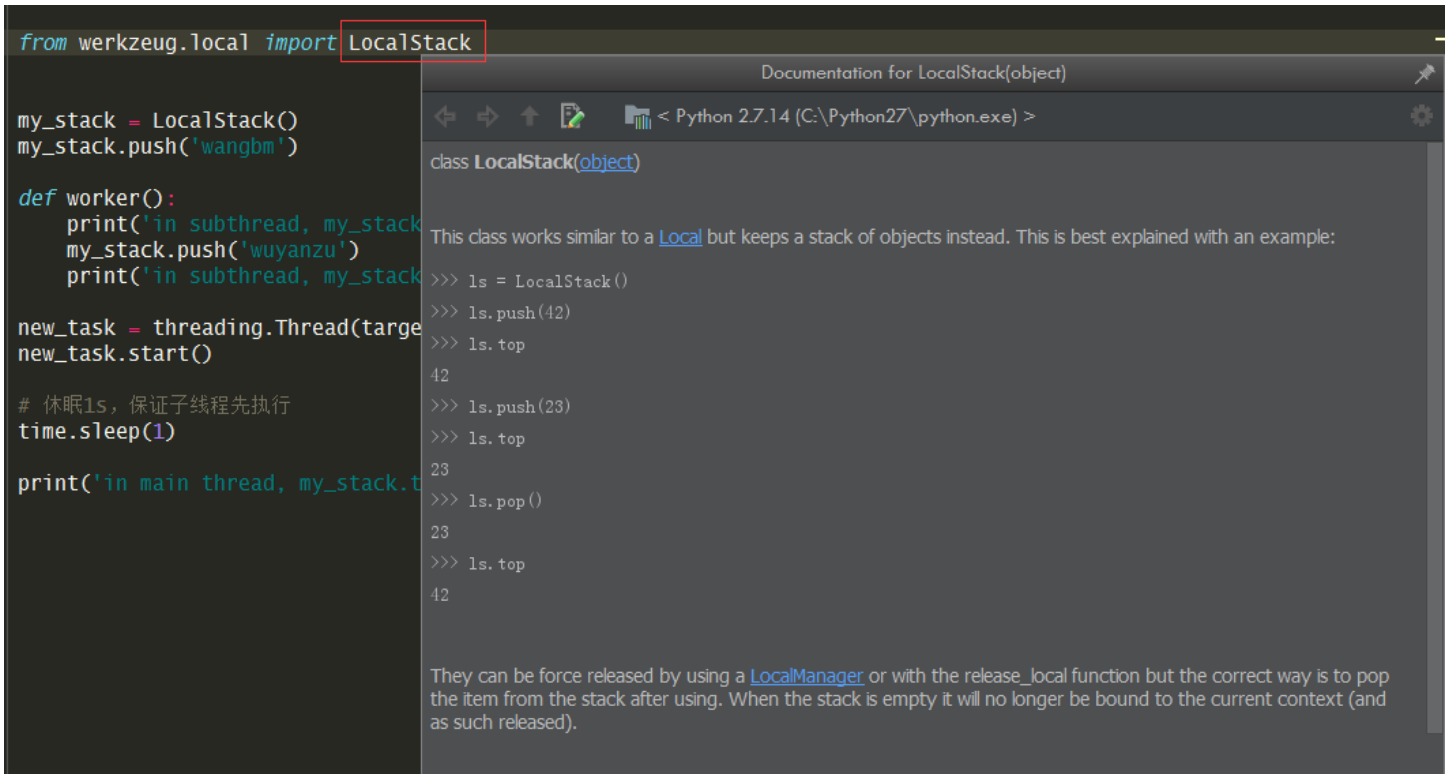
这里再另外介绍两个类似的小技巧，快速 `查看函数文档` 和 `预览源代码` 。

在函数的开头处，使用三个引号 `''' ` 包含的内容，叫做函数文档 （DocString）。

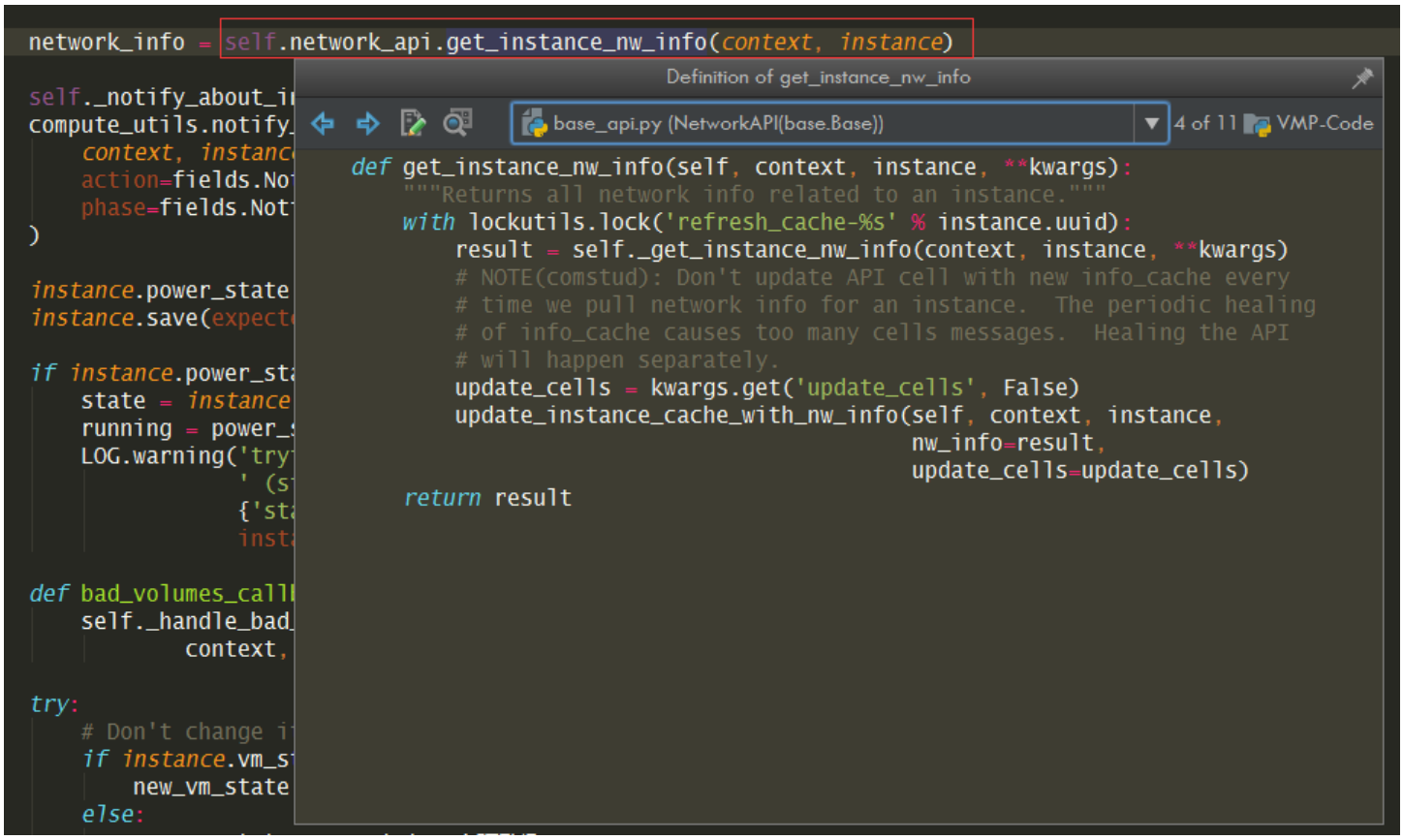
在编写代码时，顺便写好函数的接口文档，是一个很好的编码习惯。它介绍了该函数的参数类型及说明，返回值类型及范例，写得好一点的还会写出 几个简单的 Example Usage 有助于理解使用。在这一点上，Flask 可以说做得相当好。这边随便截一个 Werkzeug 的例子。

```
class LocalStack(object):

    """This class works similar to a :class:`Local` but keeps a stack
    of objects instead.  This is best explained with an example::

        >>> ls = LocalStack()
        >>> ls.push(42)
        >>> ls.top
        42
        >>> ls.push(23)
        >>> ls.top
        23
        >>> ls.pop()
        23
        >>> ls.top
        42

    They can be force released by using a :class:`LocalManager` or with
    the :func:`release_local` function but the correct way is to pop the
    item from the stack after using.  When the stack is empty it will
    no longer be bound to the current context (and as such released).

    By calling the stack without arguments it returns a proxy that resolves to
    the topmost item on the stack.

    .. versionadded:: 0.6.1
    """
```

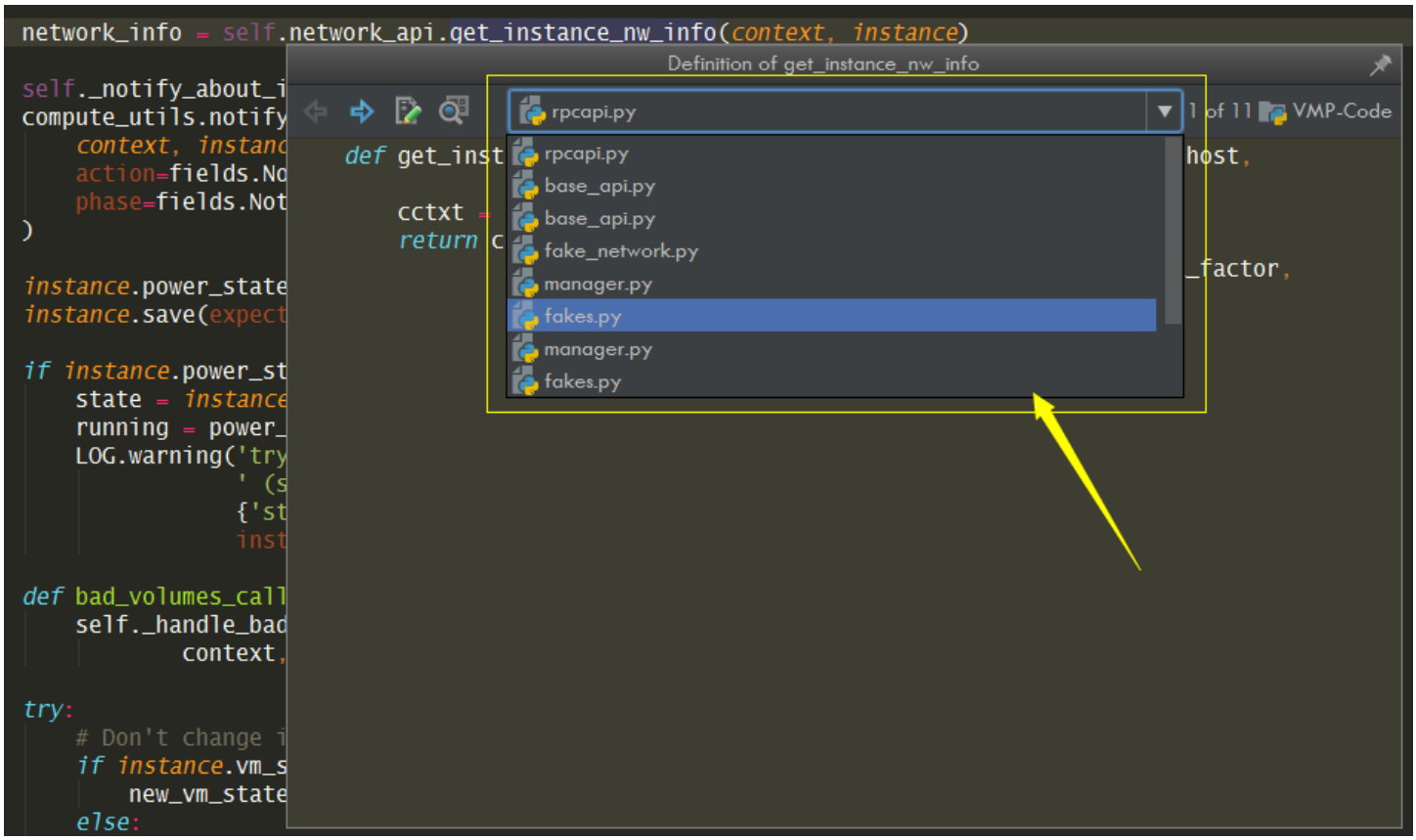
假如我们在使用这个类的时候，忘记了这个用法，可以按住 Ctrl + q（Mac暂时未找到对应快捷键），在当前页面就可以快速预览 LocalStack 的接口文档。



同样的，如果你对这个类或者函数的代码逻辑感兴趣，也可以使用快速预览的方式在当前页面展示源代码。快捷键是：Ctrl + shift + i （Mac：Command + shift + i）。效果如下



如果 PyCharm 检测到的同名函数有多个，可以点这里进行选择查看



这两个快捷键比起使用 Ctrl + 鼠标左键 跳进源代码来说，更加方便，，就像微信小程序一样，用完即焚，不会新产生一个标签页，也不需要来回跳转页面。

作者：王炳明
版本：v1.0
发布时间：2020年08月30日
微信公众号：Python编程时光
联系邮箱：wongbingming@163.com
项目主页：http://pycharm.iswbm.com
Github：https://github.com/iswbm/pycharm-guide

第五章：快捷与效率

5.1 【提高效率 01】复杂操作，录制成宏

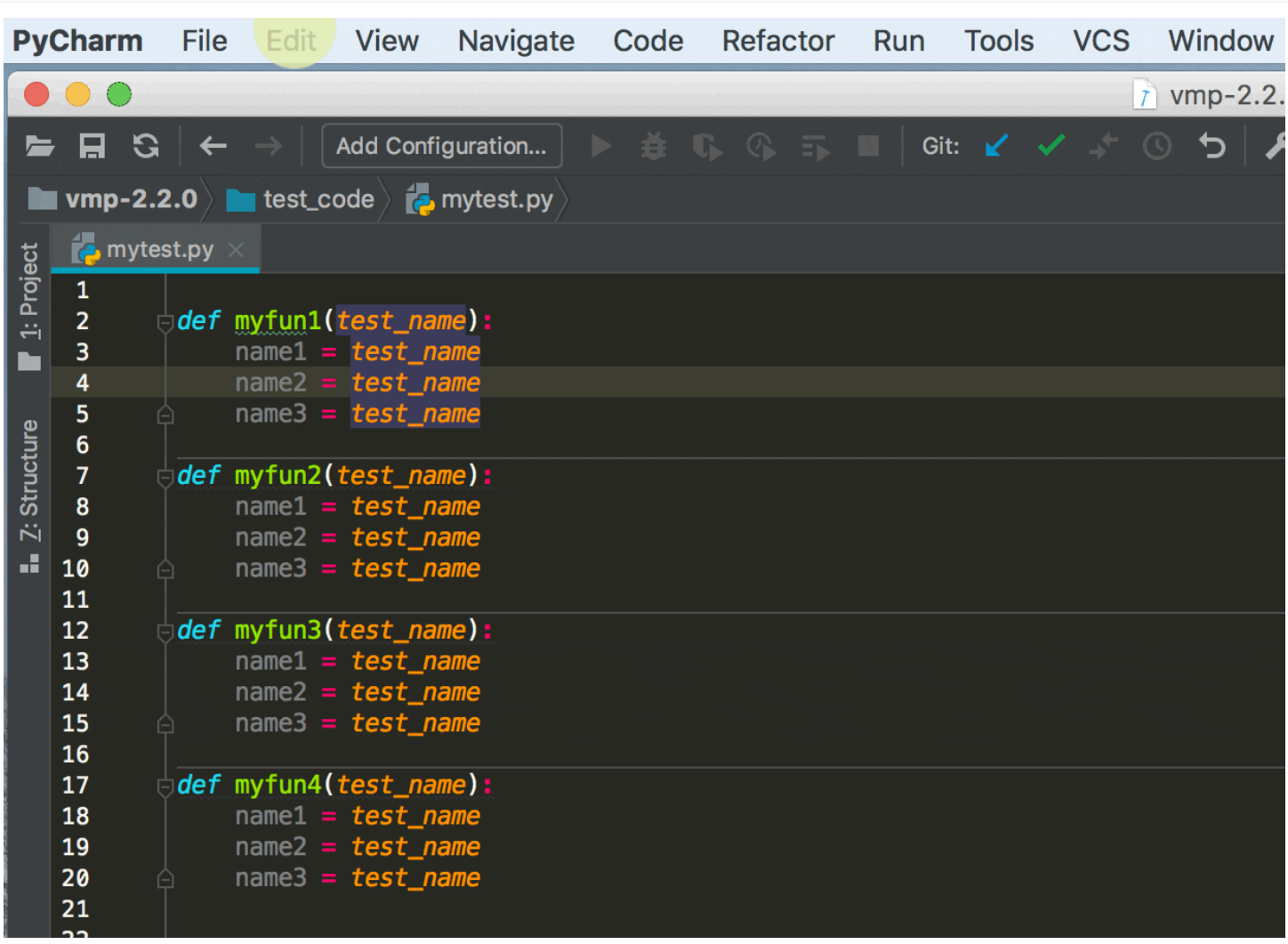
如果你在使用PyCharm 的时候，遇到有一些操作是比较复杂（步骤多），且使用频率特别高。

那可以考虑一下，使用其自带的宏录制工具。

它会将你的一连串操作，录制下来。等你想用的时候，直接调用就行了。

这边，我以录制一个 `删除函数` 的宏为例：先按上面的方法折叠函数，再按 Command+y 删除该行，就删除了该函数。

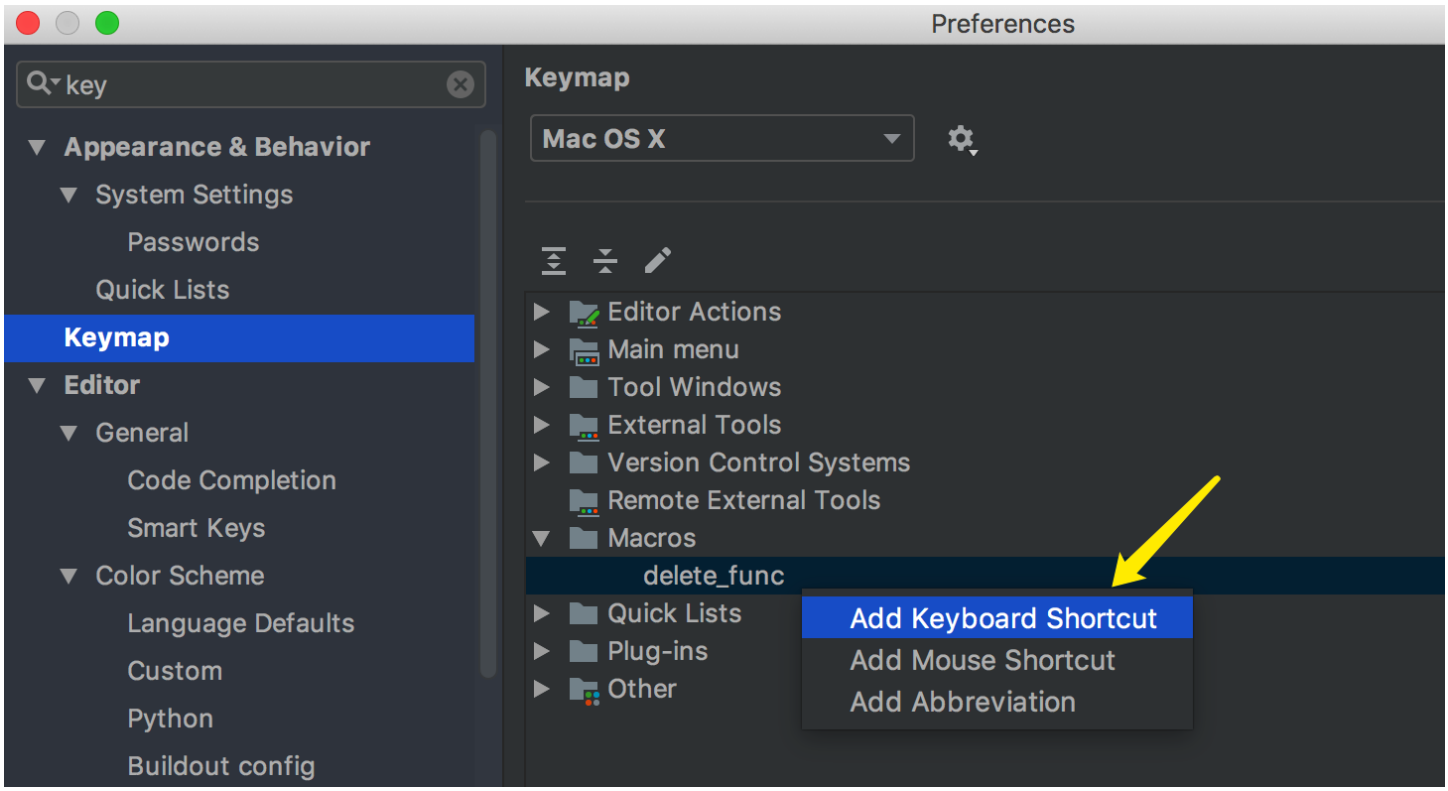
做录制方法如下：



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

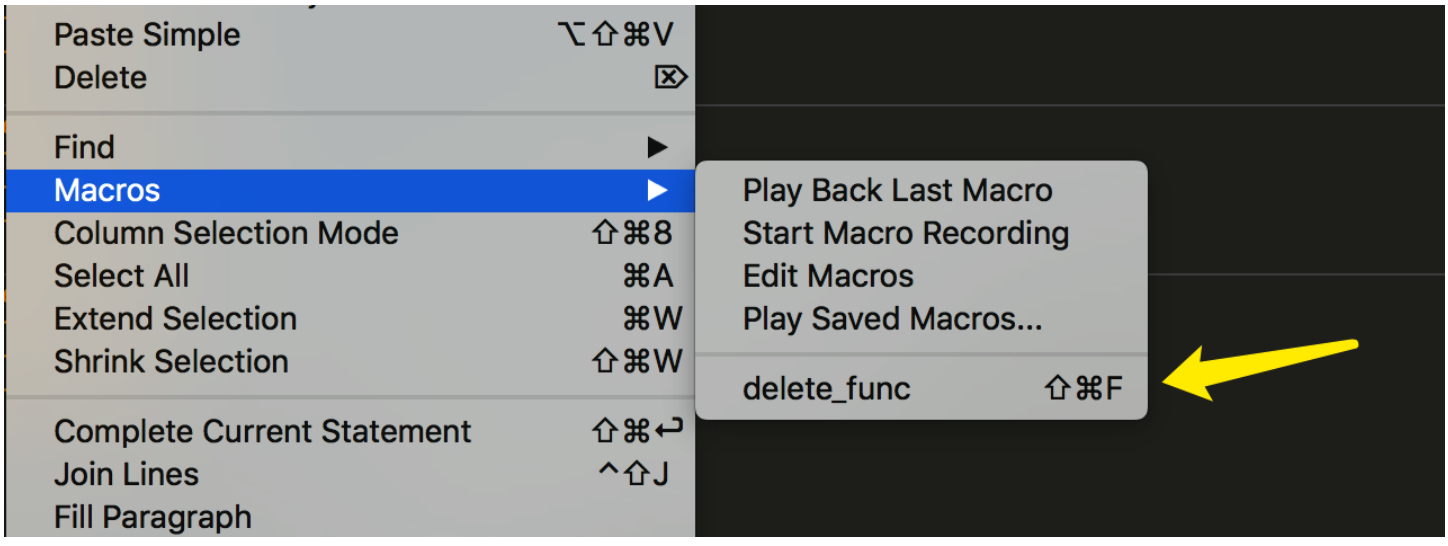
录制好后，你可以先定位到你要删除的函数处，点菜单栏 Edit - Macro 然后选择我们刚刚录制的宏，就可以播放宏了。

这样播放宏显得有点繁琐，个人建议你为这个宏定义一个快捷键，这样会更方便播放宏。



设置快捷键时，注意不要和已有的快捷键冲突。

设置好后，查看 Macro，发现PyCharm已经将这个快捷键绑定给这个宏。



之后你就可以使用这个快捷键删除一个函数（其实这只是删除一个代码块，但是这里只讨论设置方法）。

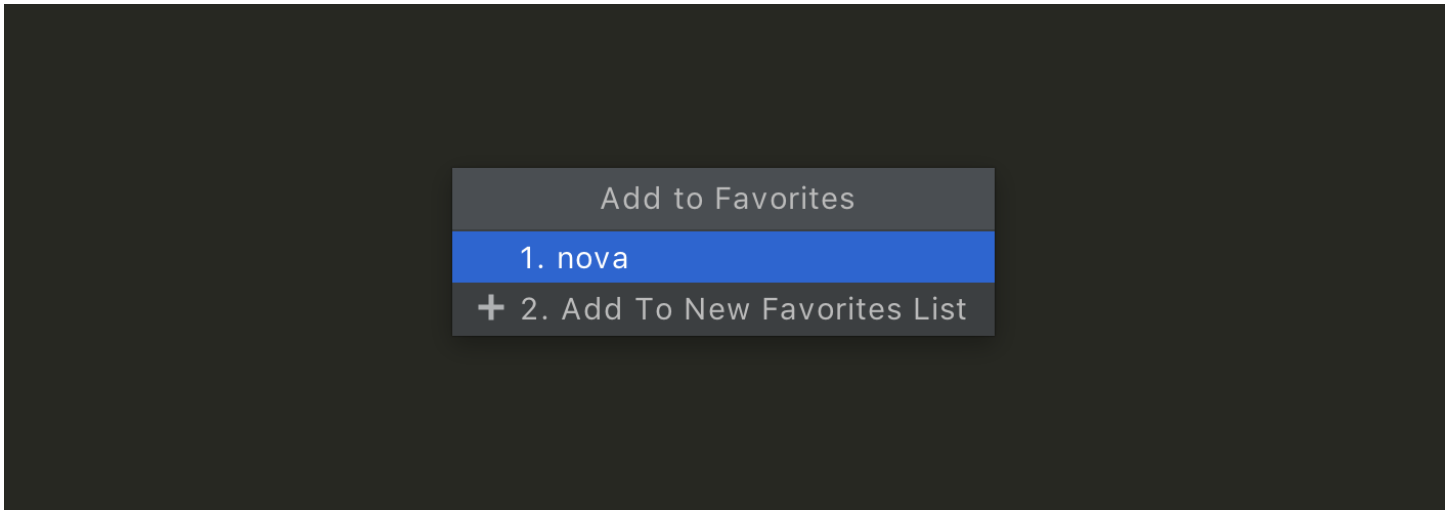
5.2 【提高效率 02】使用收藏夹，收藏关键代码位

在一个项目中，会有许多的比较关键的代码逻辑入口，比如我使用的 OpenStack 框架：

- 创建虚拟机的入口
- 删除虚拟机的入口
- 虚拟机迁移的入口
- 等等...

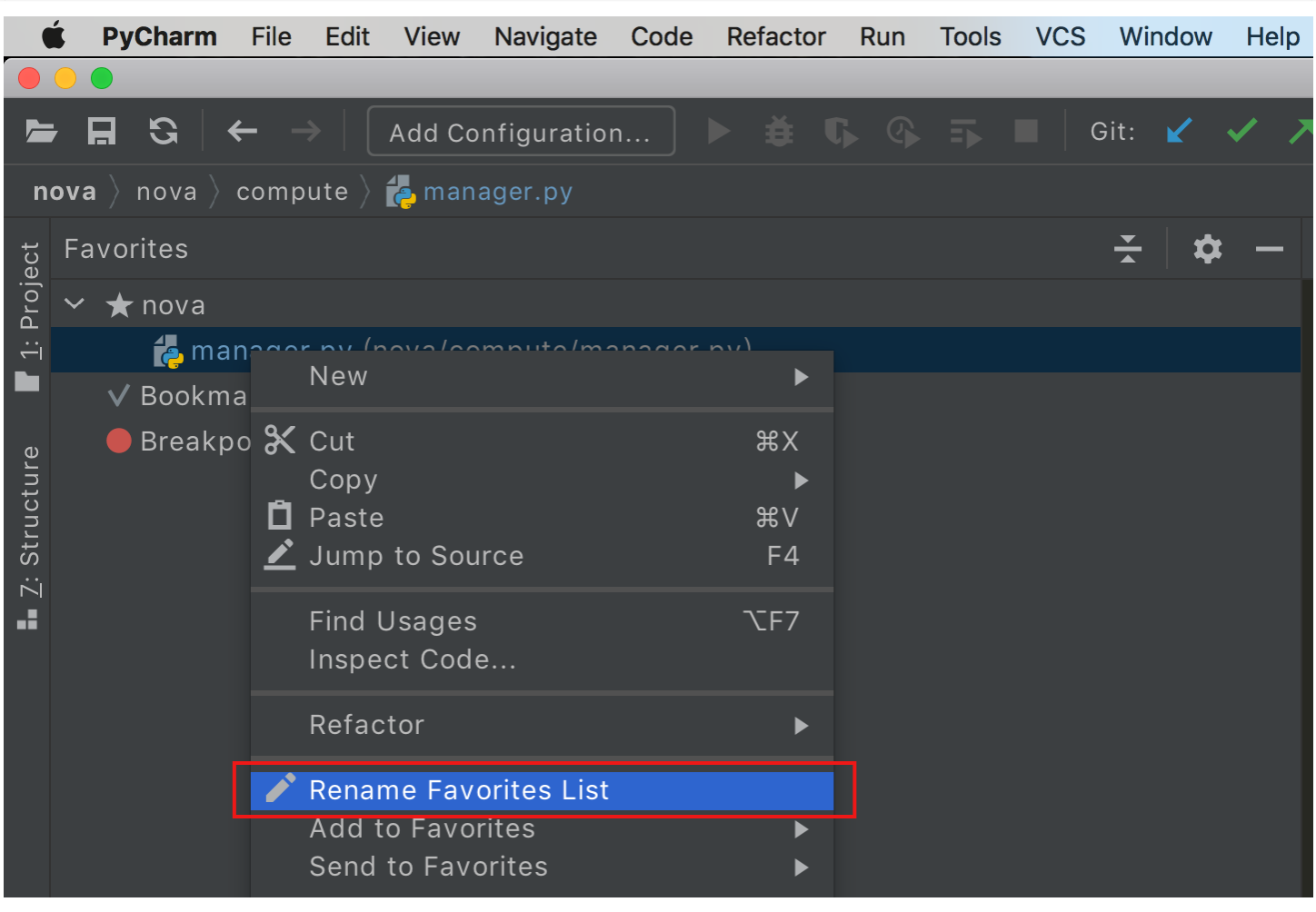
像这种比较关键且打开比较高频的代码，平时就可以收藏起来，等到要用的时候就不需要从项目树中一层一层的点开，再打开文件，再寻找对应的函数。

加入收藏夹的快捷键是：⌥ + ⇧ + F，敲下之后，会让你选择是要加入哪个收藏夹，你也可以选择新建一个收藏夹。



加入收藏夹后，可以再使用快捷键：⌘ + 2，打开收藏夹工具栏，点击相应的位置进行跳转。

如果你想对收藏夹的名字进行修改，可以右键，有一个 Rename Favorites List 的按钮。



个人感觉这个功能会书签弱好多，书签可以对位置进行重重名，而收藏夹不能对收藏的位置进行命名。

想了解 书签 的使用方法的，可以点击这里：在项目中使用书签，快速定位

5.3 【提高效率 03】一套快捷键，精准打开工具栏

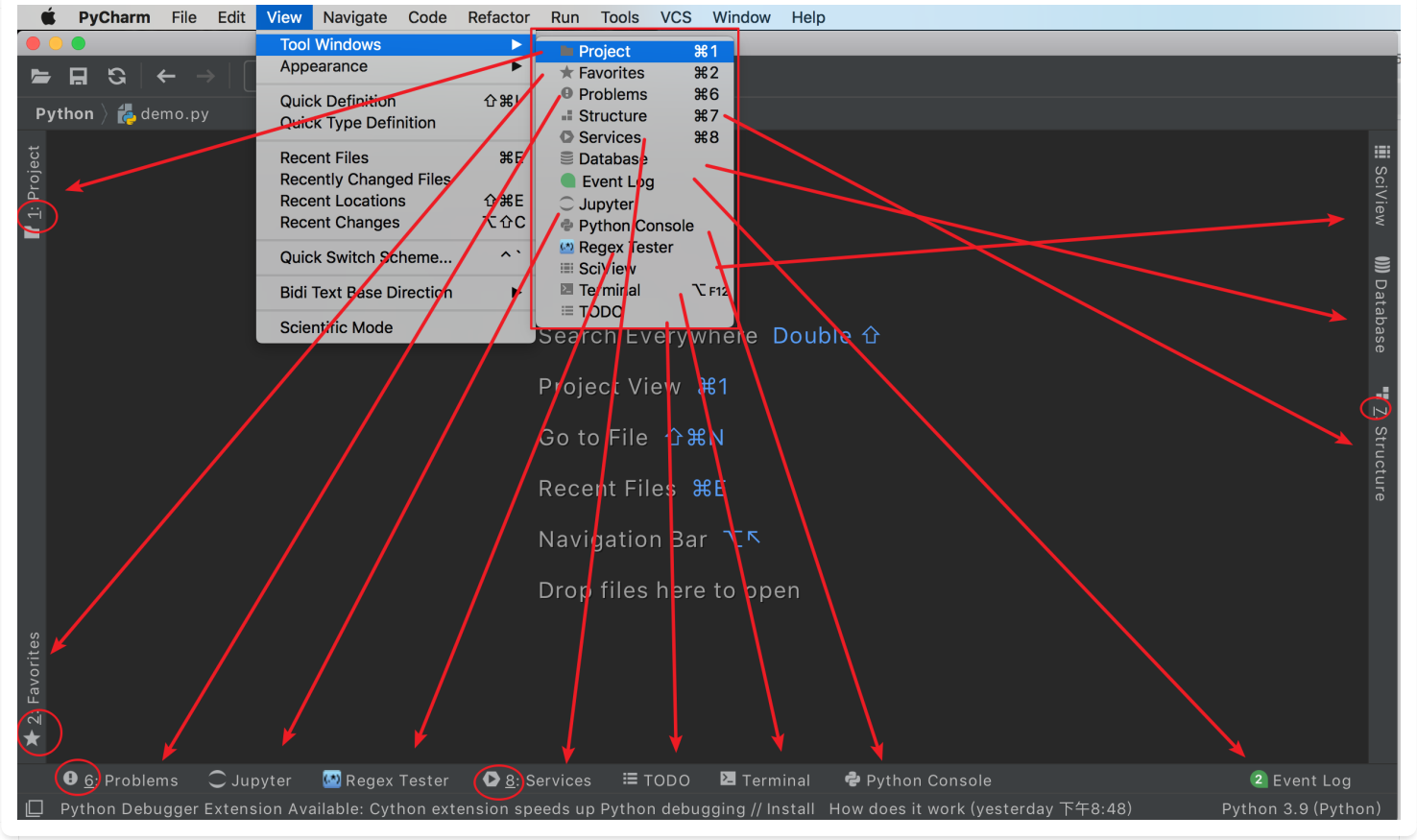
在 PyCharm 的功能强大，每一处的空间都不值得浪费。

在它的四周，我们可以看到一堆的按钮，点开这些按钮，会出来相应的功能窗口。

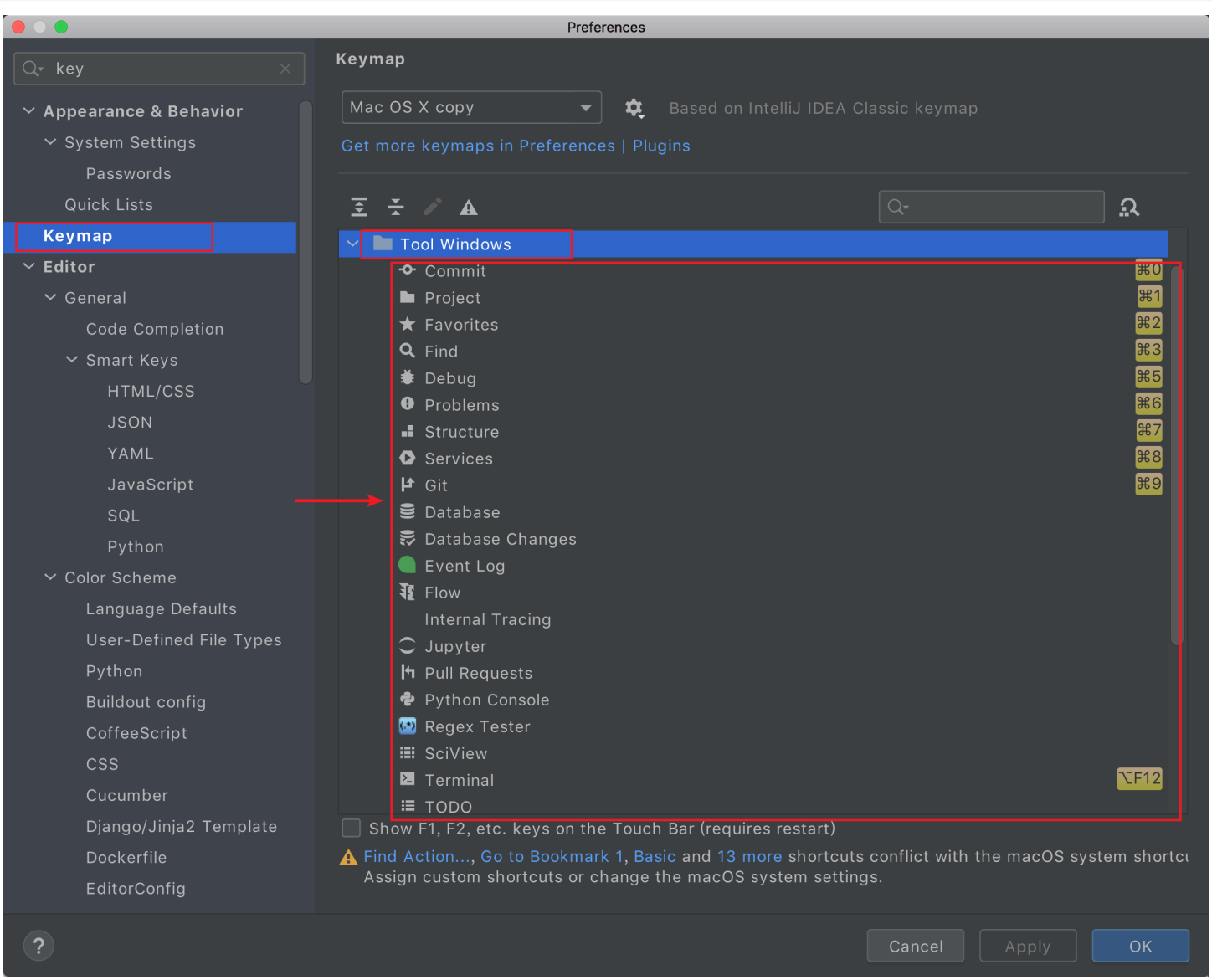
如果使用鼠标去一个一个点击，诺大的屏幕上找准一个位置点击这就是对精神的极大消耗，那有没有办法可以可以用快捷键来控制呢？

点击 `View` -> `Tool Windows` 可以看到当前打开了哪些窗口（对于未打开的窗口并不会展示在这里），同时也可以看到它们的快捷键。

仔细观察，不难发现，其实在按钮上的最前面已经提示了快捷键的序号。所以即使你忘记了也没关系，只要用眼睛瞟一下看一下序号，再在序号前面按个 Command 就行了。



如果你想设置或修改他们的快捷键，可以在 `Keymap` -> `Tool Windows` 中设置



5.4 【提高效率 04】使用模板，快速捕获异常

当你想要对一个代码块进行异常捕获时，你是怎么做的呢?

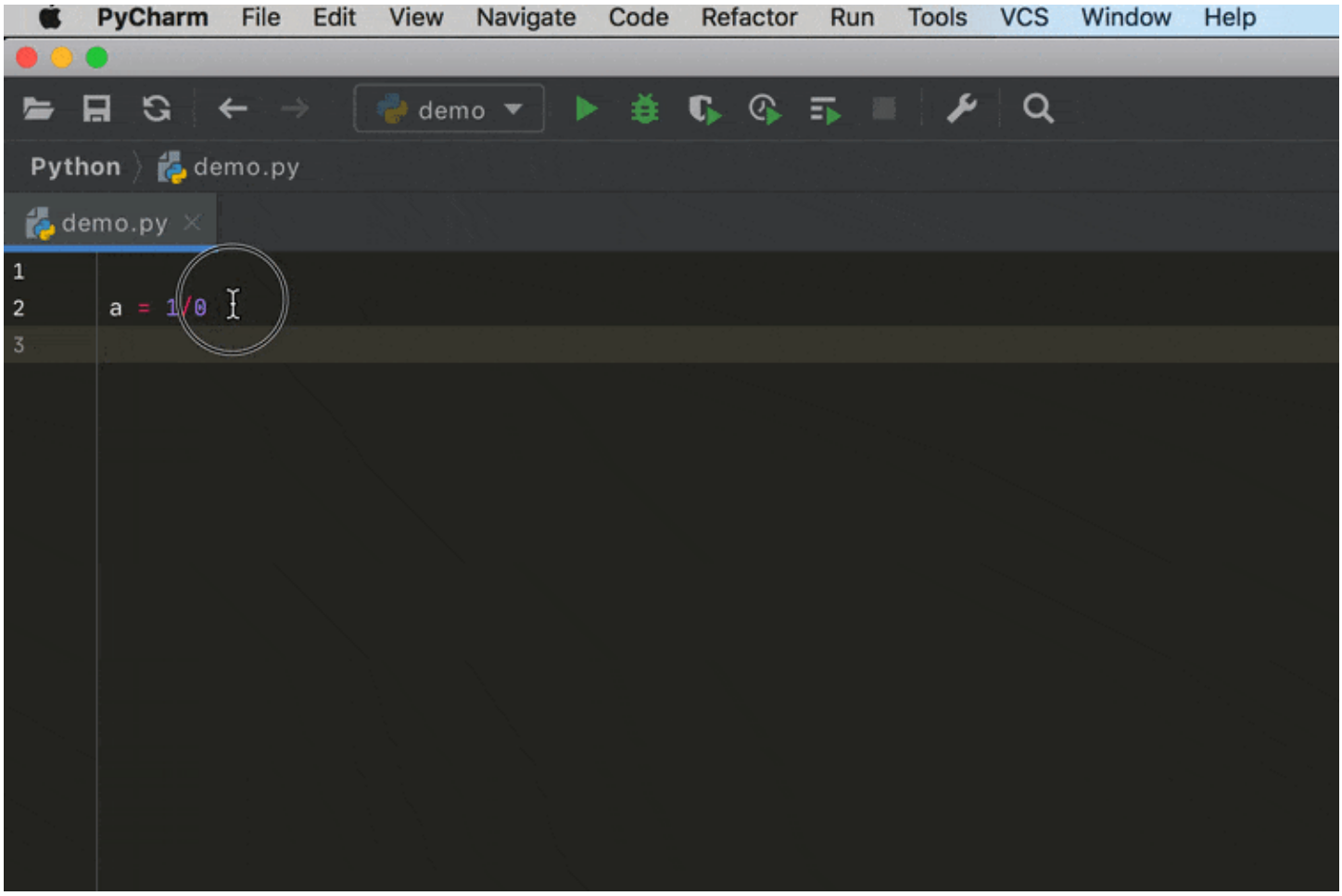
先在上面写个 try，然后对代码块缩进，然后写 excep ..

这种方法，比较生硬，而且效率极差。

这里推荐一种方法，可以使用 `try...except...` 快速包围代码。

效果如下

1. 先选中代码块
2. 按住 ⌘ + ⌥ + T
3. 选择 `try/except` 模板



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

从下拉的选项来看，除了 try/except 和 try/finally 外，还有：

- if
- while
- Comments

但是这些，相对于使用原始的方法，个人感觉并没有更简便，因为该项功能，我更多的是使用捕获异常。

5.5 【提高效率 05】快速输入自定义代码片段

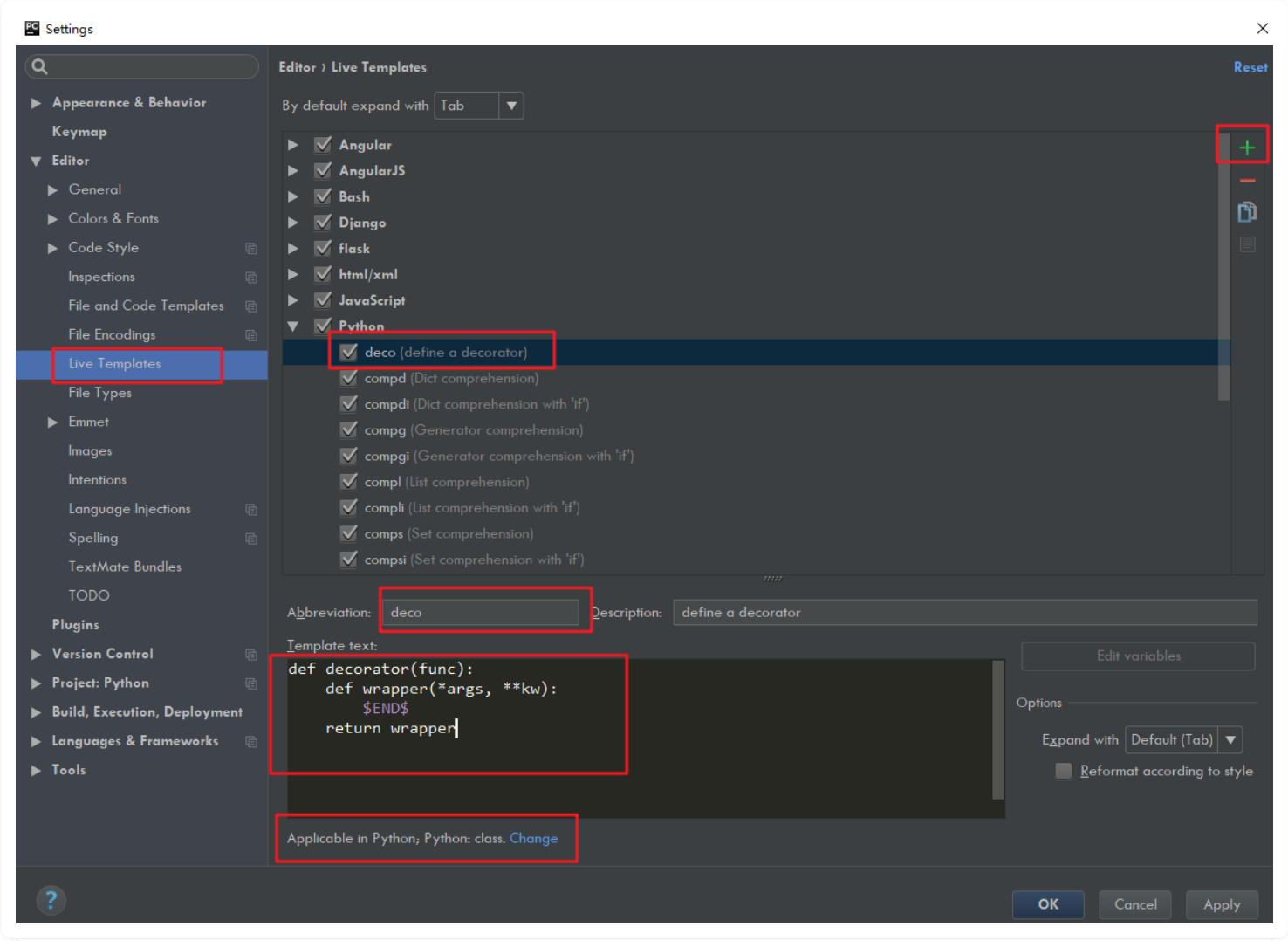
在 PyCharm 中有一个功能叫 Live Template，它可以用来自定义一些常用的代码片段。

比如下面这段，几乎是写 Python 脚本必备的

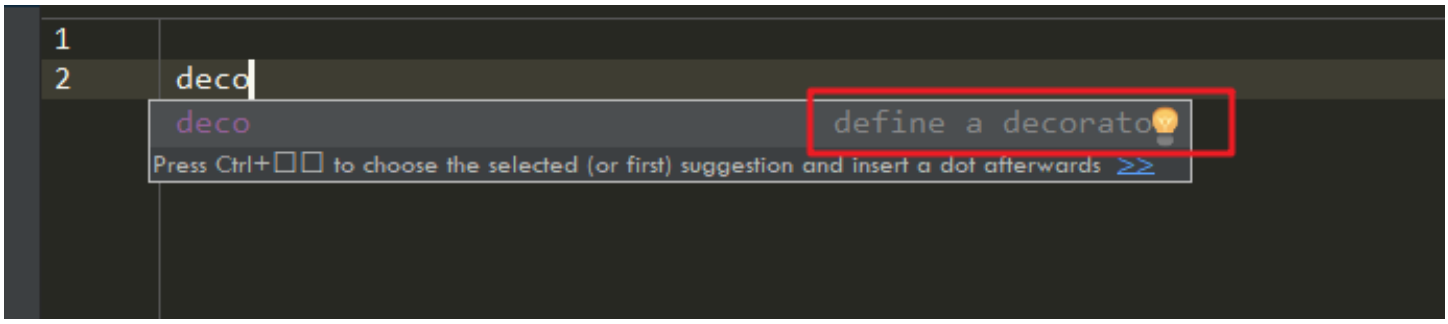
```
if __name__ == '__main__':
```

当你在PyCharm 中编码 python 代码时，只要输入 main， PyCharm 就会在 Live Template 里找到定义过的代码片段，然后只要直接键入回车，就可以生成这段代码。

再比如说，我通常会定义简单的装饰器代码

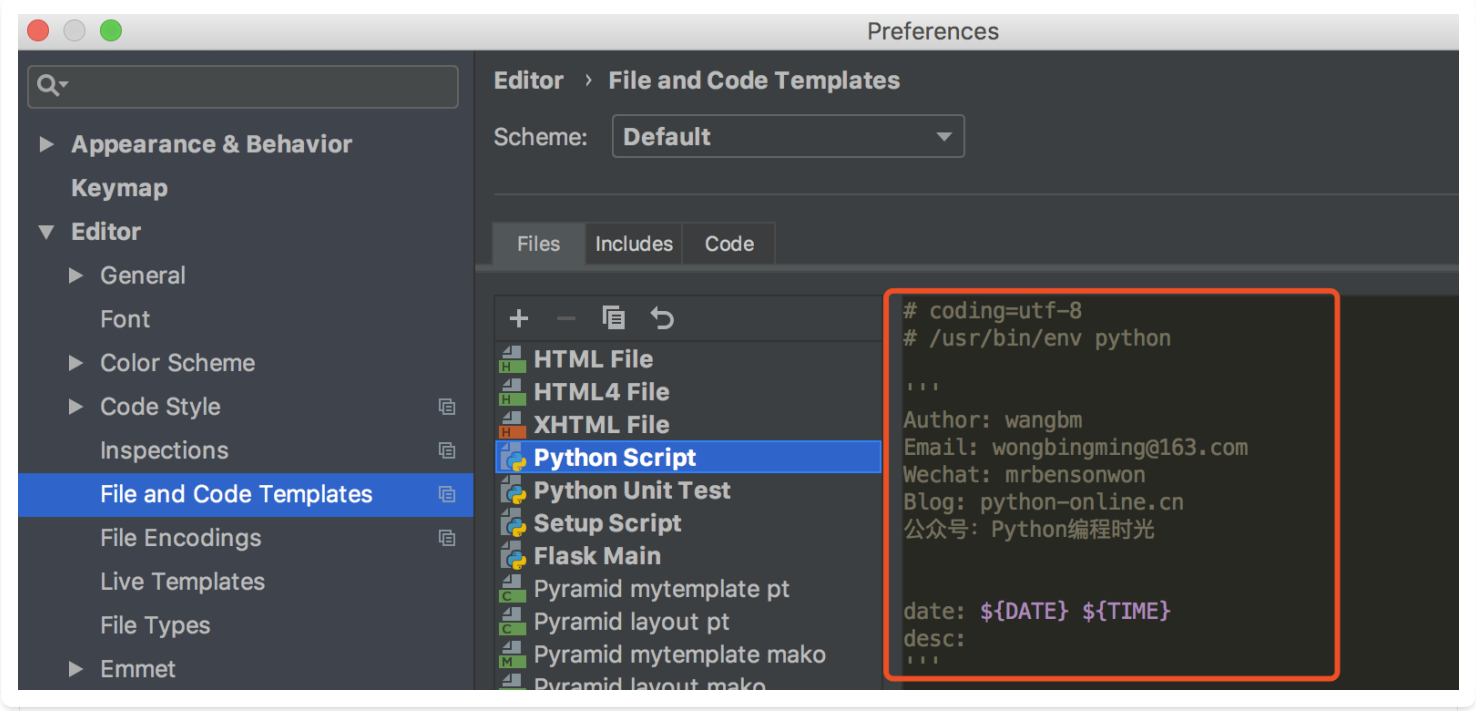


这样当我要定义一个最简单的装饰器时，只要输入 `deco` 再直接敲入回车就行啦。



5.6 【提高效率 06】代码模板，效率编码

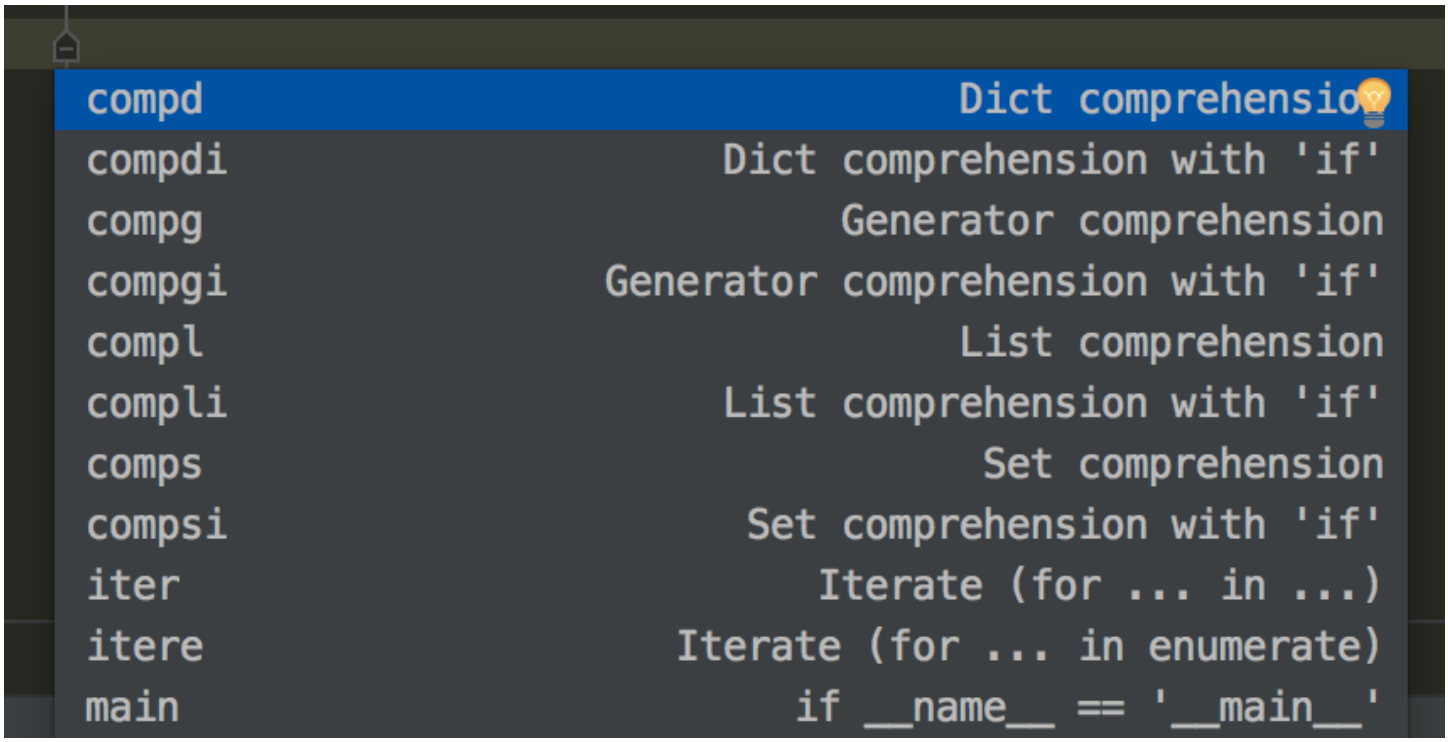
Pycharm 提供的这个代码模板，可以说是相当实用的一个功能了。它可以在你新建一个文件时，按照你预设的模板给你生成一段内容，比如解释器路径，编码方法，作者详细信息等



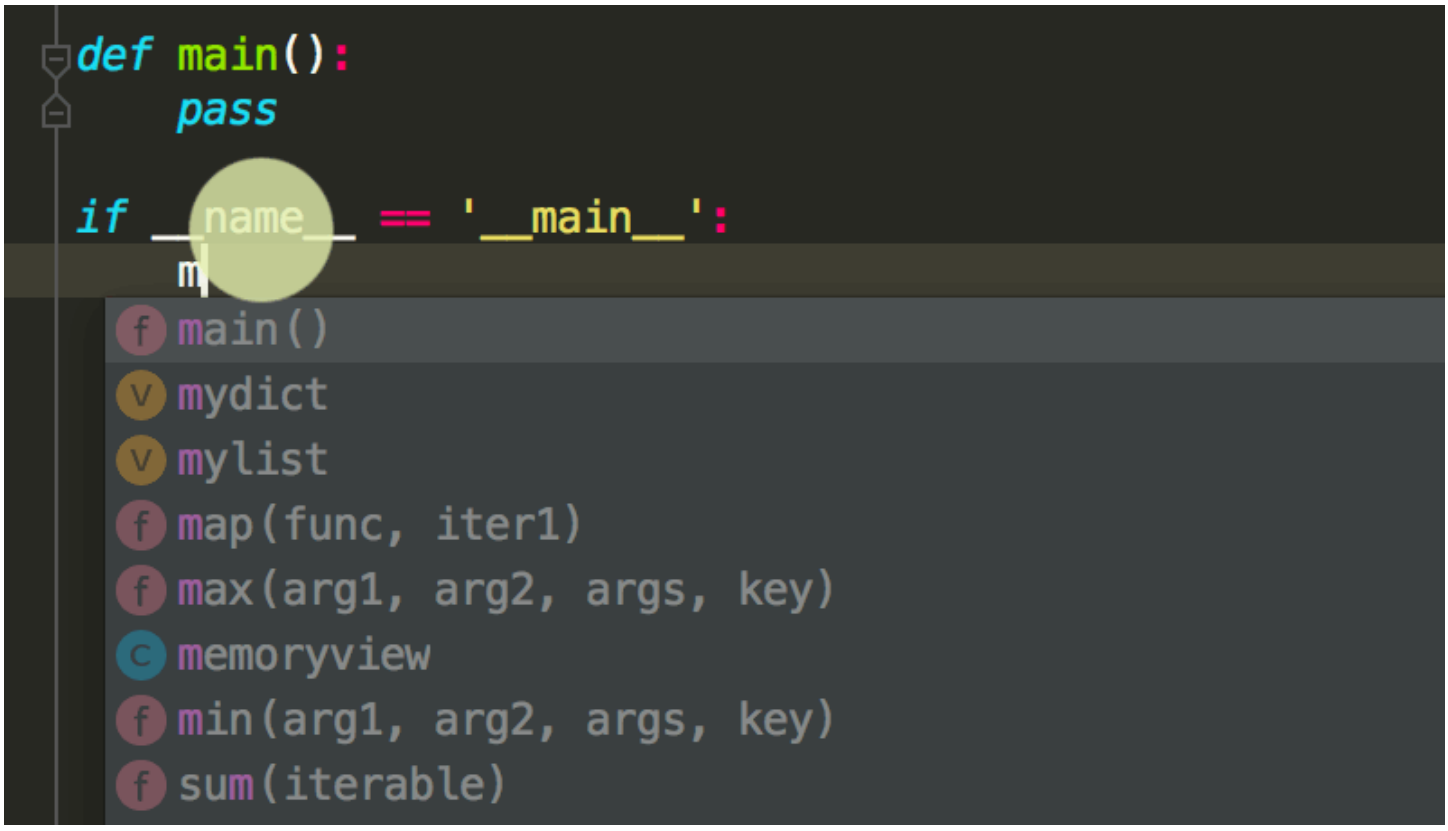
按照上图模板，生成的效果如下。

除了新建文件时可以初始化文件，在开发编写代码时，也同样使用 Pycharm 中自带的实用的代码模板，提高你的编码效率。

当你在键盘中敲入 `Command` + `J` 时，就可以调出一个面板，从下图可以看出里面有许多预设的模板。

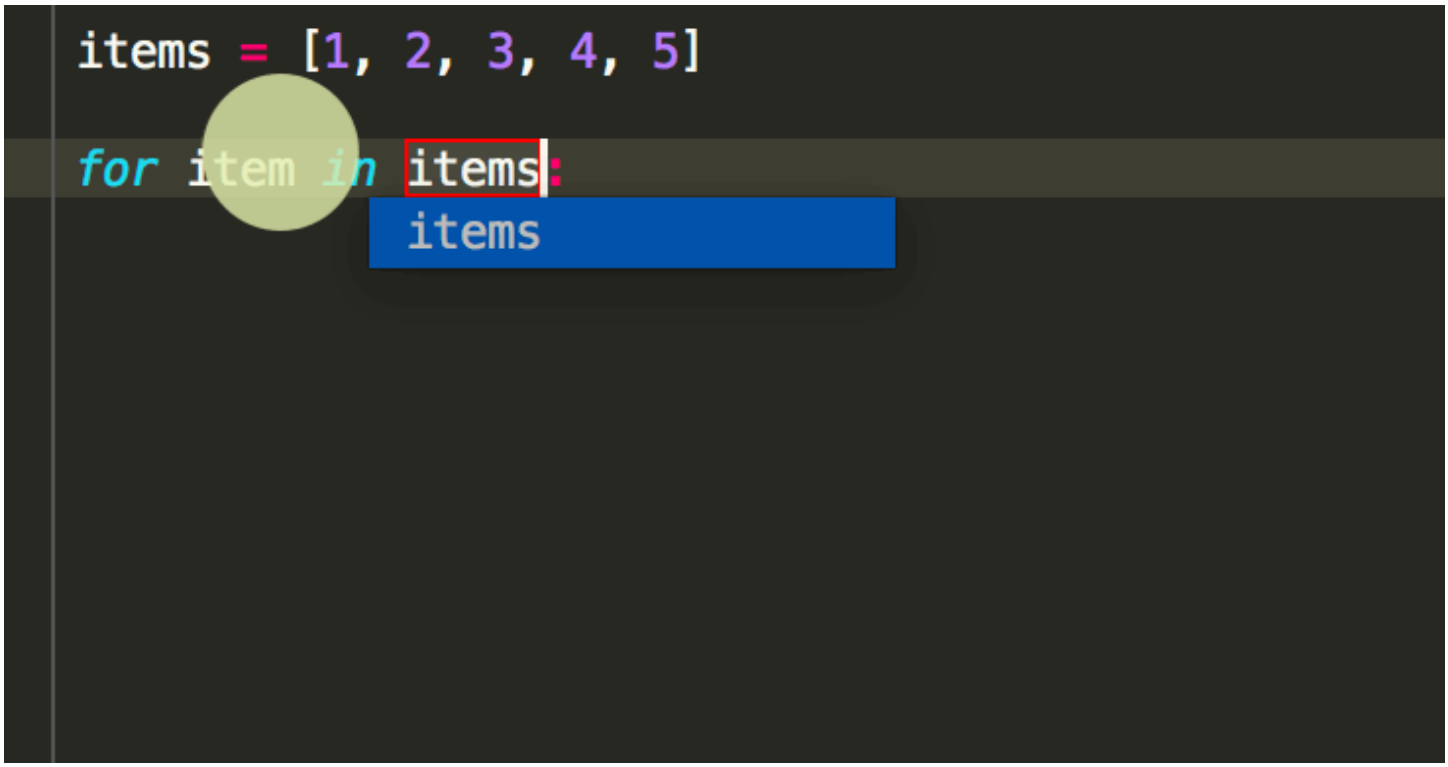


如果我们想选择最后一个 main， 可以继续键入 main，然后就可以直接生成如下这段平时都要手动敲入的代码。



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

这里再举个例子，for 循环 可以这样写。



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

5.7 【提高效率 07】代码封装，一步到位

当一个主函数的代码行数越来越多时，代码的可读性会变得越来越差。通常的做法，是按照功能将代码进行封装成多个函数。

这个过程无非是

1. 在合适的位置定义一个新的函数
2. 将原有的代码拷贝至该函数中
3. 并将原的代码替换成该函数的调用

倘若你的重构的工作量不是很大，完全可以手工来完成这些事。

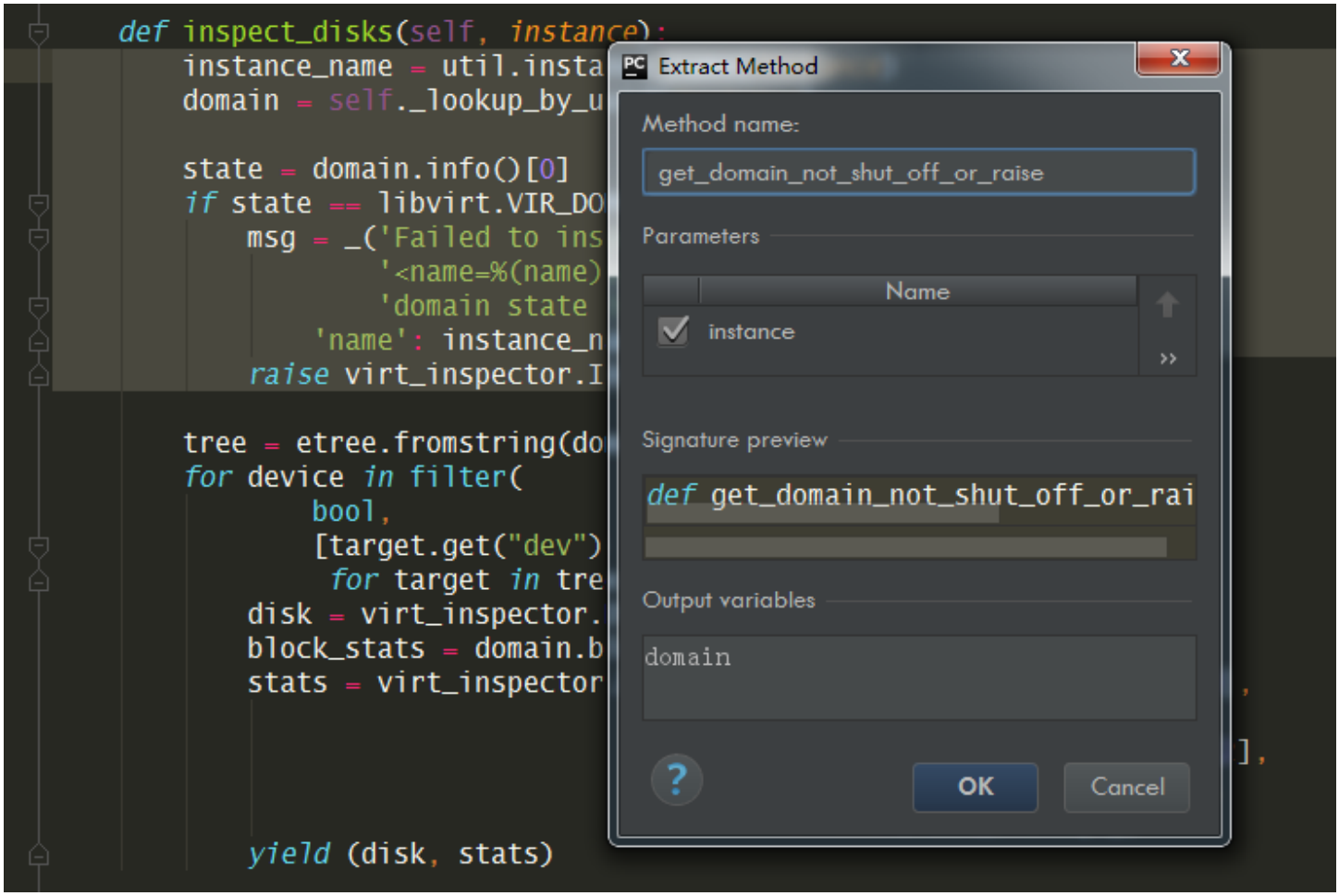
但当你是在重构一个项目代码时，你可能需要一个更高效的封装技巧。

在 PyCharm 中，提供了多种形式的代码重构快捷方法，大家比较常见的可能是重构变量名：shift+F6，而今天要给大家介绍的是方法的重构，也即代码快速封装的技巧。

假如，我现在有如下一段代码，红框标出的代码放在主函数中，有些不太合适，况且这段代码不能让人一眼就看出它是在做什么事情。如何将其进行封装，对我们理清整个主程序的逻辑会有帮助。

```
    def inspect_disks(self, instance):
        instance_name = util.instance_name(instance)
        domain = self._lookup_by_uuid(instance)

        state = domain.info()[0]
        if state == libvirt.VIR_DOMAIN_SHUTOFF:
            msg = _('Failed to inspect data of instance '
                    '<name=%(name)s, id=%(id)s>, '
                    'domain state is SHUTOFF.') % {
                'name': instance_name, 'id': instance.id}
            raise virt_inspector.InstanceShutOffException(msg)

        tree = etree.fromstring(domain.XMLDesc(0))
        for device in filter(
                bool,
                [target.get("dev")
                 for target in tree.findall('devices/disk/target')]):
            disk = virt_inspector.Disk(device=device)
            block_stats = domain.blockStats(device)
            stats = virt_inspector.DiskStats(read_requests=block_stats[0],
                                             read_bytes=block_stats[1],
                                             write_requests=block_stats[2],
                                             write_bytes=block_stats[3],
                                             errors=block_stats[4])
            yield (disk, stats)
```

选中你要封装的代码，然后按住 `Ctrl`+`Alt`+`M` 后，会跳出如下界面，根据自己的需要，修改函数名，选择参数和返回值



一切就绪点击 `OK`，PyCharm 会自动在合适的位置为你定义一个函数名，并将你选中的代码放到里面，其中参数名和返回值也都是按照你的要求，效果如下：

```
    def inspect_disks(self, instance):
        domain = self.get_domain_not_shut_off_or_raise(instance)

        tree = etree.fromstring(domain.XMLDesc(0))
        for device in filter(
                bool,
                [target.get("dev")
                 for target in tree.findall('devices/disk/target')]):
            disk = virt_inspector.Disk(device=device)
            block_stats = domain.blockStats(device)
            stats = virt_inspector.DiskStats(read_requests=block_stats[0],
                                             read_bytes=block_stats[1],
                                             write_requests=block_stats[2],
                                             write_bytes=block_stats[3],
                                             errors=block_stats[4])
            yield (disk, stats)

    def get_domain_not_shut_off_or_raise(self, instance):
        instance_name = util.instance_name(instance)
        domain = self._lookup_by_uuid(instance)
        state = domain.info()[0]
        if state == libvirt.VIR_DOMAIN_SHUTOFF:
            msg = _('Failed to inspect data of instance '
                    '<name=%(name)s, id=%(id)s>, '
                    'domain state is SHUTOFF.') % {
                     'name': instance_name, 'id': instance.id}
            raise virt_inspector.InstanceShutOffException(msg)
        return domain
```


第六章：搜索与导航

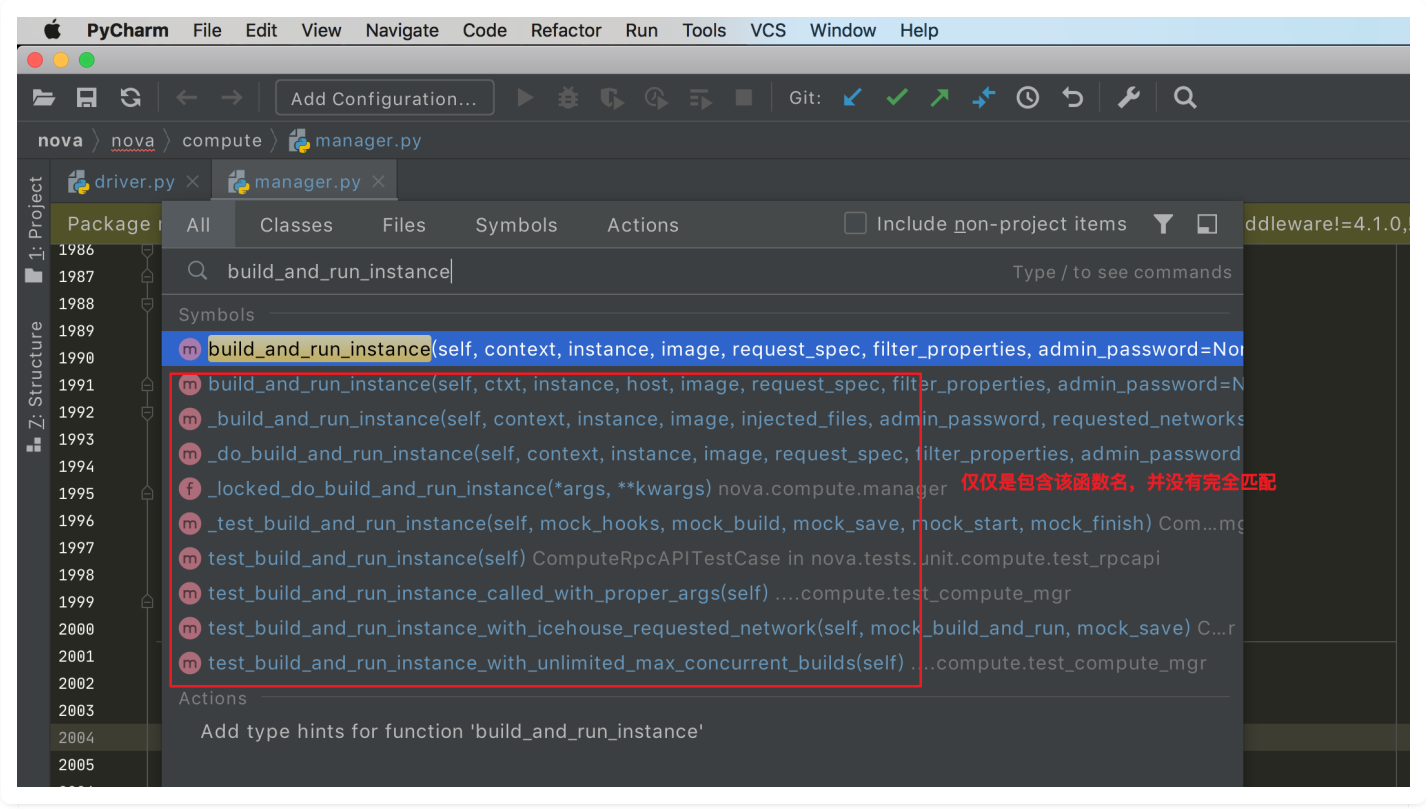
6.1 【搜索技巧 01】精准搜索函数在哪些地方被调用

在项目中搜索用该方法的地方(Find Usage)

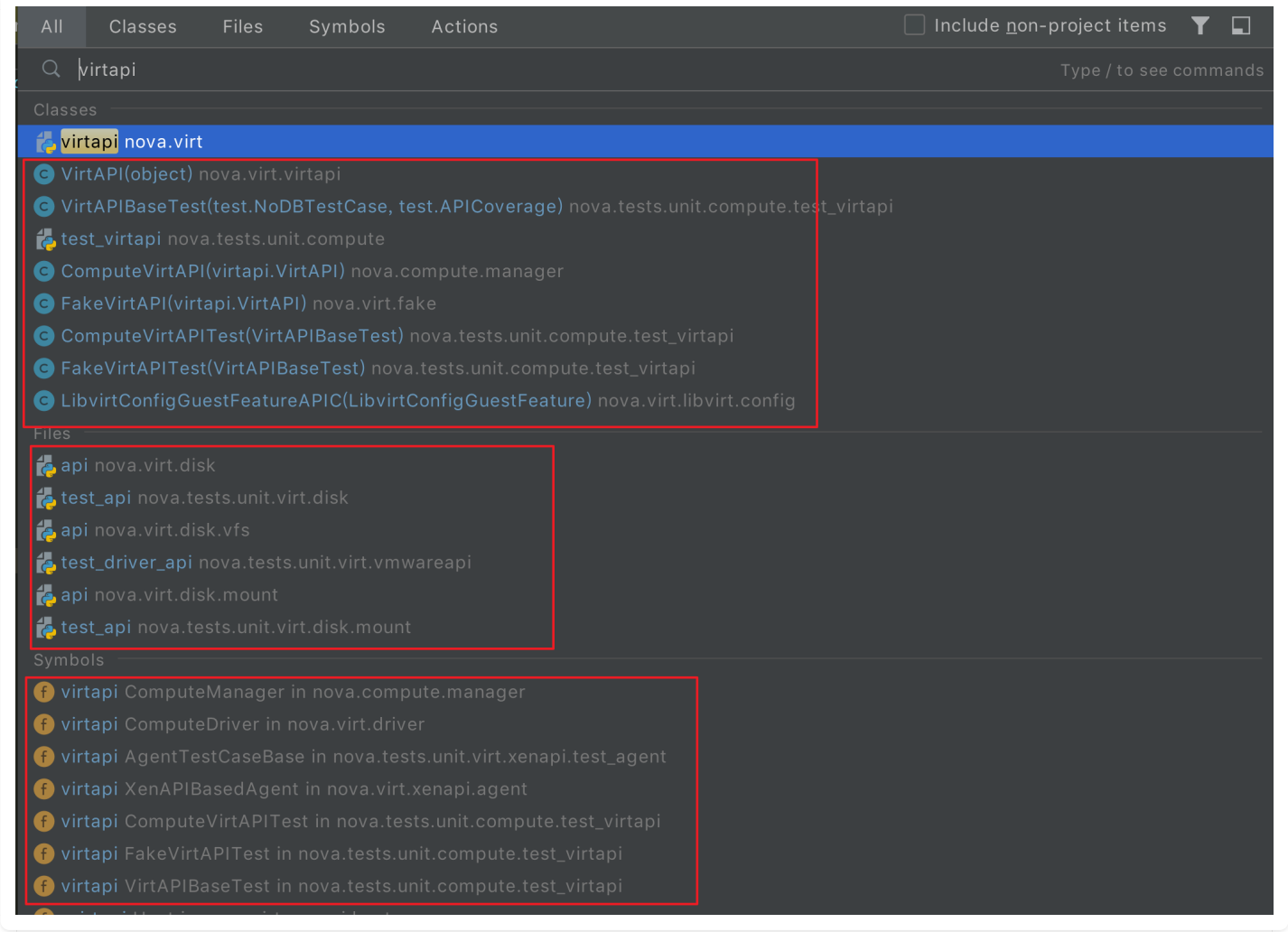
当你使用 Double + ⇧ 想搜索一个函数在整个项目中被谁调用了，会发现会有太多的干扰信息。

比如：

1. 某些函数只是包含了该函数名，也会被搜索出来



2. 不相关的文件名，符号名也会被搜索出来。



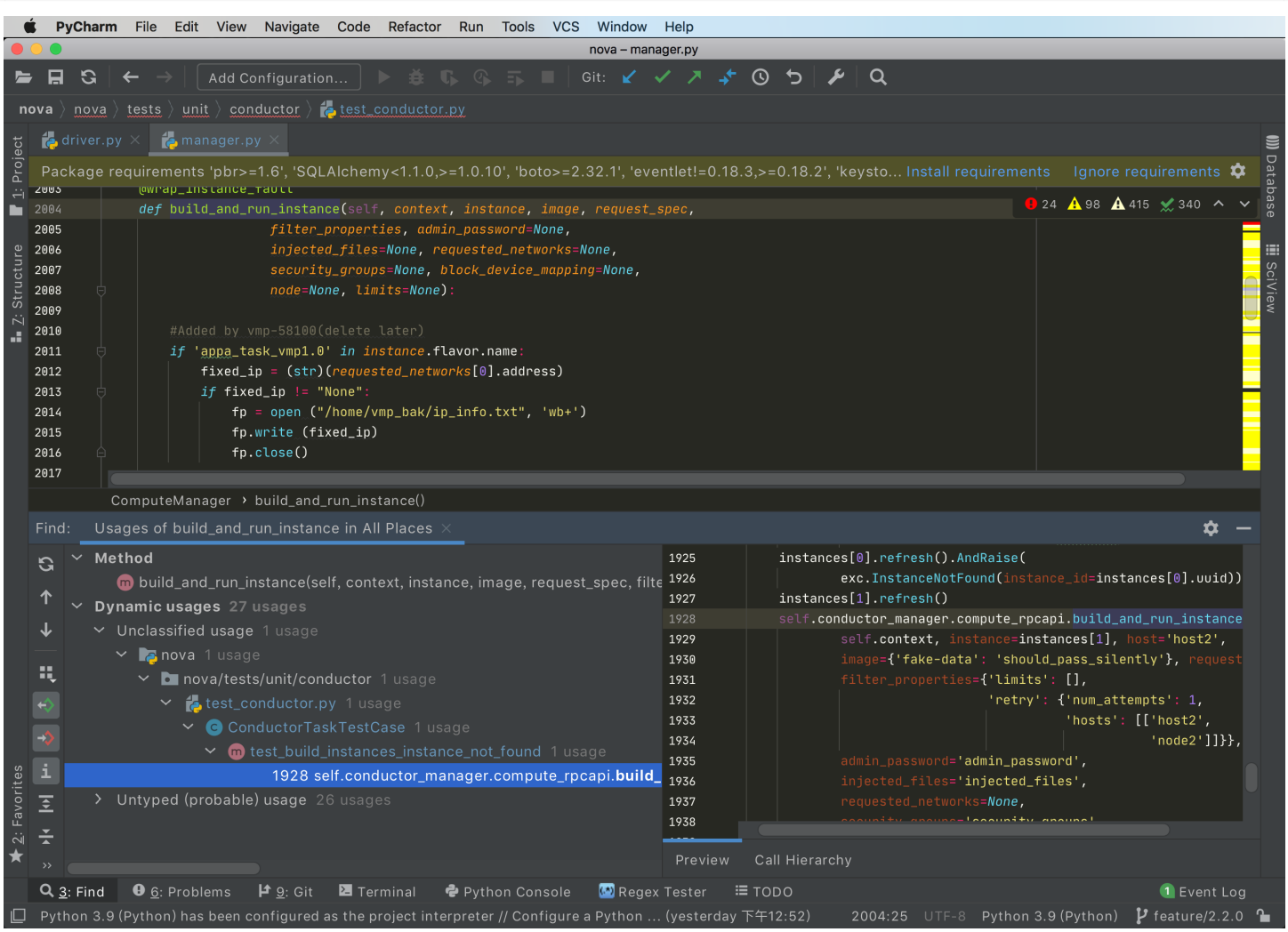
这种多的干扰信息，人工过滤掉无用的信息会消耗太多的精力。

因为对于搜索函数在所有文件中的用法会更精准的方式。

快捷键是：⌥ + F7

效果如下：

1. 函数名完全匹配才会显示
2. 只会搜索被调用的地方，定义的地方不会显示



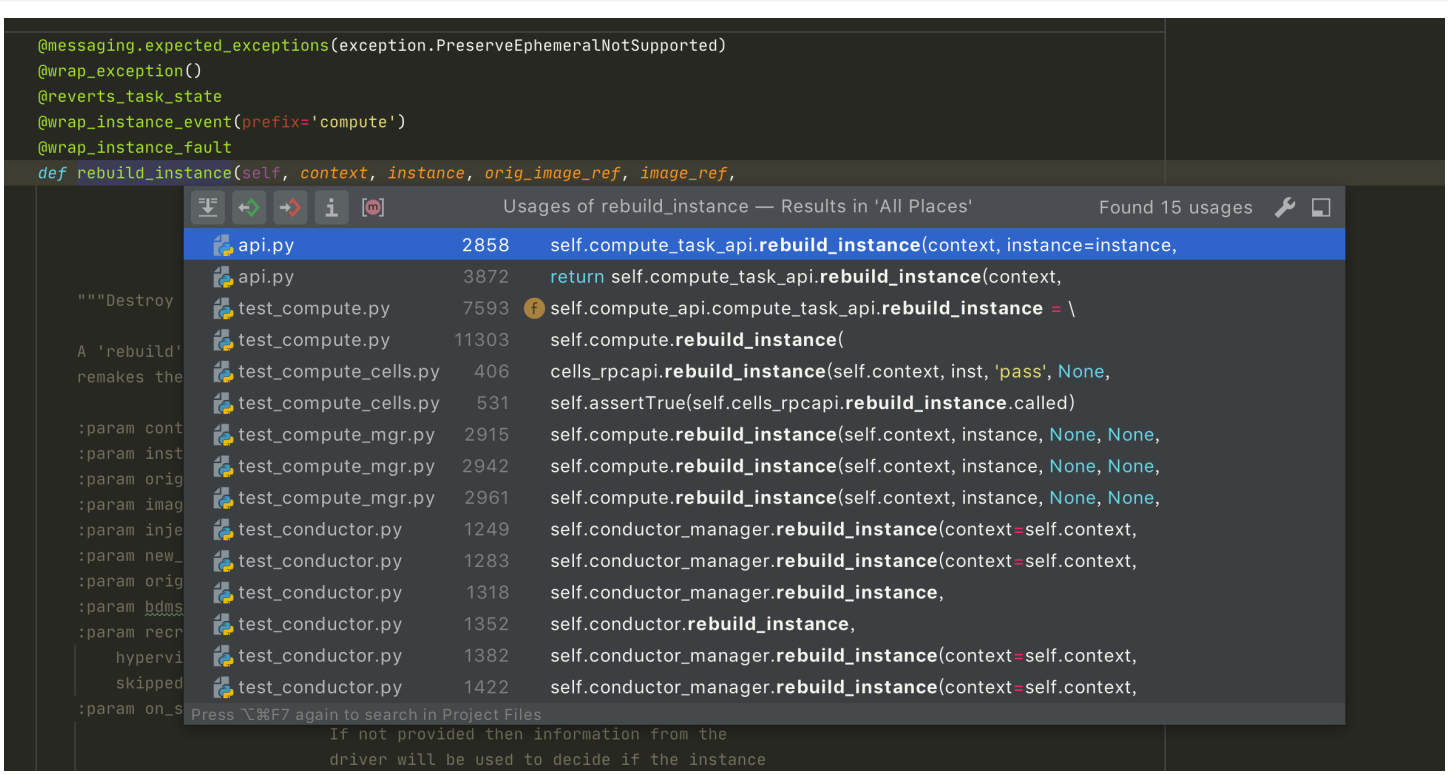
在项目中搜索用该方法的地方(Show Usage)

搜索的内容与上面第一种并无区别，但是这种显示效果会更直观一点：

- 上一种：以目标树展示，强调了层级关系
- 这一种：以文件列表展示，更加清晰易读

快捷键是：⌘ + ⌥ + F7

除了用快捷键外，还有更简便的方法，那就是直接按下 鼠标中键

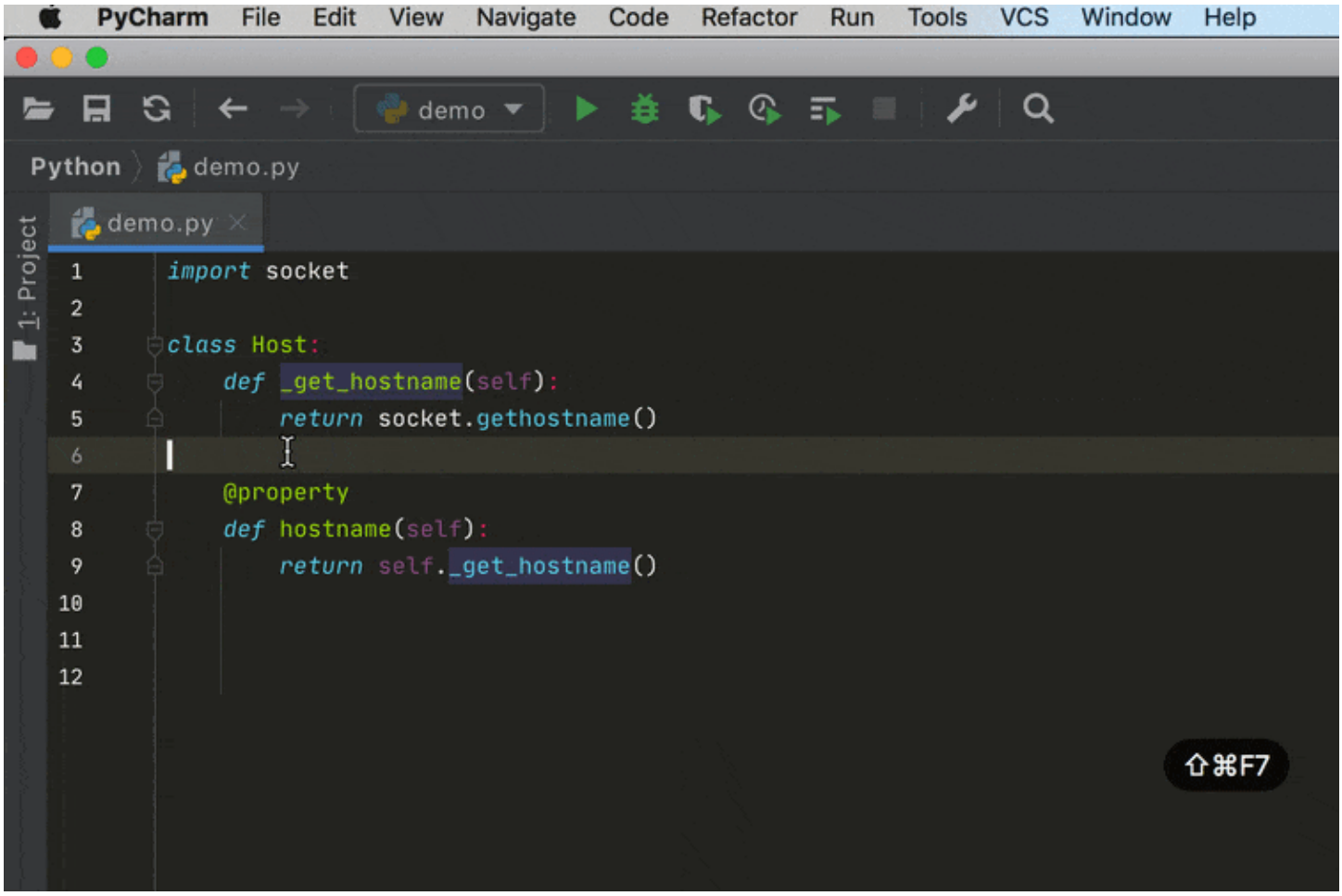


在当前文件中搜索用该方法的地方(Find Usage in File)

当你把光标放在一个函数上时，所有使用了该函数的地方都会被高亮。但是只要你把光标移走，高亮就会失效。如果一个类非常的长，你无法确保你的翻动代码时，鼠标不会点到别的地方，这时这种高亮的方法就会变得不太好用。

这里有一个更好的办法，那就是将某个函数/变量锁定在高亮状态，快捷键是 ⌘ + F7，而取消高亮状态的快捷键是 ⌘ + 仐 + F7

演示如下：



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

6.2 【搜索技巧 02】在项目中使用书签，快速定位

我在看框架的源代码时，最常使用的是 ⌘ + B （也就是 ⌘ + 鼠标左键 ）一层一层地往里深入，但是当源代码比较多，可能一整个事件过程涉及十几文件，函数调用错综复杂，对于一个庞大的项目来说，有用的可能就几个关键函数，每次要找到这几个函数，都要重头从源函数再一层一层的找下去，这样实在太麻烦了，我常常因此把自己给看晕了。

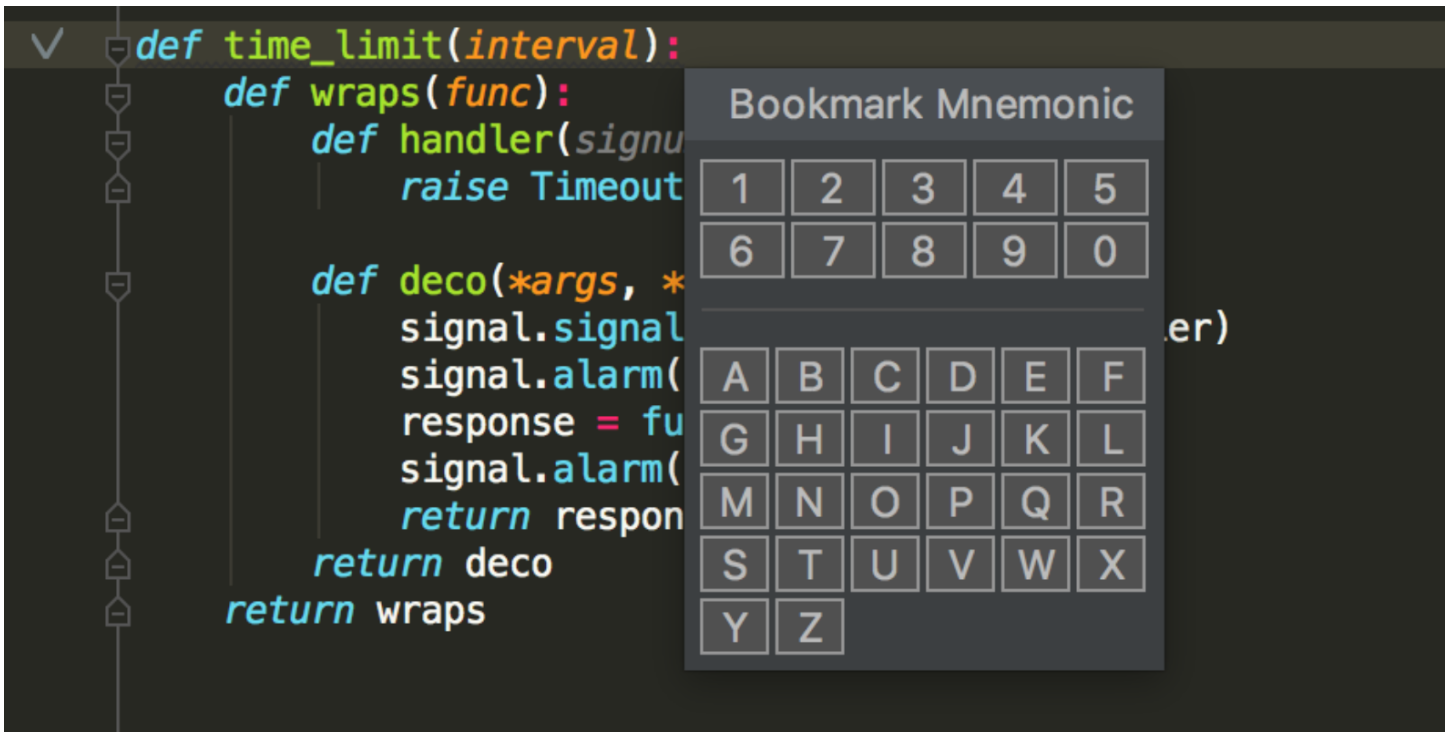
直到后来我发现了 Pycharm 这个书签功能。

使用书签功能，我可以在在关键的位置打上书签，想看的时候，调用书签，快速定位即可。

使用它，你需要记住下面下两个快捷键

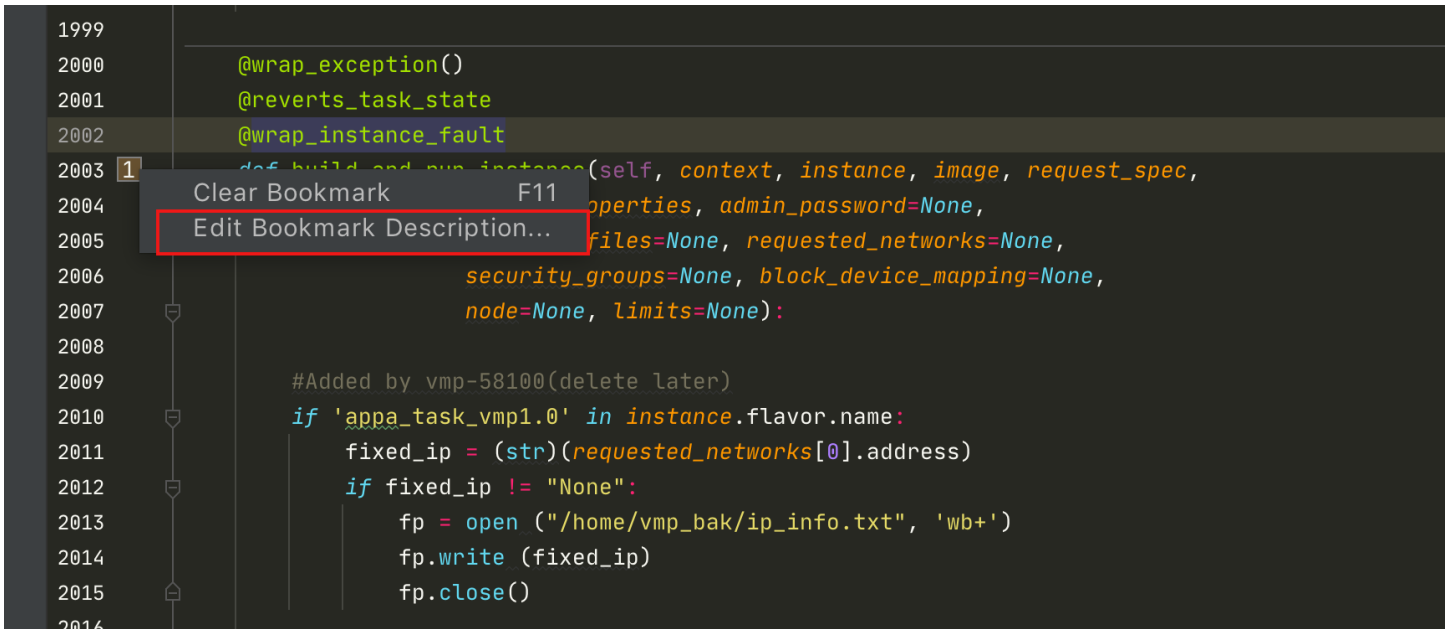
- F11：打上或清除普通书签
- ⌘ + F11：打上或清除书签（带数字，兼容普通标签）
- ⇧ + F11：展示所有书签

在你要打书签的位置，按下 ⌘ + F11 ， 你可以给这个位置加个序号，可以是数字也可以是字母，假如在下面这个位置 加了 `1` 这个序号， 下次你就可以使用 `Control` + `1` 直接跳转到这个位置。

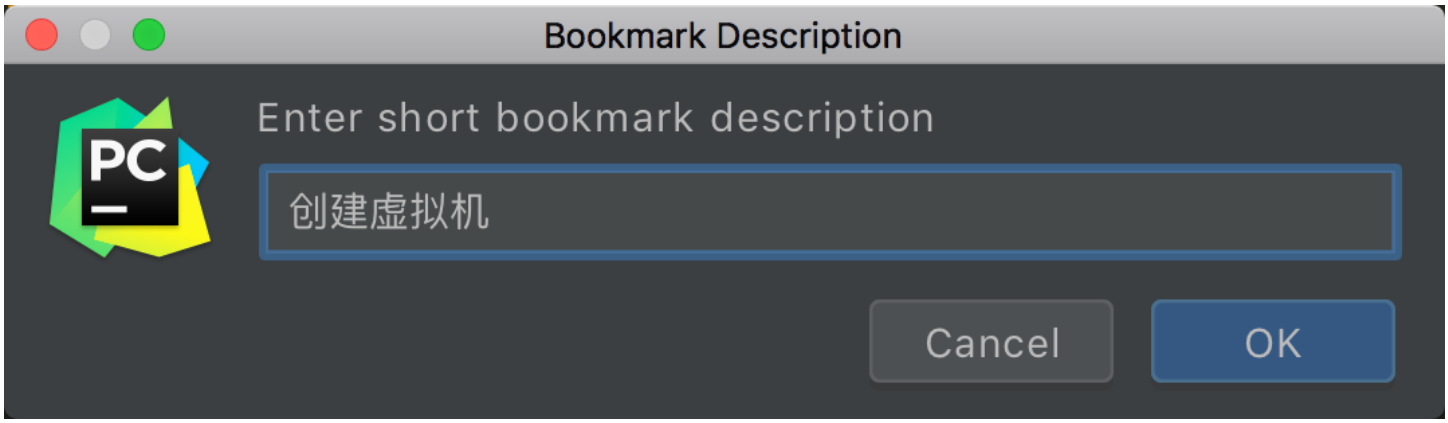


当然你也可以不加，不加的话就是匿名书签了。你可以使用 `Shift` + `F11` 展示所有的书签，再进行跳转。

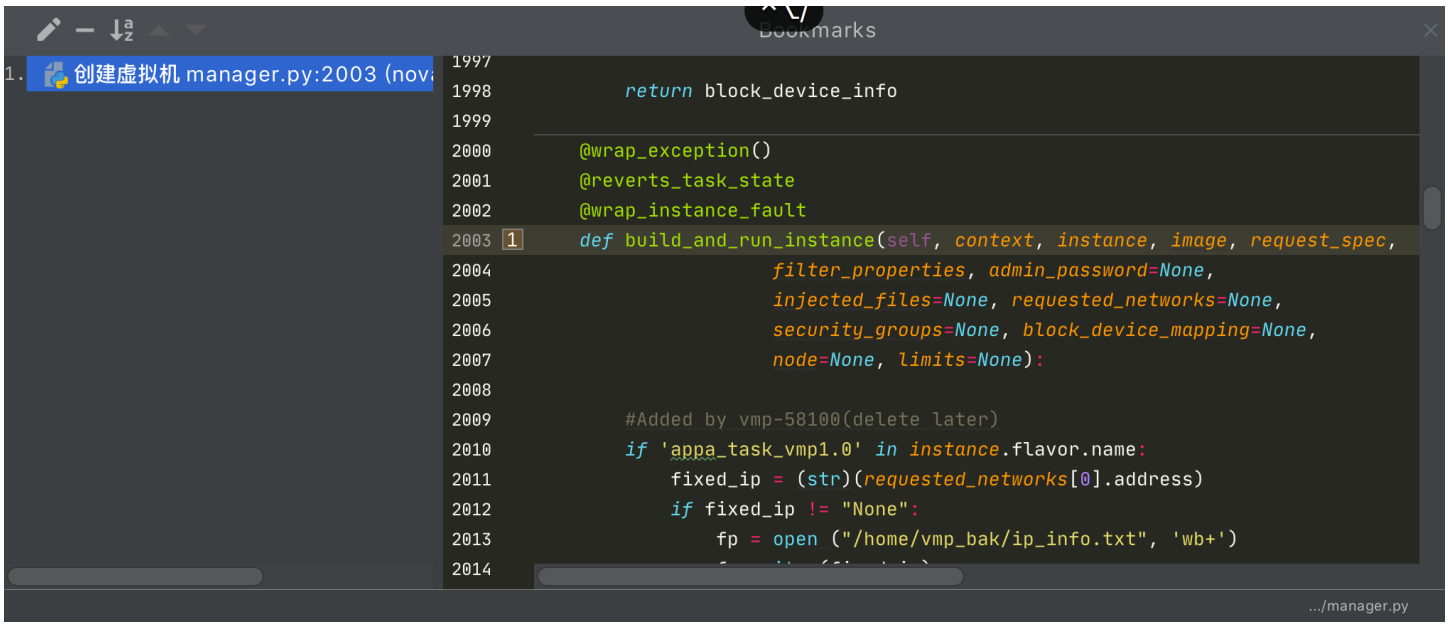
同时，你可以为书签加一段描述文字，表明这块代码是什么的



我写入如下信息



然后再使用快捷键：⇧ + F11，就可以列出所有的书签了



6.3 【搜索技巧 03】无死角搜索：搜索的八种姿势

在源代码中搜索，是一个非常高频的操作。

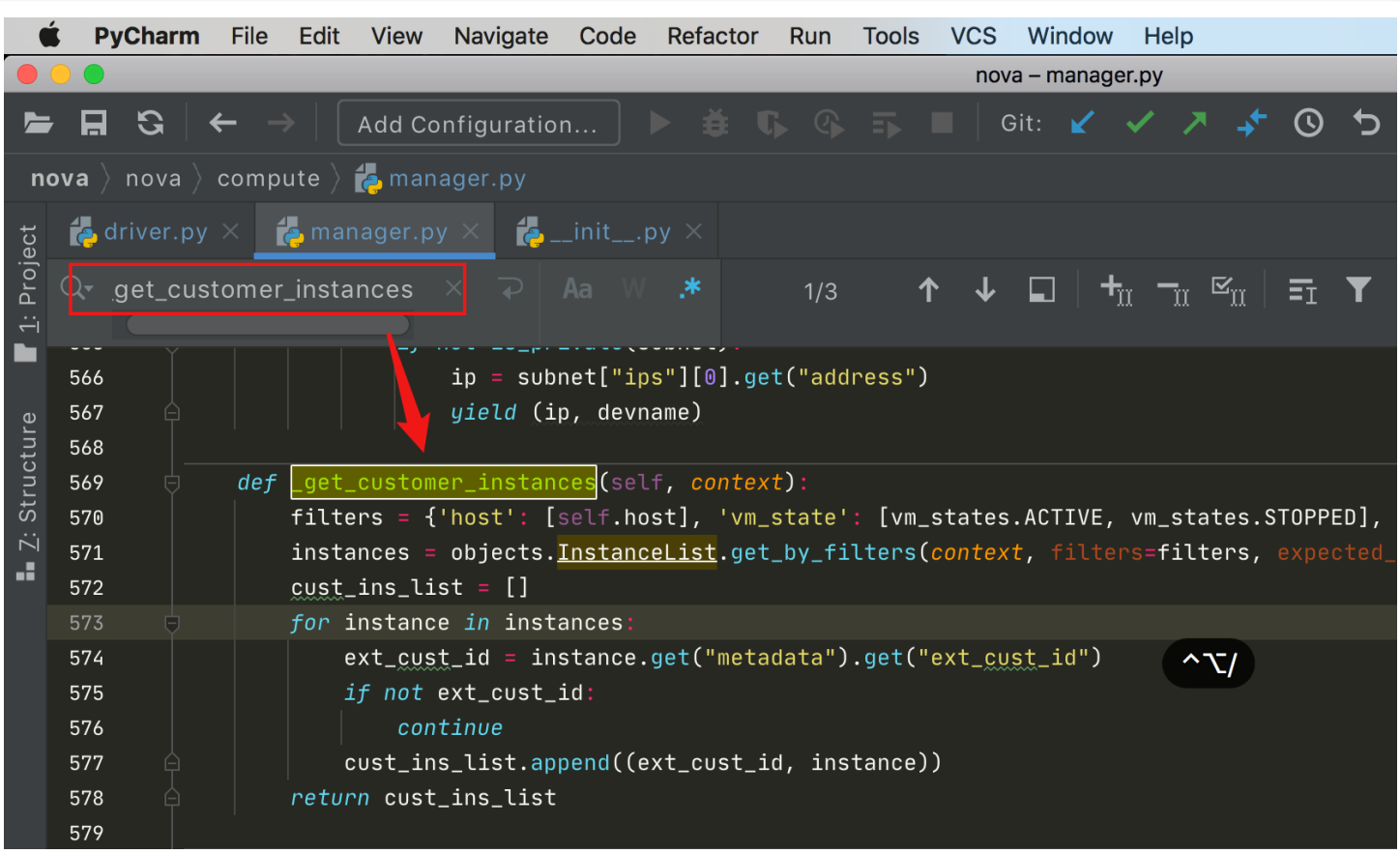
根据搜索的范围，可以分为：

1. 当前文件中搜索
2. 全局项目中搜索

当前文件中搜索

当前文件中搜索，可以使用两组快捷键，它们的功能是等同的

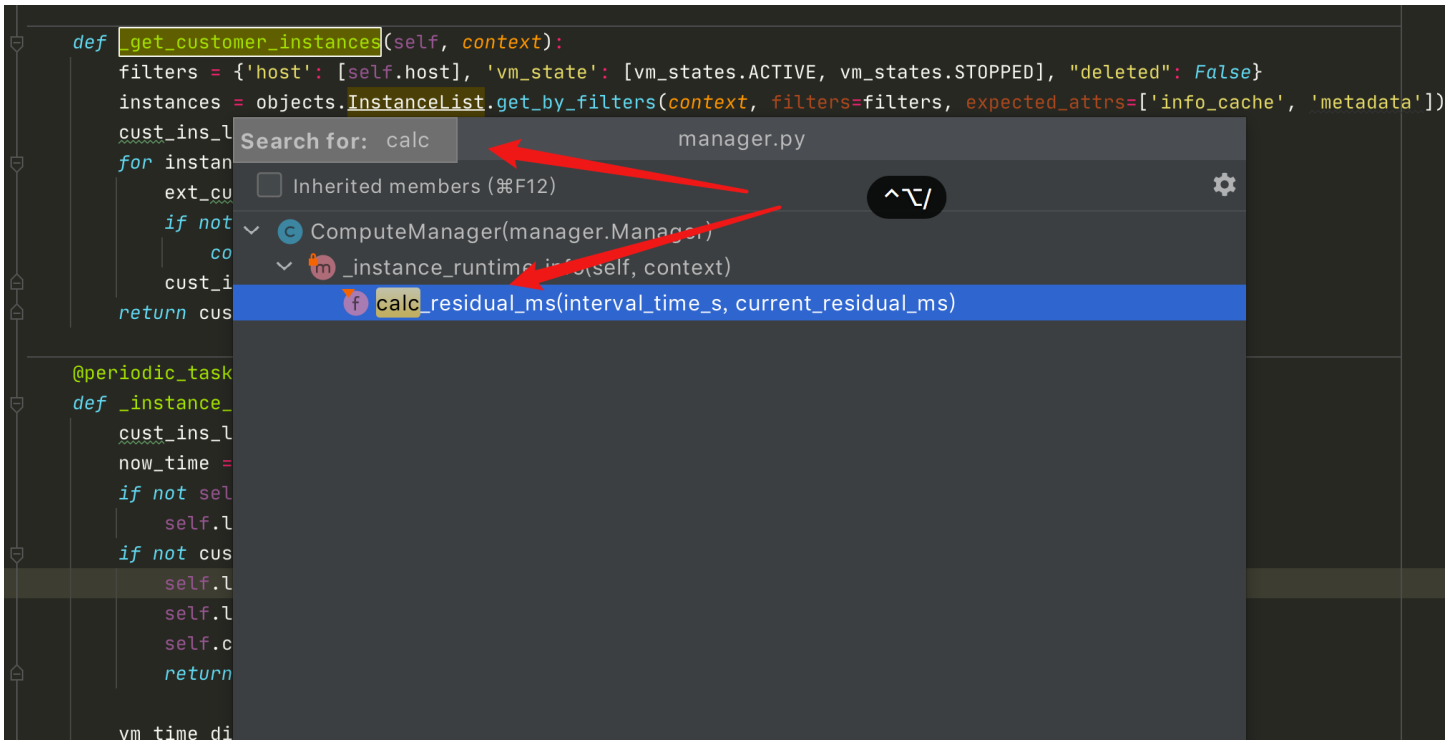
- ⌘ + F
- ⌥ + F3



除此之外呢，根据搜索对象的不同，还可以划分为：

- 普通文本
- 方法/函数名
- 类名

方法名和类名都是符号，可以使用 ⌘ + F12 ，调出结构树进行搜索。

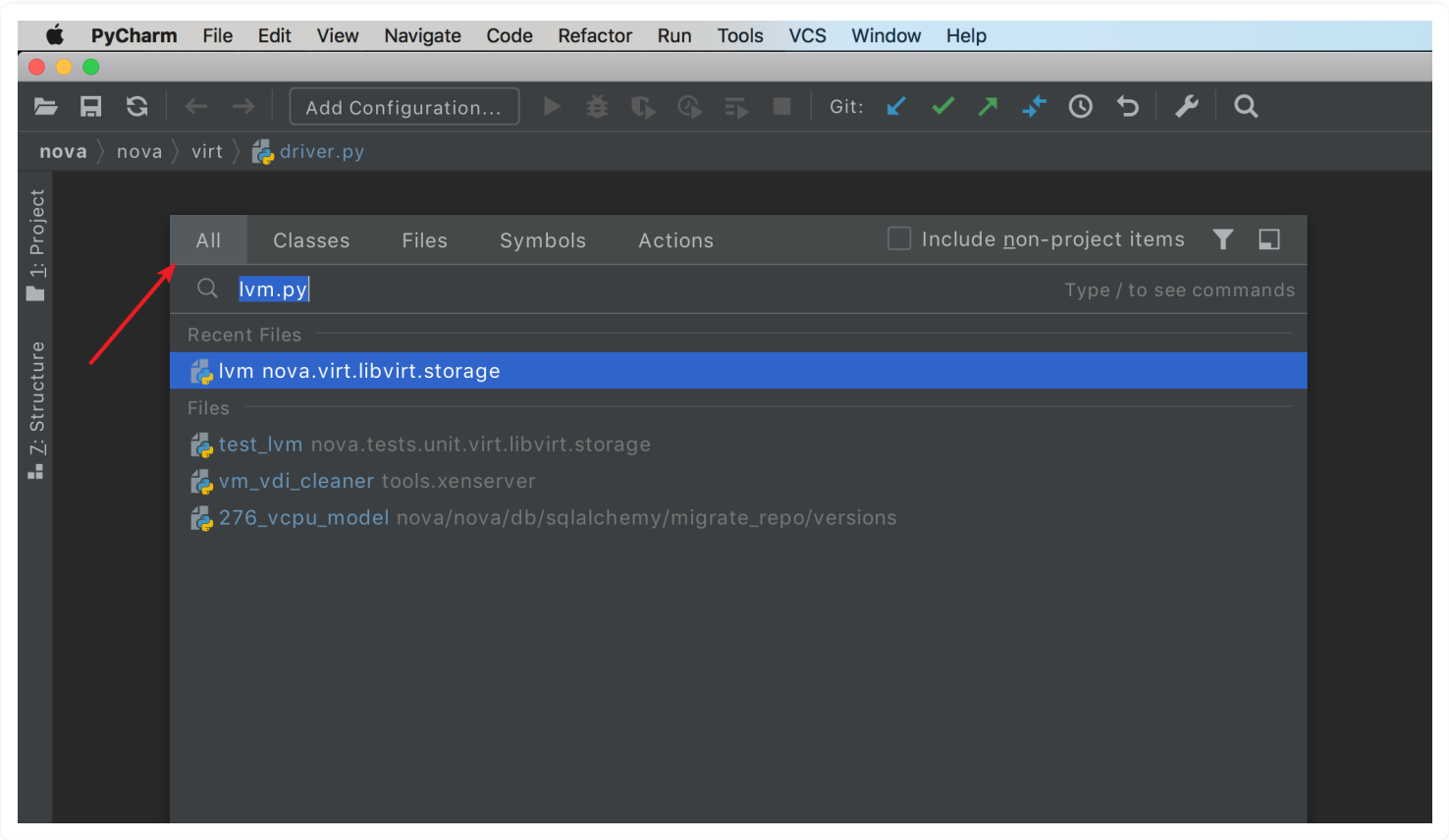


全局项目中搜索

根据搜索的对象，可以分为：

1. 文件名
2. 类名
3. 方法名
4. Action 名

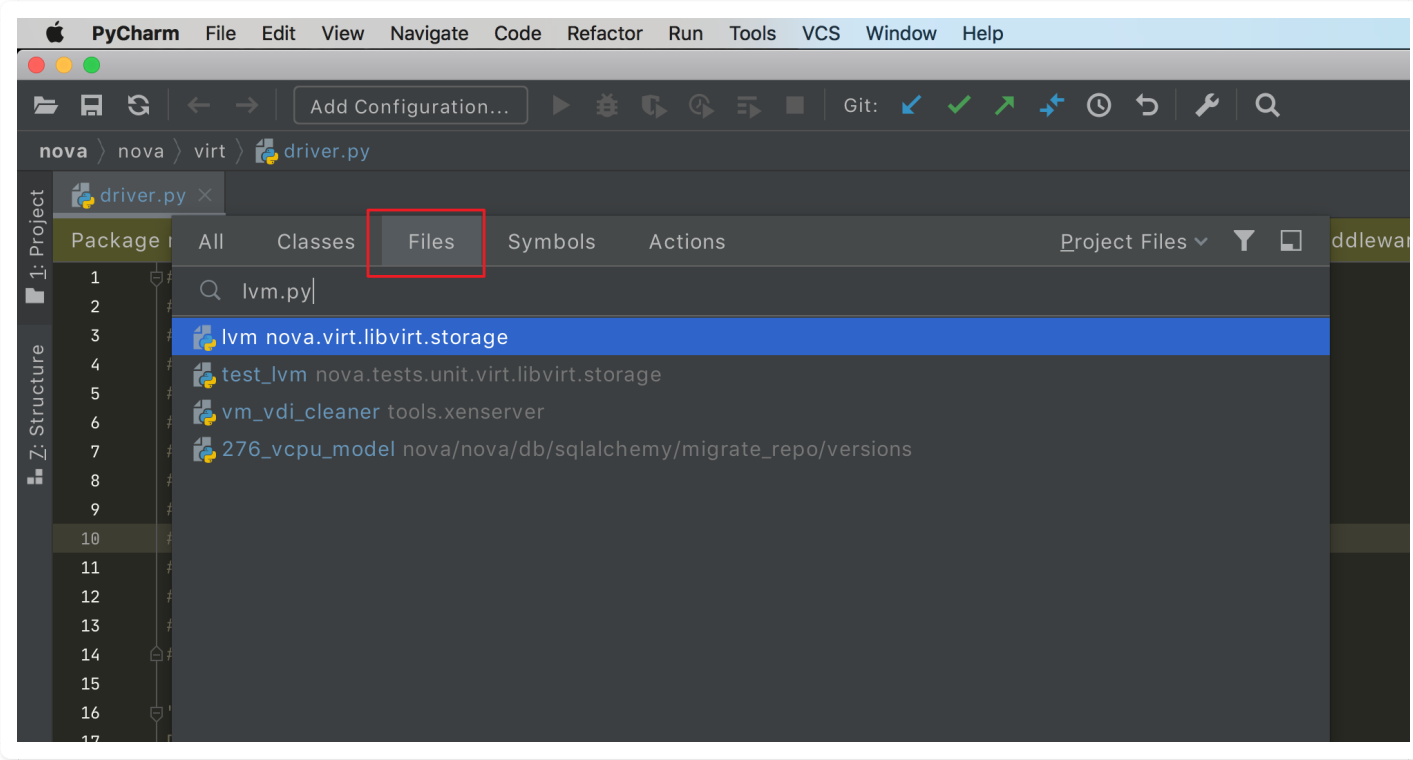
普通人都是使用 Double + ⇧ ， 来搜索所有的东西(Search everywhere)，包括



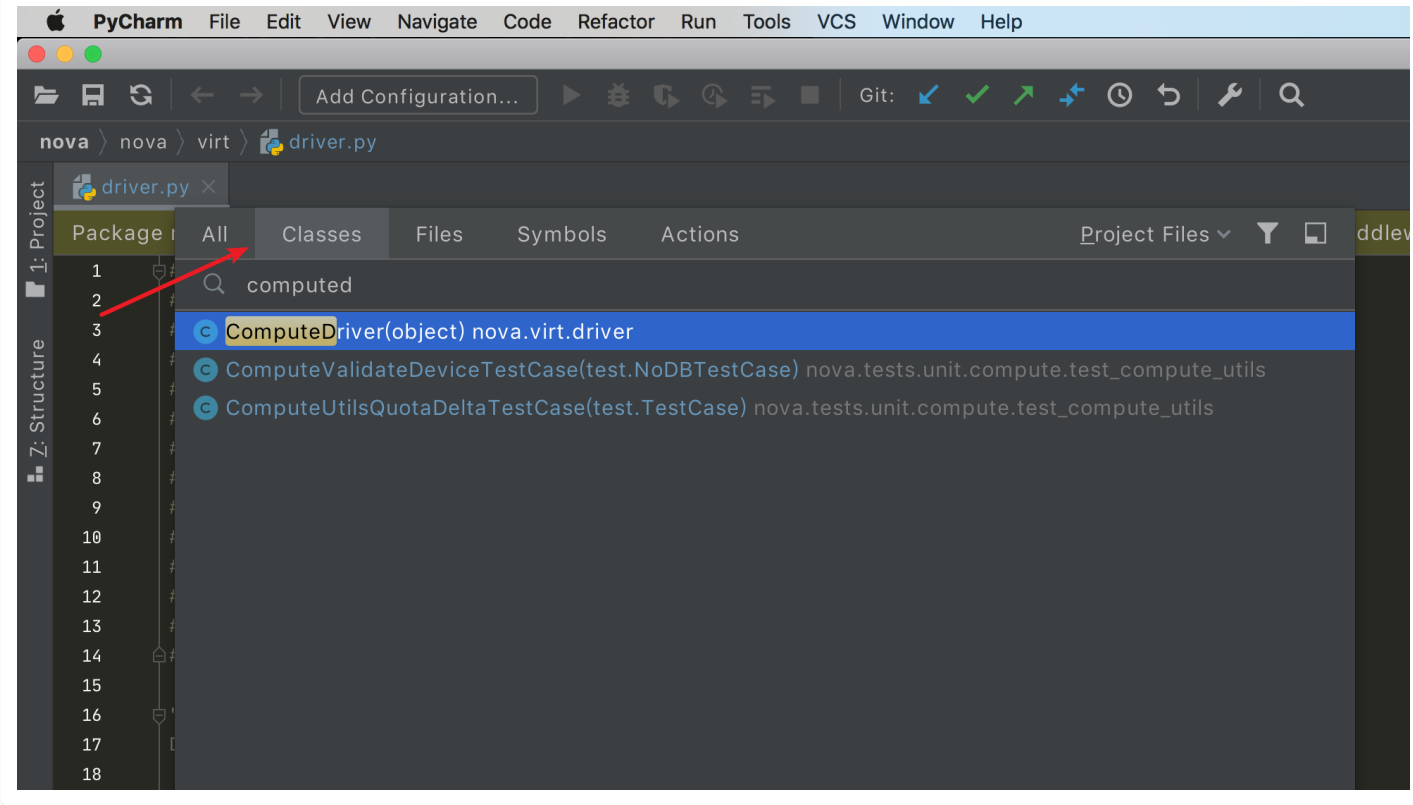
搜索的东西多了，就意味着搜索无法精准。

其实对于上面的几类，PyCharm 有提供专门的入口，下面开始介绍：

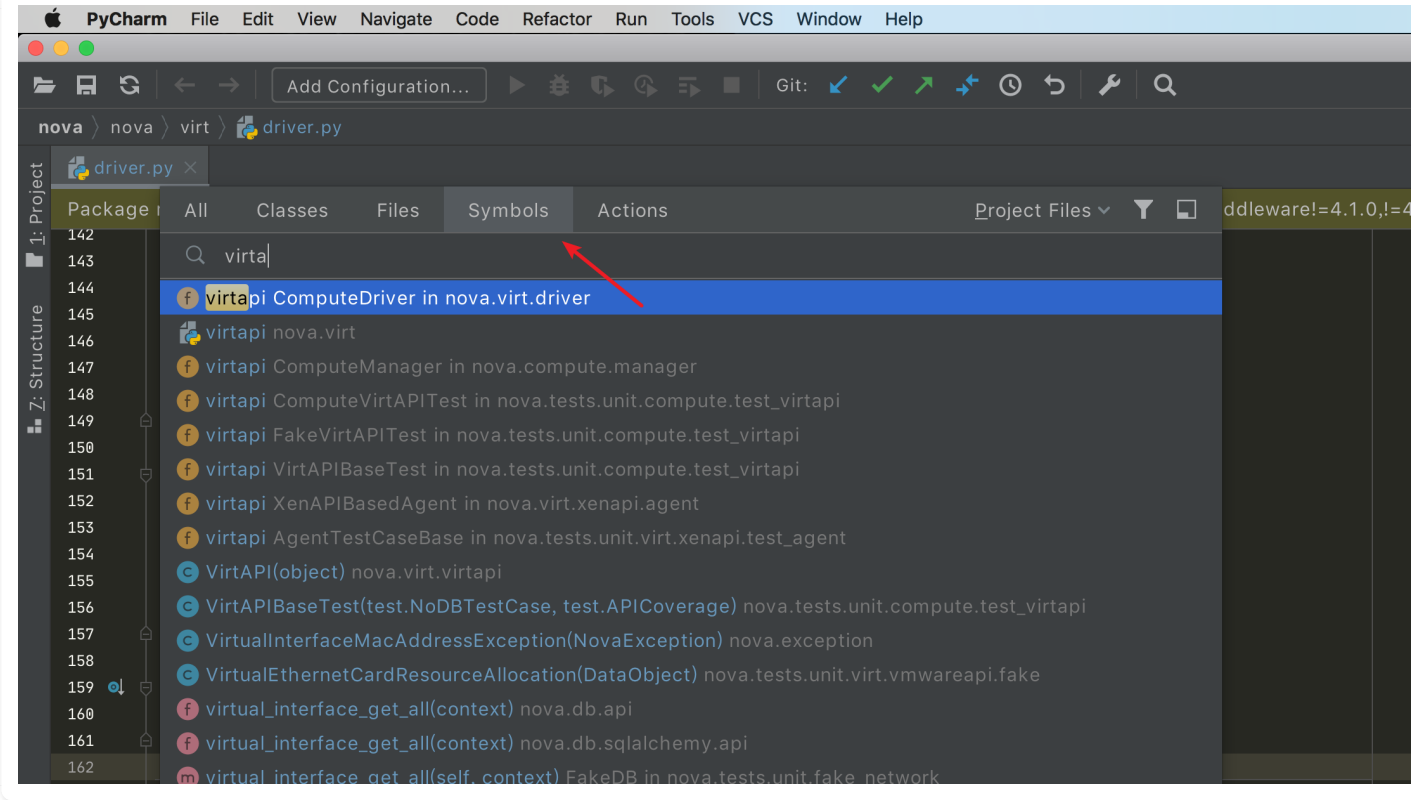
- 精准定位到文件：Windows（Ctrl+Shift+N），Mac（Command+ shift +N）



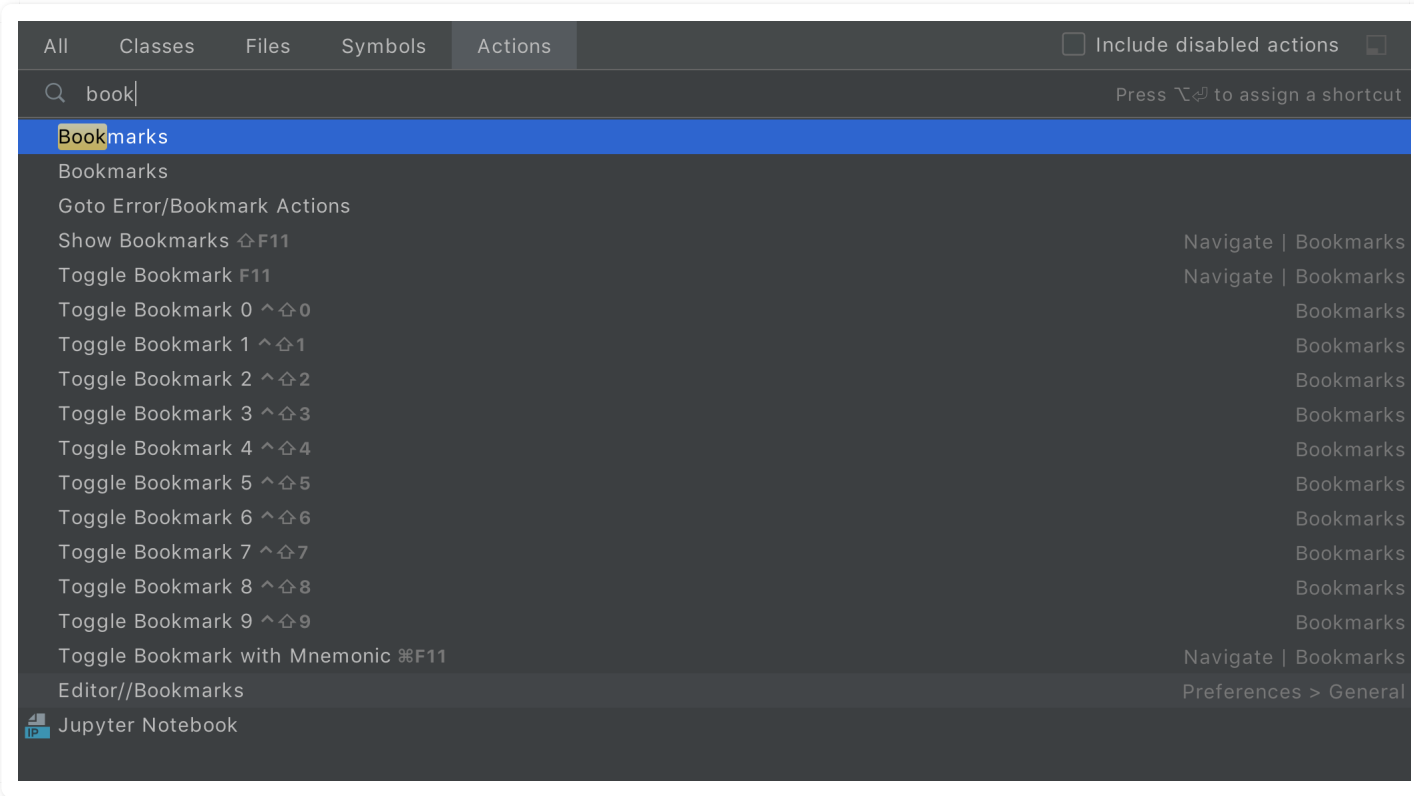
- 精准定位到类：Windows（Ctrl+N），Mac（Command+N）



- 精准定位到符号：类的所有成员（函数、变量等）都可以称之为符号，Windows（Ctrl+Alt+Shift+N），Mac（Option+Shift+Command+N）

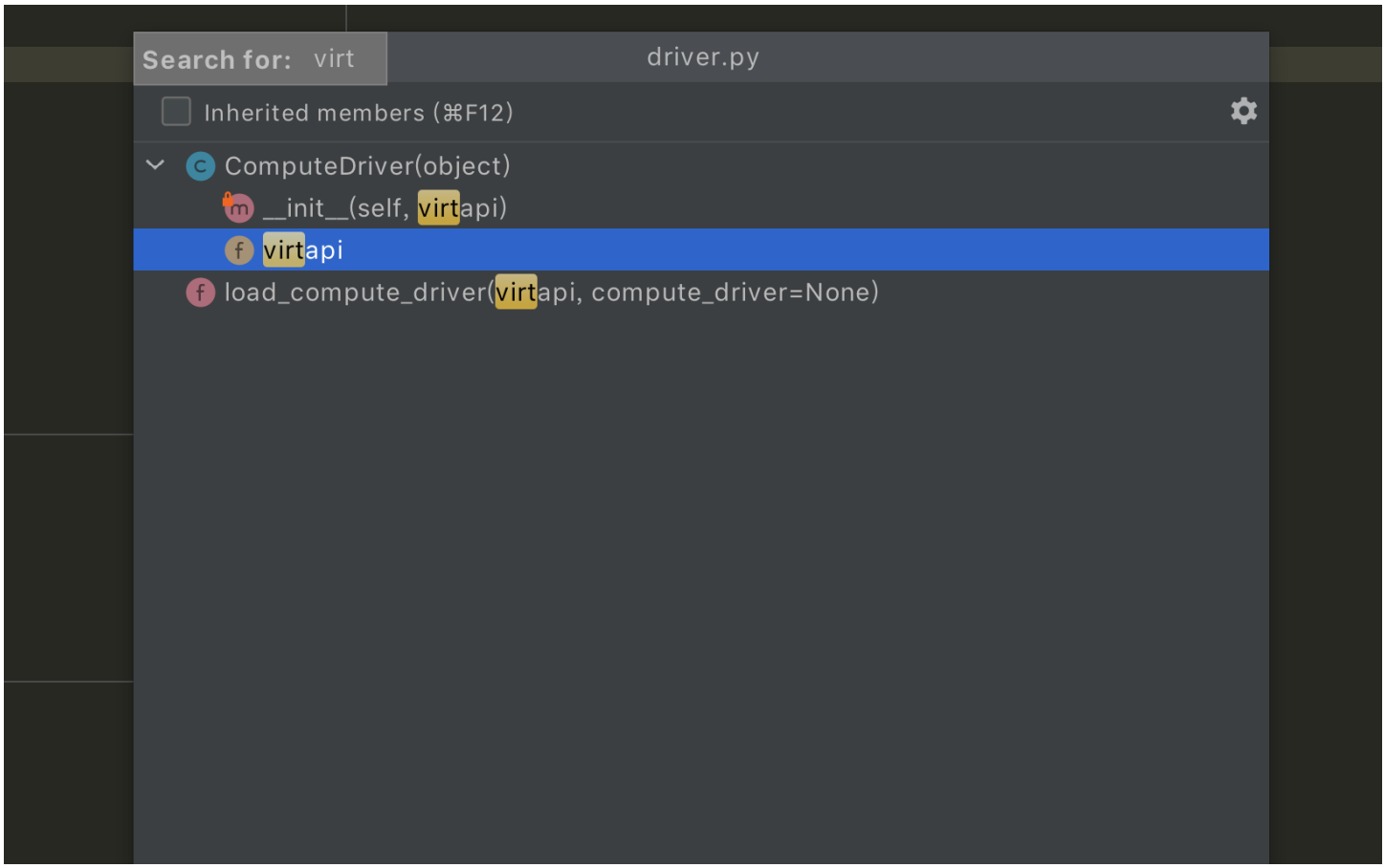


- 精准搜索 Action： ⇧ + ⌘ + A，比如下面我搜索书签的所有 Action，可以看到把相应的快捷键都给出来了。

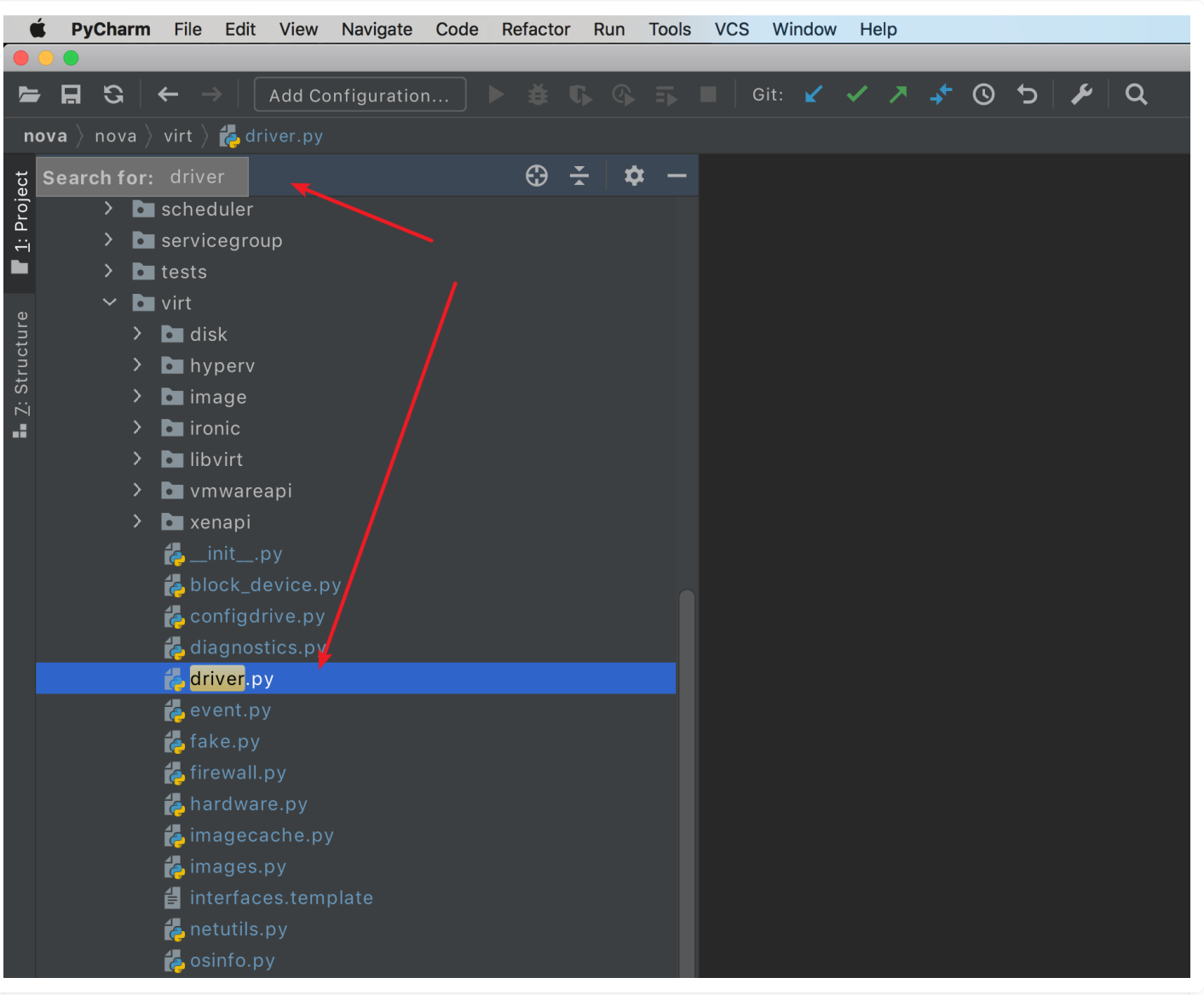


- 精准定位到文件结构：文件结构包括类、函数、变量，这说明上面定位到类和定位到符号的方法，你都可以用这个来代替。

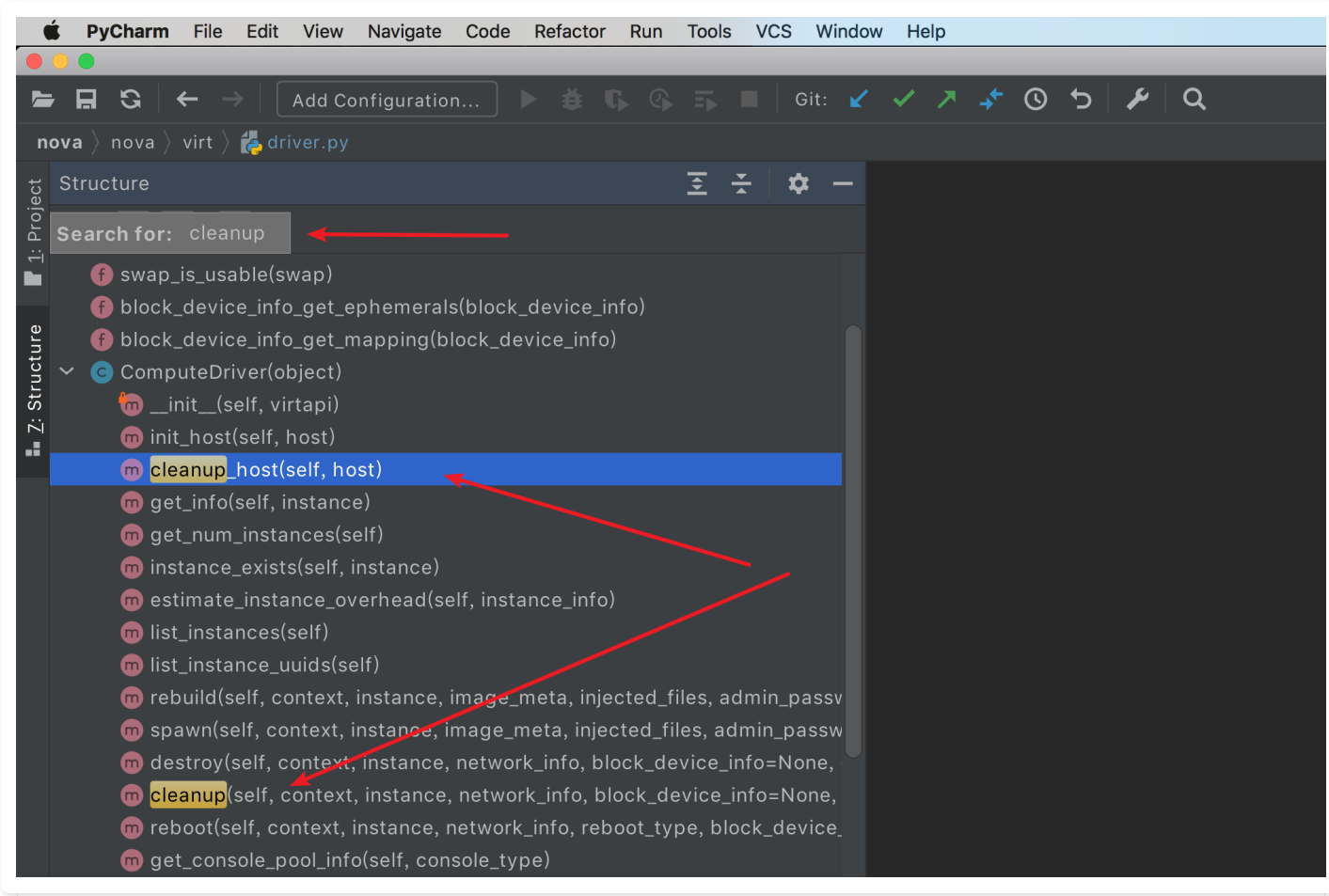
  Windows：Ctrl+F12，Mac：Command+F12，如果和我一样是Mac是带touchbar的，键盘上是没有F12的，那你应该先按住 Command + fn，这时 touchbar 上会出现 F12，再按F12即可。



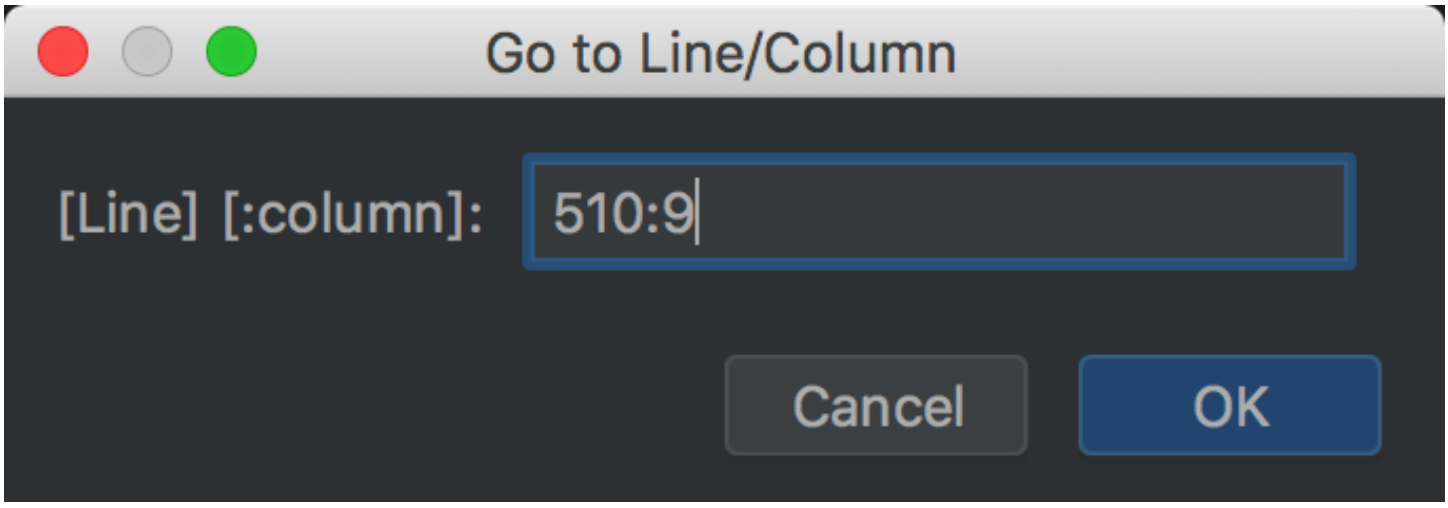
- 指定文件夹下搜索文件：直接在项目树中输入你要搜索的文件名，若要清空以往输入，可以按 esc 清空。



- 指定文件中搜索项目结构：直接在项目结构中输入你要搜索的结构名（可以是类、函数、变量等），若要清空以往输入，可以按 esc 清空。



- 精准定位到某行：Windows（Ctrl+G），Mac（Command+G），如下图定位到第510行第9个字符处。

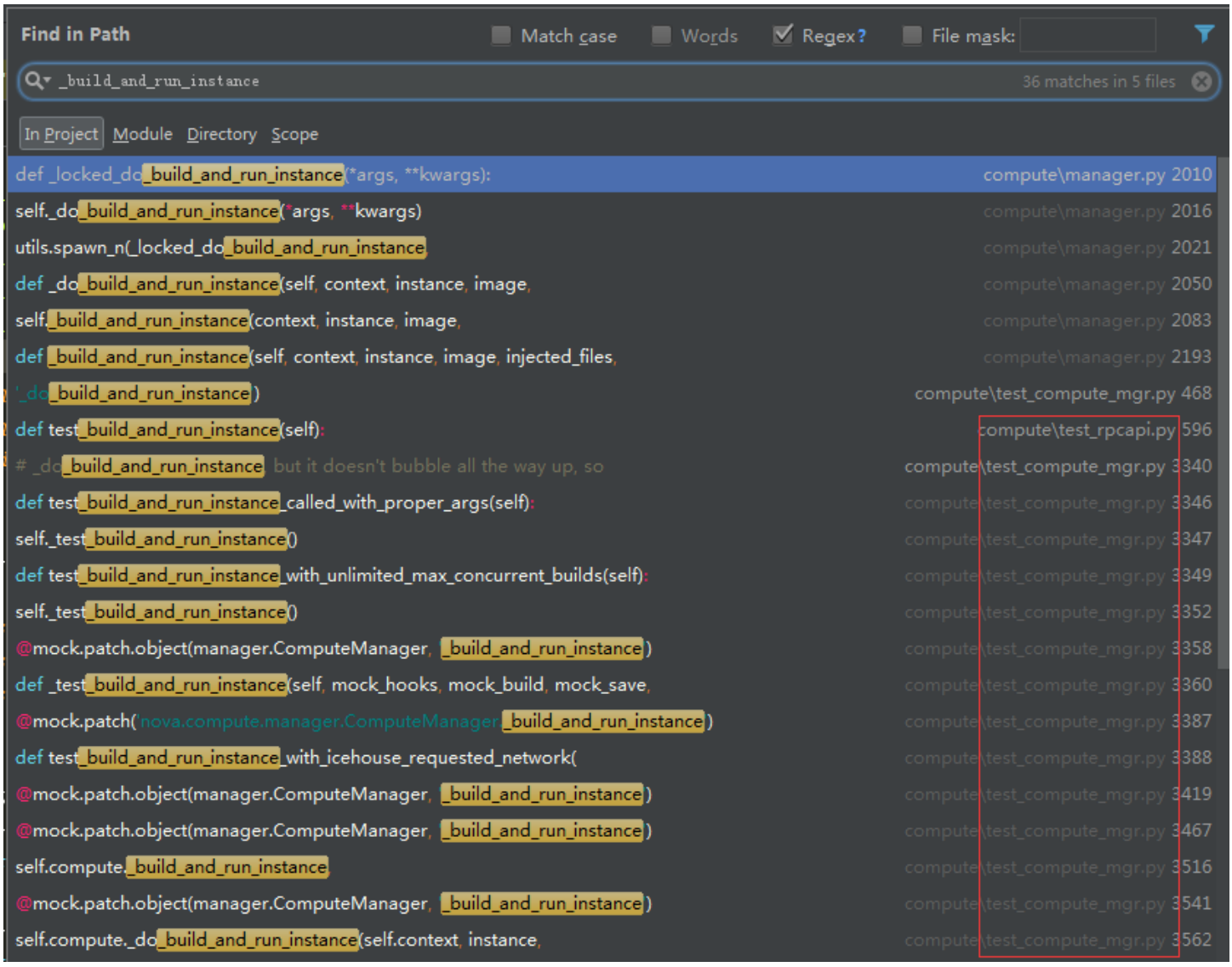


6.4 【搜索技巧 04】搜索时过滤测试文件

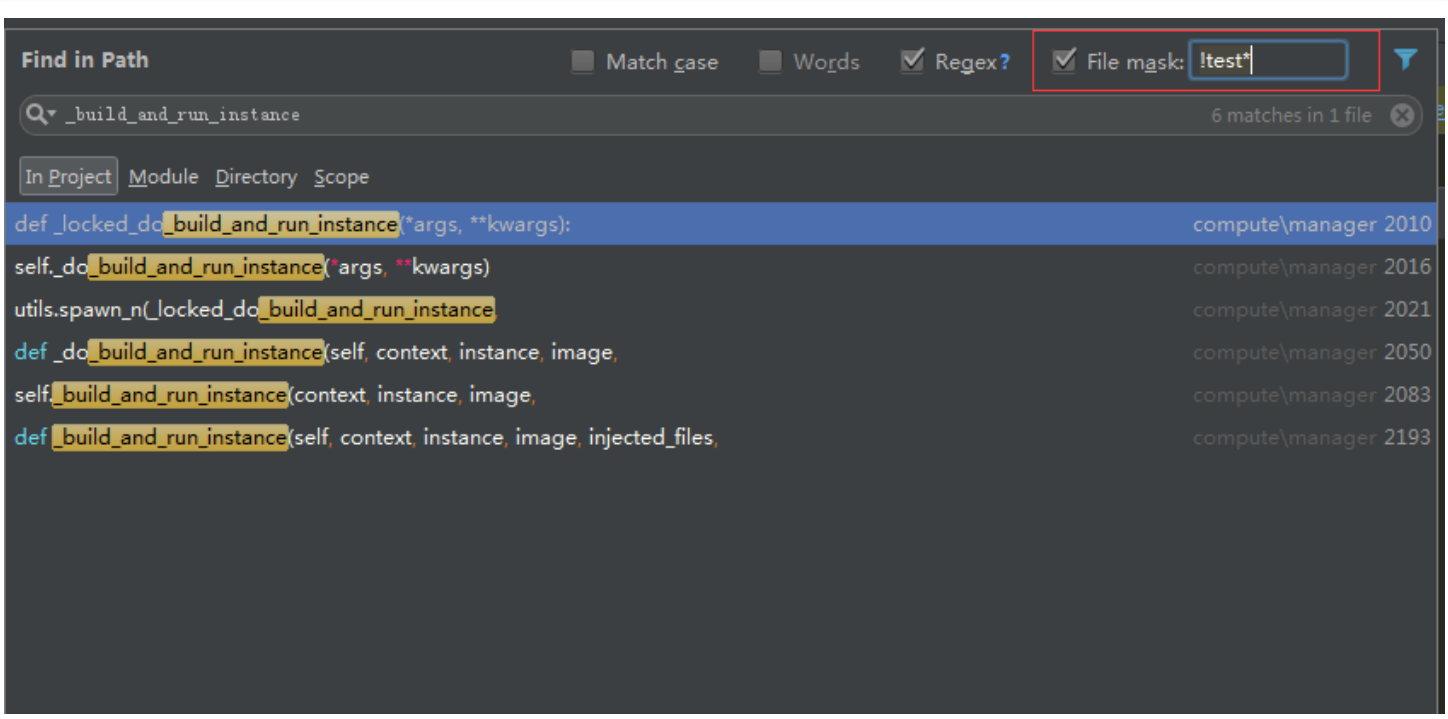
接下来，介绍一个，我看框架源码的时的一个小技巧，可能只适用一小部分人吧。

我平时会看的框架是 OpenStack，我不知道其他框架是怎样的，但在 OpenStack 里面带有大量（真的很多）的单元测试文件。这给我在使用 `Find in Path` 时带来了不小的困扰，你可以从下图

的搜索结果中感受一下，搜索一个函数，test 文件里的结果比 正常文件要多很多。



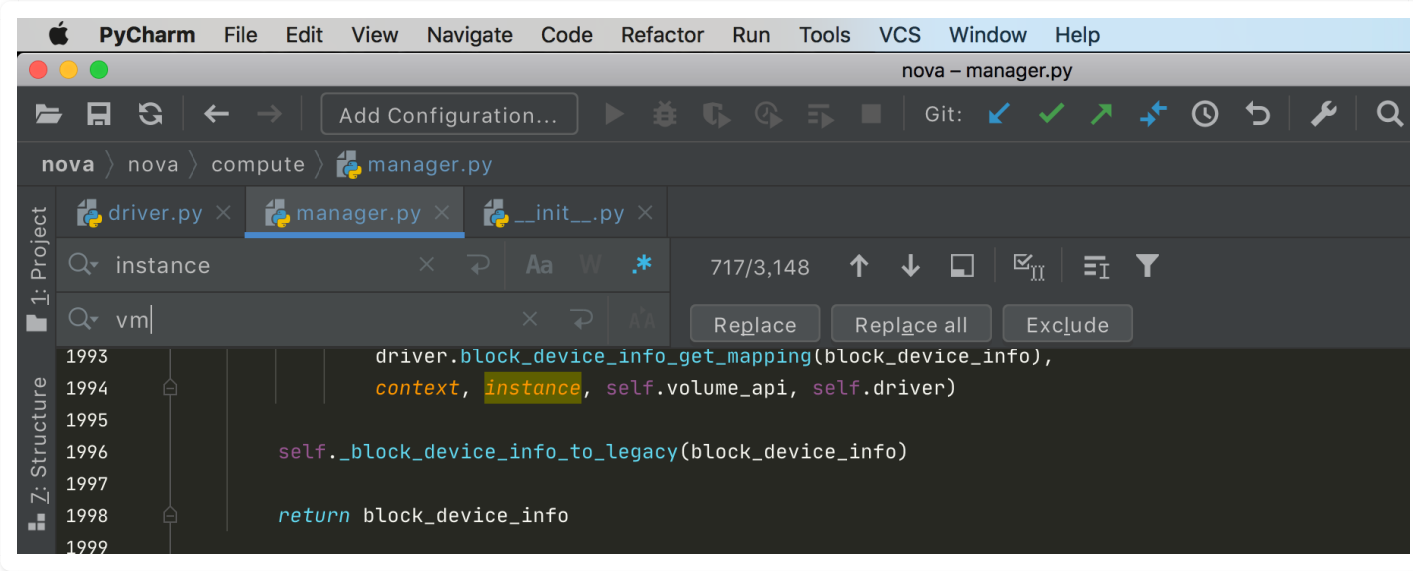
这些测试文件的搜索结果，对于我们看源代码不仅没有任何帮助的，更重要的是还干扰视线。于是我就研究了一下，从文件名入手，只要在 `File mask` 里填写 `!test*` 可以将这些test文件过滤掉。搜索结果一下子清晰很多。



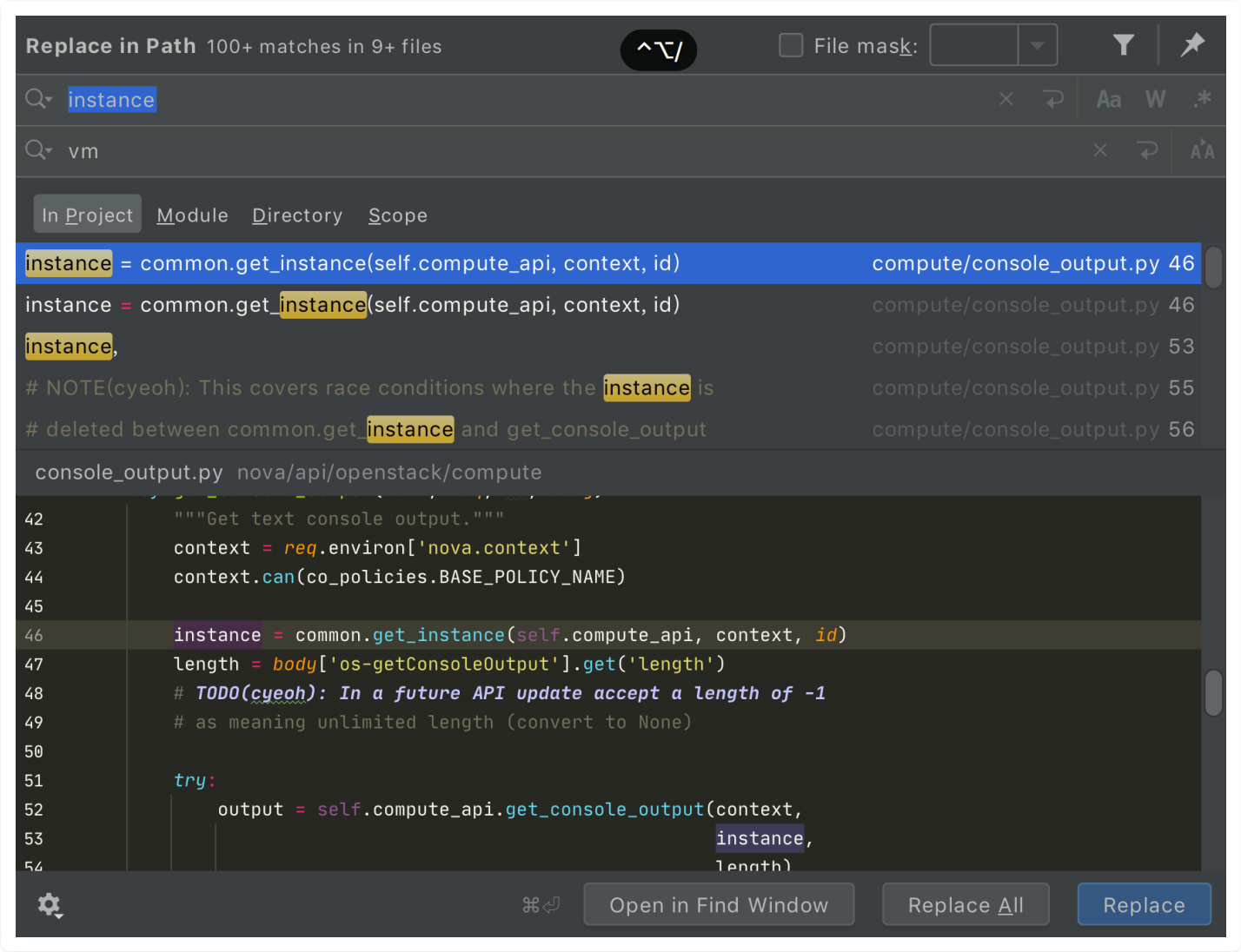
6.5 【搜索技巧 05】当前文件替换与全局替换

替换主要分两种：

1. 在当前文件里替换，快捷键是：⌘ + R



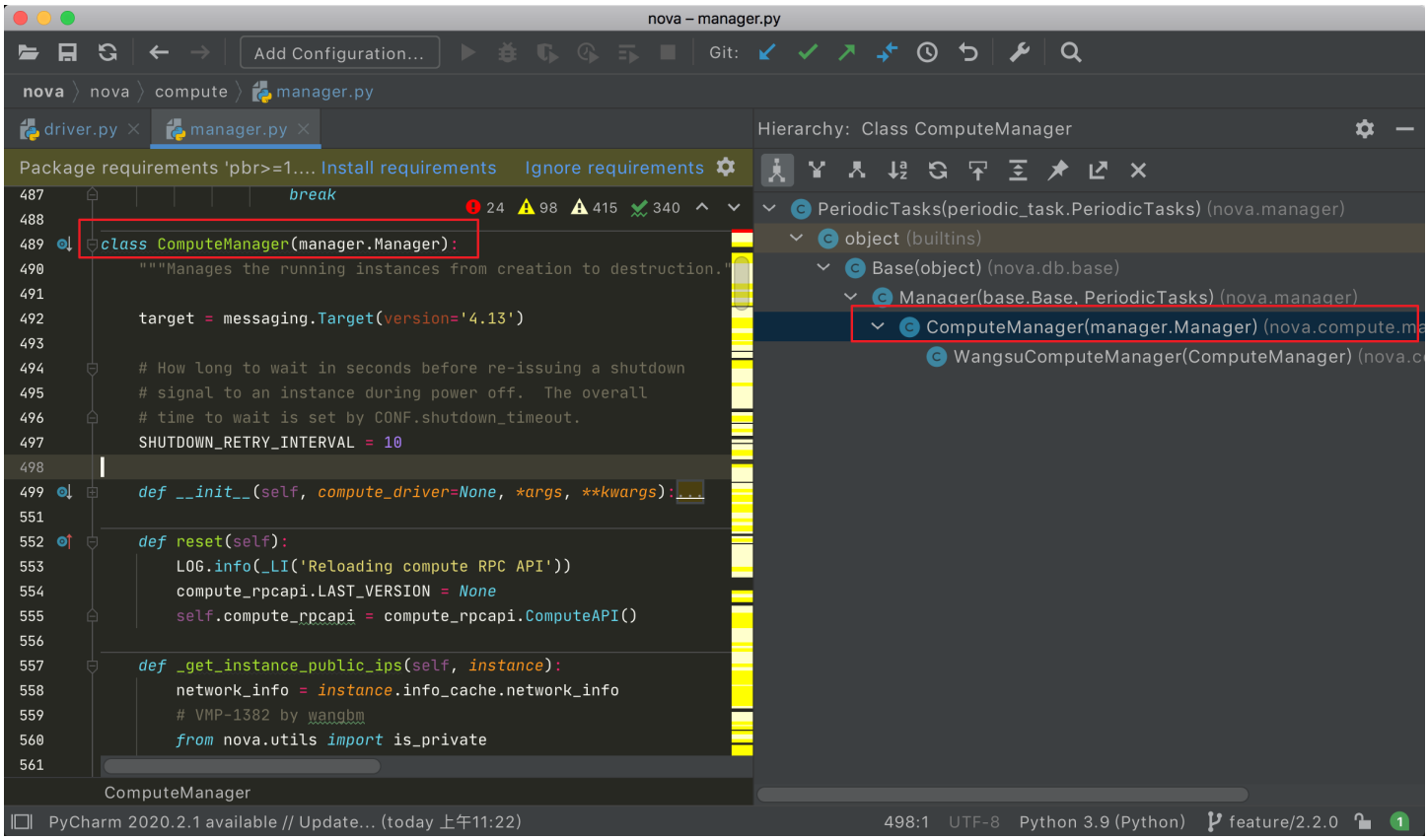
2. 在全局项目下替换，快捷键是：^ + ⇧ + R



6.6 【搜索技巧 06】显示当前类的继承树：Type Hierarchy

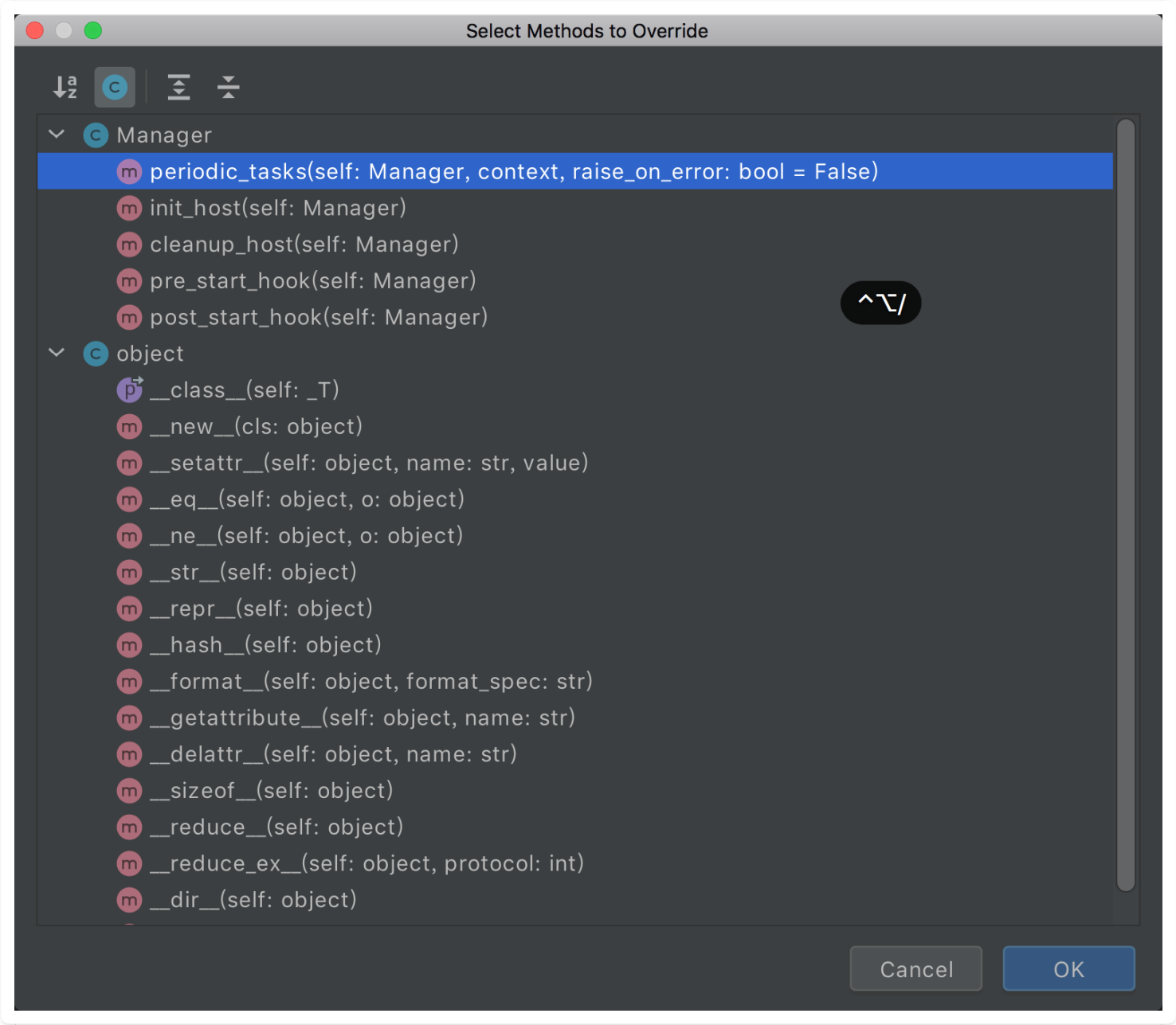
第一种方法

使用快捷键 `Ctrl + H`



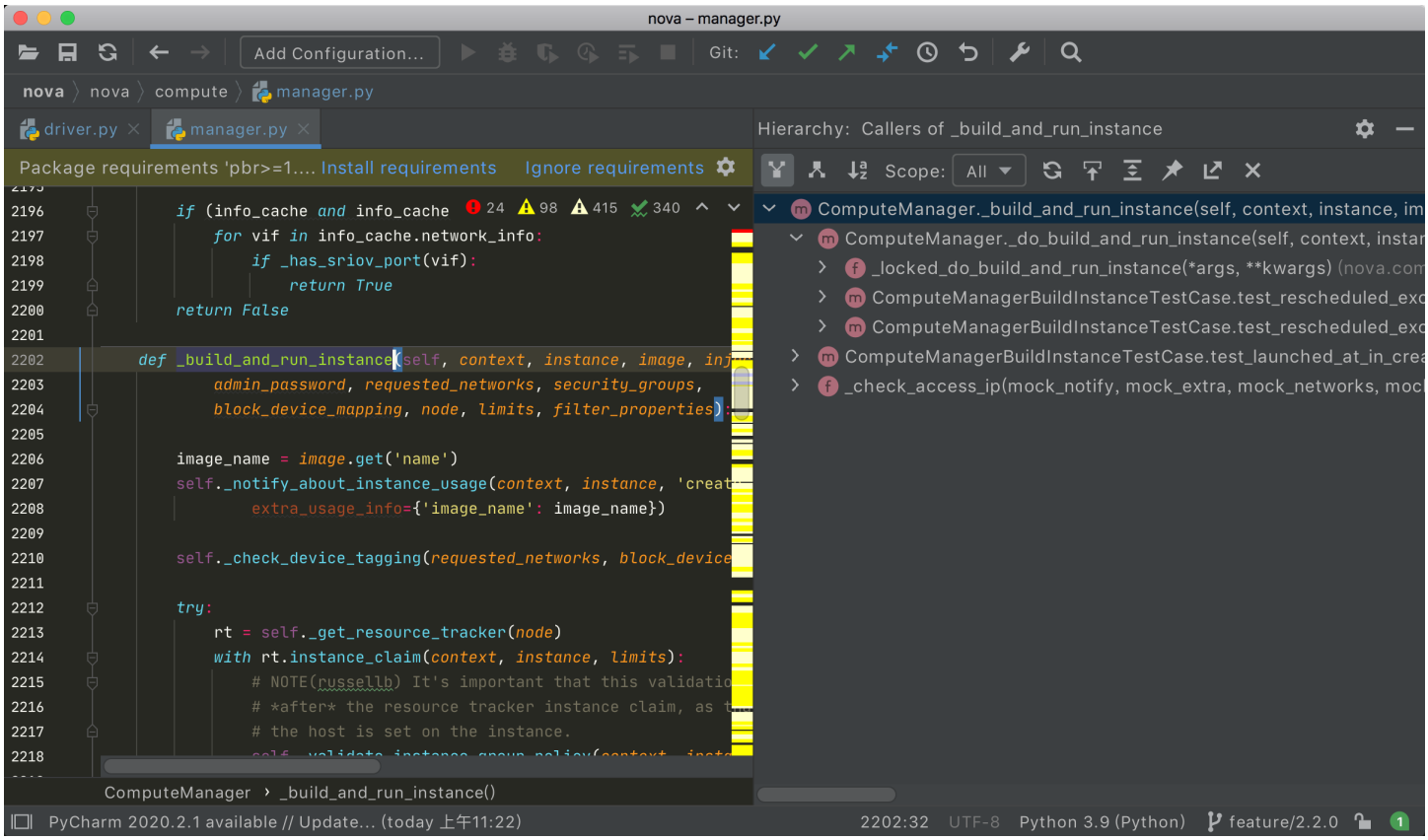
第二种方法

使用快捷键 ⌘ + O



6.7 【搜索技巧 07】显示当前方法的调用树：Method Hierarchy

按住快捷键 Ctrl + ⌥ + H



6.8 【导航技巧 01】跳转到最后编辑的地方

根据开源框架定制功能，是我每天都要做的事情。

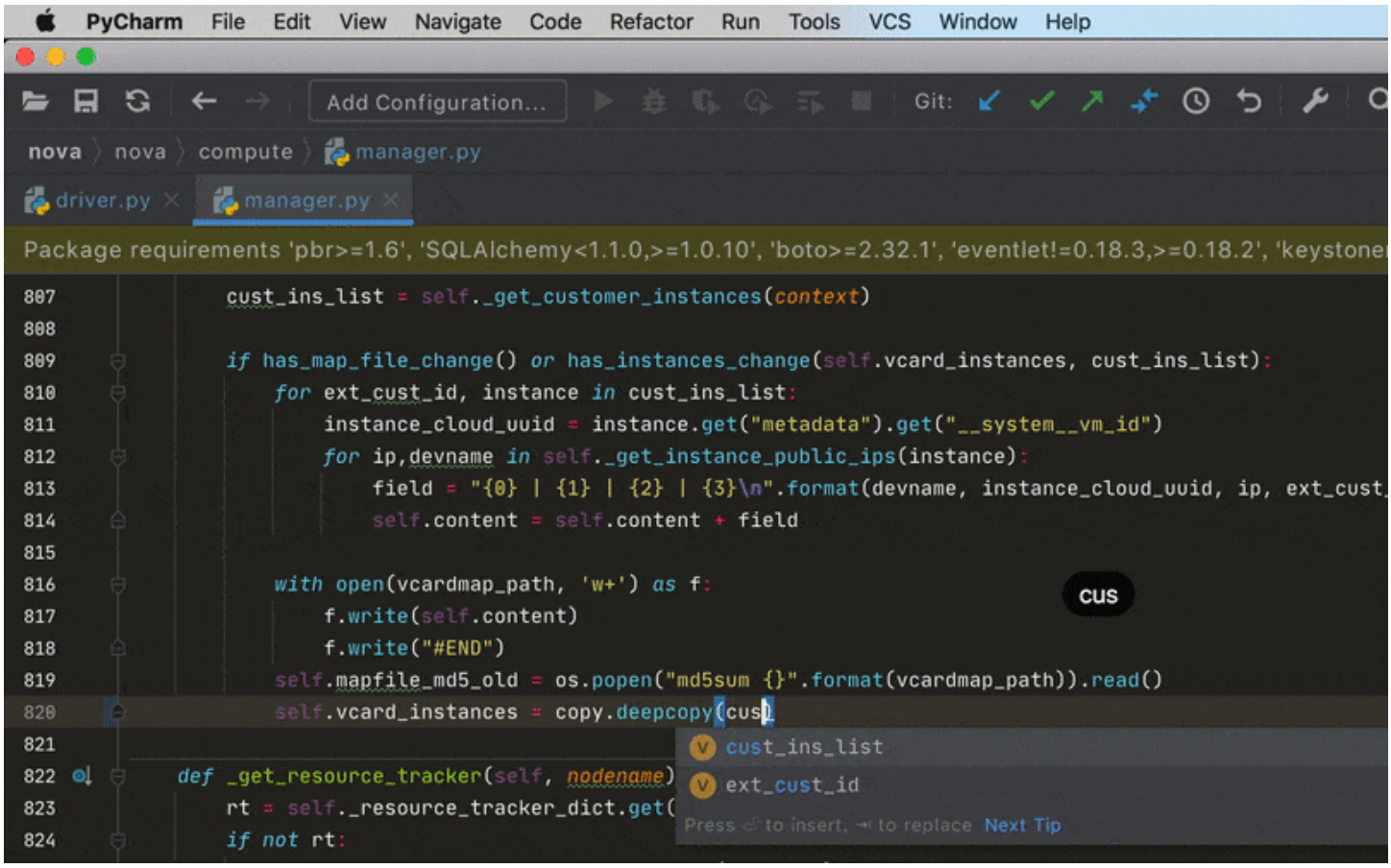
每次我在某处编写代码时，都有可能会卡住，要去其他地方看下代码的实现，才能继续写下去。

有可能是该文件下的其他代码，看完后还要回来继续写原来的代码，但是在一个数千行的文件下，我要跳回到刚刚写代码的位置，事实上还是比较麻烦的。

- 可以用查找功能，搜索你刚刚的代码，进行跳转，前提你得记得你的代码。
- 也可以在原来的代码上加个固定的注释，到时直接搜索就行。

以上都比较死板，这里介绍一个更好的办法。

你只要按下快捷键：⌘ + ⇧ + delete，不管你在哪里，是同文件下，还是不同文件下，都可以回到你最后编辑的位置，下面我来演示下

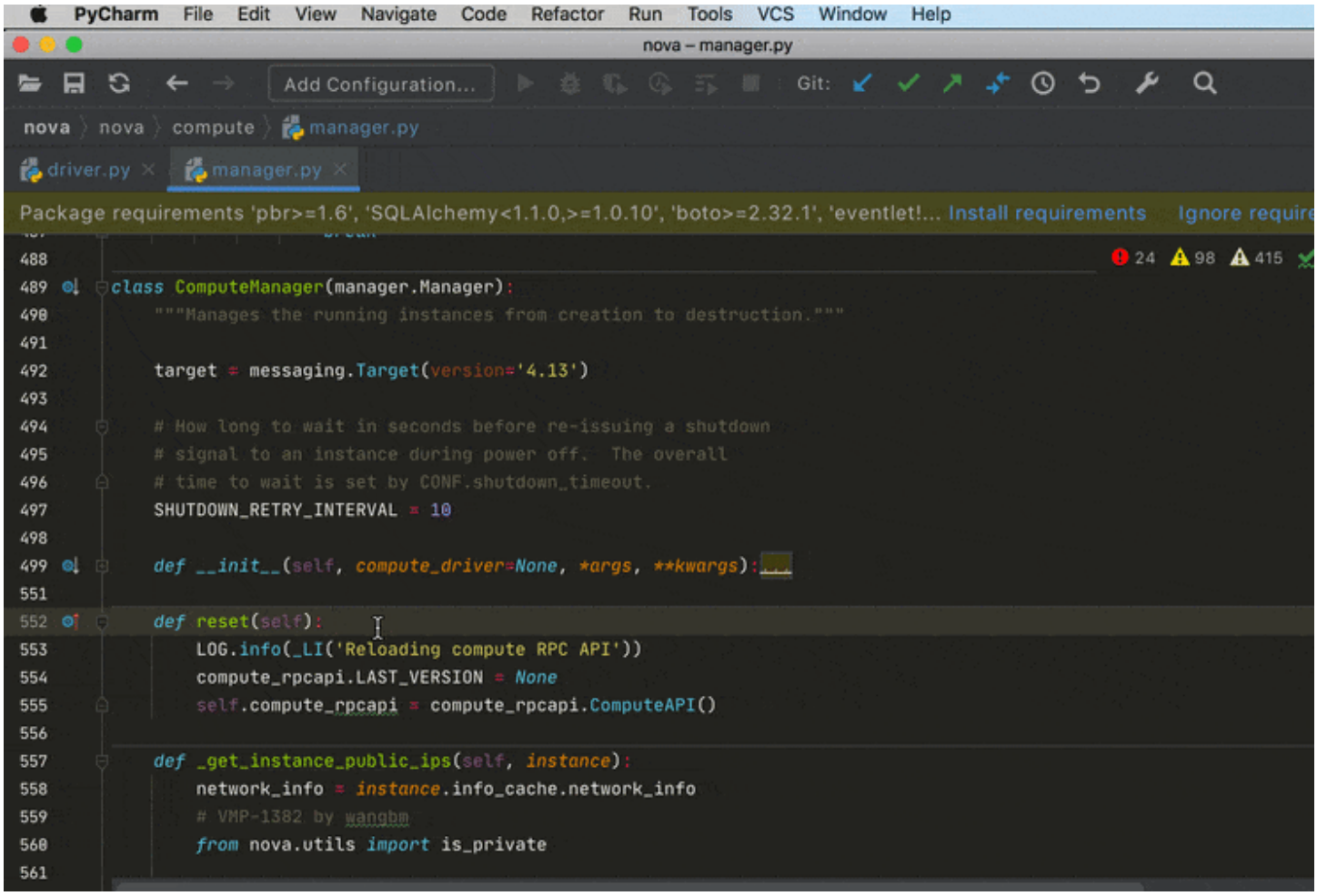


> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

6.9 【导航技巧 02】在子类方法中快速进入父类方法

如下图所示

1. 当前光标处于子类的 reset 方法
2. 按下快捷键 ⌘ + U， 就会进入父类的 reset 方法



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

6.10 【导航技巧 03】前进/后退 到上次"点击"的地方

本节写的前进/后退，是指光标点击的位置。

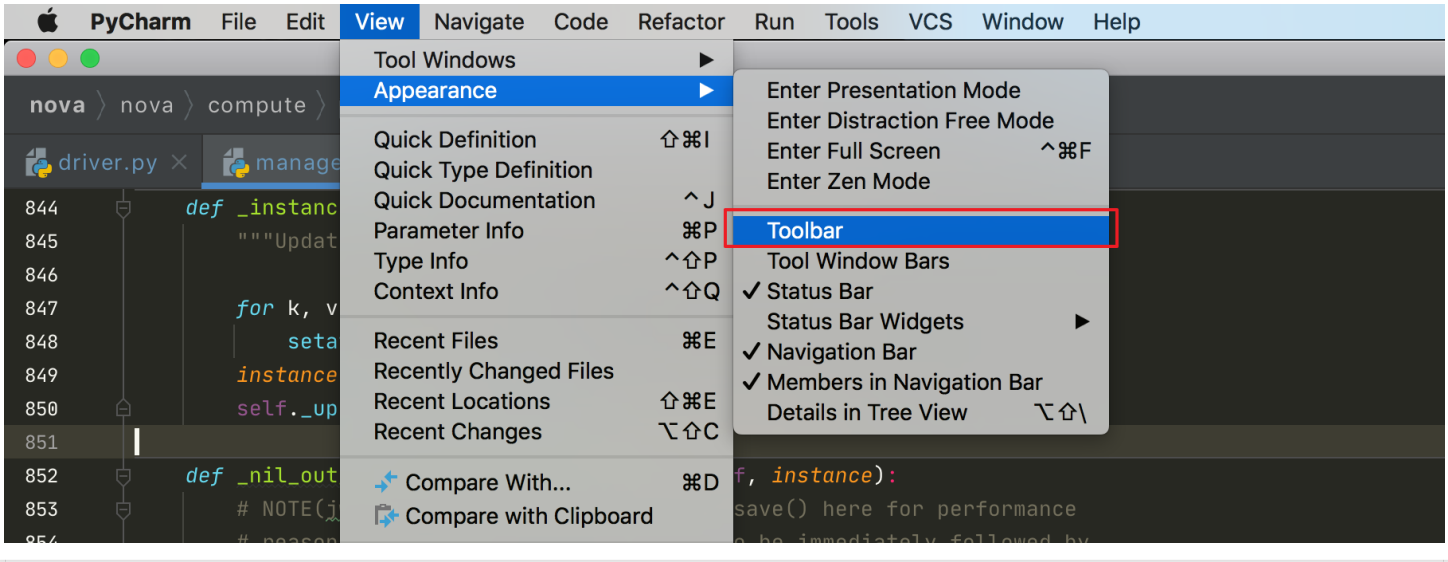
有三种方法

第一种：使用工具栏

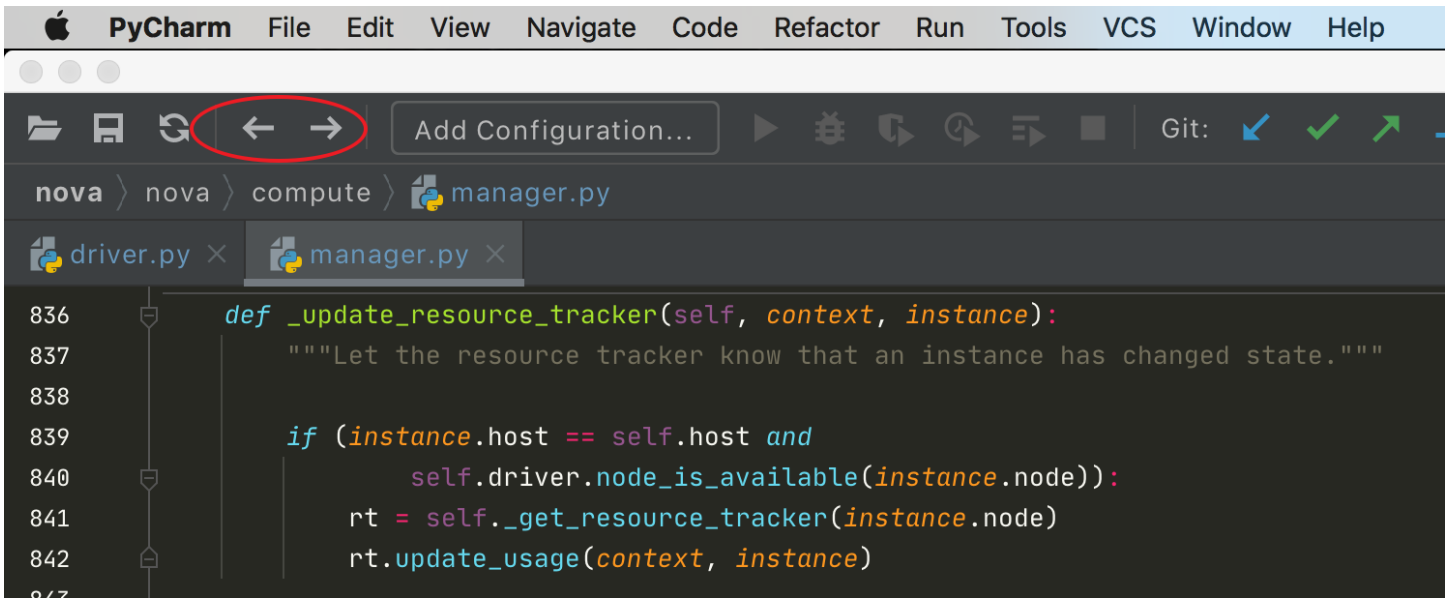
首先要让 PyCharm 显示这两个按钮

因为你的界面上可能没有这两个按钮

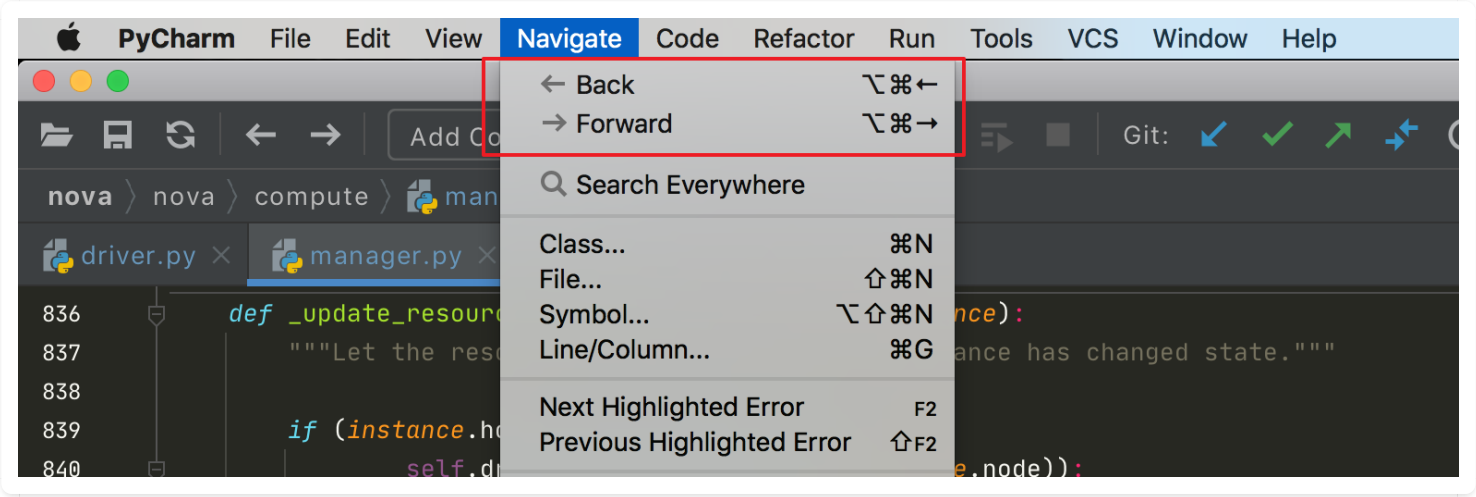
怎么打开这个工具栏呢，看下图



设置显示工具栏后，在界面上就会有如下两个按钮



第二种：使用菜单栏



第三种：使用快捷键

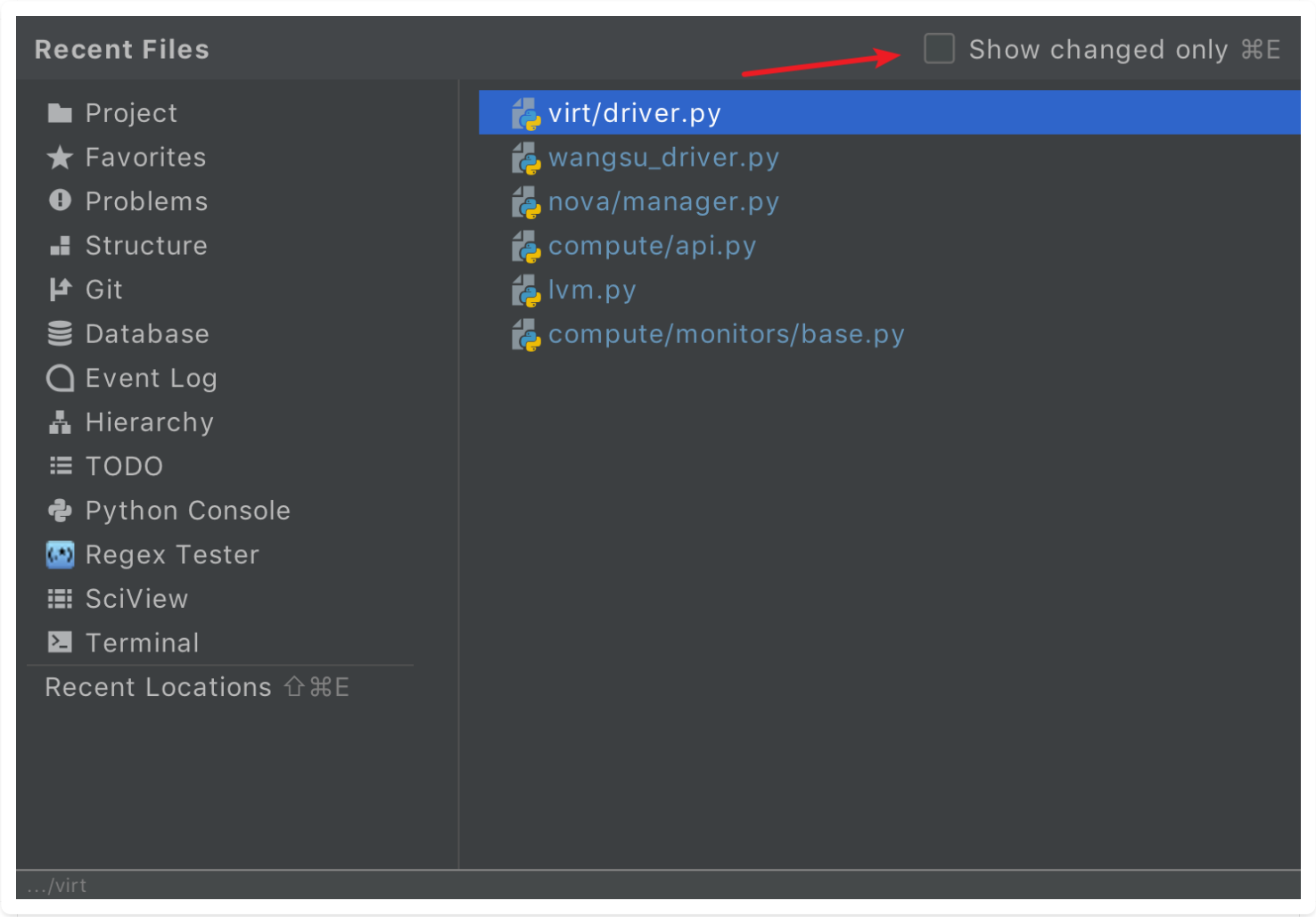
后退：回到上一次光标的位置，⌥ + ⌘ + ←

前进：回到后一次光标的位置，⌥ + ⌘ + →

6.11 【导航技巧 04】显示最近打开过的文件

如果你关掉了一个文件标签页，而后面还想打开。再去项目树中一个一个查找打开是比较麻烦的。

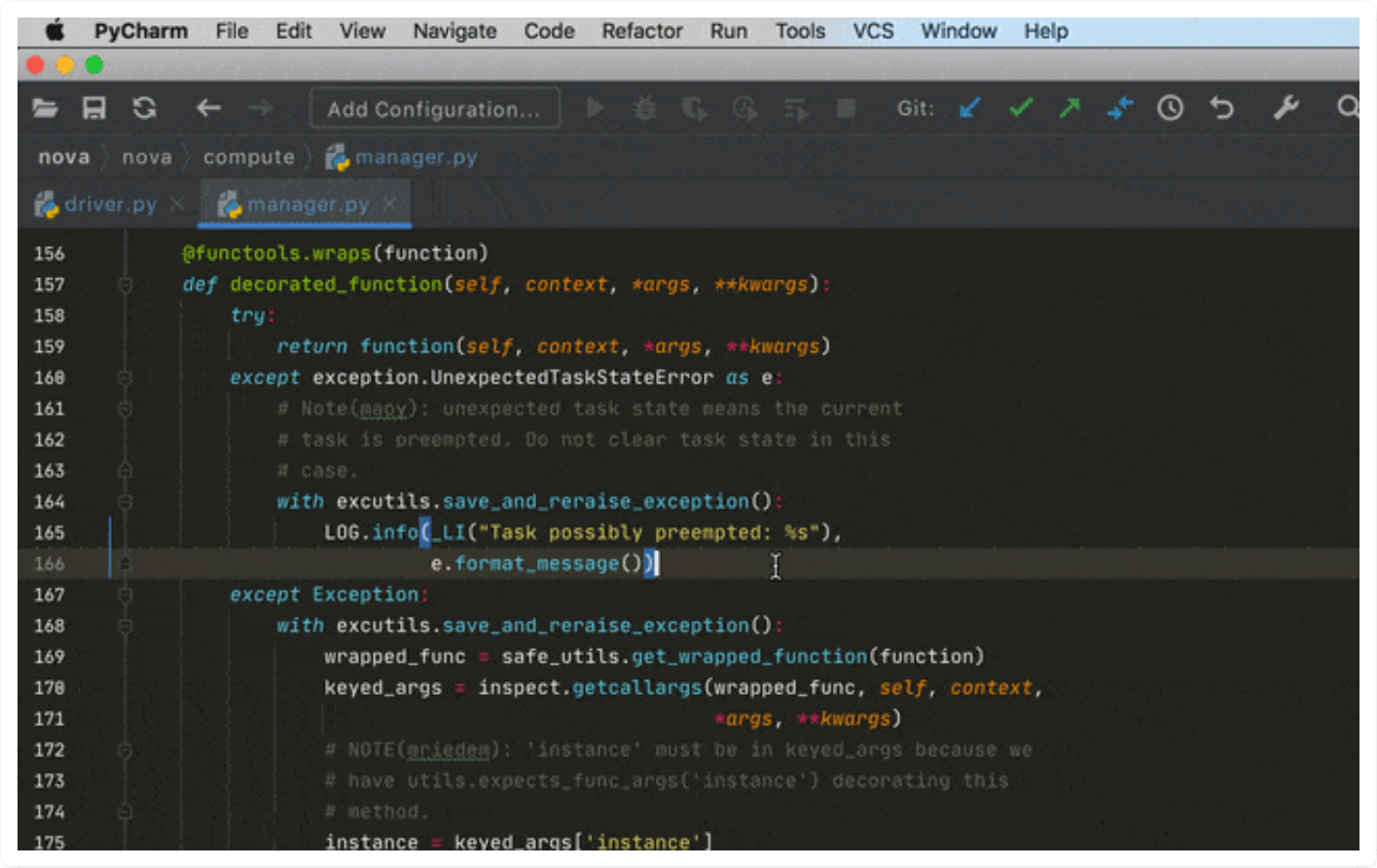
不防试试快捷键： ⌘ + E ， 可以直接调出你曾经打开过的文件，甚至可以点击如下按钮选择有修改过的文件。



6.12 【导航技巧 05】不使用鼠标，操作目录打开文件

在隐藏目录树的情况下，如果你想切换到一个还未打开的文件，你可以试一下快捷键： ⌥ + Home

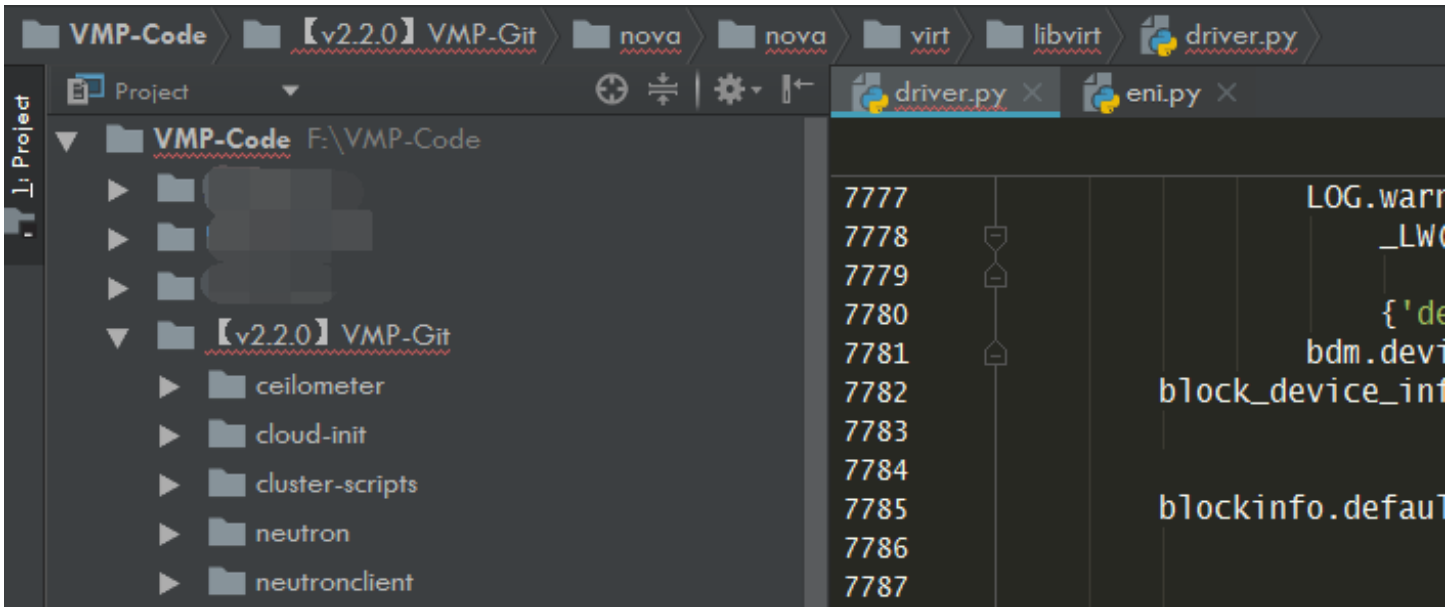
- ←：进入父级目录
- →：进入子级目录
- ↑：打开目录，展示目录下的所有文件
- Enter：进入选中的目录，或者打开选中的文件



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

6.13 【导航技巧 06】快速跳转到有 ERROR 的行

前几天打开 PyCharm，发现在导航栏这里出现了很多波浪线，有过 PyCharm 使用经验的同学，就会知道，这是代码中出现了错误。

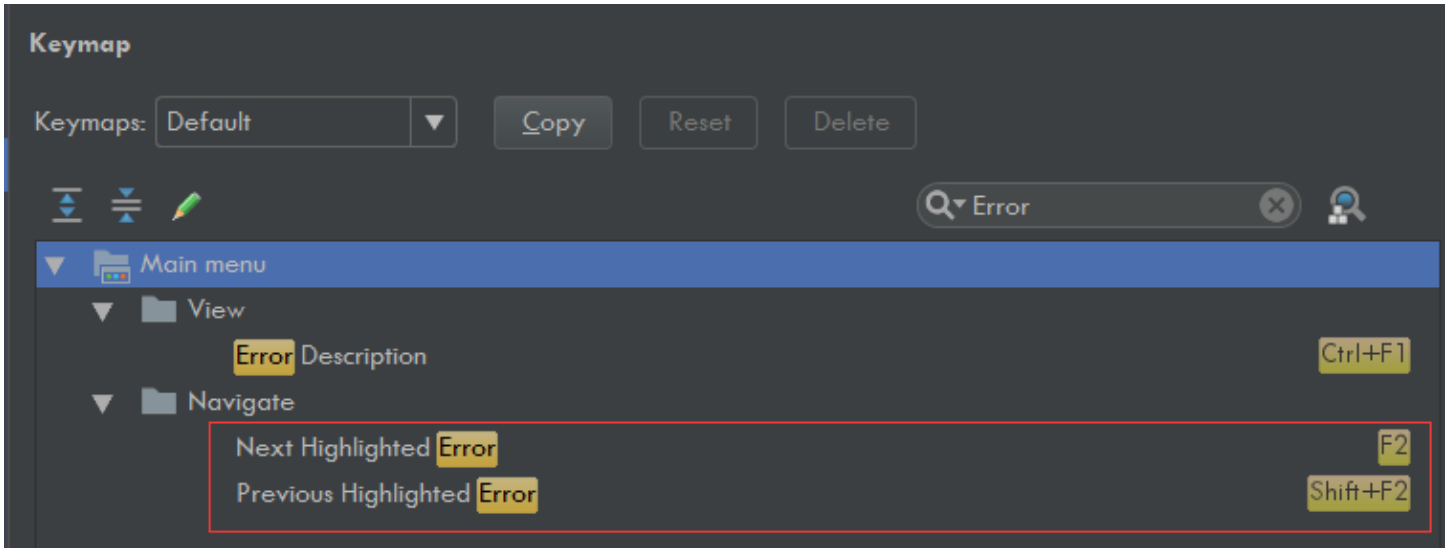


顺着波浪线，我一层一层地展开目录树，终于找到了那个包含错误的文件。由于是手误，我也不知道我改动了哪一行，看了下这个文件，有将近8000行的代码，难道一行一行地去找？

遇到问题，就应该尝试去寻找快捷方法，有没有办法，可以一下子定位到错误代码呢？

这时候，我想起了PyCharm 有提供给我们一个 Keymap 的面板，可以很方便的设置、查询快捷键。说不定我在那里可以找到我想要的答案

我在搜索框输入 Error，就找到了快速定位到错误位置的快捷键 `F2` 和 `Shift+F2` 可以快速的定位到错误行。



使用快捷键 F2 查看了下原来是这里缩进有问题。

```
            hv.spinlocks = True
            # Increase spinlock retries - value recommended by
            # KVM maintainers who certify Windows guests
            # with Microsoft
            hv.spinlock_retries = 8191
            hv.vapic = True
            guest.features.append(hv)

            if (virt_type in ("qemu", "kvm") and
                image_meta.properties.get('img_hide_hypervisor_id')):
            guest.features.append(vconfig.LibvirtConfigGuestFeatureKvmHidden())
```

对应的快捷键还有一个：⇧ + F2

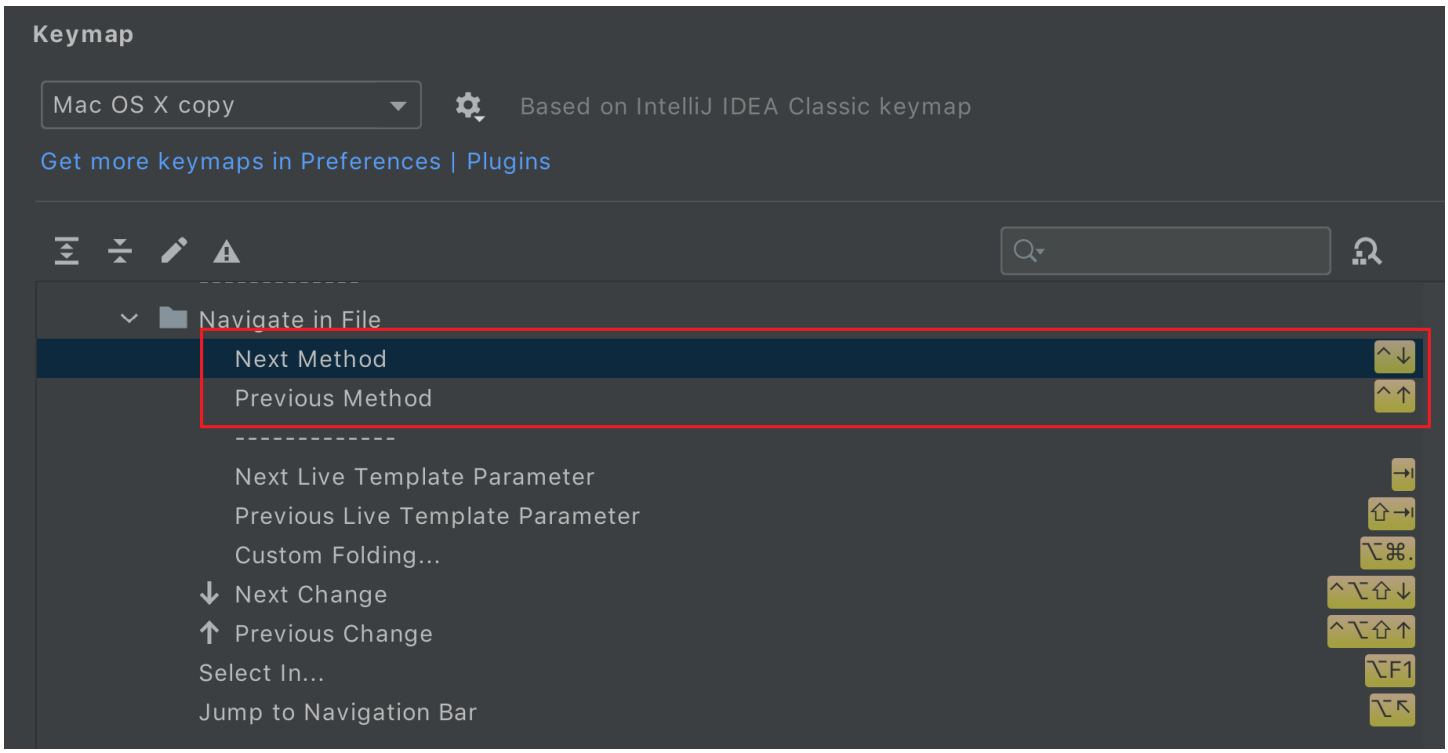
F2：跳转到下一个有错误的行

⇧ + F2：跳转到上一个有错误的行

6.14 【导航技巧 07】跳转到上/下一个方法

PyCharm 原生跳转到上/下一个方法的快捷键是

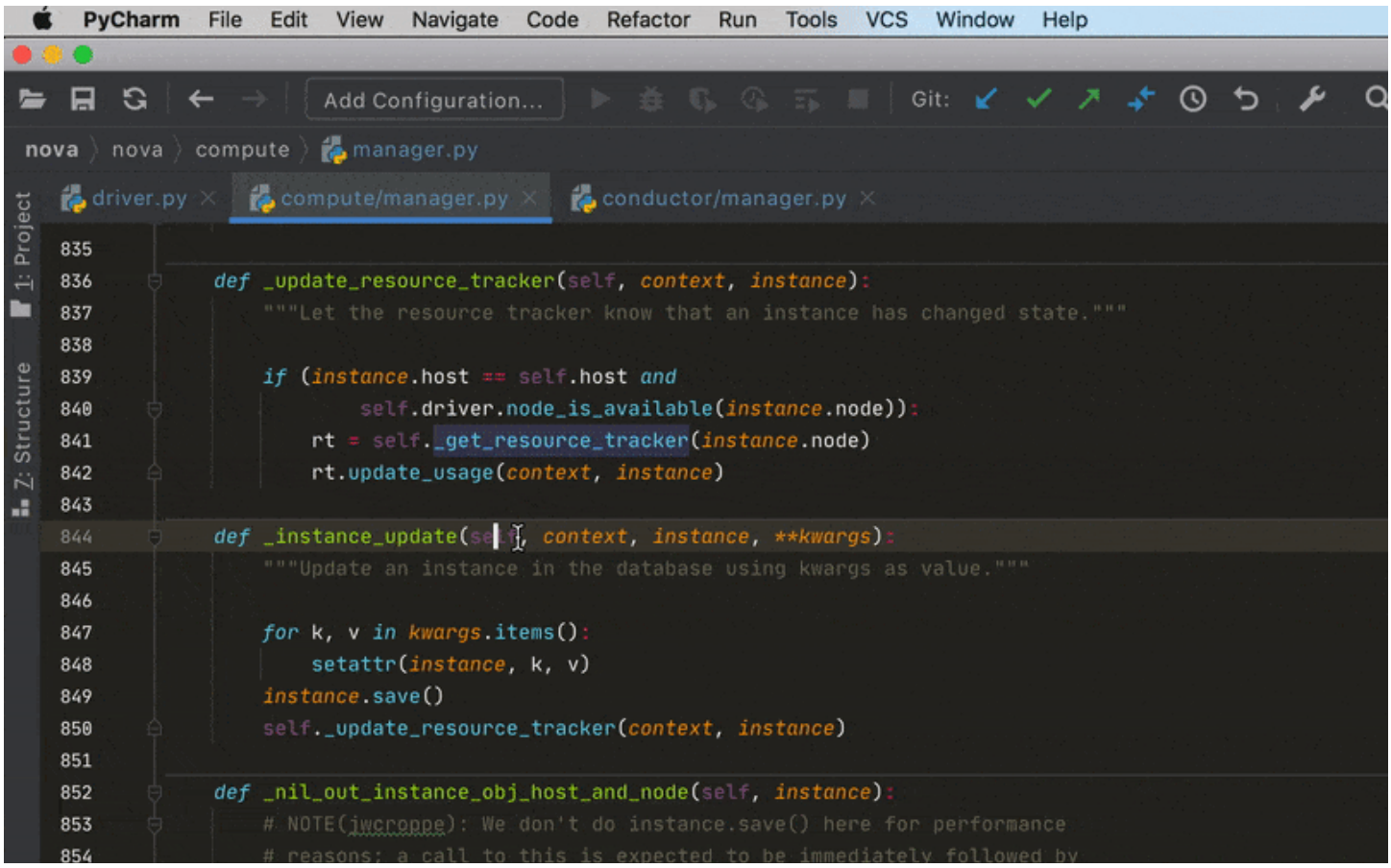
- ^ + ↑ ： 跳转到上一个方法
- ^ + ↓ ： 跳转到下一个方法



但是这组快捷键在 Mac 上跟系统快捷键冲突了。

因此 我将其改成

- ^ + ⇧ + ↑ ： 跳转到上一个方法
- ^ + ⇧ + ↓ ： 跳转到下一个方法

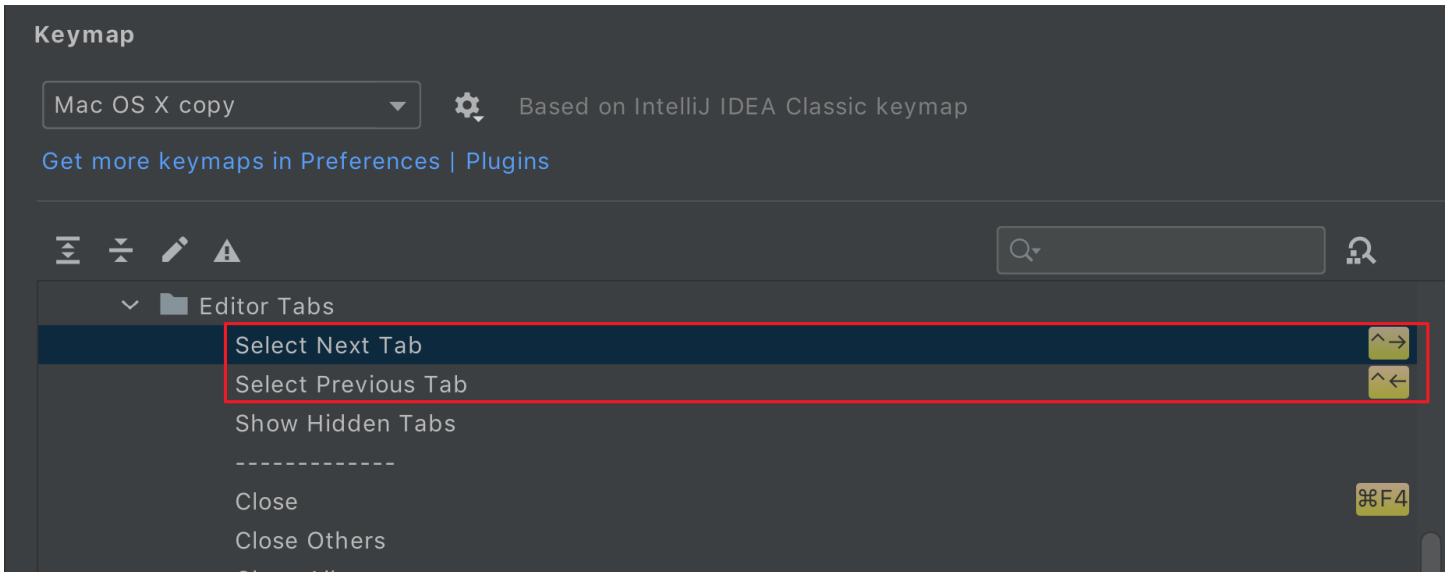


> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

6.15 【导航技巧 08】向左/向右切换标签页

PyCharm 原生切换到左/右标签页的快捷键是

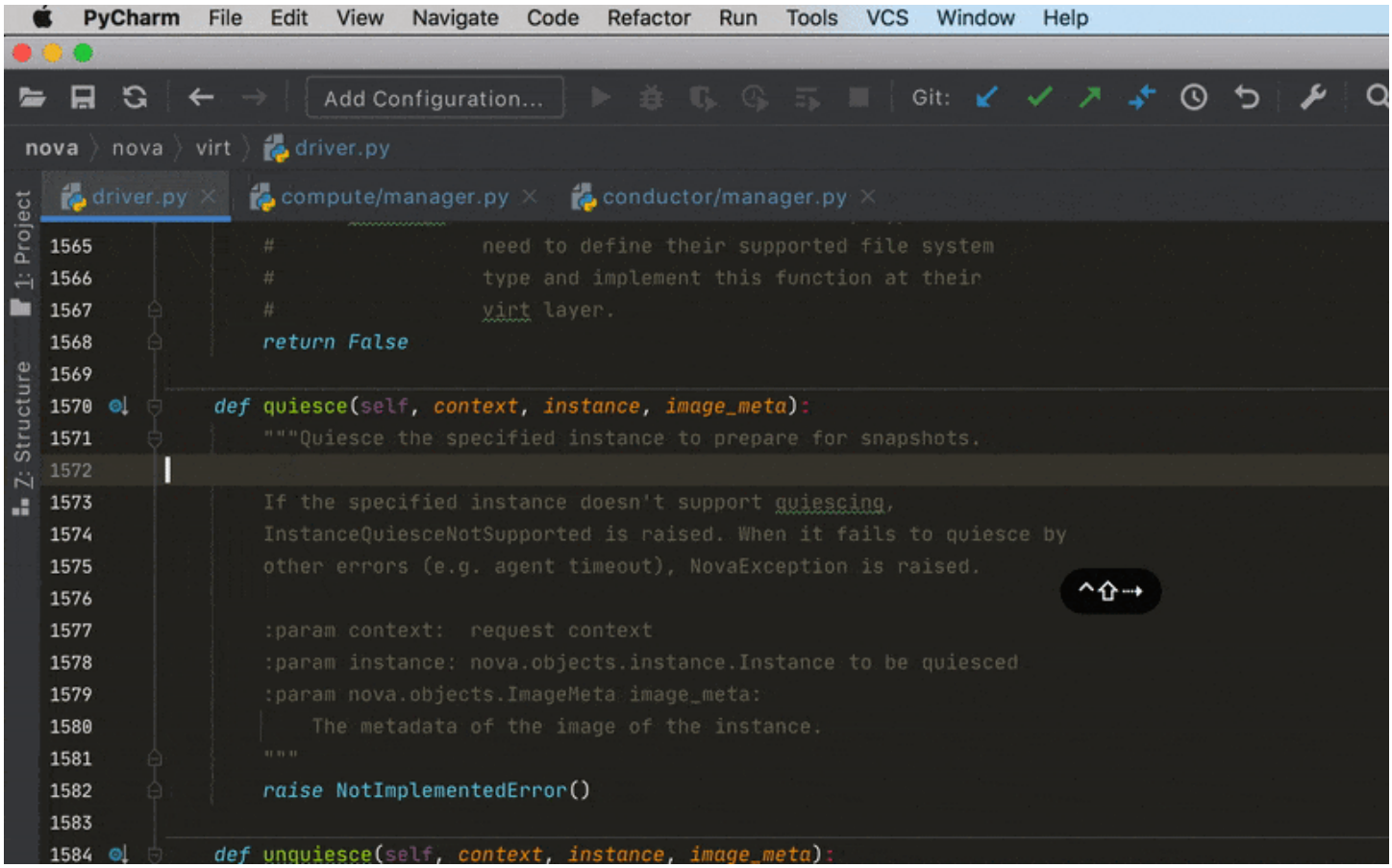
- ^ + ← ： 切换到左边标签页
- ^ + → ： 切换到右边标签页



但是这组快捷键在 Mac 上跟系统快捷键冲突了。

因此 我将其改成

- ^ + ⇧ + ←： 切换到左边标签页
- ^ + ⇧ + →： 切换到右边标签页

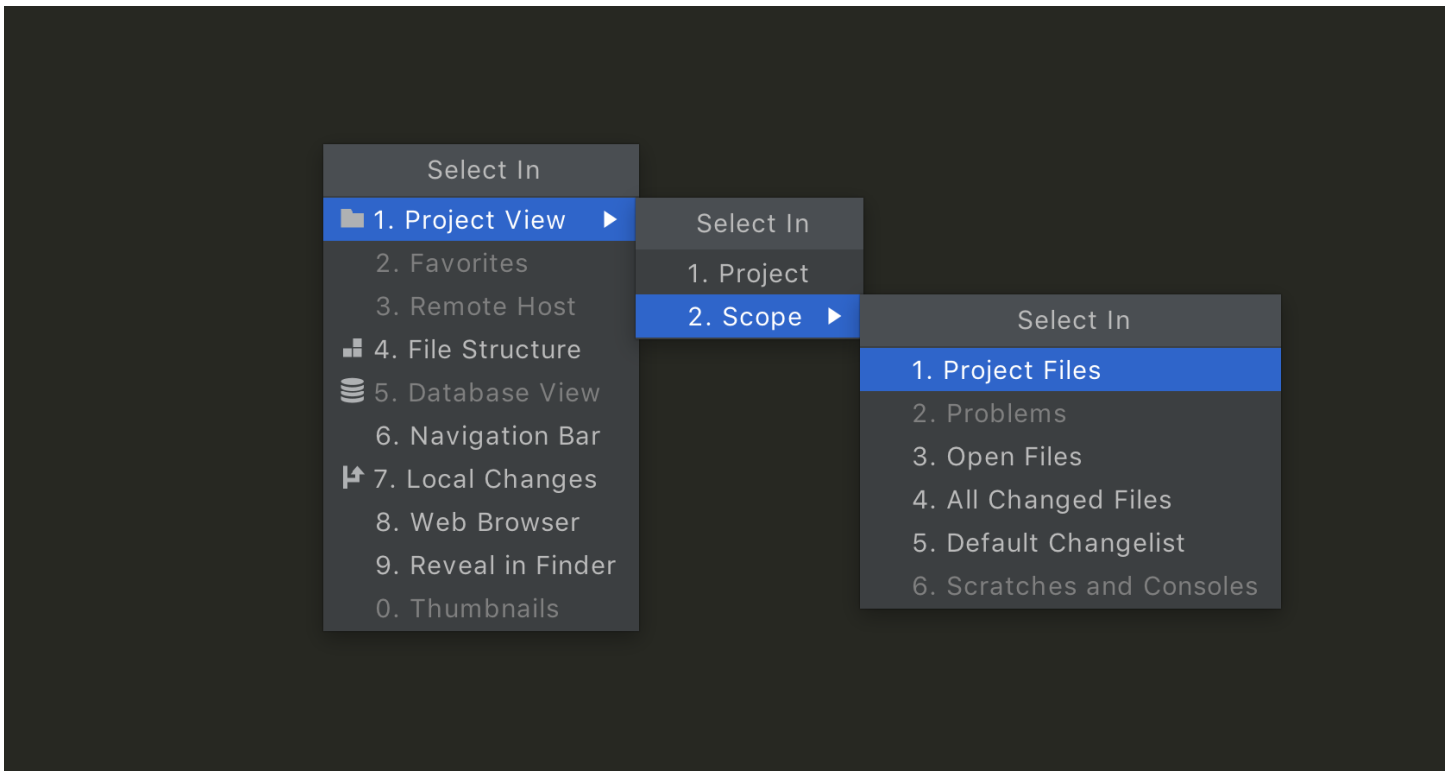


> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

6.16 【导航技巧 09】快速打开文件可用的工具栏

使用快捷键：⌥ + F1

会弹出一个窗口，窗口会显示该文件可用的工具窗口有哪一些，使用方向键进行选择。



6.17 【导航技巧 10】学会跨级别跳转代码块

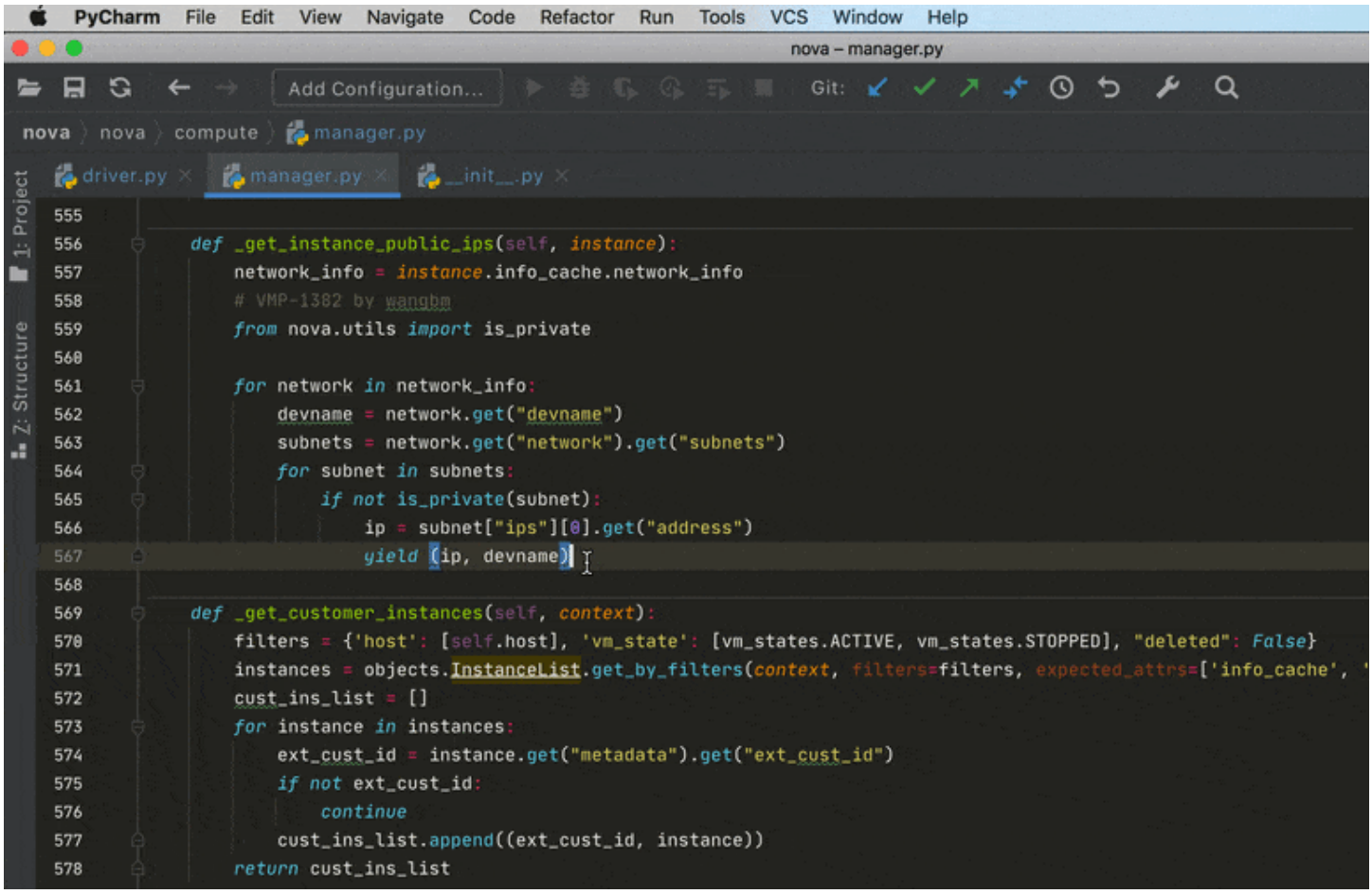
根据代码块的在模块中的位置，可以将代码块分为：

1. 行代码块：
2. 流程控制代码块
3. 函数内函数代码块
4. 函数代码块
5. 类代码块

在 PyCharm 中如何实现跨级别代码块（从下一级跳到上一级，不可逆向）的跳转呢？

只要记住这一组快捷键就可以：

- ⌘ + [：跳到上一级代码块开始的地方
- ⌘ +]：跳到上一级代码块结束的地方



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

6.18 【导航技巧 11】善用 TODO 记录待办事项

一个程序员，如果能够一天都只和代码打交道，是一件多么难得的事情。

可能外行人不知道，做为同样是程序员的你，是不是和我有一样的烦恼。

代码写着写着，测试突然就喊道：小明，你的代码有bug，ug，g（回声）。。

代码写着写着，运维突然一个弹窗：小明，这个线上问题赶紧排查一下。。

代码写着写着，产品突然就跳出来：小明，能做一个根据手机壳颜色自动改变app主题的app不？？

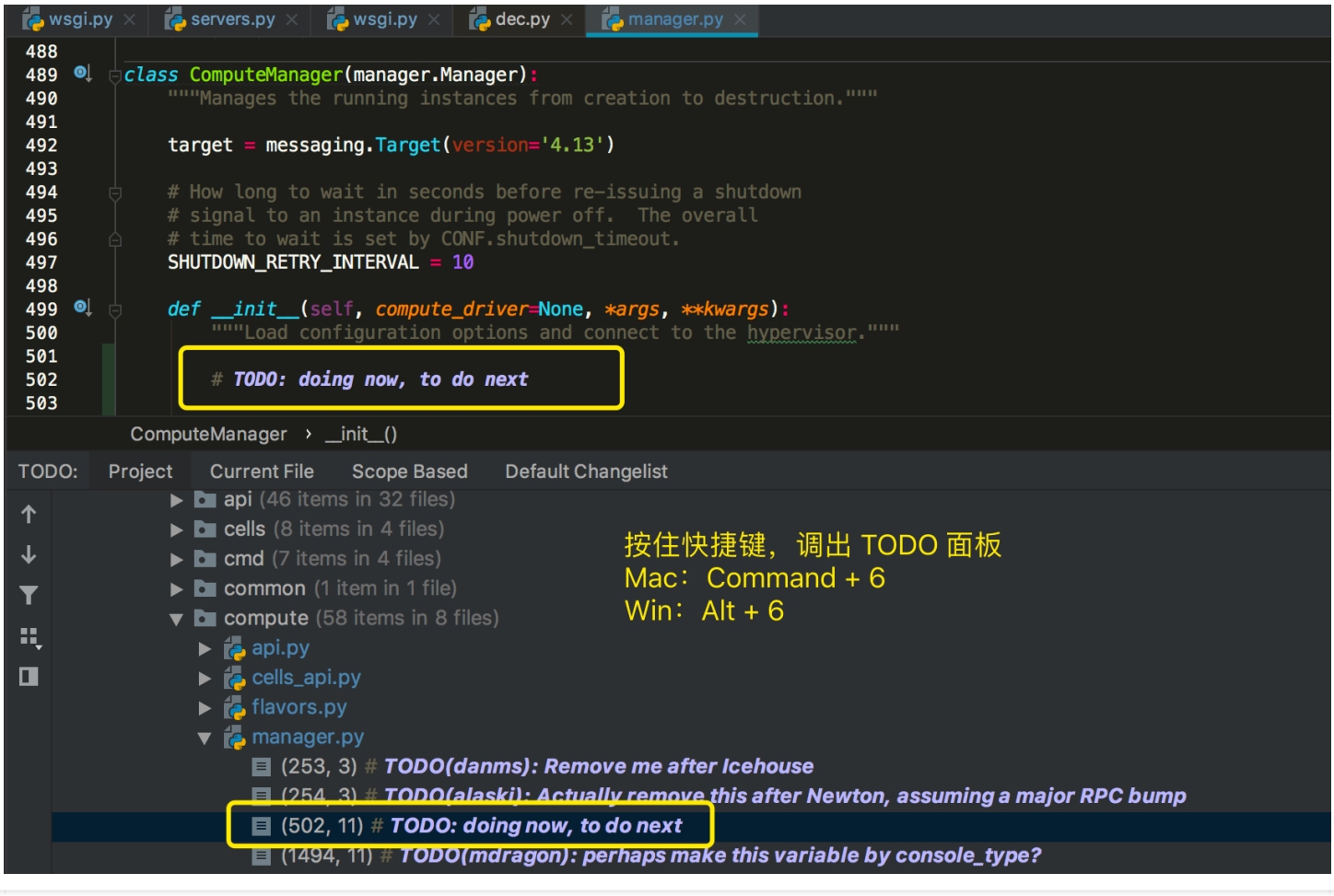
这样的噩梦每天都在重复不间断地上演着，或许我知道了为什么程序员要在深夜里码代码了，因为那是白日里得不到的宁静。

所以 王建硕 在<< 入静和入世 >>一文中写道：

> *“当看到一个程序员冥思苦想的时候，不要过去打扰，甚至在极端的情况下，一句友好的问候都是多余的。”*

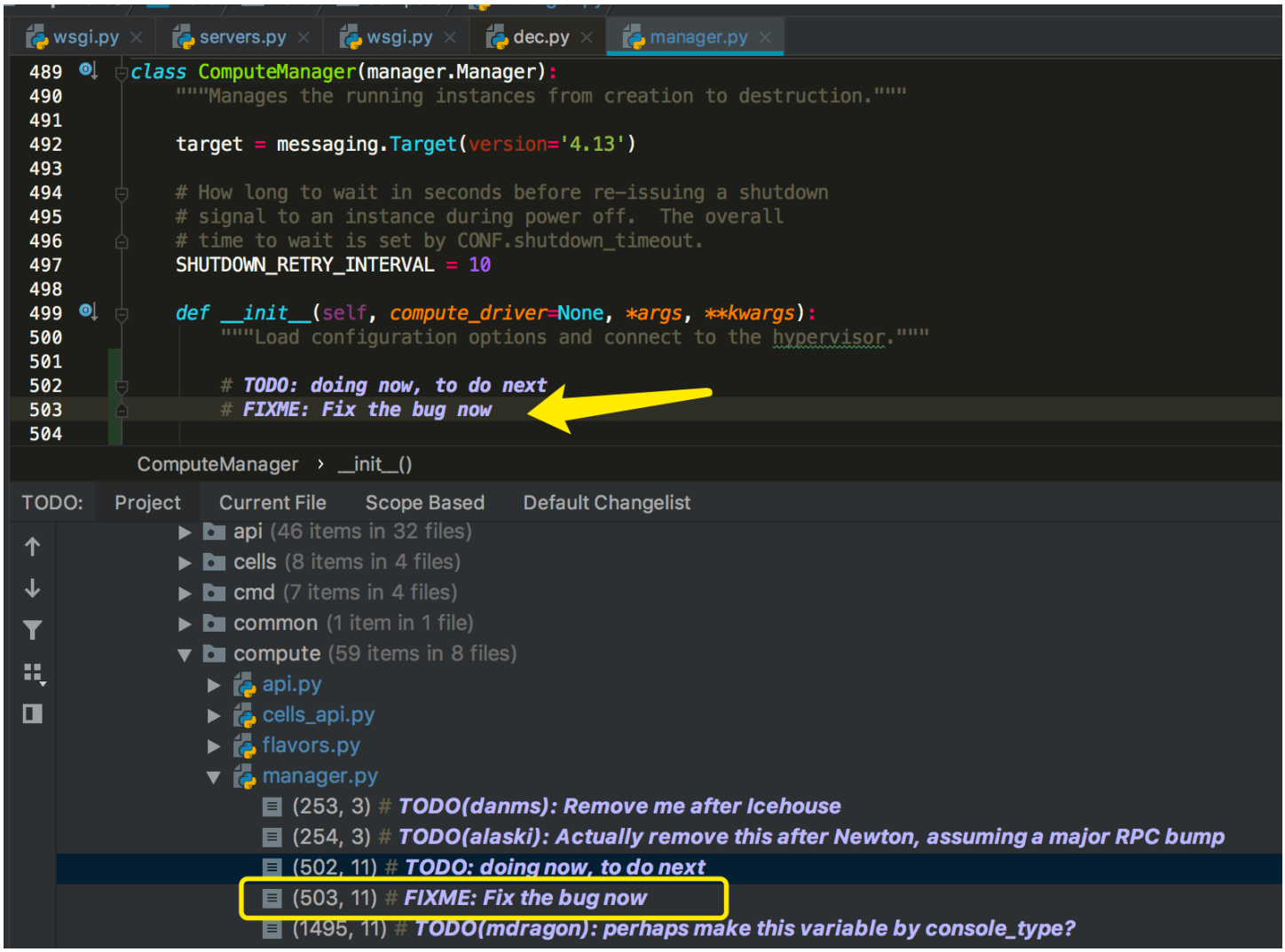
为了避免这个情况，我通常在别人打断我的时候，请对方给我一分钟的时间，使用PyCharn 的 TODO 功能快速记录下当前的思绪状态，以及下一步要做的事情。

使用方法跟注释差不多，只要固定要以 TODO 开头。然后，你要查看全局项目中的所有 TODO 事项的时候，可以使用快捷键调出 TODO 面板。如果你是 Mac， 快捷键 是Command + 6，而 Windows 是 Alt+6。



另外，我还使用这个来记录下个版本要优化的代码逻辑，要添加的功能。

如果是比较紧急的 BUG，可以使用类似 TODO 的标记 – `FIXME` 来区分紧急程度。



作者：王炳明
版本：v1.0
发布时间：2020年08月30日
微信公众号：Python编程时光
联系邮箱：wongbingming@163.com
项目主页：http://pycharm.iswbm.com
Github：https://github.com/iswbm/pycharm-guide

👆👆👆

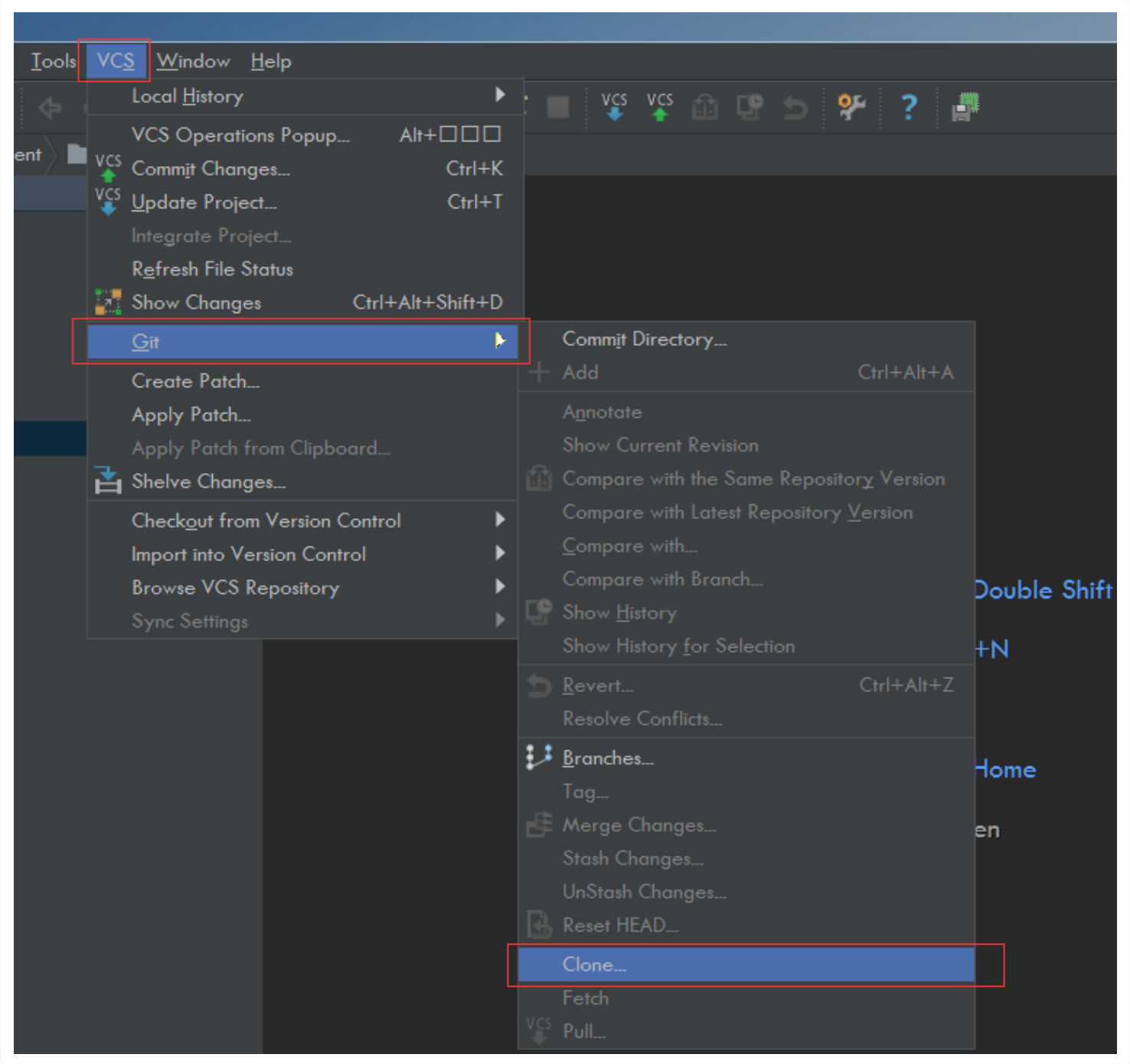
回复 "pycharm"，获取最新版 PDF

第七章：版本与管理

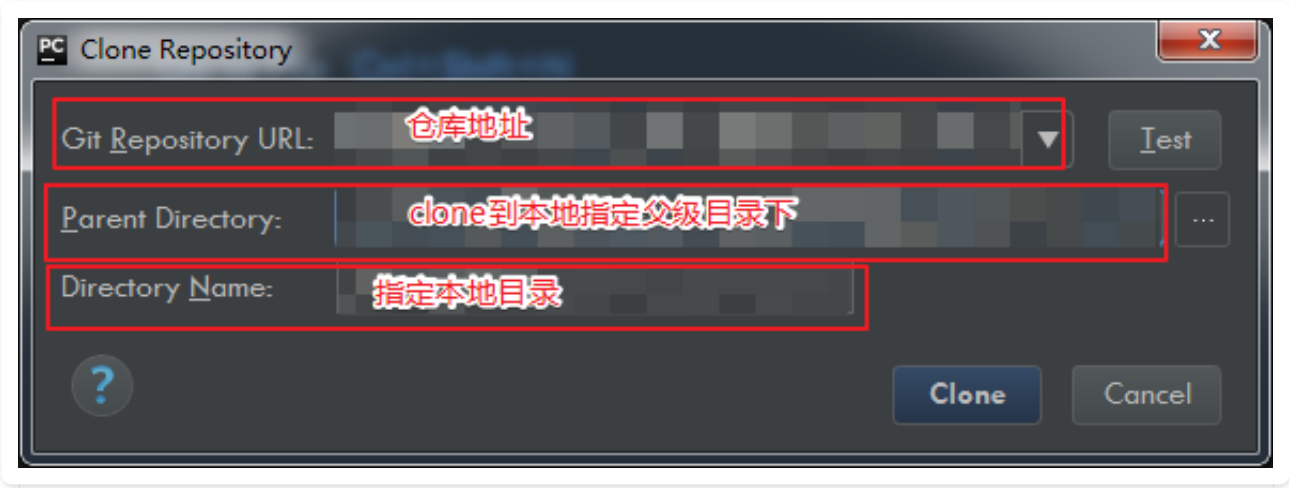
7.1 【版本管理 01】使用 Git 进行版本管理

开启版本控制

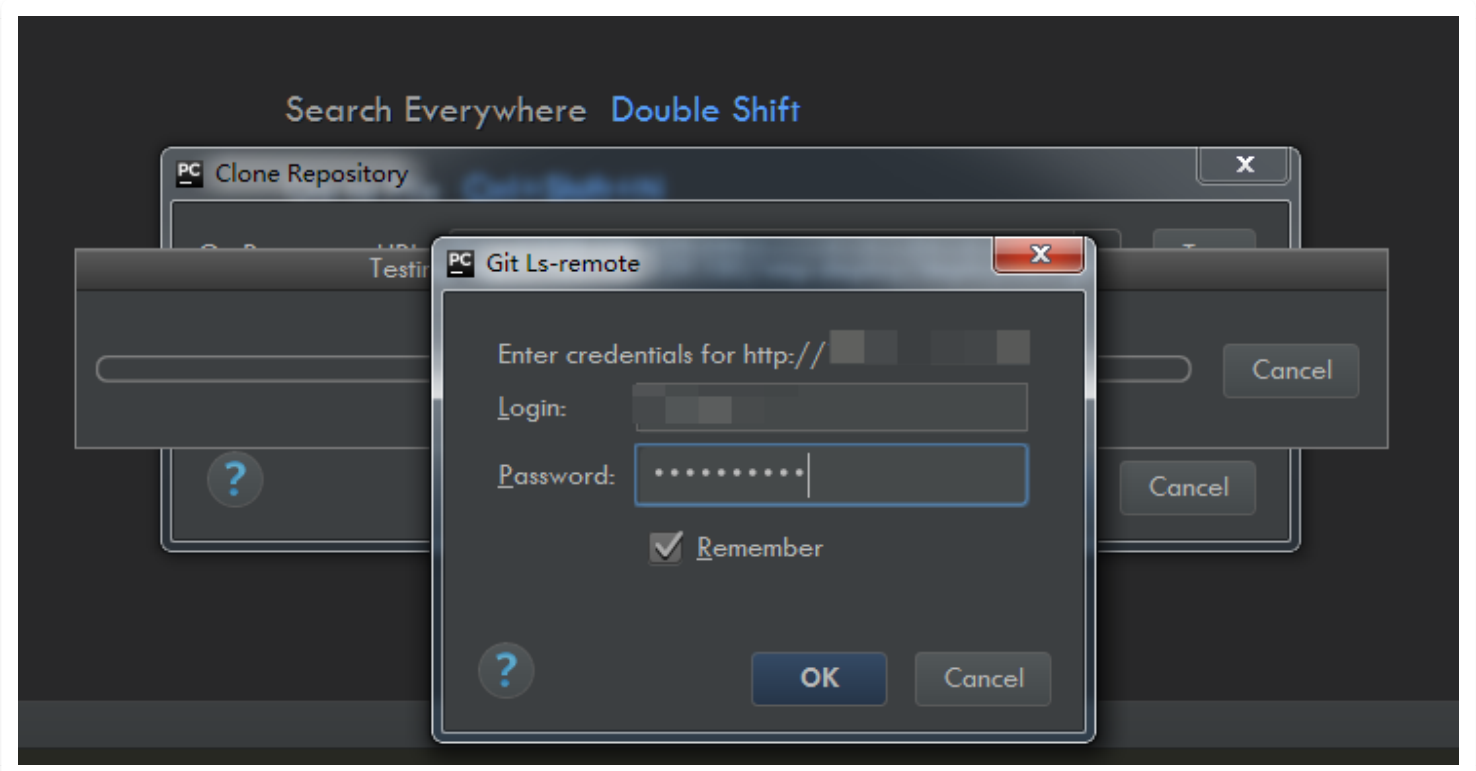
点击 `VCS` -> `Git` -> `Clone`



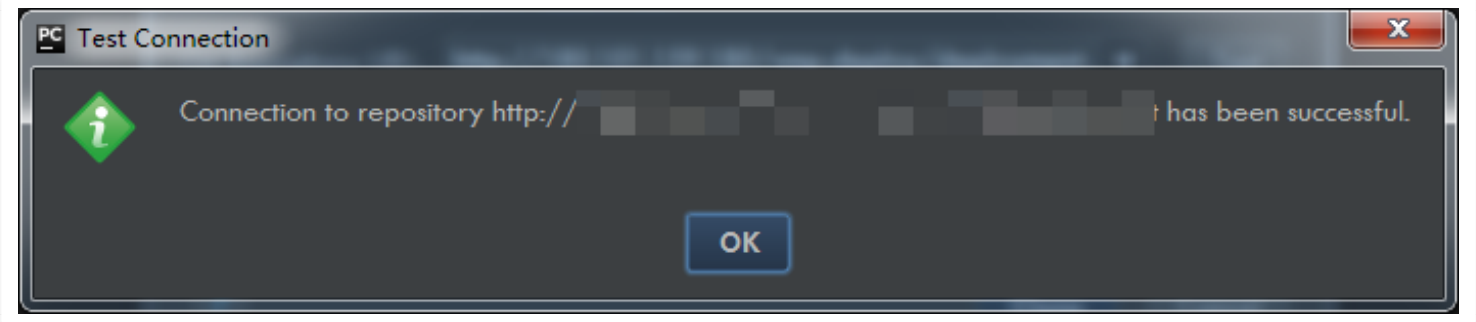
填写git仓库相关信息



点击 `Test`，会尝试连接 git 服务器，中间会让你输入登陆的帐号和密码。



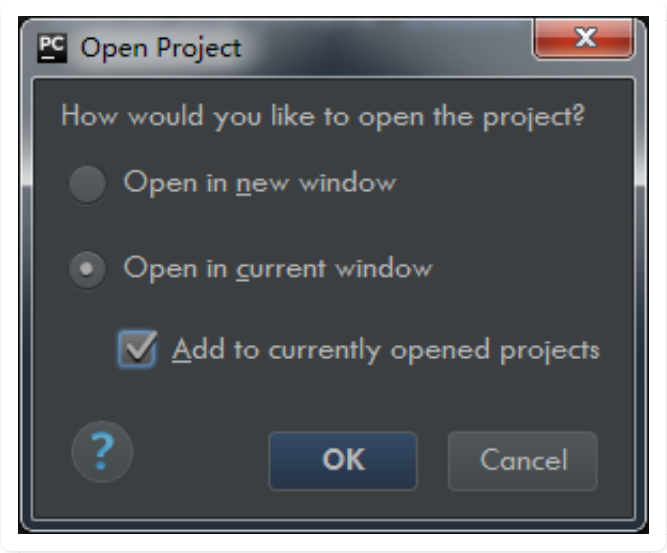
点击`OK`后，若一切正常会提示连接成功。



点击 `OK` 后，PyCharm 需要你选择如何打开这个 Git 仓库目录，是在当前窗口中打开，还是新建一个窗口？

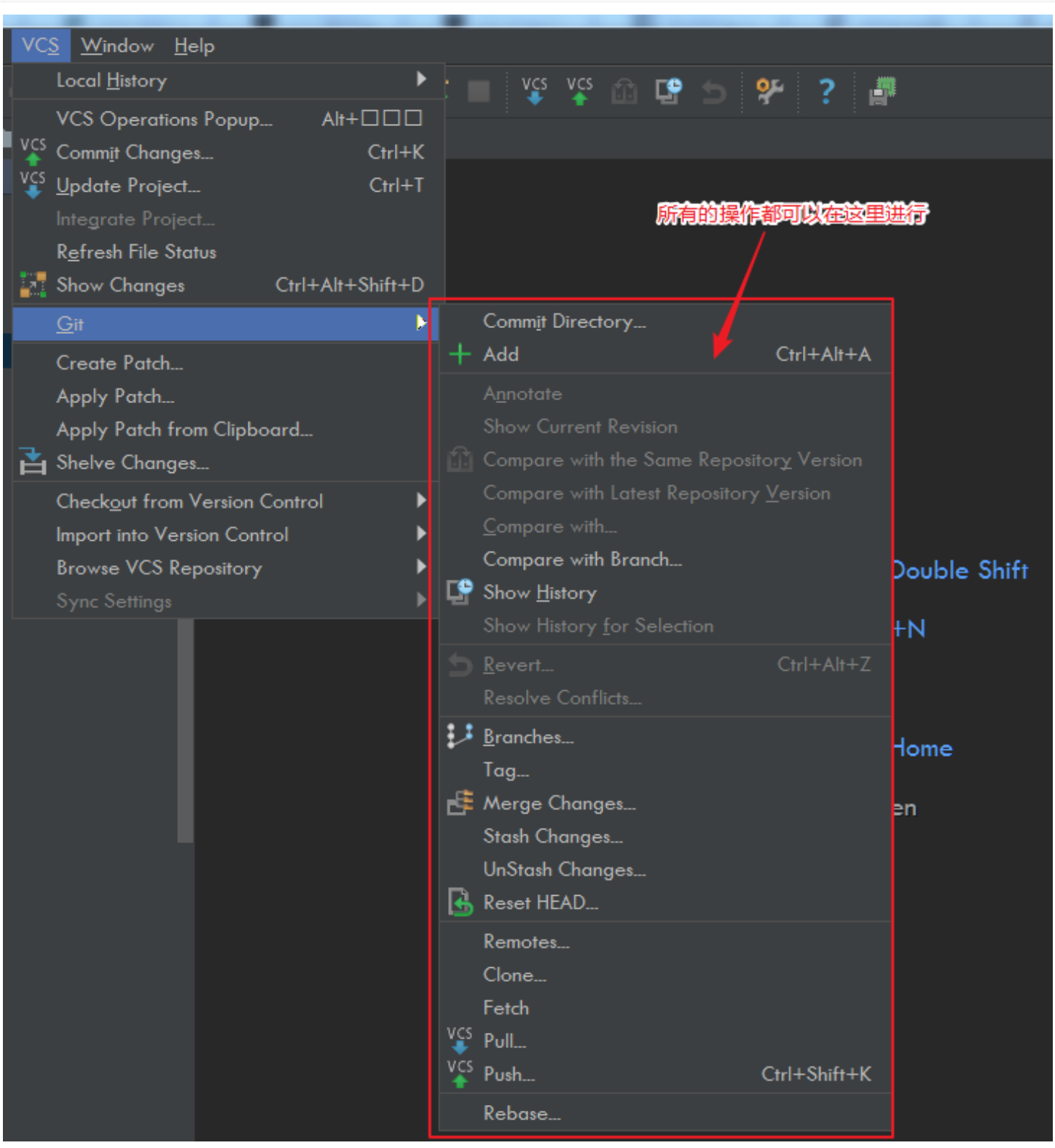
由于我在一个 PyCharm 下会有多个 Git 仓库，为了方便，我选择在当前窗口中打开（注意勾选

`Add to currently opened projects`）。



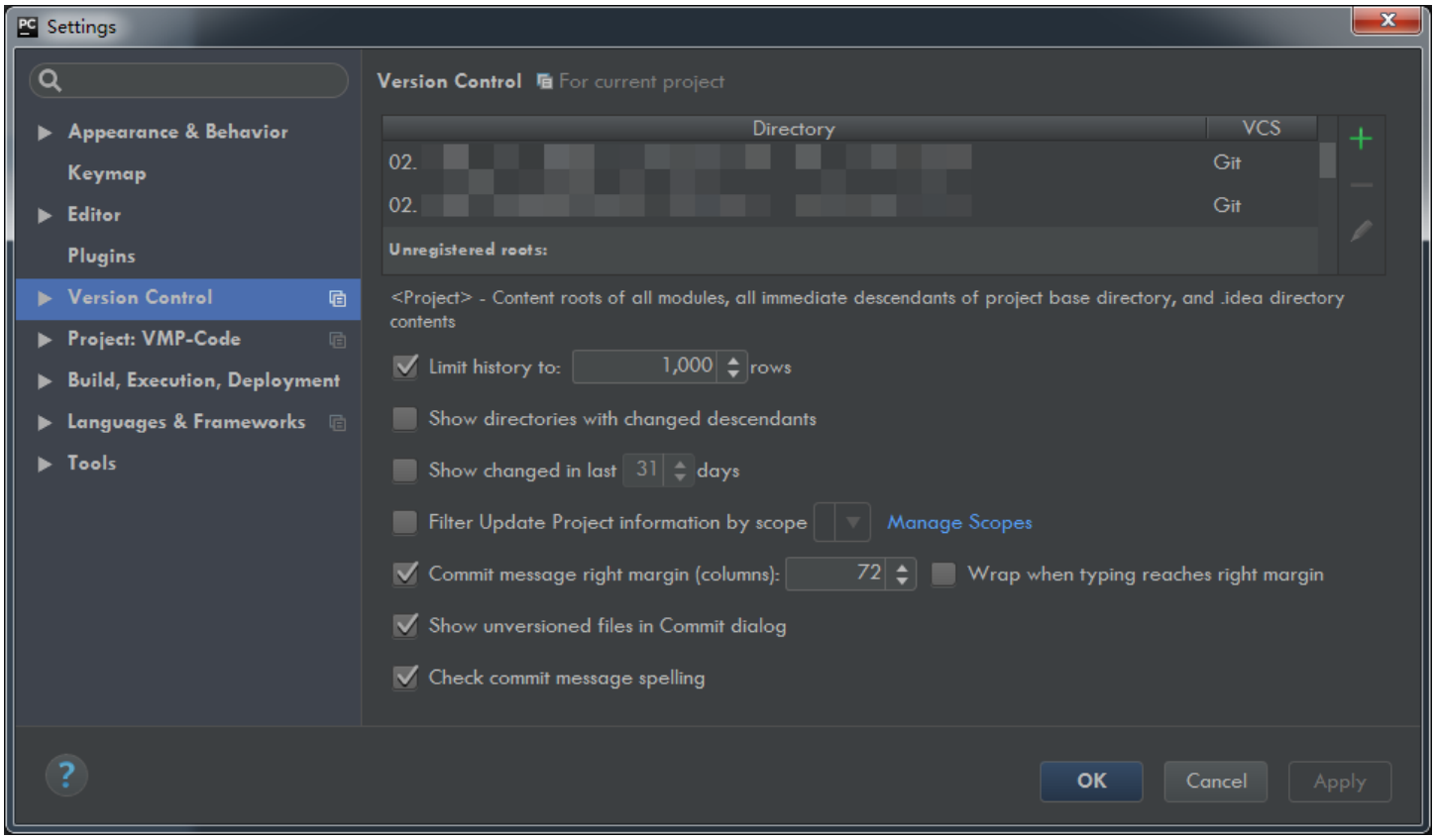
至此，Git 配置完成。

此时你可以 `VCS` -> `Git` 查看，发现之前这些灰色不可用的按钮都可以使用了。

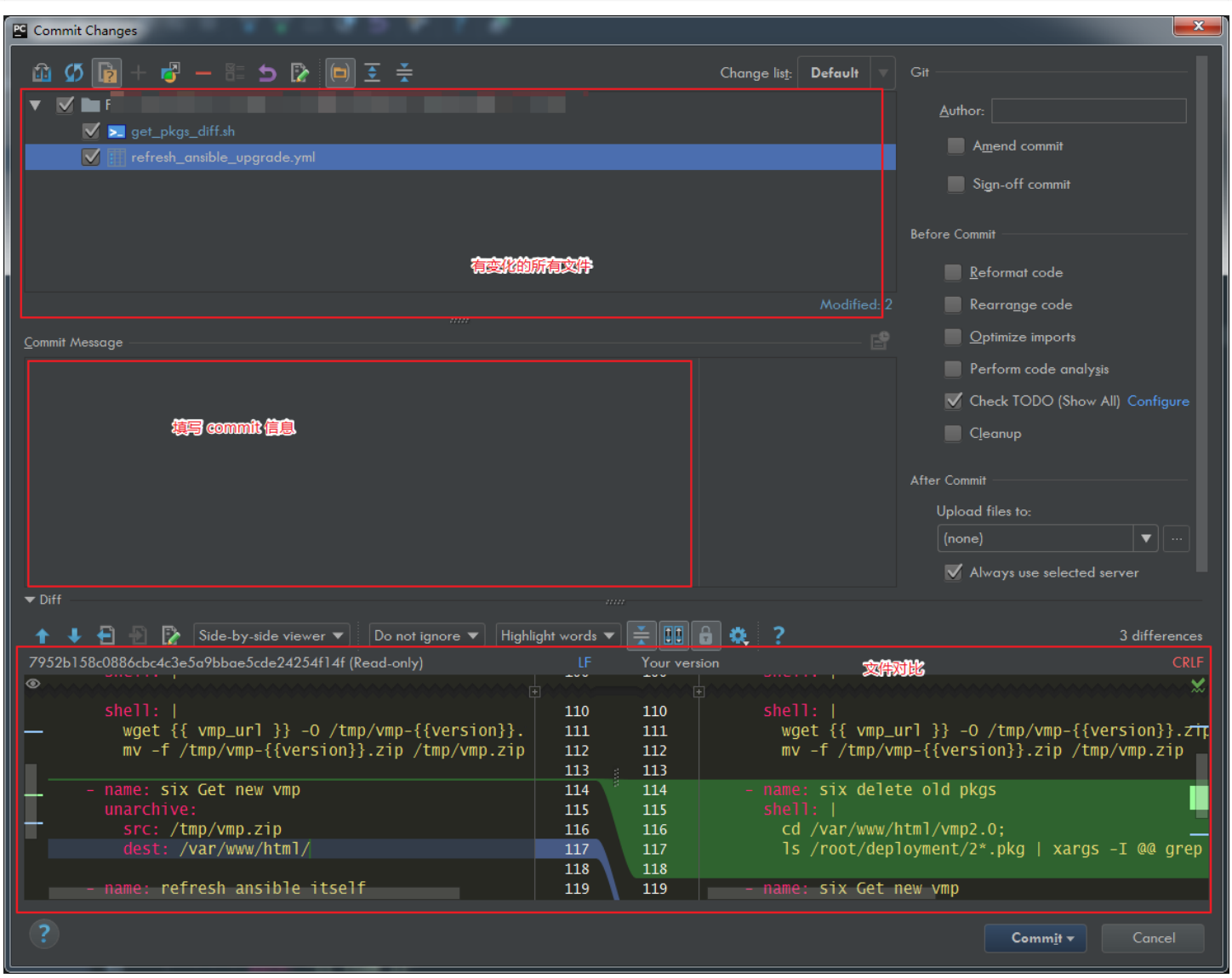


本篇重在讲解 PyCharm 的配置，关于Git 的操作，不属于本篇重点，就不再展开讲了。

若你想对已配置的Git仓库进行修改，可点击 `File` -> `Setting` -> `Version Control` 调出如下界面。



不得不说 PyCharm 的这 UI 做得可以，随便改了个东西提交一下



与 VSC 有关的快捷键

⌘ + K： 提交代码到版本控制器

⌘ + T： 从版本控制器更新代码

⌥ + ⇧ + C：查看最近的变更记录

^ + C：快速弹出版本控制器操作面板

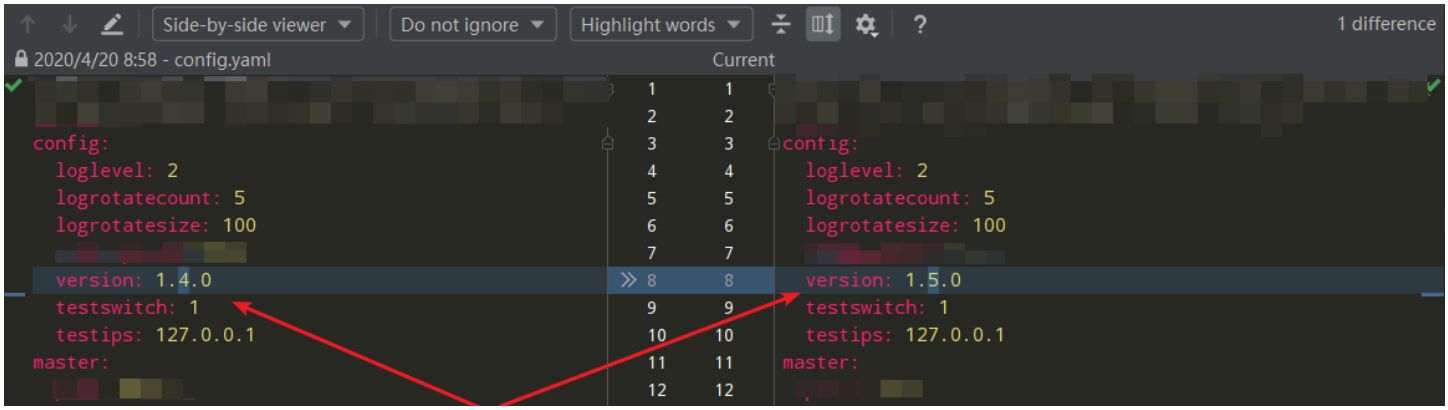
7.2 【版本管理 02】三种查看文件的修改

如果你的项目在 git 的管理之下，在你修改了文件后，你会有很多种方法来查看自己到底修改了什么？

第一种当然是使用 git diff

```
$ git diff config.yaml
diff --git a/etc/config.yaml b/etc/config.yaml
index f1beaf9..a8ed5be 100644
--- a/etc/config.yaml
+++ b/etc/config.yaml
@@ -5,7 +5,7 @@ config:
   logrotatecount: 5
   logrotatesize: 100
   listenport: 65521
-  version: 1.4.0
+  version: 1.5.0
   testswitch: 1
   testips: 127.0.0.1
 master:
```

第二种是使用之前写的 show history



第三种，也是今天要介绍的，是最简便，也是直接的方法。

在有文本变动的位置，PyCharm 会有提示，如下红色箭头标识处，点击它就可以直接查看，还可以快速回滚。

7.3 【版本管理 03】媲美beyond compare 的差异对比功能

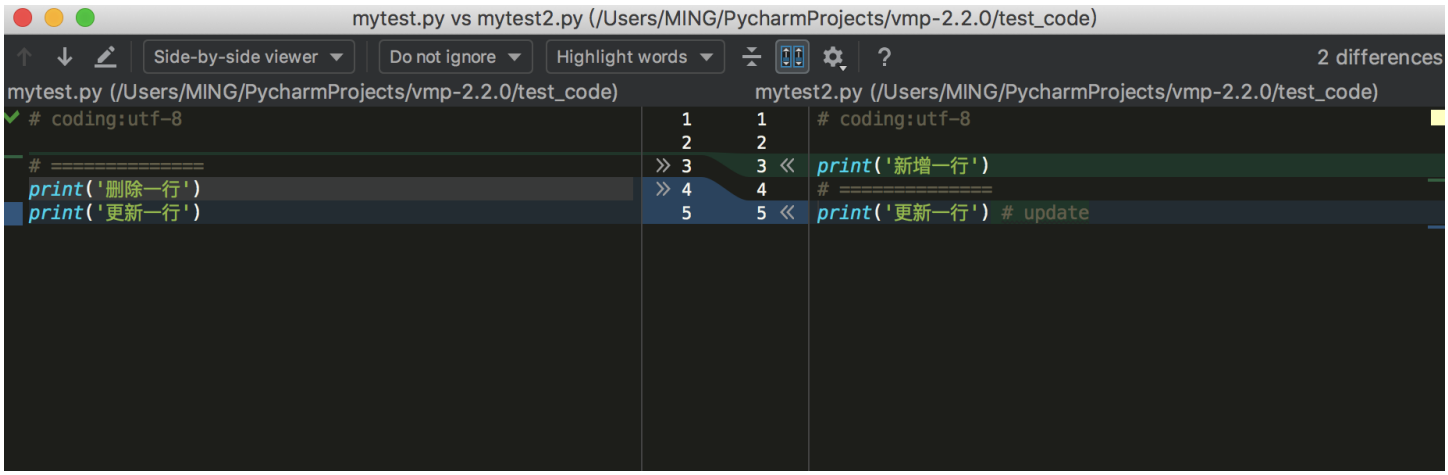
程序开发必备神器中，beyond compare 绝对可以排一号。

虽说好用，但这东西，是收费的。

如果是简单的单个文件的比对，其实可以使用PyCharm里自带的。

点击源文件，再点击`View` -> `Compare With …` -> 选择目标文件

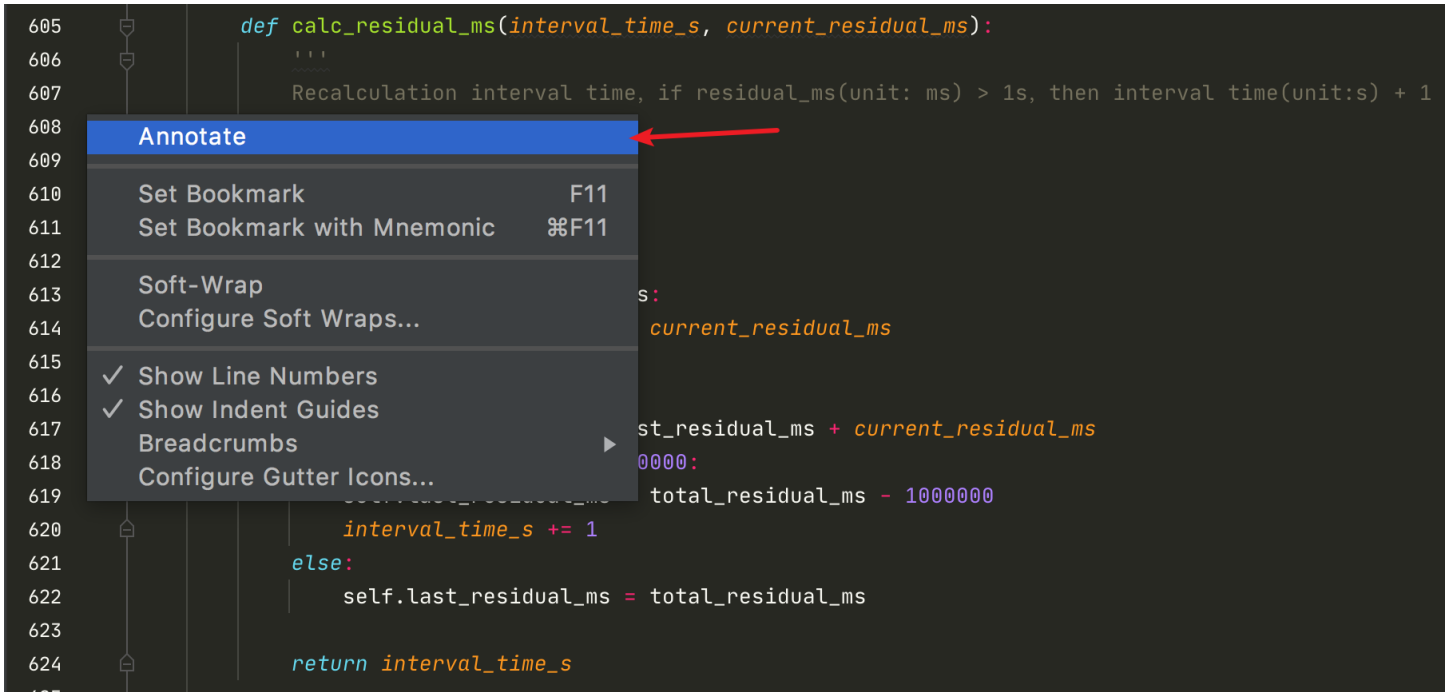
对比示例，可以查看下面这张图，UI做的还是挺好看的。



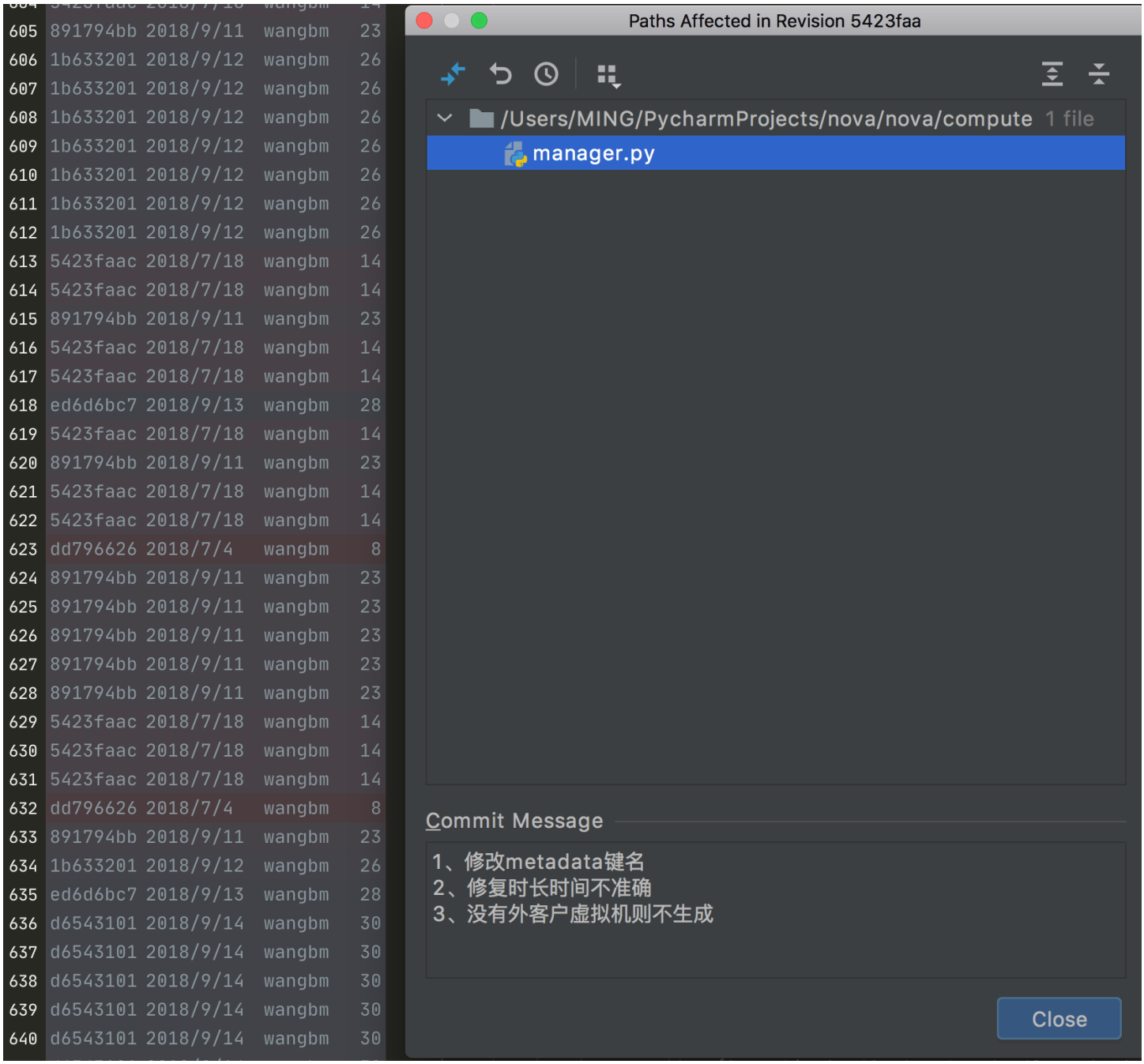
7.4 【版本管理 04】查看文件修改记录：Annotate

当你的项目处于受控状态（开启了版本控制），你对项目里的文件的修改都会留下记录。

那怎么查看这些记录呢？在编辑框的左边右键，然后选择`Annotate`



就会出来如下图所示的界面。



在这里界面里记录着，哪个人在哪一天修改了该文件的哪一行，commit 号是多少？

非常方便我们对代码进行追溯。

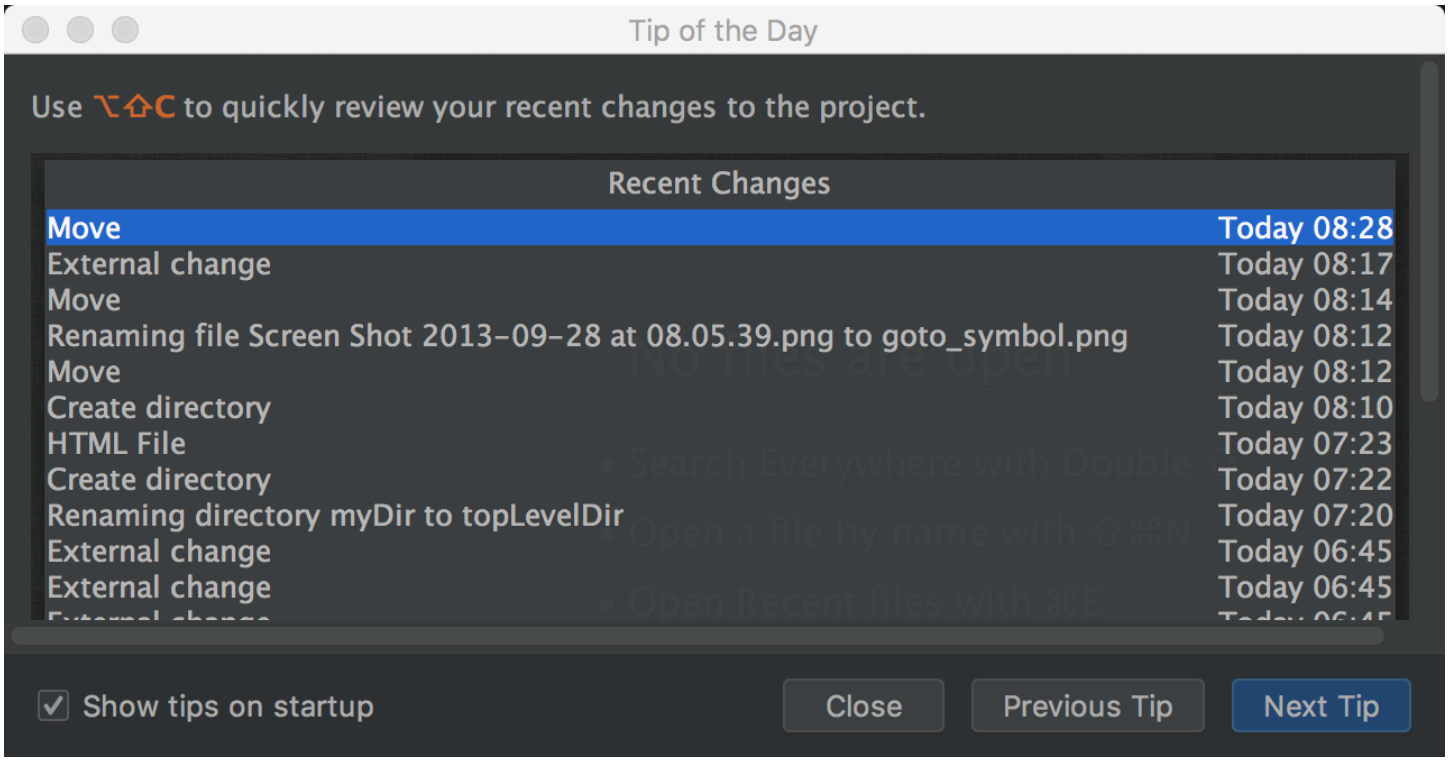
7.5 【版本管理 05】查看文件的所有操作记录

之前为了恢复因为手误造成的语法错误，我使用了快捷键来定位错误行，虽然解决了问题，但总感觉姿势不对，如果没有造成语法错误呢？如何追溯到是哪里的改动影响到了呢？

假如有种方法，可以项目查看最近的修改记录的话（没有git做版本控制的情况下），那就太好了。

太巧的是，今天我打开 PyCharm，就给我推了这条 tip，（在Mac上）使用 ⌥ + ⇧ +C 可以快速查

看最近修改的内容（windows 上是 alt+shift+c）



作者：王炳明
版本：v1.0
发布时间：2020年08月30日
微信公众号：Python编程时光
联系邮箱：wongbingming@163.com
项目主页：http://pycharm.iswbm.com
Github：https://github.com/iswbm/pycharm-guide

回复 "pycharm"，获取最新版 PDF

第八章：插件与工具

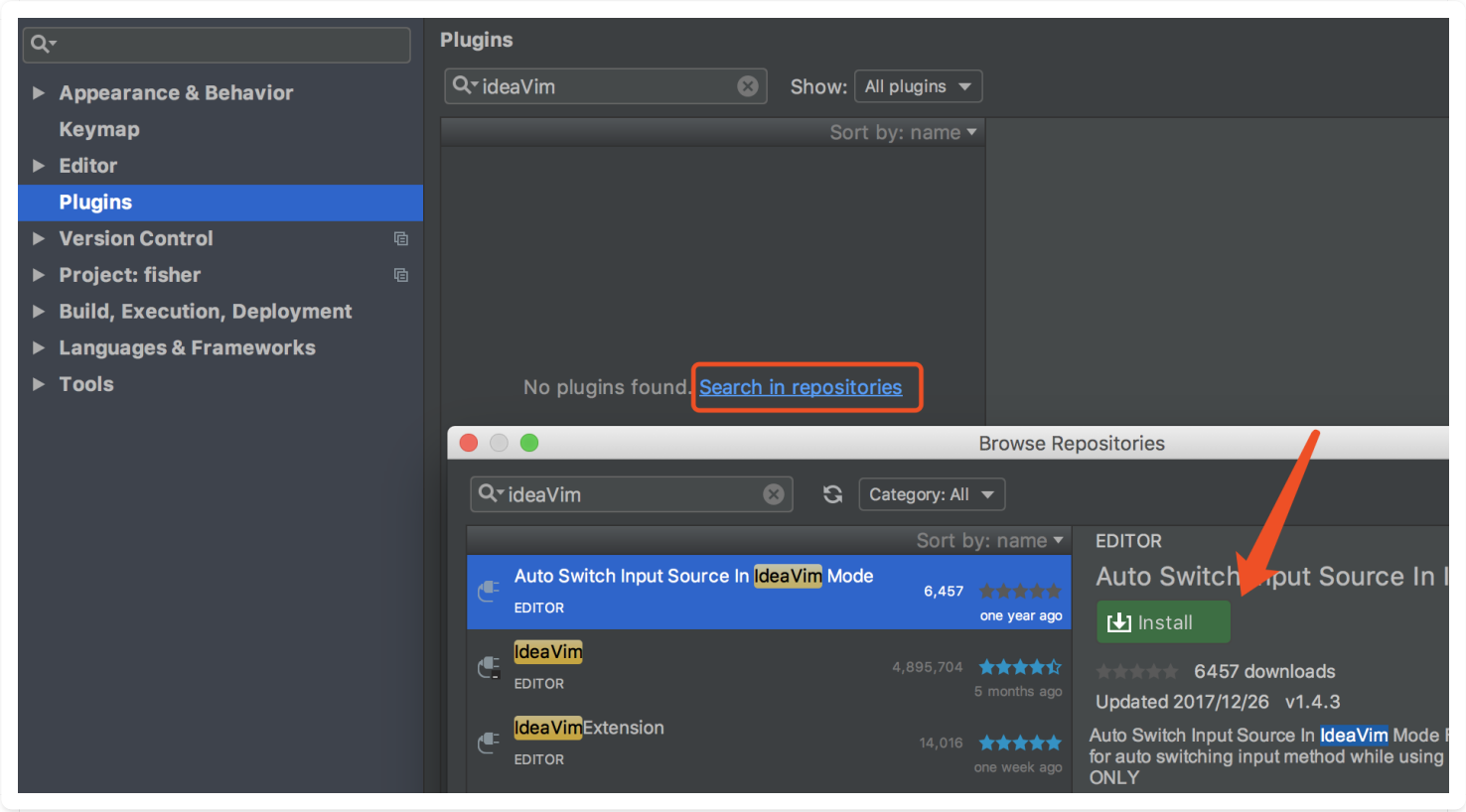
8.1 【插件神器 01】在 PyCharm 中使用 vim

在大多数场景之下，使用鼠标的效率和精准度，是远不如键盘快捷键的（前提是你已经相当熟练的

掌握了快捷键），这个你得承认吧。

Vi 可以满足你对文本操作的所有需求，比可视化界面更加效率，更加 geek。如果你和我一样，是忠实的 vim 粉。在安装完 Pycharm 完后，肯定会第一时间将 `ideaVim` 这个插件也装上，它可以让我们在 Pycharm 中 使用 vim 来编辑代码。

安装方法如下，安装完后需要重启 Pycharm 生效。

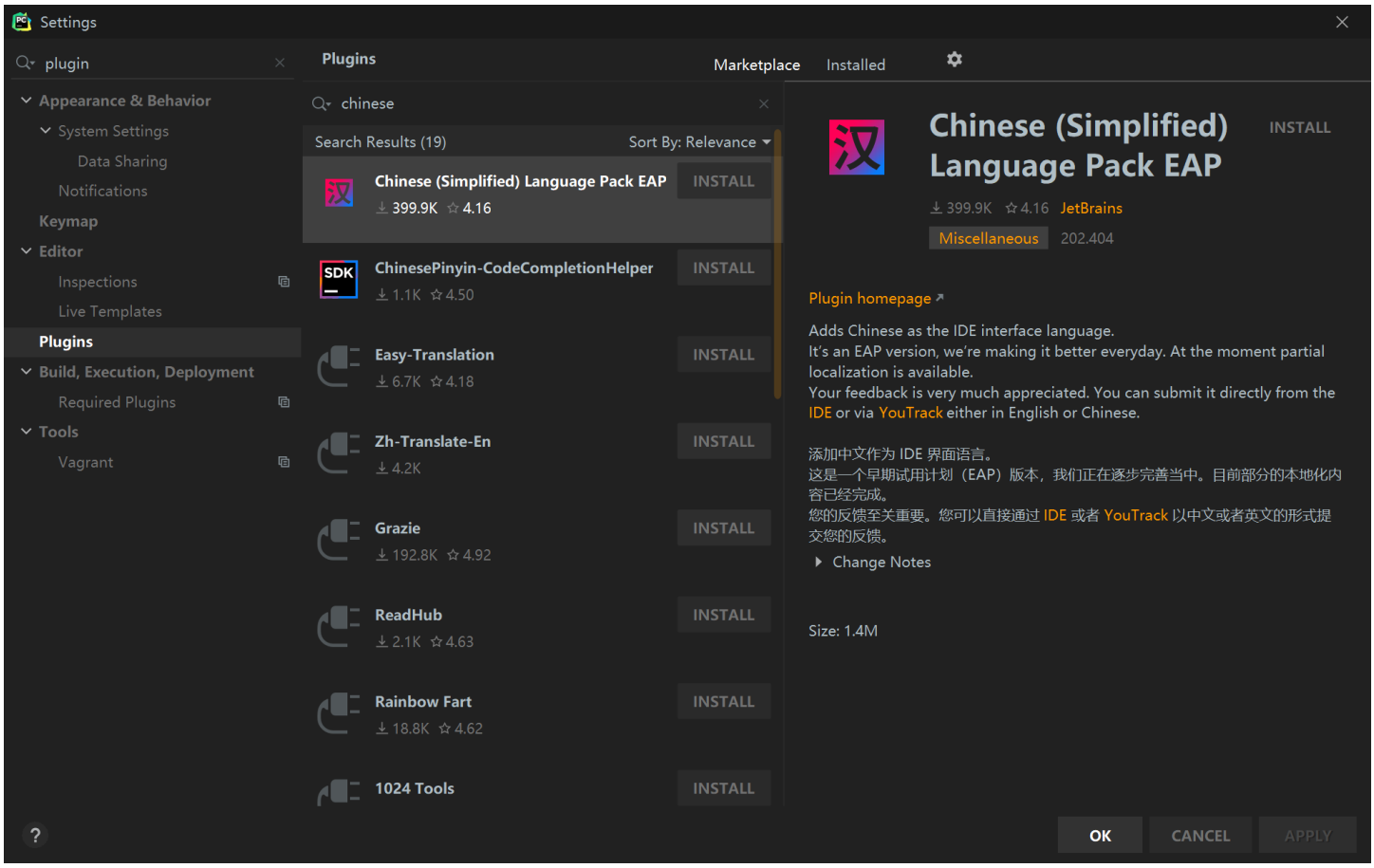


8.2 【插件神器 02】JetBrains 官方推出了汉化插件

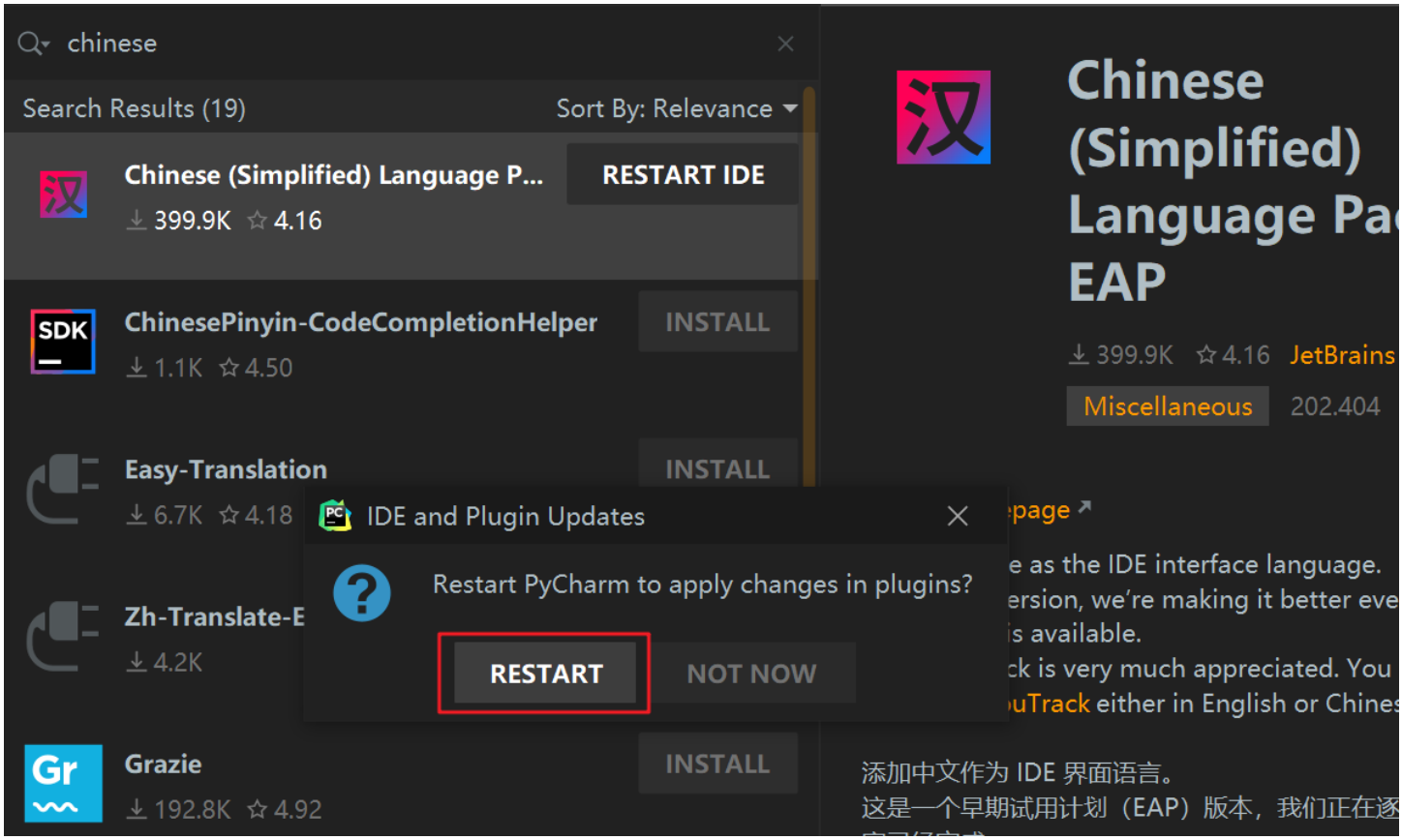
经常听到很多初学者抱怨说，PyCharm 怎么是全英文的？学起来好难啊。

在以前，我会跟他们说，学习编程语言，英文是一项非常重要的能力，千万不能惧怕它，逃避它，而要是去学习它，适应它，如果连个 IDE 都适应不了，那就别学编程了。

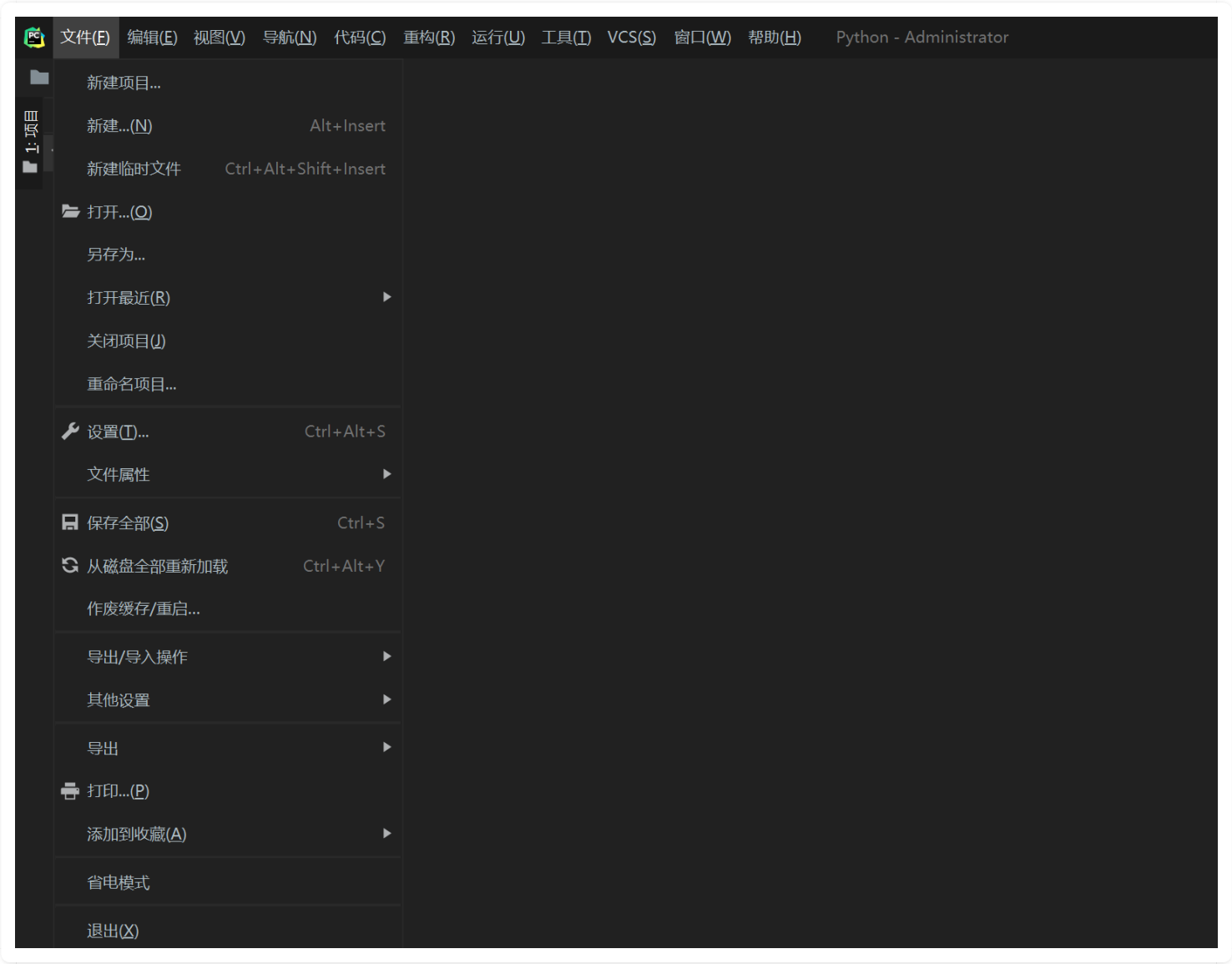
而现在，JetBrains 官方自己出了汉化插件，名字就叫： chinese，在插件市场里一搜，排名第一便是它，下载量已经 40 万，对比排名第二的民间汉化插件，简直不是量级的。



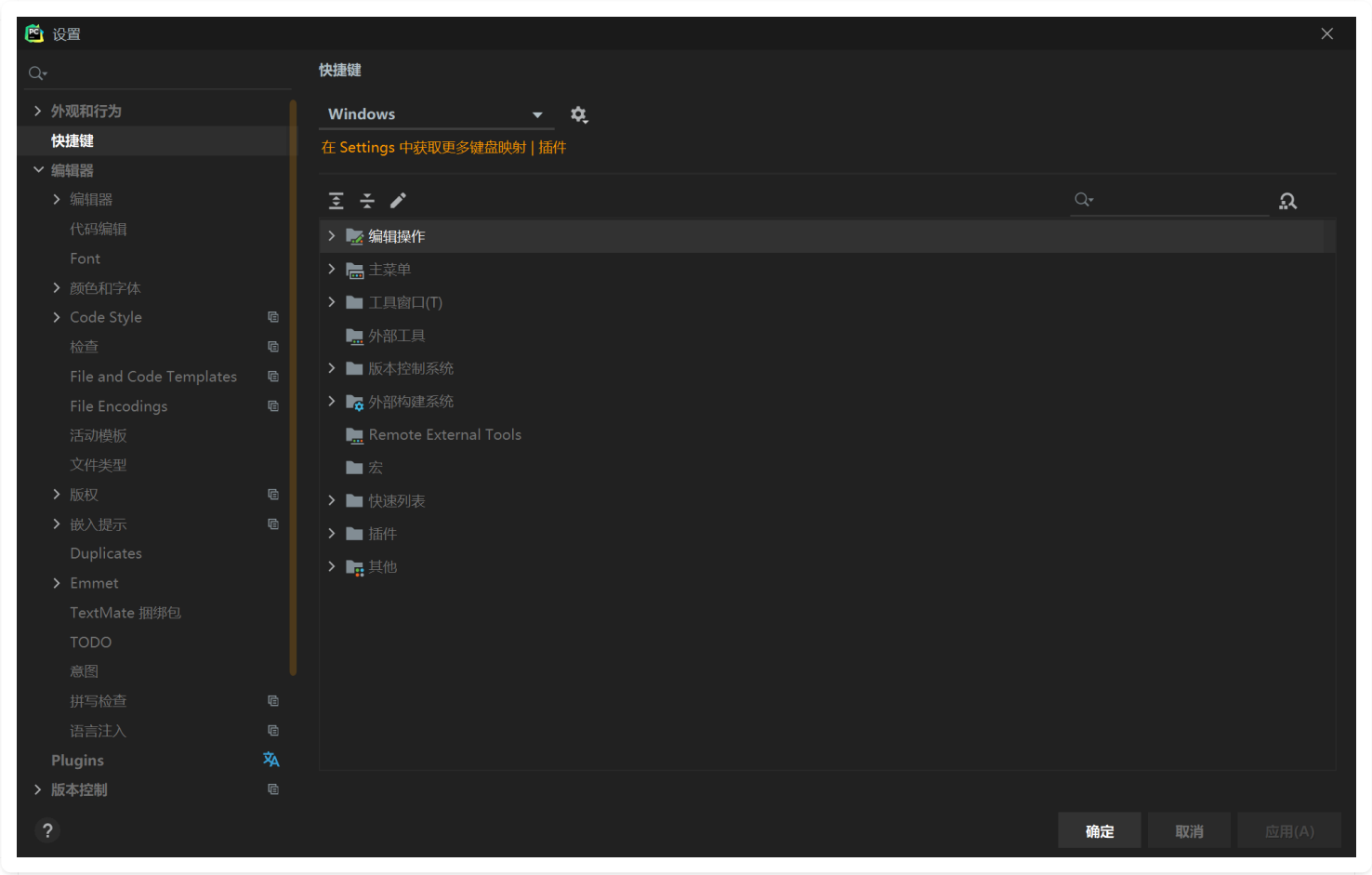
点击 `INSTALL` 安装后，会提示你进行重启，才能生效。



重启完成后，展现在我们面前的是一个既熟悉又陌生的界面，所有的菜单栏全部变成了中文。



点进设置一看，可以说基本实现了汉化，只剩下一小撮的英文（难道是因为这些词保留英文会比翻译后更容易理解吗？就像 socket 和套接字一样。），不过个人感觉完全不影响使用了。



8.3 【插件神器 03】在 PyCharm 中写 Markdown

富文本排版文档是一件非常痛苦的事情，对于程序员写文档，最佳的推荐是使用 Markdown，我所有的博客日记都是使用 Markdown 写出来的。

从 Github下载的代码一般也都会带有README.md文件，该文件是一个Markdown格式的文件。

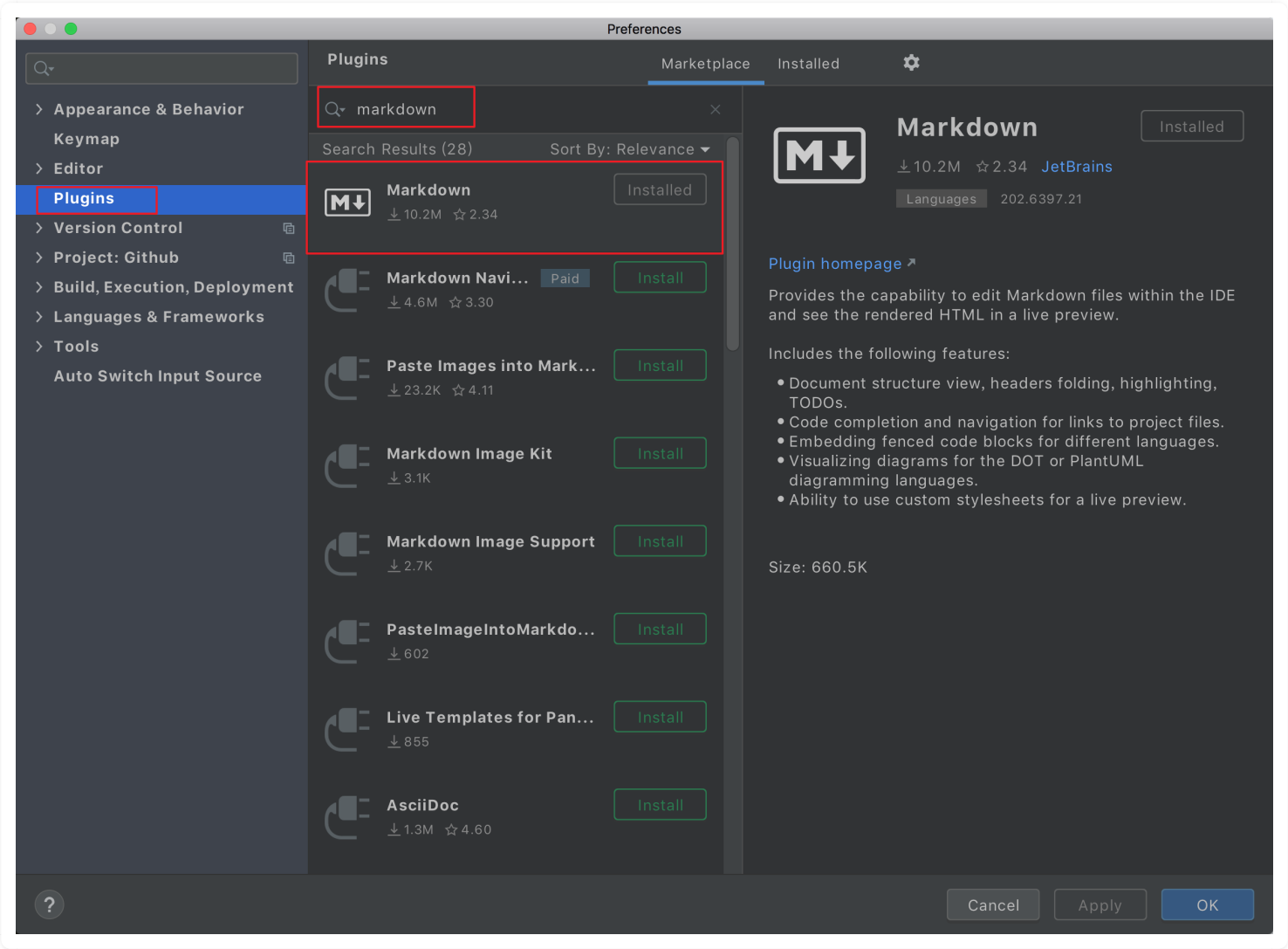
PyCharm是默认没有安装Markdown插件的，所以不能按照Markdown格式显示文本，显示的是原始文本。

因此，如果要在 PyCharm 中阅读 Markdown 文档，可以装一下 Markdown support 这个插件。

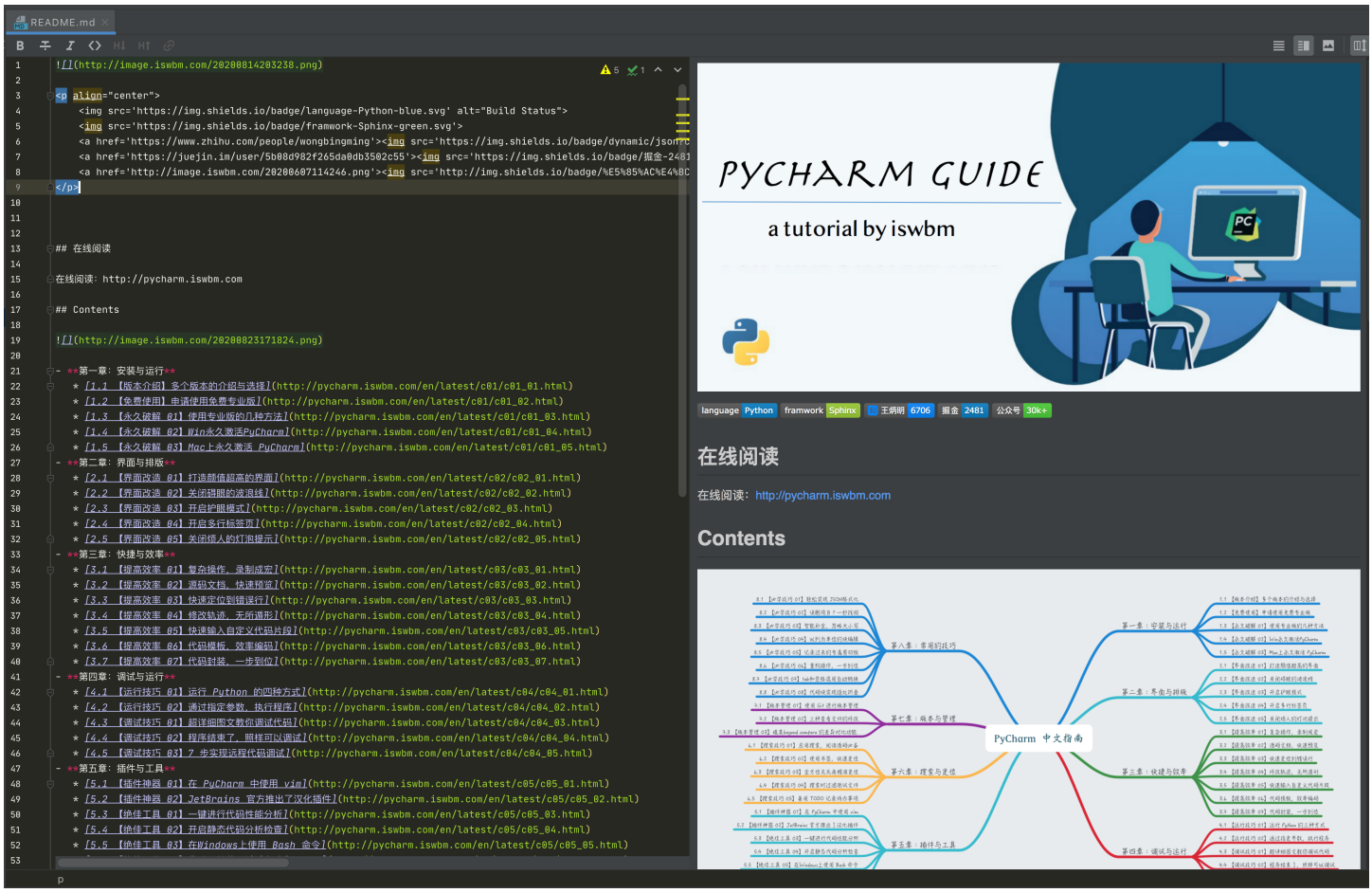
安装的方法有两种:

1、第一种，最方便的，就是你打开一个 MD 的文档，PyCharm 就会提示你安装它。

2、从插件商店中搜索安装。


Preferences
Plugins
Marketplace
Installed
markdown
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Project: Github
Build, Execution, Deployment
Languages & Frameworks
Tools
Auto Switch Input Source
Search Results (28)
Sort By: Relevance
Markdown
10.2M 2.34
Installed
Markdown Navi...
Paid
4.6M 3.30
Install
Paste Images into Mark...
23.2K 4.11
Markdown Image Kit
3.1K
Markdown Image Support
2.7K
PasteImageIntoMarkdo...
602
Live Templates for Pan...
855
AsciiDoc
1.3M 4.60
Markdown
10.2M 2.34 JetBrains
Languages
202.6397.21
Plugin homepage
Provides the capability to edit Markdown files within the IDE and see the rendered HTML in a live preview.
Includes the following features:
Document structure view, headers folding, highlighting, TODOs.
Code completion and navigation for links to project files.
Embedding fenced code blocks for different languages.
Visualizing diagrams for the DOT or PlantUML diagramming languages.
Ability to use custom stylesheets for a live preview.
Size: 660.5K
Cancel
Apply
OK

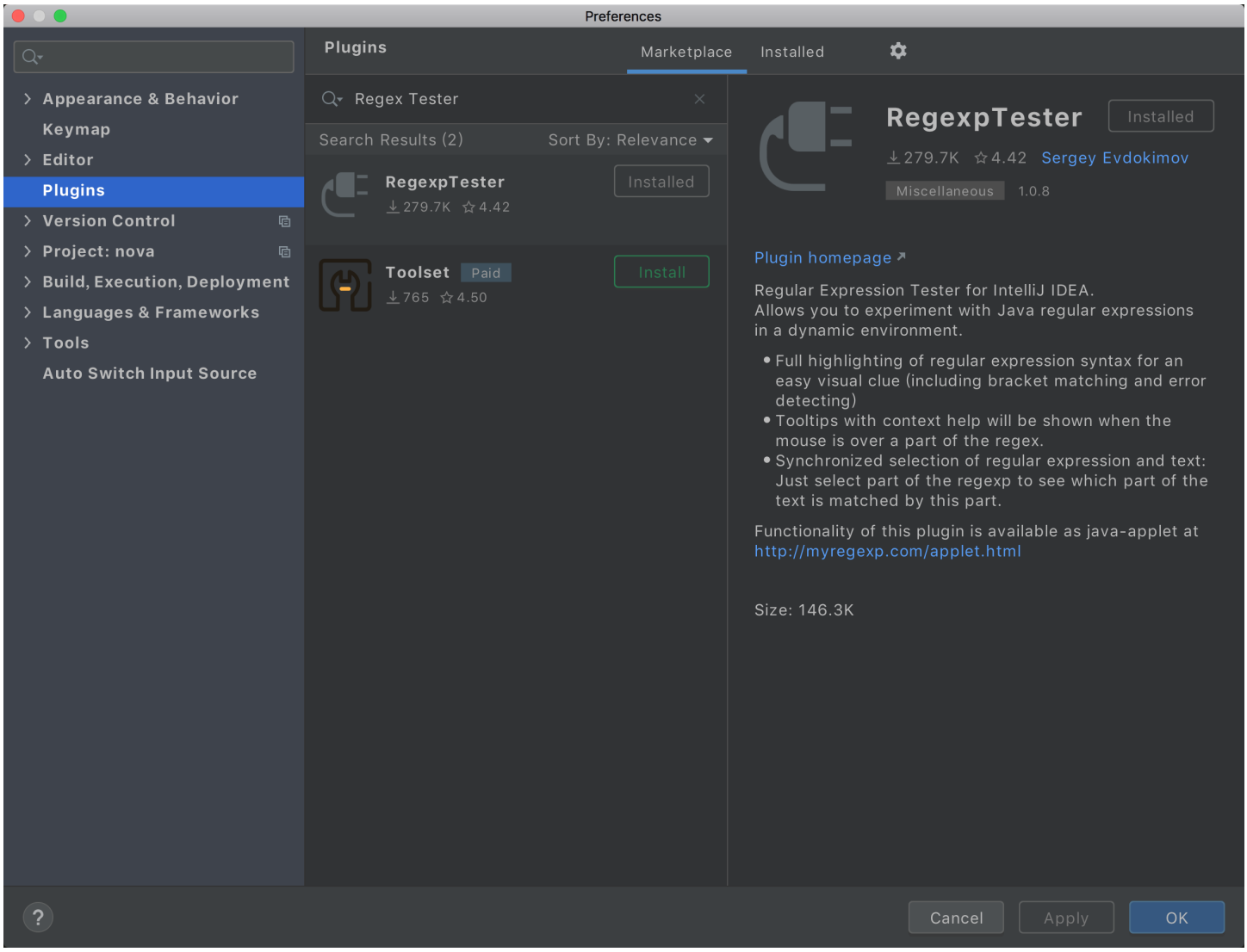
效果如下



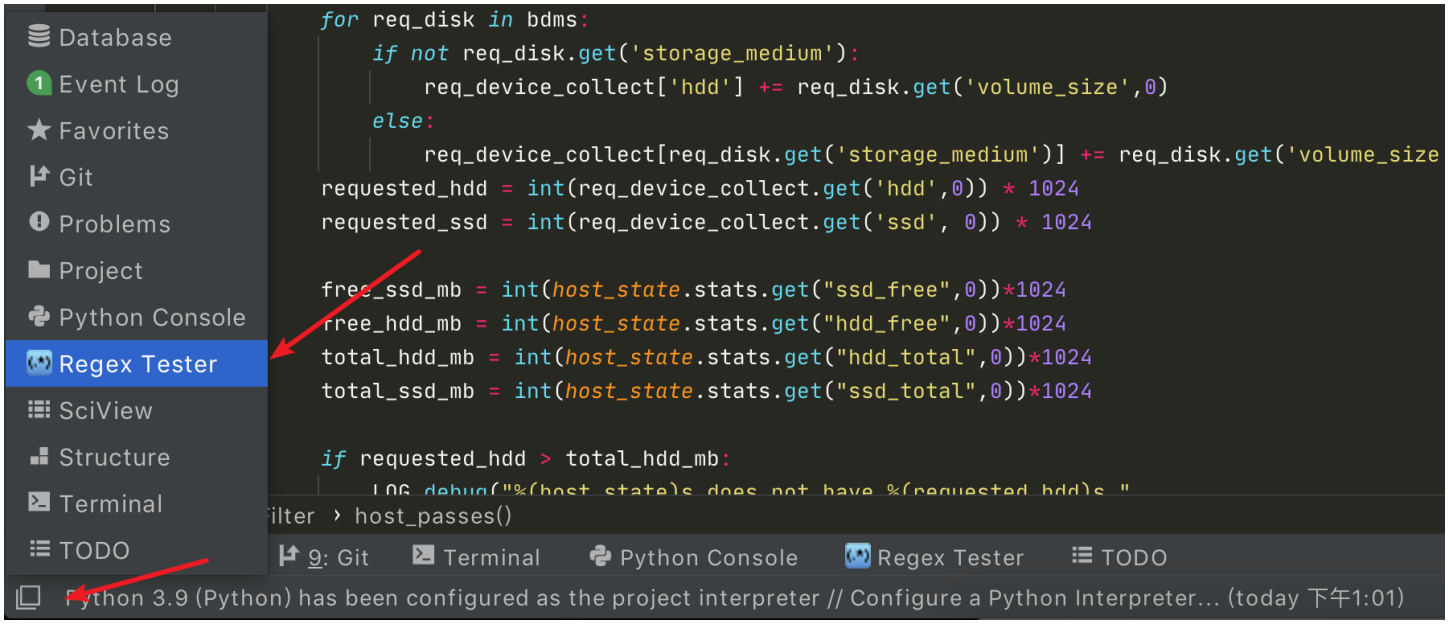
8.4 【插件神器 04】正则表达式测试：Regex Tester

Regex Tester是PyCharm的第三方插件，可以测试正则表达式。

按照下图入口，安装 Regex Tester 插件：



安装完成后，无需重启 PyCharm ， 点击 PyCharm 界面左下方的小矩形按钮，就能找到 Regex Tester 选项。



点击进入后，就出现了如下界面。我随手写了个匹配手机号码的正则（不一定准确），匹配到的字

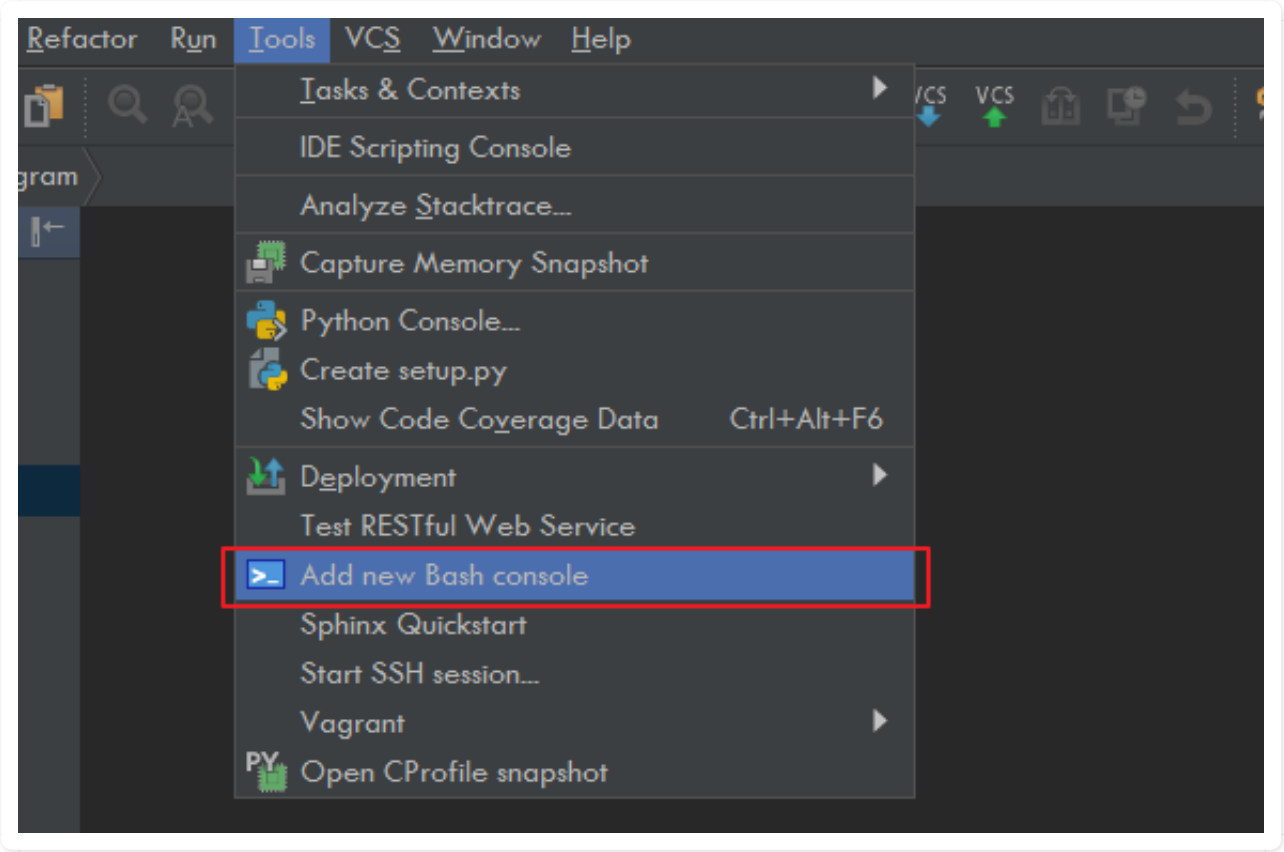
符串背景会被高亮。右上方还有一些选项如大小写敏感，多行模式等，可根据需要进行选择。Regex Tester 还提供了Split，Replace功能等。

使用效果如下：

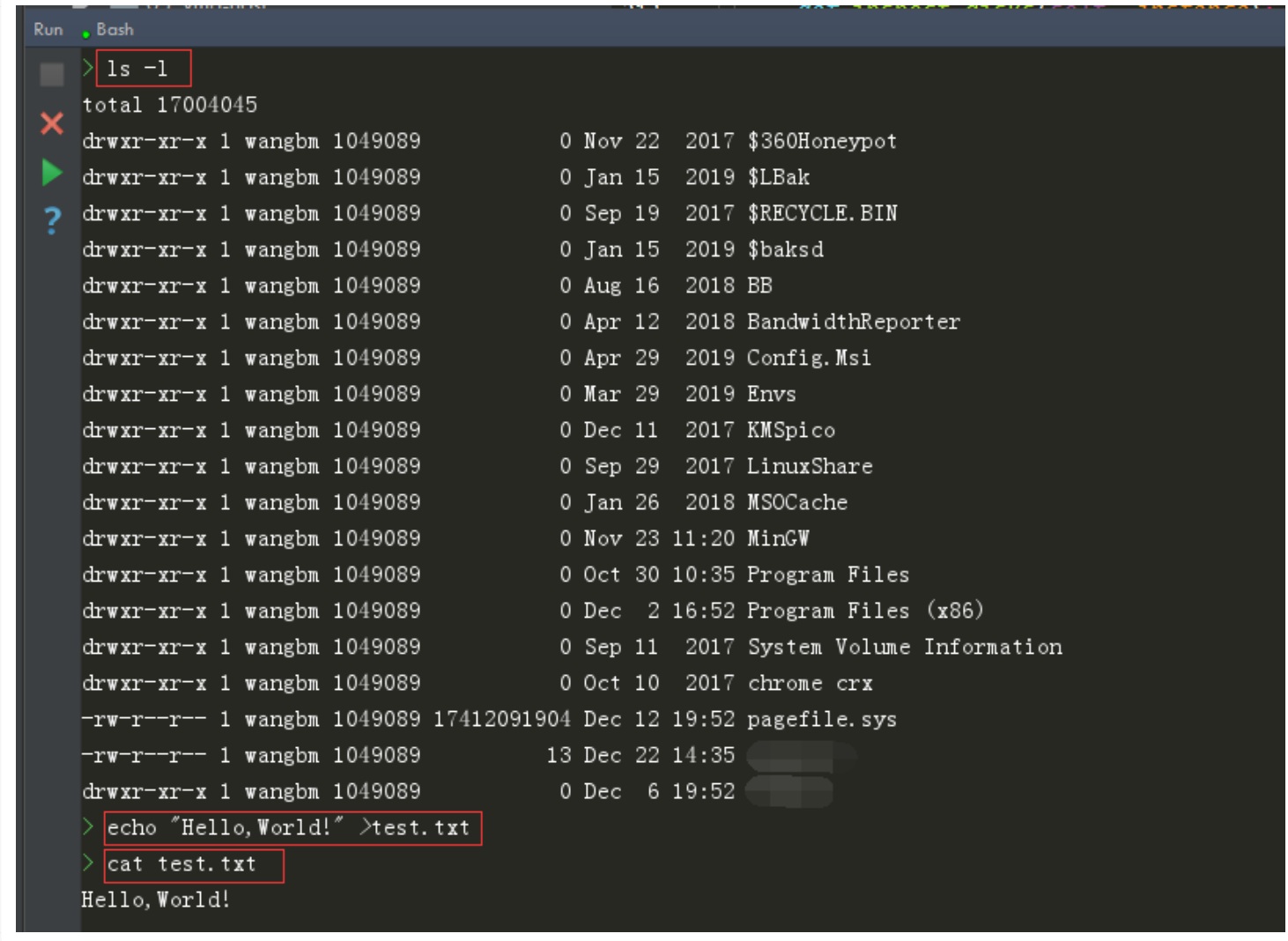
8.5 【绝佳工具 01】在Windows上使用 Bash 命令

在 Windows 上的 cmd 命令和 Linux 命令有不少的差异，比如要列出当前目录下的所有文件，Windows 上是用 `dir` ，而 Linux 上则是用 `ls -l` 。

对于像我这样熟悉 Linux 的开发者来说，Windows 的 那些 CMD 命令带来的糟糕体验是无法忍受的。



在弹出的 Bash 窗口，你可以敲入你想使用的 Linux 命令，是不是舒服多了。



8.6 【绝佳工具 02】代码不规范？试试自动化 PEP8

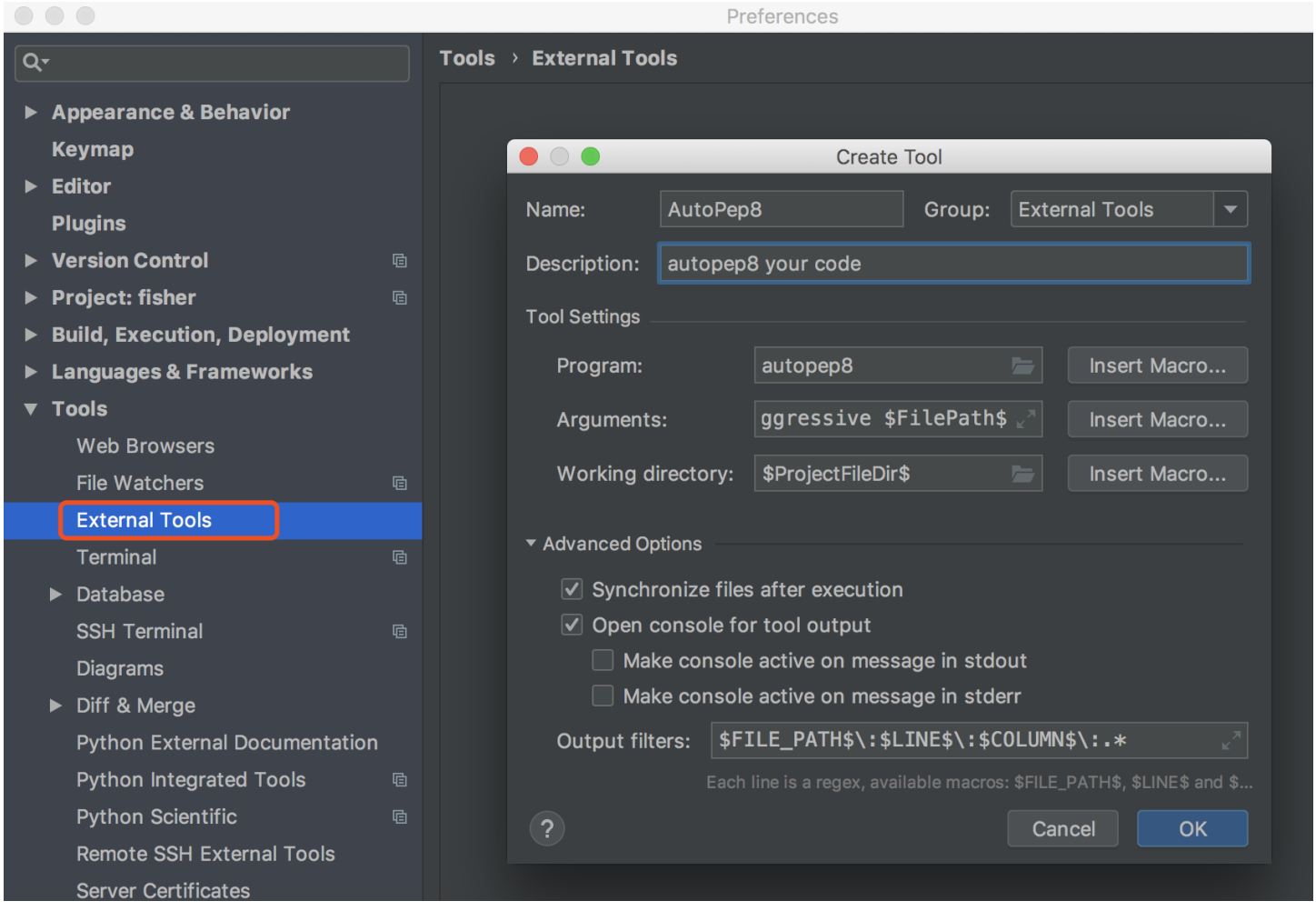
`pep8` 是Python 语言的一个代码编写规范。如若你是新手，目前只想快速掌握基础，而不想过多去注重代码的的编写风格（虽然这很重要），那你可以尝试一下这个工具 - `autopep8`

首先在全局环境中（不要在虚拟环境中安装），安装一下这个工具。

```
$ sudo pip install autopep8
```

然后在 PyCharm 导入这个工具，具体设置如下图

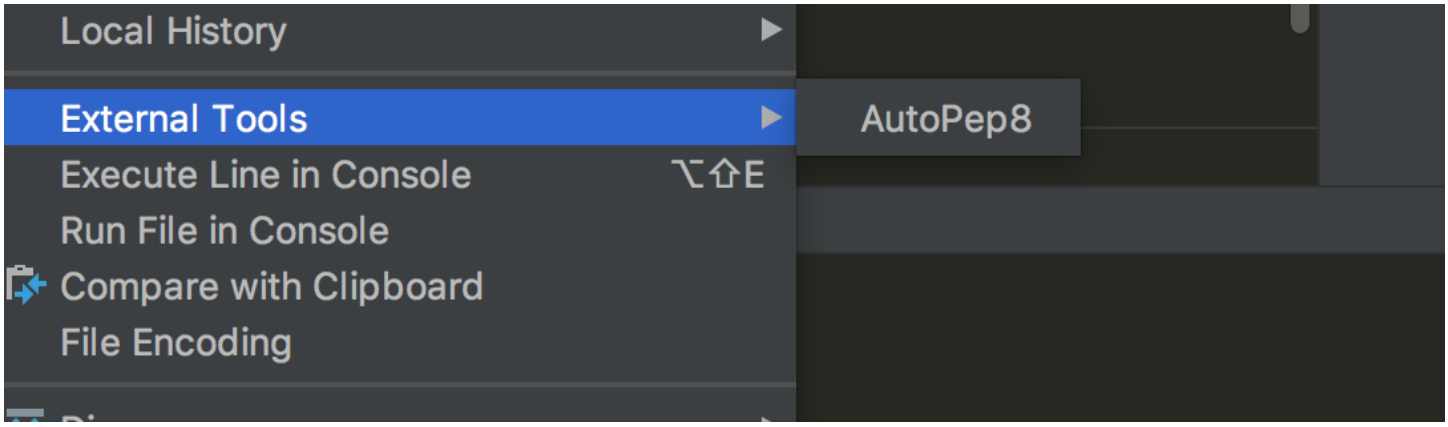
```
Name: AutoPep8
Description: autopep8 your code
Program: autopep8
Arguments: --in-place --aggressive --aggressive $FilePath$
Working directory: $ProjectFileDir$
Output filters: $FILE_PATH$\:$LINE$\:$COLUMN$\:.*
```



我随意写了一段不符合 pep8 规范的代码。

```
import os,sys

def example1():
    some_tuple = (1, 2, 3, 'a')
    some_variable = {
        'long': 'Long code lines should be wrapped within 79 characters.',
        'other': [math.pi, 100, 200, 300, 9876543210,'This is a long string that goes on'],
        'more': { 'inner': 'This whole logical line should be wrapped.'}}
    return (some_tuple, some_variable)
```

点击右键，选择 `External Tools` -> `AutoPep8`



看一下效果，还是挺明显的。

```
import os
import sys


def example1():
    some_tuple = (1, 2, 3, 'a')
    some_variable = {
        'long': 'Long code lines should be wrapped within 79 characters.',
        'other': [
            math.pi,
            100,
            200,
            300,
            9876543210,
            'This is a long string that goes on'],
        'more': {
            'inner': 'This whole logical line should be wrapped.'}}
    return (some_tuple, some_variable)
```

你可能会说，Pycharm 本身就自带这个功能了呀，快捷键 `Command` + `Option` + `L` ，就可以实现一键pep8了。你可以对比一下，Pycharm 自带的代码 pep8 化功能 并没有像这个 `autopep8` 来得彻底。 我相信你最终的选择肯定是后者。

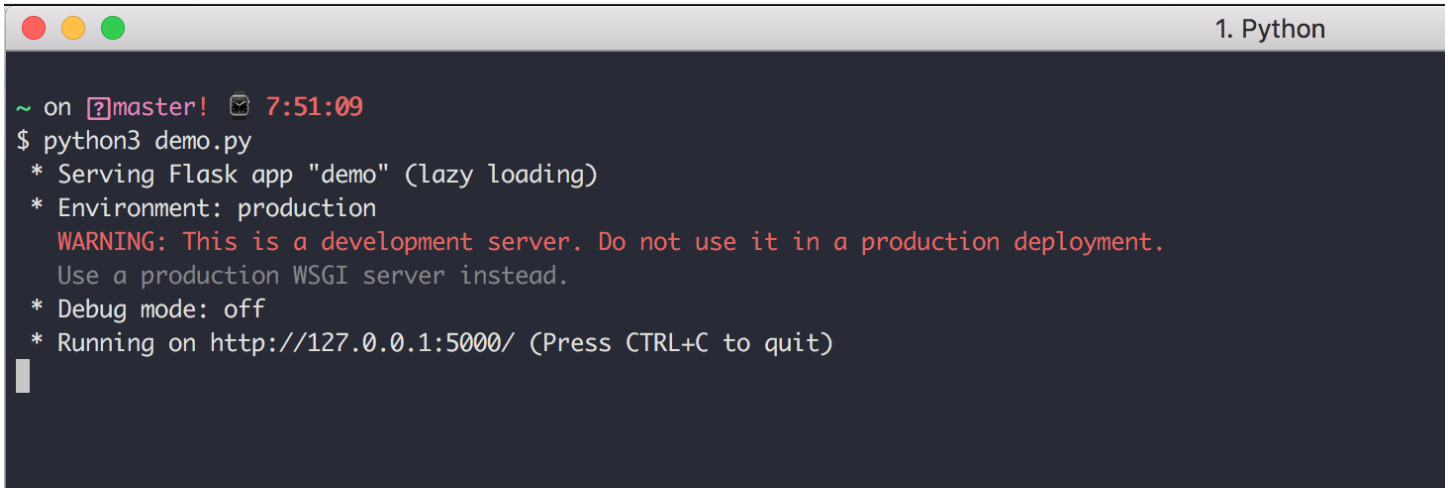
8.7 【绝佳工具 03】HTTP接口调试：Test RESTful Web Service

PyCharm 的 Test RESTful Web Service工具提供了RESTful接口测试界面，如下图所示，提供了get、post，put等http方法，其中的Request子界面headers，Parameters，Body等功能，Response子界面用于显示返回值，Response Headers用于显示返回的消息头。

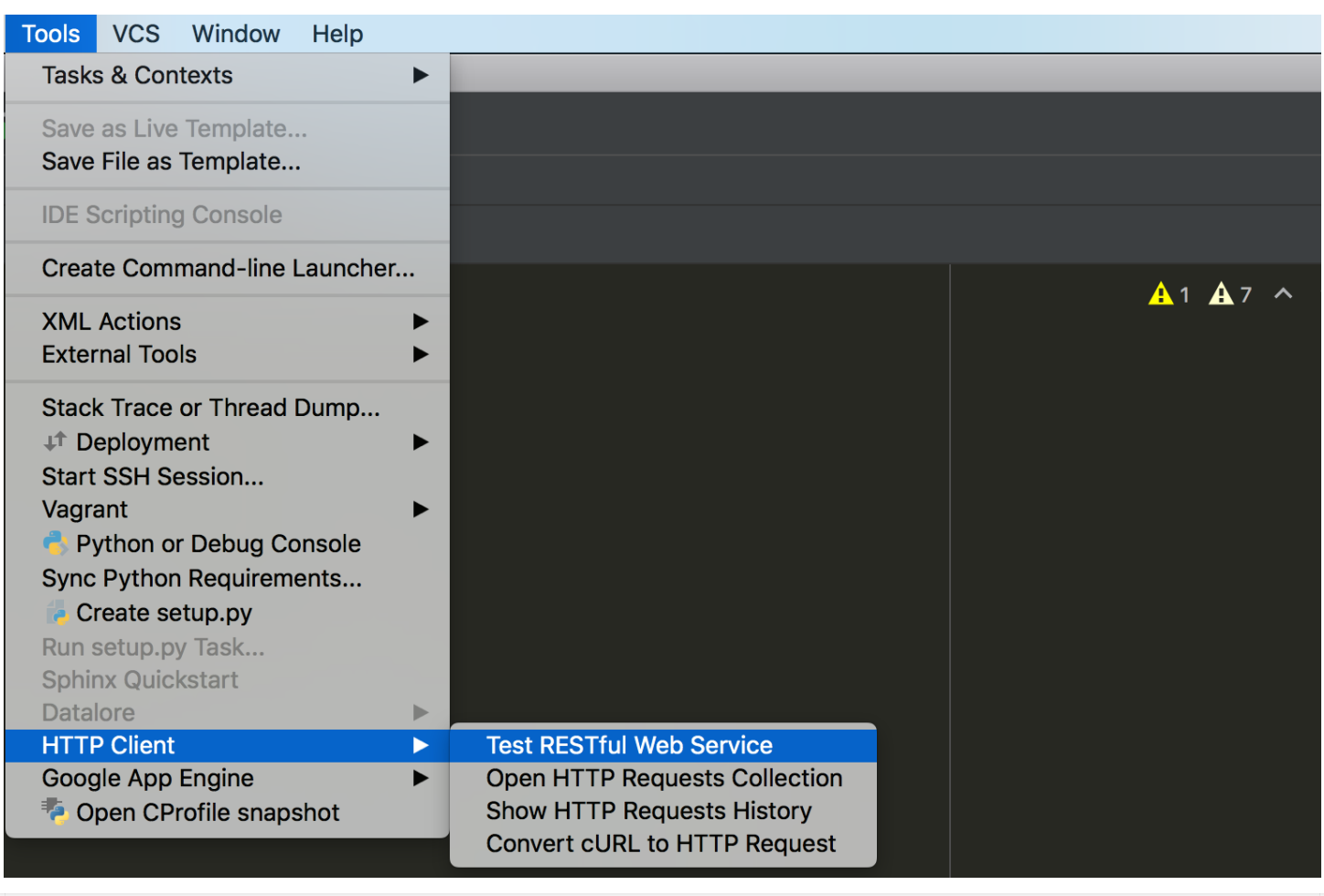
为了演示，我先使用 Flask 写一个 HTTP 接口

```
from flask import Flask, request

app = Flask(__name__)


@app.route('/hello')
def index():
    name = request.args.get('name')
    return '你好, ' + name

if __name__ == '__main__':
    app.run()
```

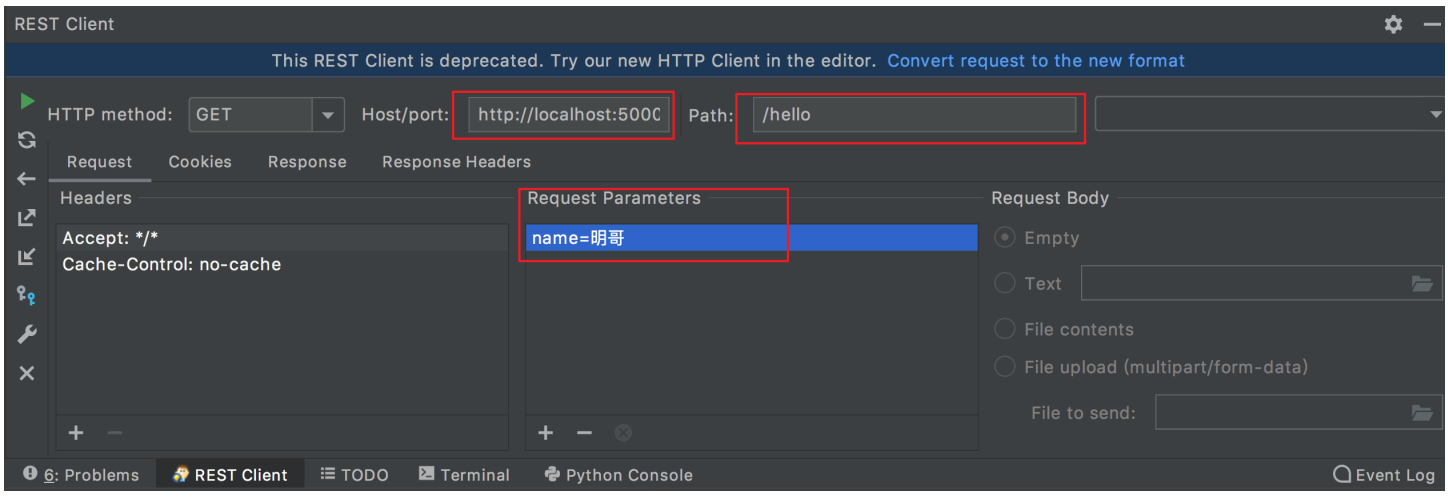
并运行它开启服务，访问地址是：http://127.0.0.1:5000/



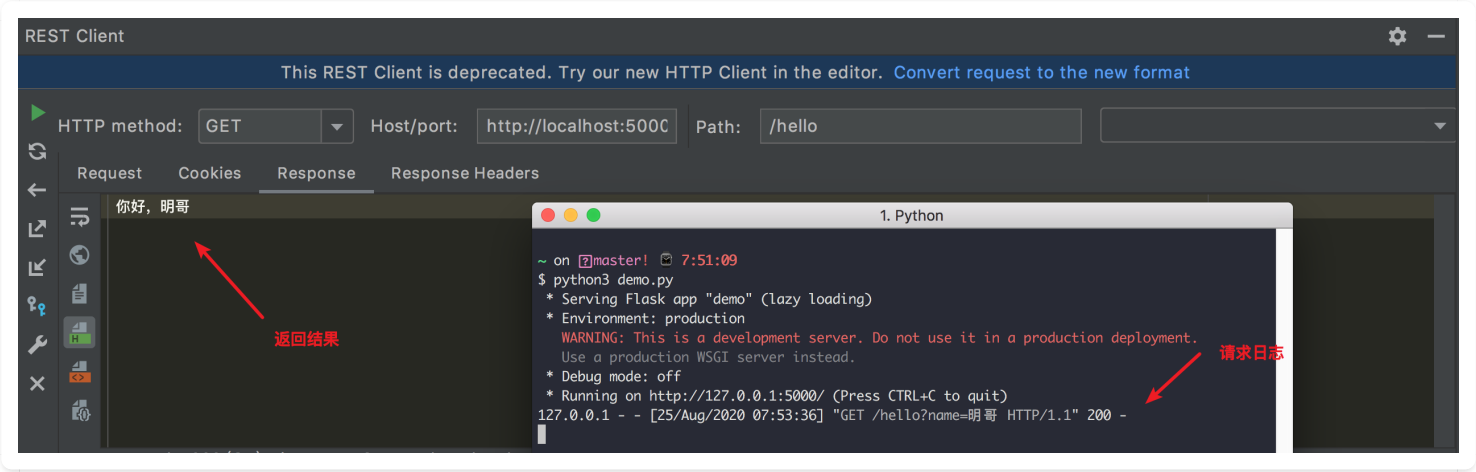
通过下图方式打开 `Test RESTful Web Service`



会出现如下界面，在红框处填写如下信息



然后点击最左边的运行按钮，即可向服务器发送 http 请求。



8.8 【绝佳工具 04】选择执行：Execute Selection in Console

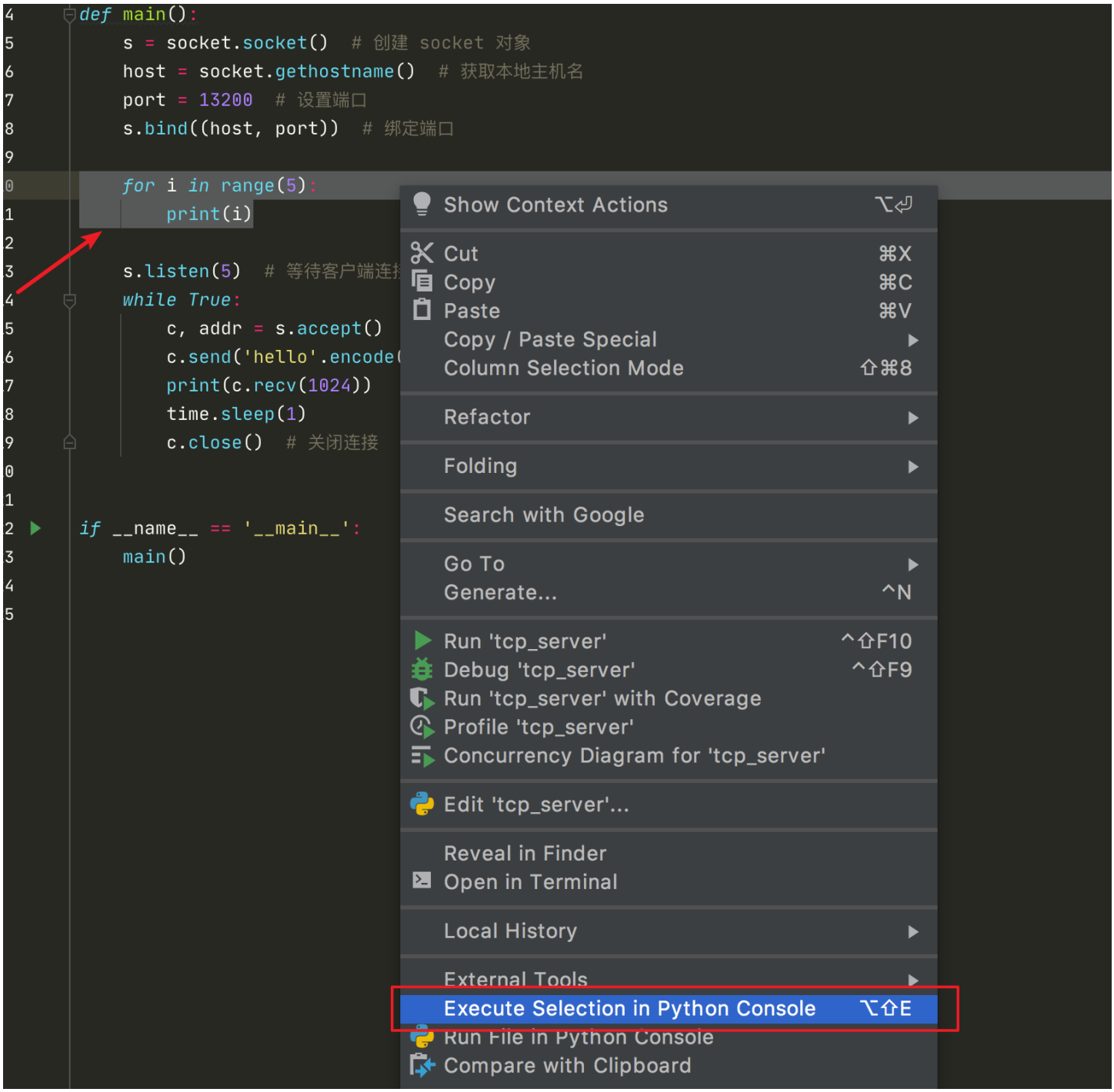
当你想写一段简单的测试代码时，或许你会这样子

1. 使用 Python Shell 直接写。缺点是没有自动补全。
2. 在 PyCharm 中新开一个文件。缺点是要新创建一个文件，完了后还要删除。

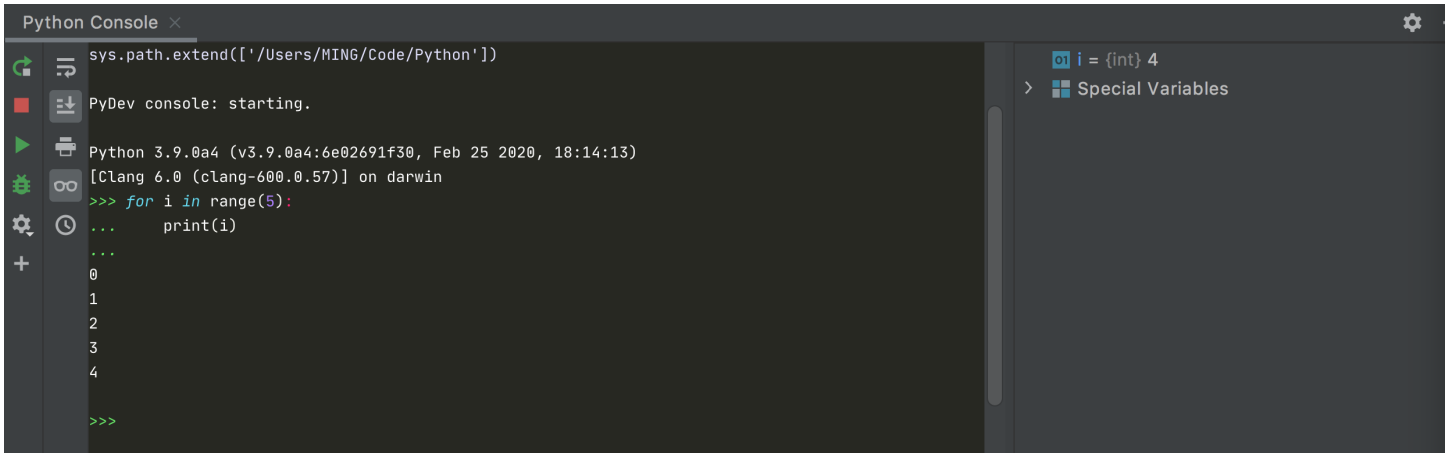
今天再给大家介绍一种新的方法，可以完全避开上面两种方式的缺点。

那就是 `Execute Selection in Console`，可以说是 `Run in Anywhere`.

只要在当前文件中，写好代码，然后光标选择后，右键点击 `Execute Selection in Python Console` 或者 使用快捷键 option + shift + E (windows 上是 alt + shift + E)。



接着 PyCharm 就会弹出一个 Python Console 窗口，然后运行你所选择的代码。



可以发现其中的一个亮点，就是使用这种方法，PyCharm 会自动帮我们处理好缩进（我们选择时，前面有缩进，可是在执行时，会自动去掉前面多余的缩进）

8.9 【绝佳工具 05】一键进行代码性能分析

在 Python 中有许多模块可以帮助你分析并找出你的项目中哪里出现了性能问题。

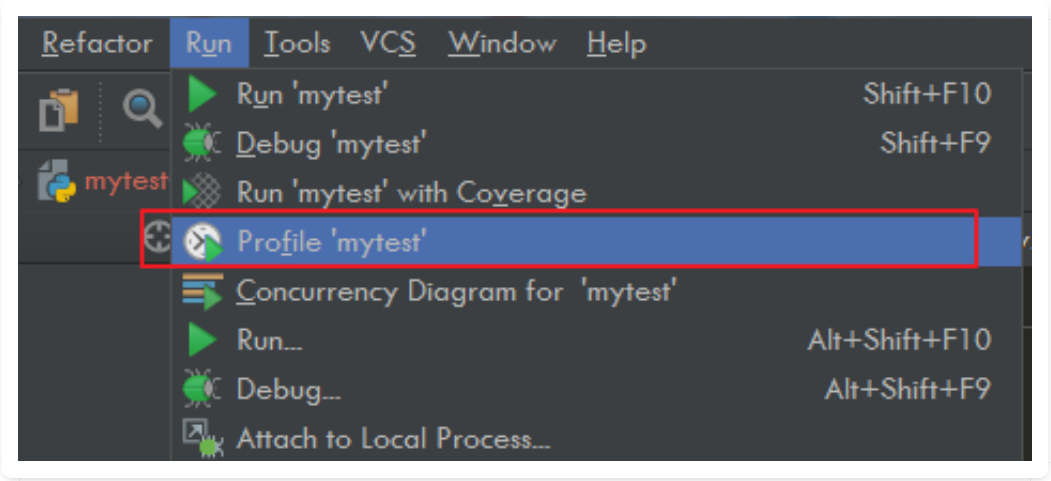
比如，常用的模块有 cProfile，在某些框架中，也内置了中间件帮助你进行性能分析，比如 Django，WSGI。

做为Python 的第一 IDE， PyCharm 本身就支持了这项功能。而且使用非常方便，小白。

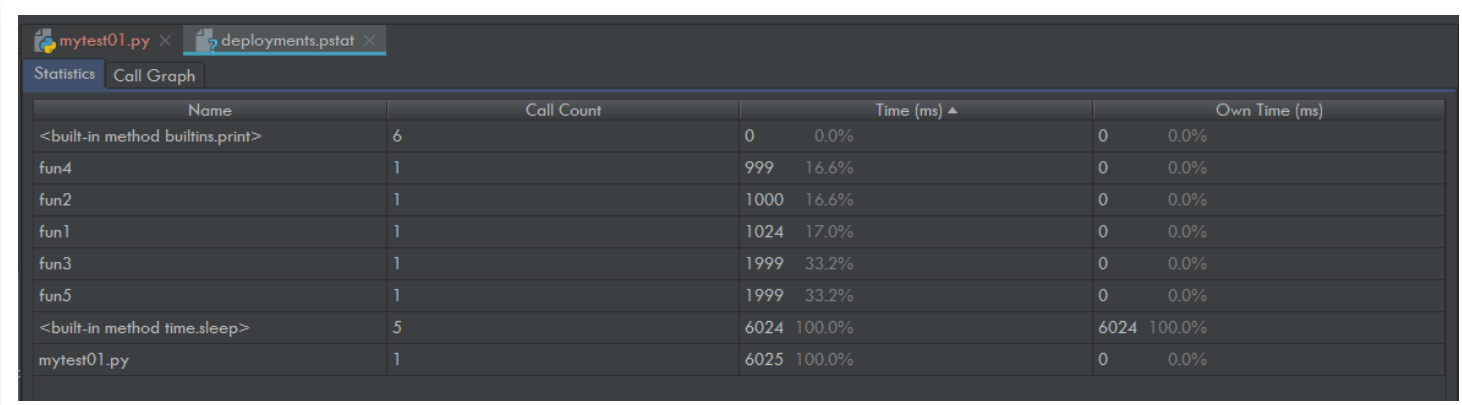
假设现在要分析如下这段代码的性能损耗情况，找出到底哪个函数耗时最多

```
import time

def fun1():
    time.sleep(1)

def fun2():
    time.sleep(1)

def fun3():
    time.sleep(2)

def fun4():
    time.sleep(1)

def fun5():
    time.sleep(1)
    fun4()

fun1()
fun2()
fun3()
fun5()
```

点击 Run -> Profile '程序'，即可进行性能分析。



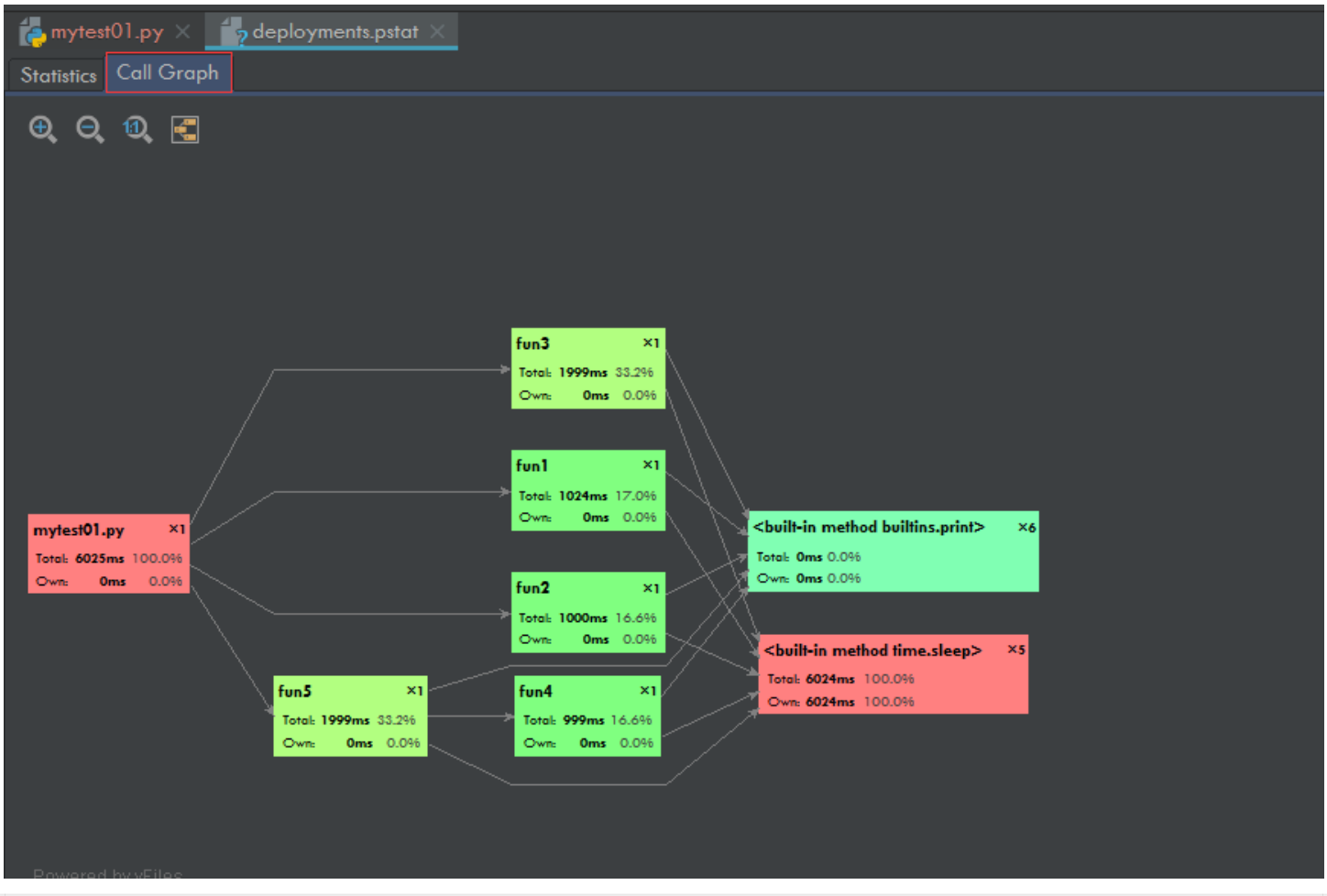
运行完毕后，会自动跳出一个性能统计界面。



性能统计界面由Name、Call Count、Time(ms)、Own Time(ms)，4列组成一个表格，见下图。

1. 表头Name显示被调用的模块或者函数；Call Count显示被调用的次数；Time(ms)显示运行时间和时间百分比，时间单位为毫秒（ms）。
2. 点击表头上的小三角可以升序或降序排列表格。
3. 在Name这一个列中双击某一行可以跳转到对应的代码。
4. 以fun4这一行举例：fun4被调用了一次，运行时间为1000ms，占整个运行时间的16.7%

点击 Call Graph（调用关系图）界面直观展示了各函数直接的调用关系、运行时间和时间百分比，见下图。



右上角的4个按钮表示放大、缩小、真实大小、合适大小；

1. 箭头表示调用关系，由调用者指向被调用者；
2. 矩形的左上角显示模块或者函数的名称，右上角显示被调用的次数；
3. 矩形中间显示运行时间和时间百分比；
4. 矩形的颜色表示运行时间或者时间百分比大小的趋势：红色 > 黄绿色 > 绿色，由图可以看出fun3的矩形为黄绿色，fun1为绿色，所有fun3运行时间比fun1长。
5. 从图中可以看出Test.py直接调用了fun3、fun1、fun2和fun5函数；fun5函数直接调用了fun4函数；fun1、fun2、fun3、fun4和fun5都直接调用了print以及sleep函数；整个测试代码运行的总时间为6006ms，其中fun3的运行时间为1999ms，所占的时间百分比为33.3%，也就是 1999ms / 6006ms = 33.3%。

8.10 【绝佳工具 06】开启静态代码分析检查

对于编译型的语言，如 Java，需要将代码编译成机器可识别的语言才可运行，在编译过程中，就可以通过分析或检查源程序的语法、结构、过程、接口等来检查程序的正确性，找出代码隐藏的错误和缺陷。这个过程叫做静态代码分析检查。

那对于 Python 这种解释型的语言来说，代码是边运行边翻译的，不需要经过编译这个过程。很多肉眼无法一下子看出的错误，通常都是跑一下（反正跑一下这么方便）才能发现。

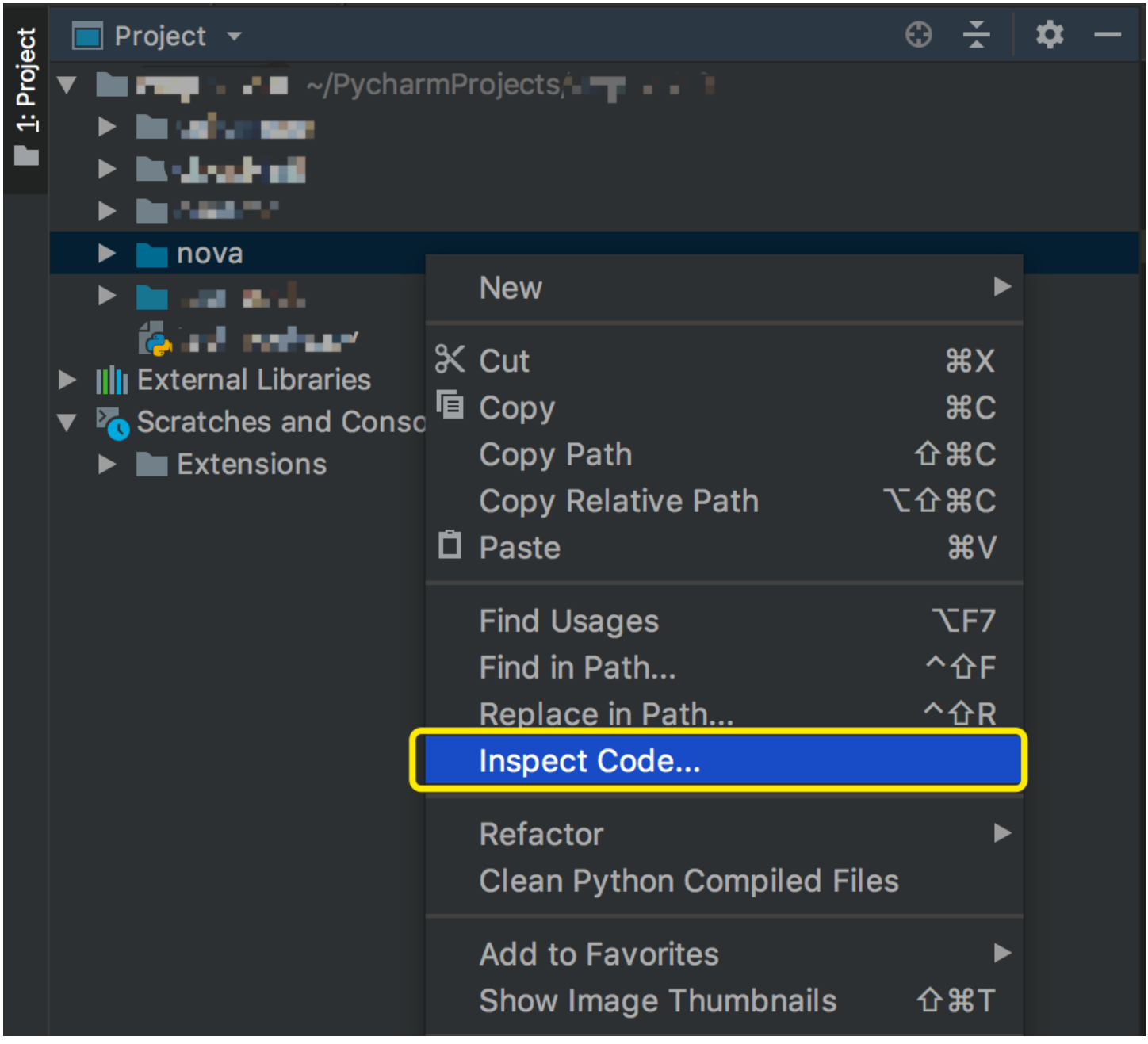
由于Python 运行是如此的方便，以至于我们都不太需要关注静态分析工具。

但也不是说，静态分析工具完全没有用武之地，我认为还是有。

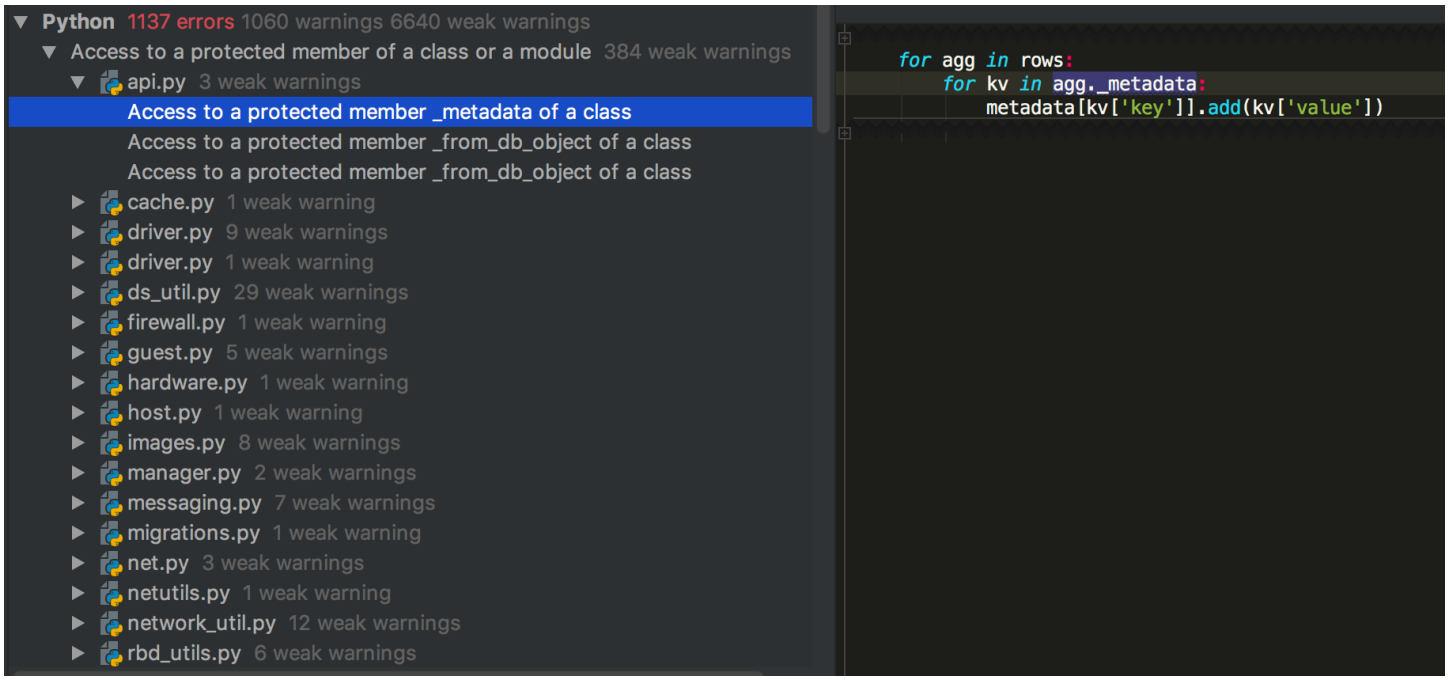
如果你的编码能力还没有很成熟，代码中可以有许许多多的隐藏bug，由于 Python 是运行到的时候才解释，导致一次运行只能发现一个错误，要发现100个bug，要运行100次，数字有点夸大，其实就是想说，如果这么多的错误都能通过一次静态检查发现就立马修改，开发调试的效率就可以有所提升。当然啦，并不是说所有的错误静态分析都能提前发现，这点希望你不要误解。

做为 Python 最强 IDE，PyCharm本身内置了这个功能，不需要你安装任何插件。

你只需要像下面这样点击项目文件夹，然后右键，选择 `Inspect Code`，就可以开启静态检查。



我对开源组件 nova 的静态检查发现，其有不规范的地方有数千处。

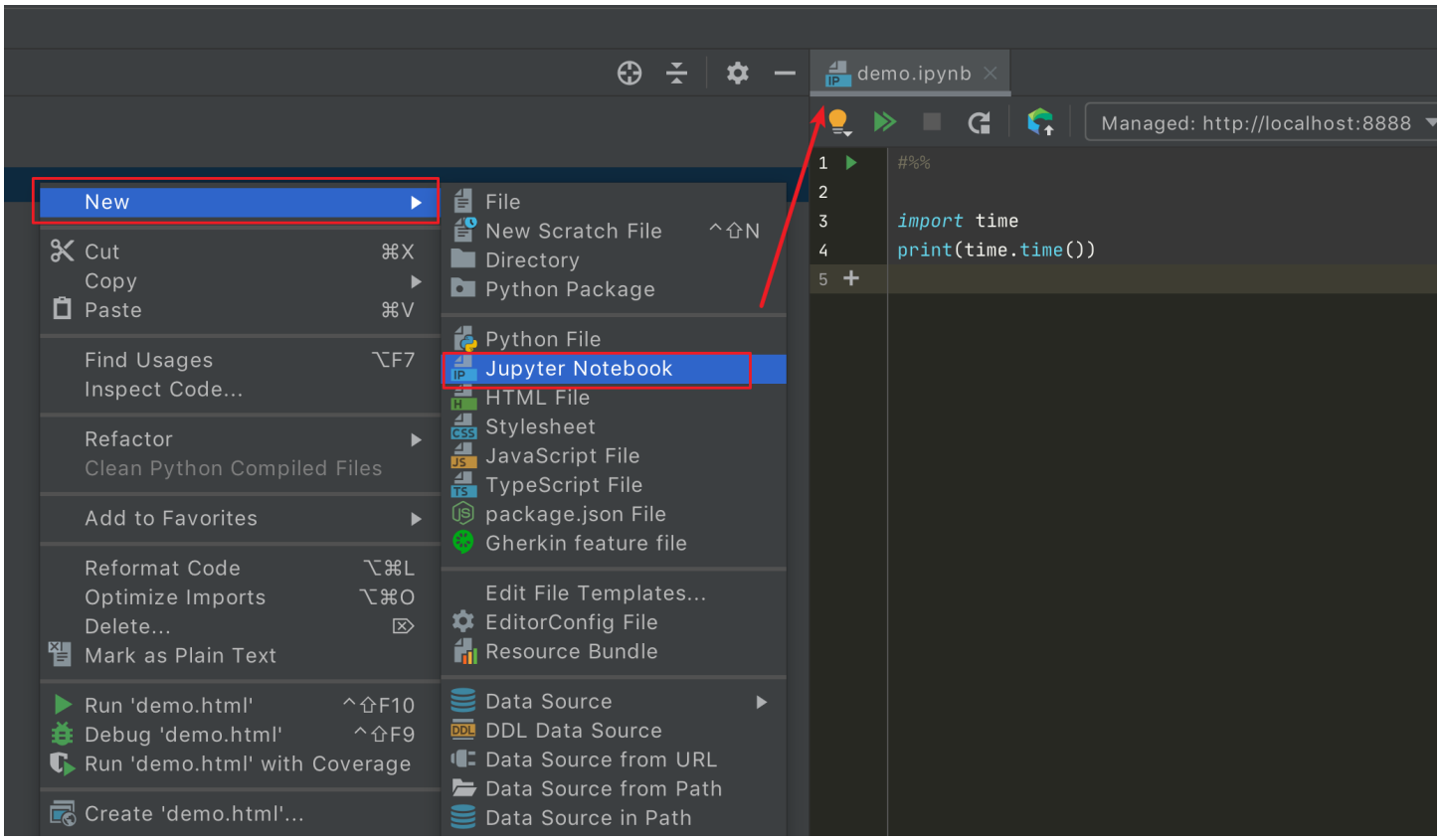


8.11 【绝佳工具 07】在 PyCharm 运行 Jupyter Notebook

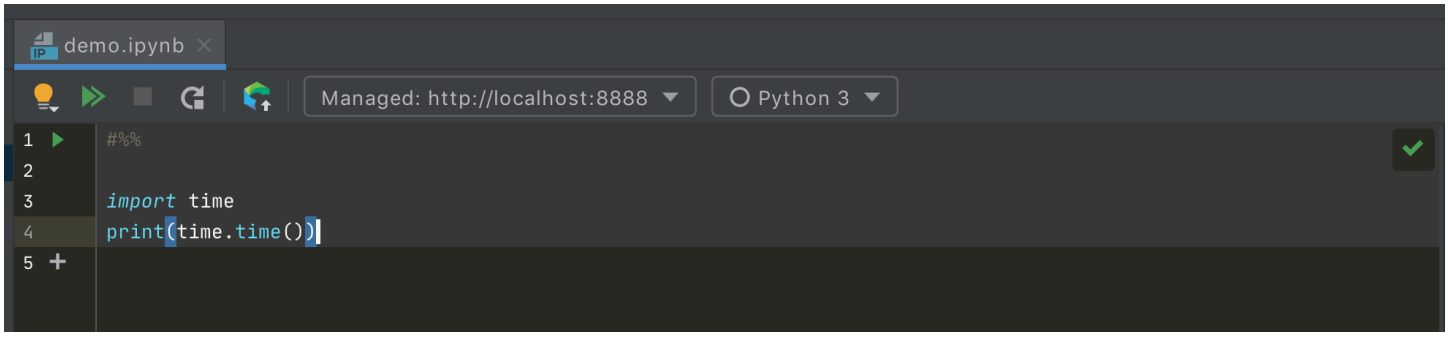
使用 Jupyter 之前， 先要安装它

```
$ pip install jupyter
```

然后按照下图指示新建一个 Notebook， 就可以开始运作了。

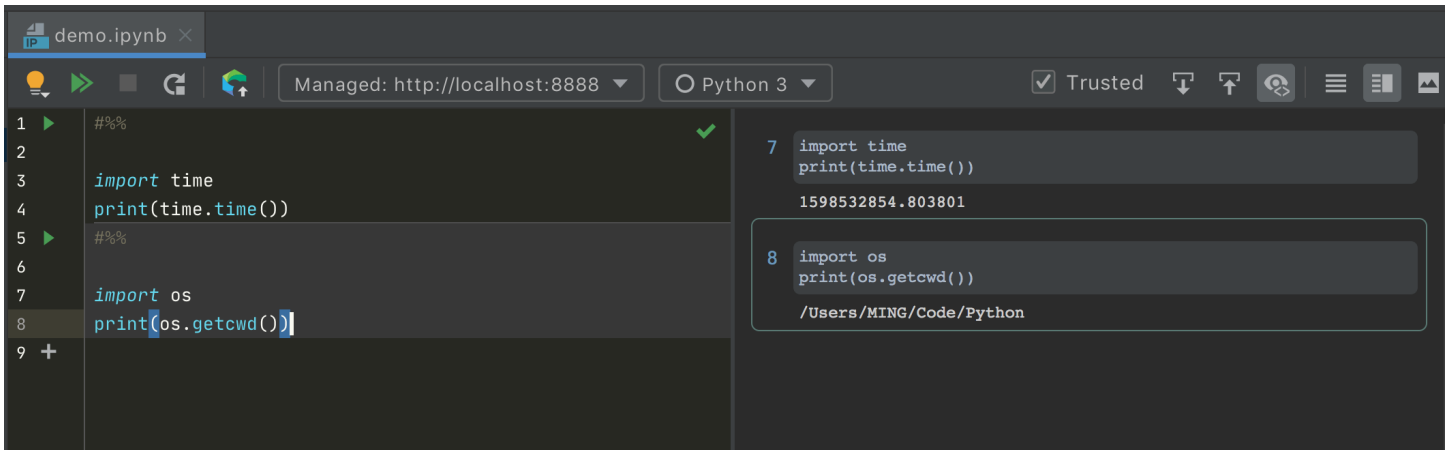


这个界面感觉和 Jupyter 的风格不太符

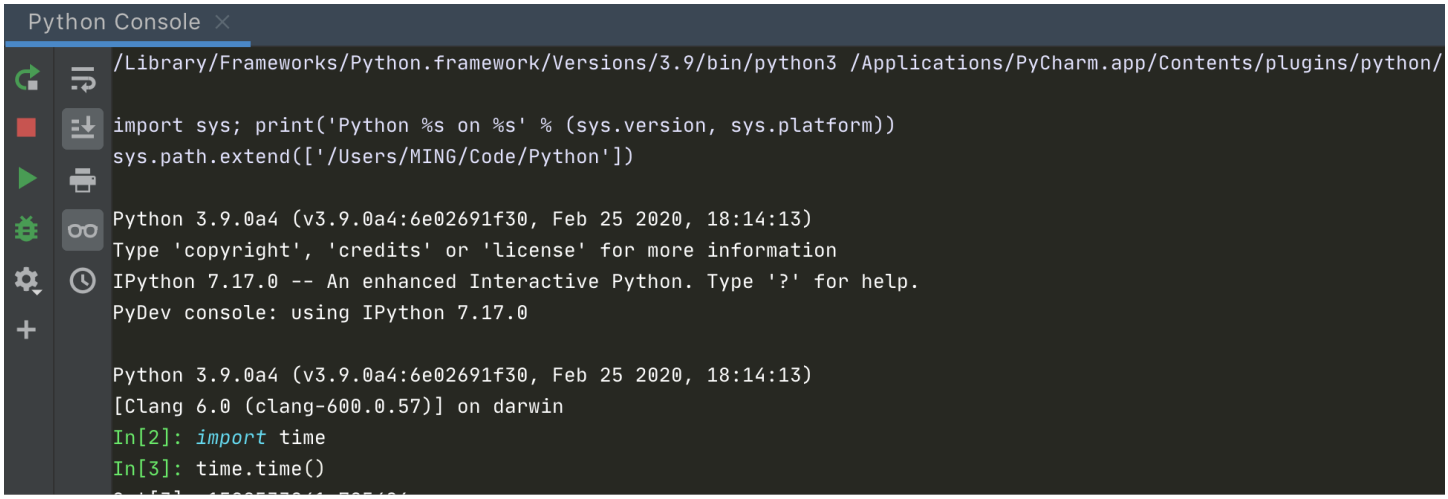


但是使用上是没有什么区别的，记住三个快捷键就好(下面指的是 Mac 上的，Windows 上的有所不同)

- Ctrl+Enter：运行该 cell
- Option + shift + Enter：调试该 cell
- Shift + Enter：插入一个新的 cell



只要你安装了 Jupyter 后，你使用 Python Console 也会自动变成 Jupyter 的模式



作者：王炳明
版本：v1.0
发布时间：2020年08月30日
微信公众号：Python编程时光
联系邮箱：wongbingming@163.com
项目主页：http://pycharm.iswbm.com
Github：https://github.com/iswbm/pycharm-guide

👆👆👆

回复 "pycharm"，获取最新版 PDF

第九章：常用的技巧

9.1 【必学技巧 01】轻松实现 JSON格式化

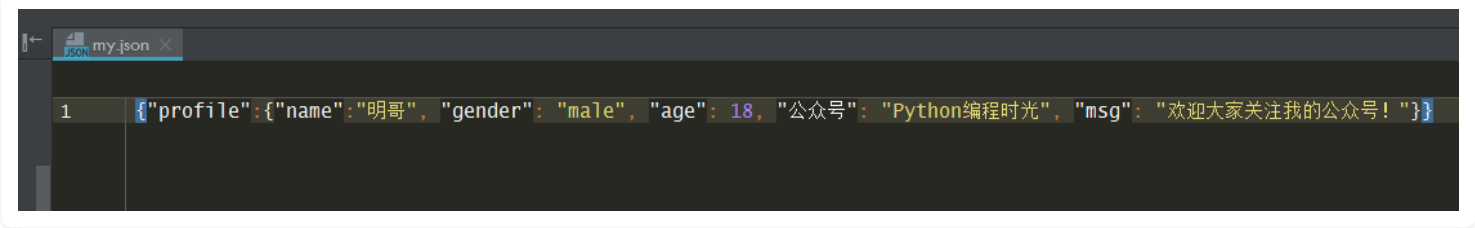
如下是一个未经美化的 json 文件，当一个 json 文件的内容很多时，若不经过工具重新美化，想要从中提取出有效的信息是一件很困难的事情。

```
{"profile":{"name":"明哥", "gender": "male", "age": 18, "公众号": "Python编程时光", "msg": "欢迎大家关注我的公众号！"}}
```

以前我经常使用一些在线的网站，比如：https://tool.oschina.net/codeformat/json

如果你的电脑无法连网，或者不喜欢多记一个网址，完全可以使用 PyCharnm 来解决这一诉求

没有经过美化是这样的：



按住 `Ctrl+Alt+L` 经过美化后是这样的

9.2 【必学技巧 02】误删项目？一秒找回

有一次由于自己的误操作，在没有任何备份的情况下，将一个自己写了两个星期的项目给删除了。待我回头神来的时候，我甚至都记不起是何时进行的删除操作。

做为一名老司机，当然是临危不惧地打开了回收站，进行一番搜寻，几个月没有清理过的回收站，真是一片狼藉，什么 jpg，avi 都有，不堪入目呀。

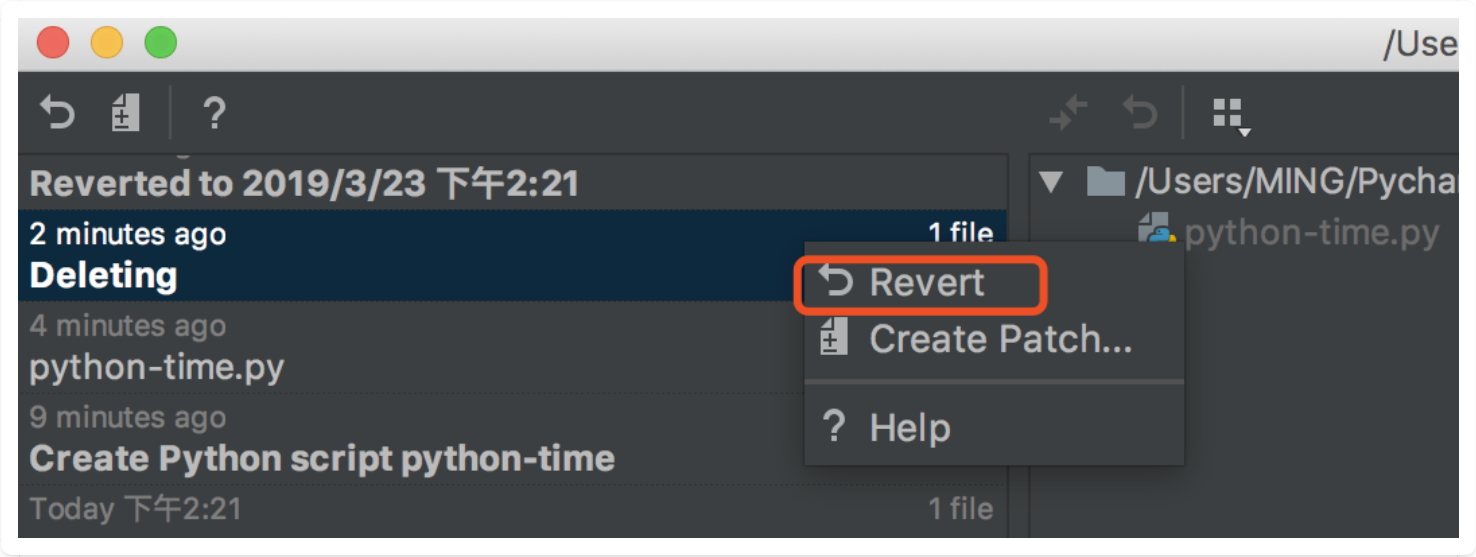
我用一分钟快速浏览了一下，没有发现我要找的那几个 py 文件，我心想，应该是文件太多了，看叉掉了。由于项目是最近写的，文件名我还清楚地记得，既然有文件名，那就利用windows 自带的搜索功能，结果还是没有，这下我才开始意识到事态的严重性，文件可能真的「没了」。

我已经很久没有清理过回收站了，为什么回收站里会没有我的文件呢?

我想这可能是一次非同寻常的 delete，会不会是在 Pycharm 里的 发起删除操作，不会往回收站里丢呢？经过一番测试，在回收站还真的找不着，但是这次尝试也无意中发现了 Pycharm 的隐藏的一个功能 `Local History`，它会保存你对文件的所有操作记录。

就拿我的刚刚测试的文件来举例，我先是新建了一个文件，然后对在这个文件里添加了几行代码。最后我将这个文件删除了。

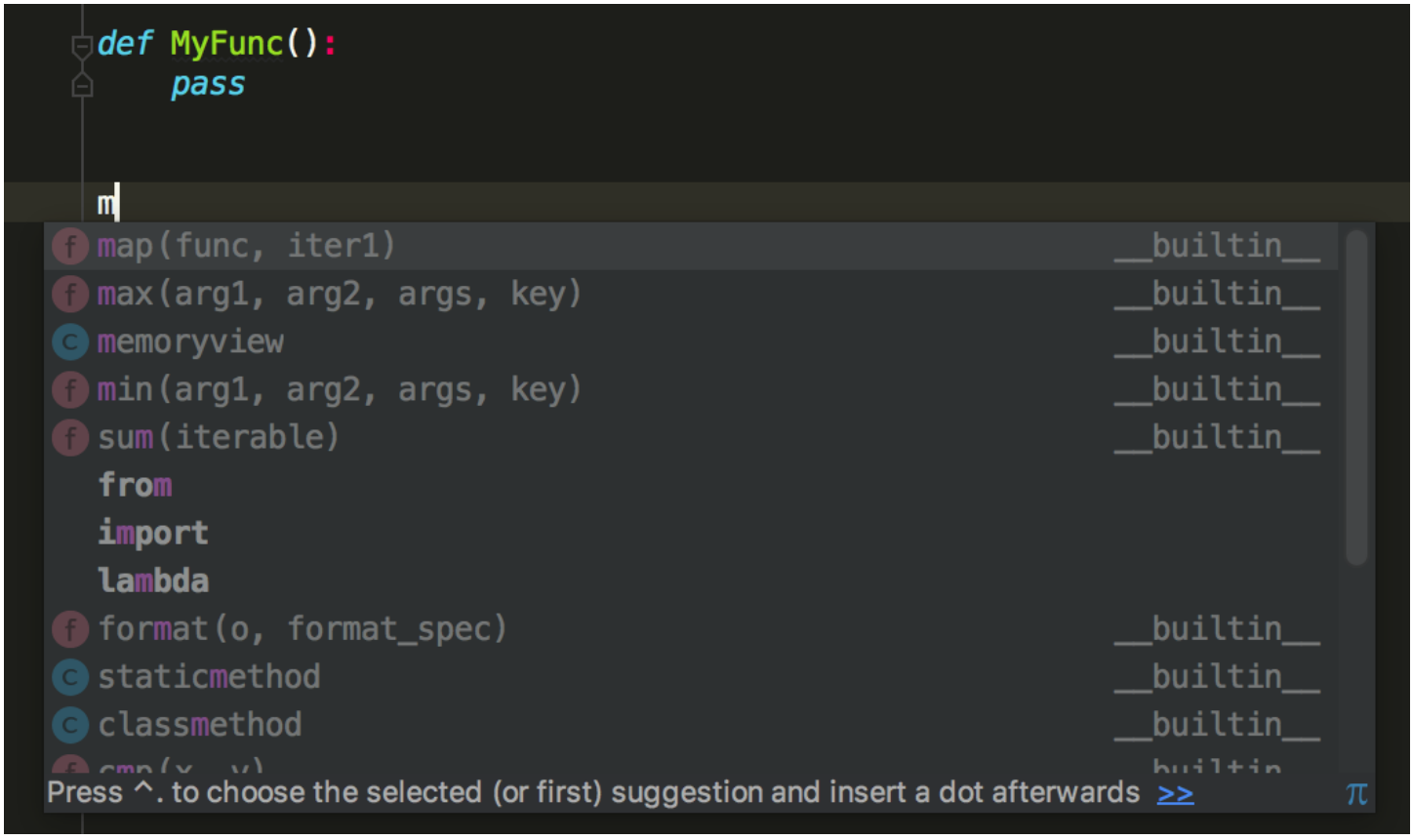
此时你可以在你的项目目录里，点击右键，有个 `Local History` 的选项，再点击子选项 `Show History`，你可以看到这里有个记录板。如果你想恢复删除的文件，就在删除的记录项点击右键，选择 `Revert` 即可恢复。



9.3 【必学技巧 03】智能补全，忽略大小写

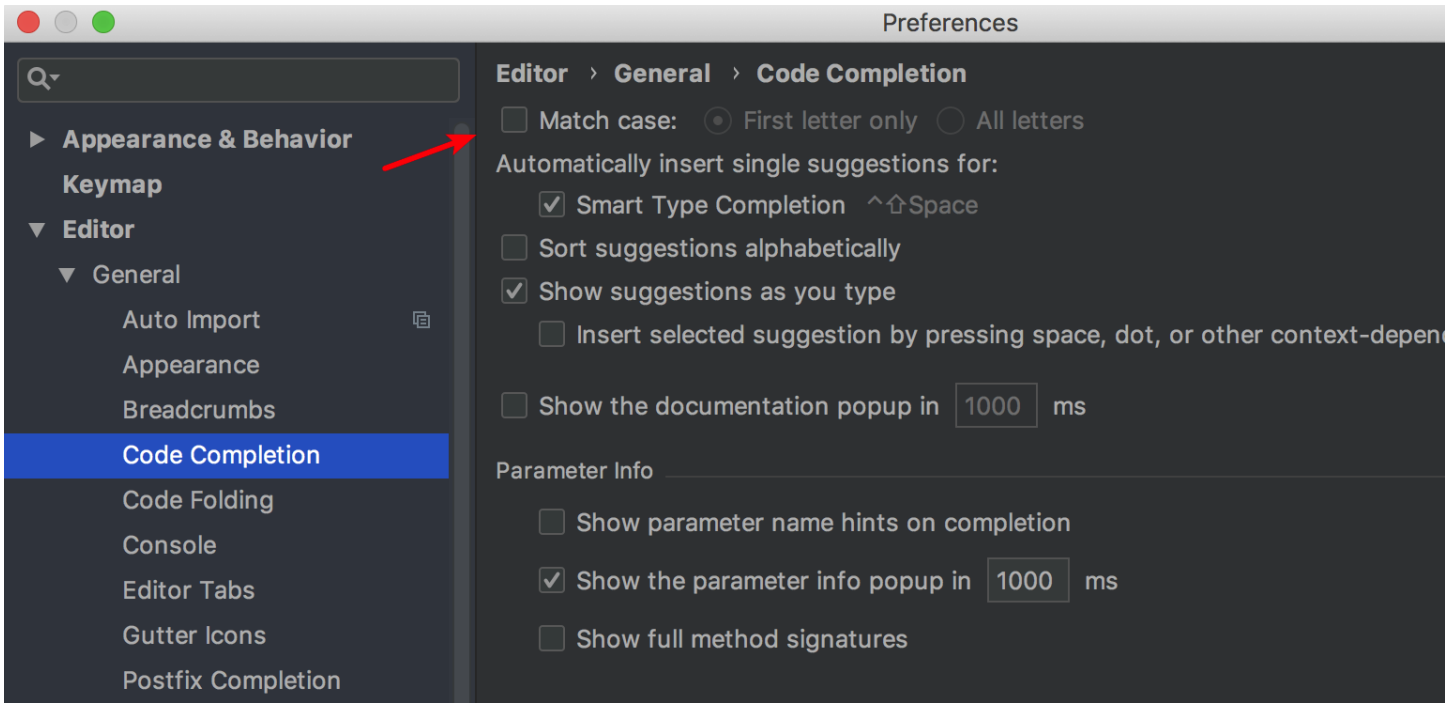
智能搜索补全，是IDE的最吸引人的功能之一。

当你的对象是以大写字母开头时，而你使用小写字母编写代码时，是不能查找到该函数的，你必须得先切换成大写再输入一遍。

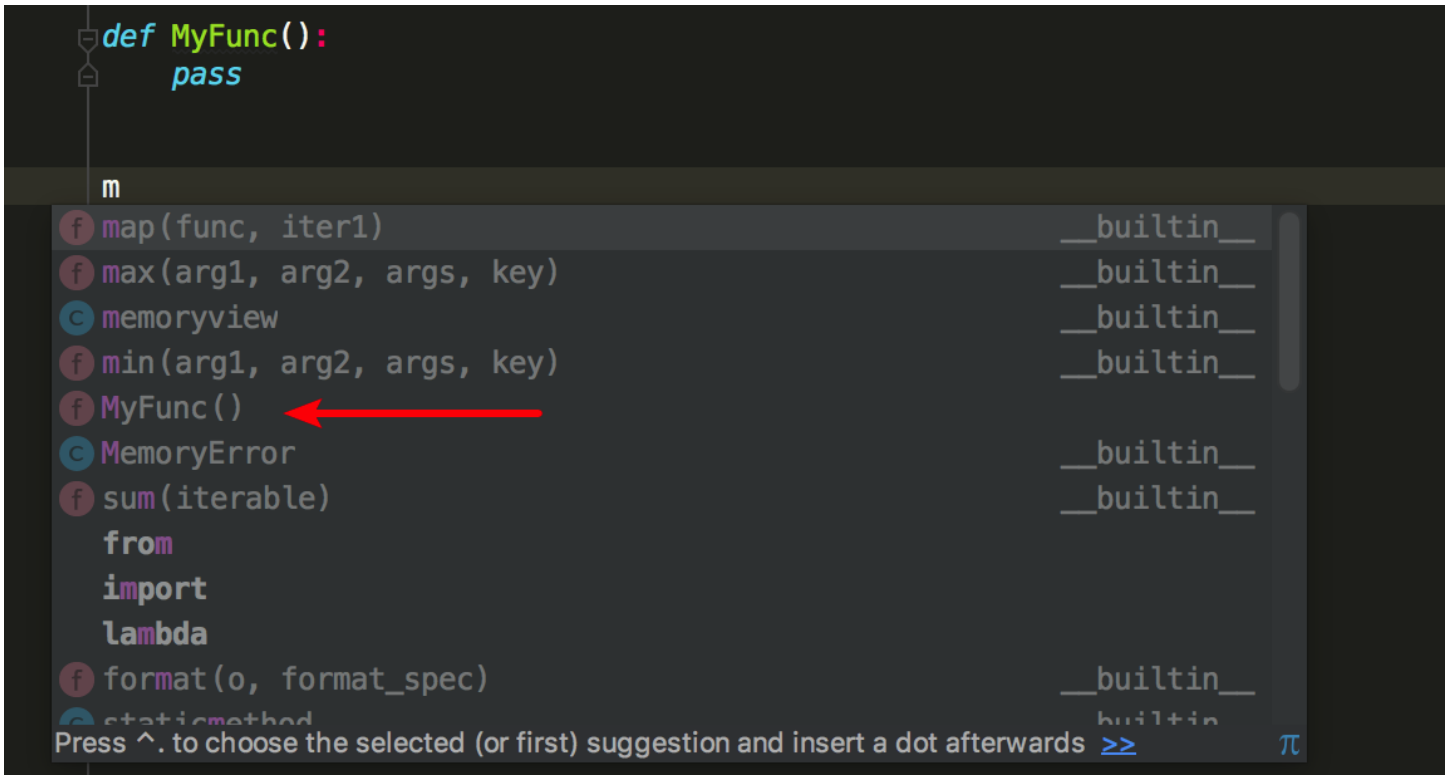


如何避免这种尴尬的情况？

只要在配置中关闭大小写匹配即可。



效果如下：

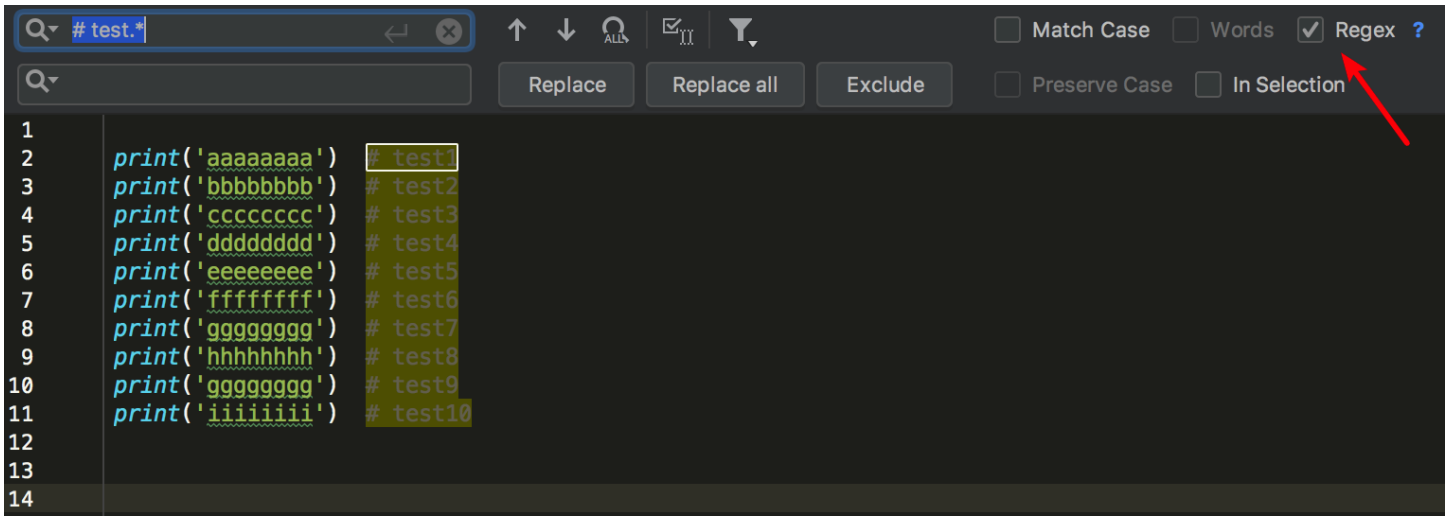


9.4 【必学技巧 04】以列为单位的块编辑

先给你出道小题，像下面这段代码，如果在不影响代码的情况下，快速删除后面代码后面的注释呢？

```
print('aaaaaaaa')  # test1
print('bbbbbbbb')  # test2
print('cccccccc')  # test3
print('dddddddd')  # test4
print('eeeeeeee')  # test5
print('ffffffff')  # test6
print('gggggggg')  # test7
print('hhhhhhhh')  # test8
print('gggggggg')  # test9
print('iiiiiiii')  # test10
```

我能想到的有两种方法，如果像如上这种有规律的注释，可以使用 正则匹配 + 替换 来实现。



对于这个场景我想到了可以用 vim来轻松的解决，vim 支持块编辑，可以以列为单位选择区域然后进行操作，这在vim中是很常用的一个取消注释的操作。

同样回到 PyCharm 中来，你会发现它也支持块编辑。

如果你使用的是旧版本的 PyCharm ， 当你按住 alt（windows）或者option（mac）， 然后使用鼠标进行选择，你会发现这样一件神奇的事情。

```
1
2   print('aaaaaaaa')  # test1
3   print('bbbbbbbb')  # test2
4   print('cccccccc')  # test3
5   print('dddddddd')  # test4
6   print('eeeeeeee')  # test5
7   print('ffffffff')  # test6
8   print('gggggggg')  # test7
9   print('hhhhhhhh')  # test8
10  print('gggggggg')  # test9
11  print('iiiiiiii')  # test10
12
13
14
```

> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

如若上面的快捷键不生效，说明你的 PyCharm 是旧版本，在较版本中，有两种方法开启列选择模式

1、使用快捷键 Command + shift + 8

2、点击右键，选择 『Column Selection Mode』

在新版本中，列选择的功能变成了一种模式，开启才能使用，使用完后还需要关闭。相比旧版本，个人认为这个改变不好，不能即用即走。

9.5 【必学技巧 05】阅读源码的六种方法

在你使用一个模块的函数时，如果想查看这个函数的源码，有两个思路

第一个思路

进入函数声明的位置，就可以看到源码了。

对应的快捷键有如下四组：

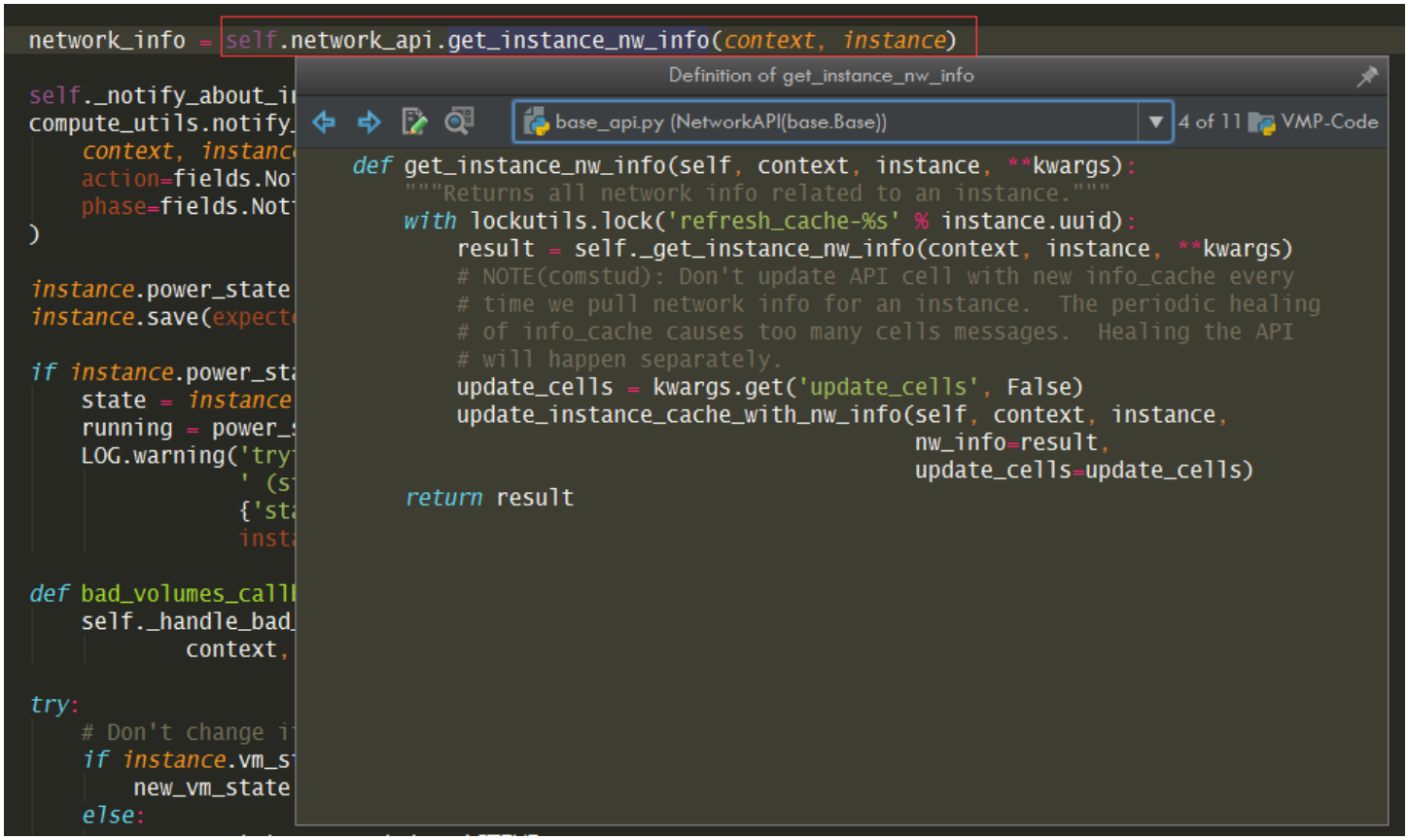
1. ⌘ + B：Go to Declaration or Usages
2. F4：Jump to Source
3. ⌘ + ⌥ + B ： Go to Implementation(s)

4. ⌘ + 鼠标左键
5. ^ + ⇧ + B：跳转到类型声明处

第二个思路

在当前页面弹出一个小窗口，直接显示 源代码，不用像上面一样跳转到另外一个页面。

快捷键是：⌘ + ⇧ + I



9.6 【必学技巧 06】快速重构，修改所有函数与变量

在某同事离职交接代码时，有时候会在心里默默吐槽。

为了后期维护的方便，我通常会花个几天的时间对其代码进行了大量的重构。

变量重命名

重构代码，免不了要对变量进行重命名。

如果一个一个改，显然不太智能，要知道我们是在用IDE，你也许会说，用搜索全部替换不就行了？还真不行。

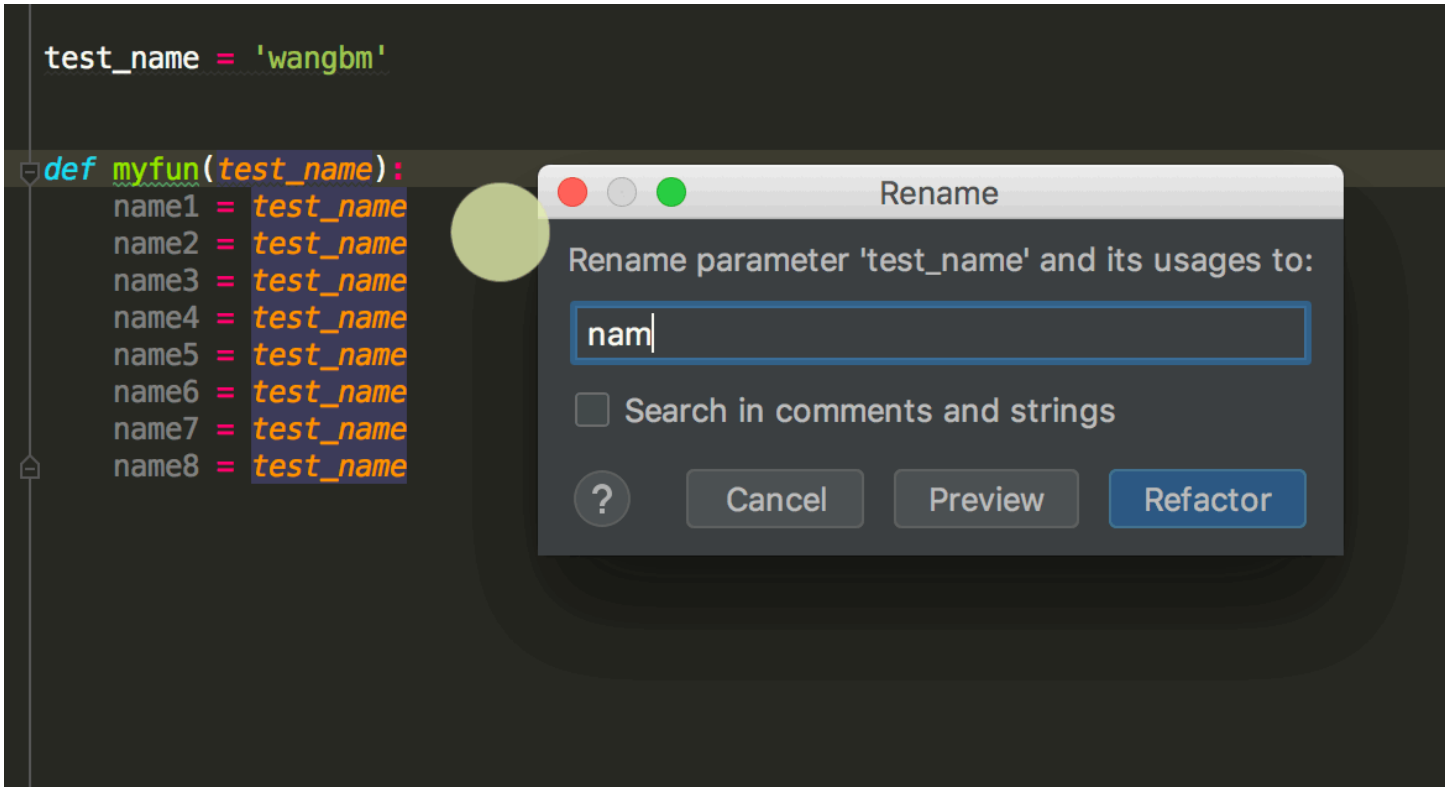
比如下面这段代码，我只想改myfun 里的的test_name，而对于全局下的同名变量是不应该修改的。如果你全局替换，就会有误伤。

```
test_name = 'wangbm'

def myfun(test_name):
    name1 = test_name
    name2 = test_name
    name3 = test_name
    name4 = test_name
    name5 = test_name
    name6 = test_name
    name7 = test_name
    name8 = test_name
```

这时候，我们如何做呢?

可以使用 PyCharm 的 Refactor 功能，它会自动匹配作用域，既做到批量更改，也做到不误伤。

操作方法很简单，先选中你的变量，然后使用快捷键 Shift+F6，就可以直接重命名了。



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

函数重命名

如果是对函数重命名，使用 Shift + F6 也是可以的，只不过要点很多下。

更合适的方法是使用 Command + F6，演示过程如下

```python
import socket  # 导入 socket 模块
import time

def main_init():
    s = socket.socket()  # 创建 socket 对象
    host = socket.gethostname()  # 获取本地主机名
    port = 13200  # 设置端口
    s.bind((host, port))  # 绑定端口

    for i in range(5):
        print(i)

    s.listen(5)  # 等待客户端连接
    while True:
        c, addr = s.accept()  # 建立客户端连接
        c.send('hello'.encode("utf-8"))
        print(c.recv(1024))
        time.sleep(1)
        c.close()  # 关闭连接


if __name__ == '__main__':
    main_init()
```

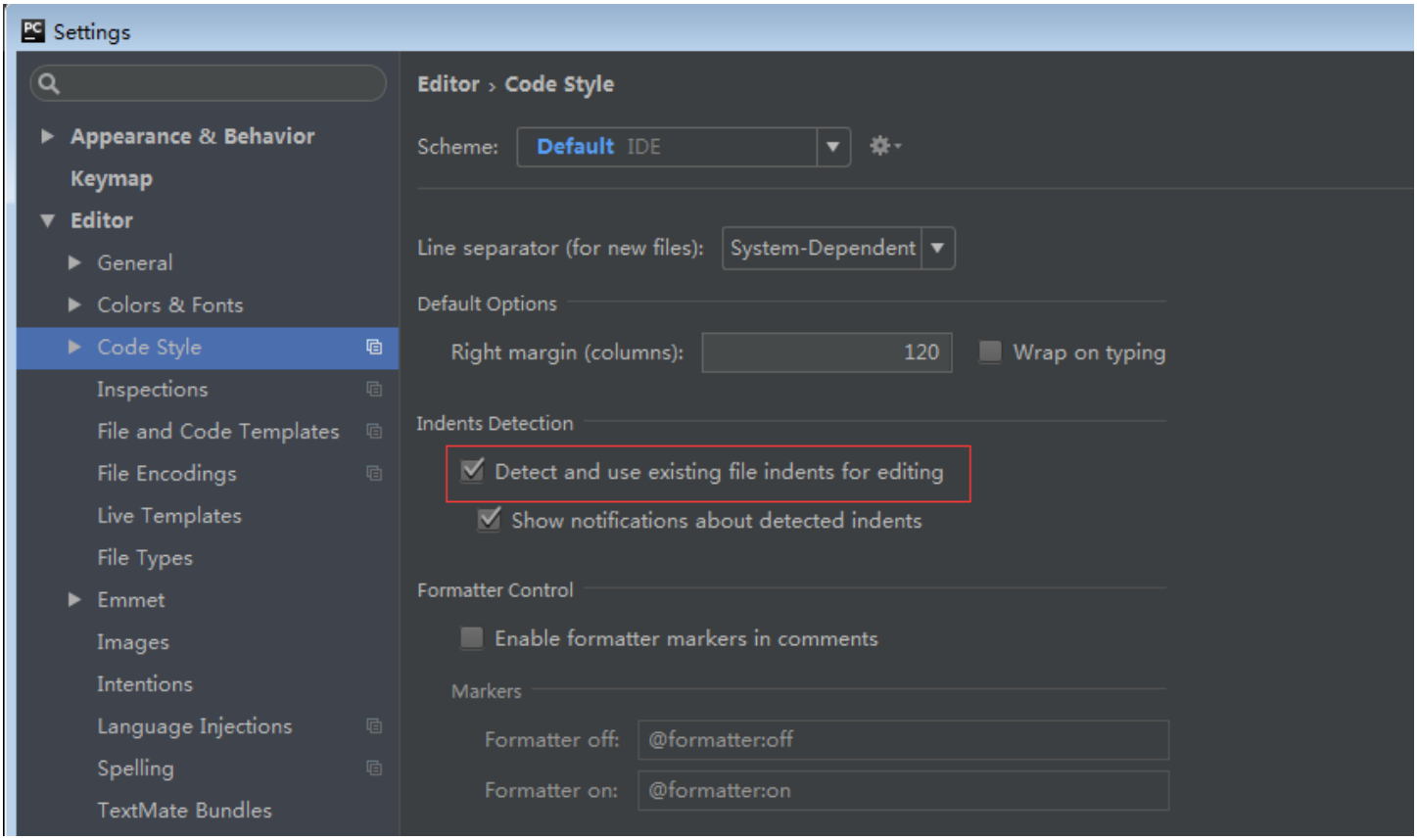
> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

9.7 【必学技巧 07】tab和空格混用自动转换

在团队协作中，你难免会动到别人编辑的文件，有的人喜欢做tab做缩进，有的人喜欢用四个空格做缩进。

但是在同一个Python文件模块里，tab 和 四个空格缩进两种风格是不能共存的。这就需要你按照该文件原来的缩进风格来进行编码，在 Pycharm 里，可以设置自动检测原文件的缩进风格来决定当你使用tab键缩进的时候，是TAB还是四个空格。

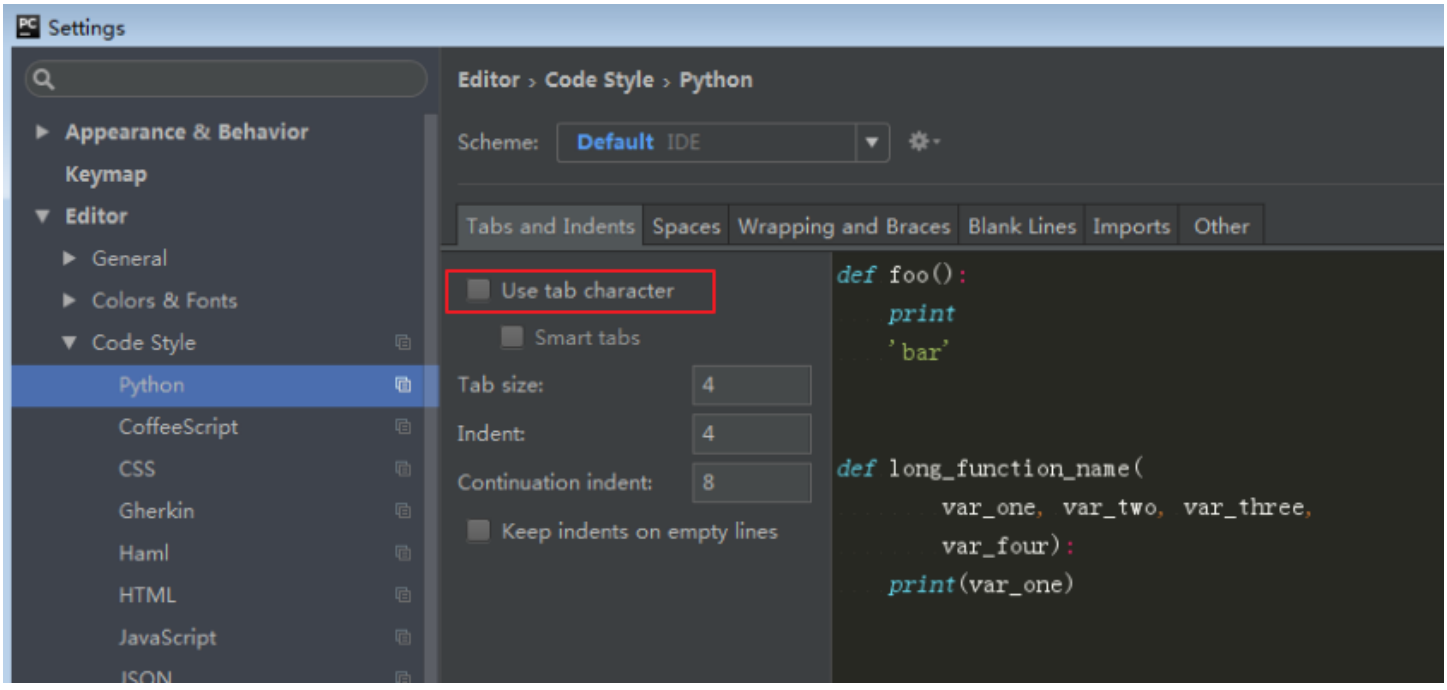
在图示位置打勾即可开启自动检测。



上面是对一个旧的 Python 模块进行修改时，如何决定当前编辑的缩进方式。

而对于新建模块，默认的缩进方式，是如何确定的？

如下图，若在 `Use tab character` 打上勾，则你新建一个 Python 后，就会使用 TAB 进行缩进，反之，则使用四个空格进行缩进。



9.8 【必学技巧 08】脱离鼠标的代码区域选择：Extend Selection

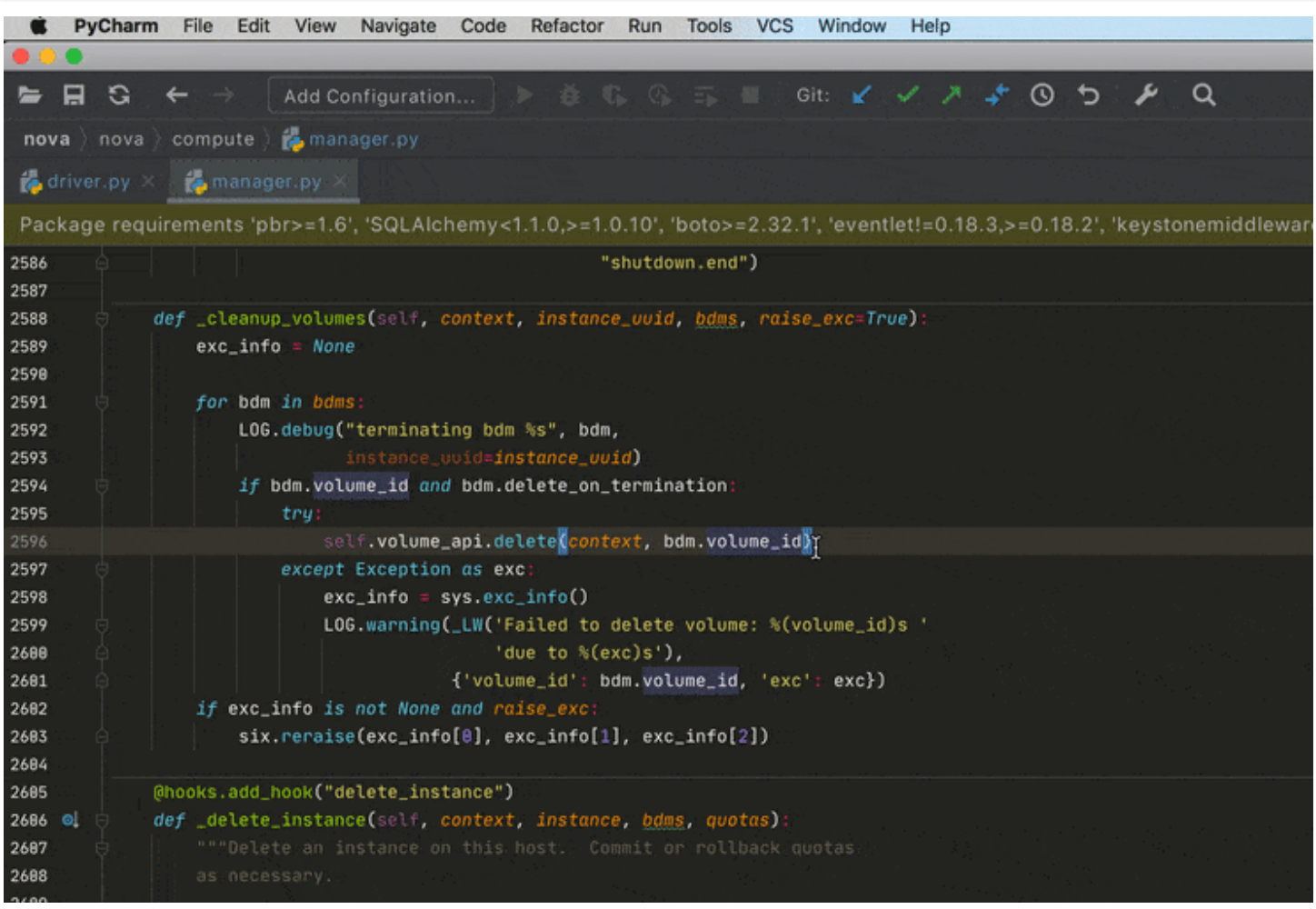
根据选中的区域的大小，可以分为：

1. 选中单词
2. 选中表达式
3. 选中单行
4. 选中代码块
5. 选中函数
6. 选中类

对于代码区域，通常都要借助鼠标才能完成，这里给你推荐一组快捷键，可以让脱离鼠标进行区域的选择：

- ⌘ + W：扩大选中的区域
- ⌘ + ⇧ + W：缩小选中的区域

演示的效果如下：



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

这种用法，适用于：

1. 使用 mac 笔记本，不使用鼠标，只使用触控板的人群
2. 想要选中一个上千行的类或函数，进行操作的人

9.9 【必学技巧 09】从可视化 Python 包管理器

PyCharm 在配置了解释器后，下方会列出你的该环境下所有已安装的包。

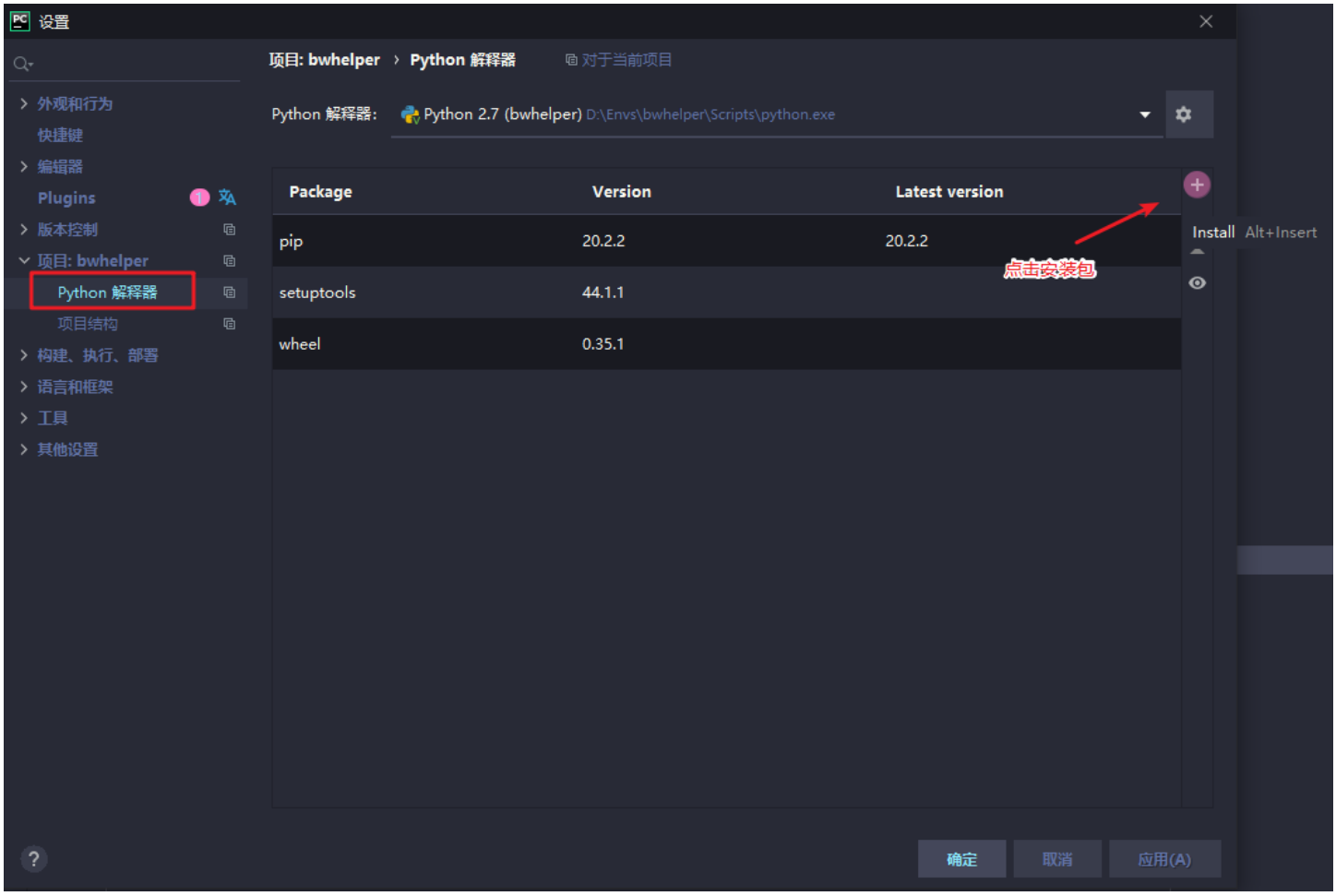
在右边有四个按钮

1. 安装
2. 卸载
3. 升级
4. 查看早期版

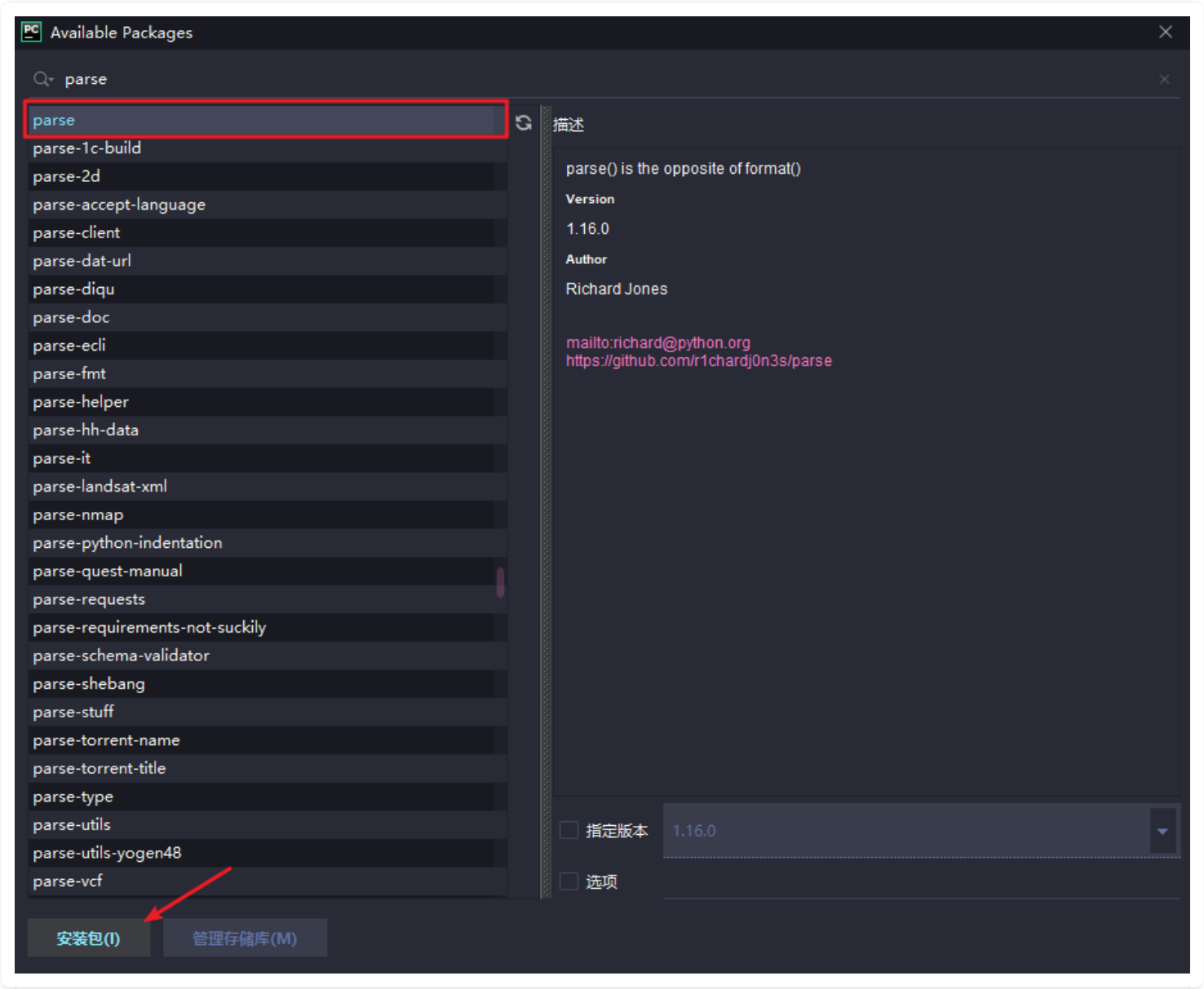
你可以通过他们对这些包进行管理。

下面以安装 parse 包为例：

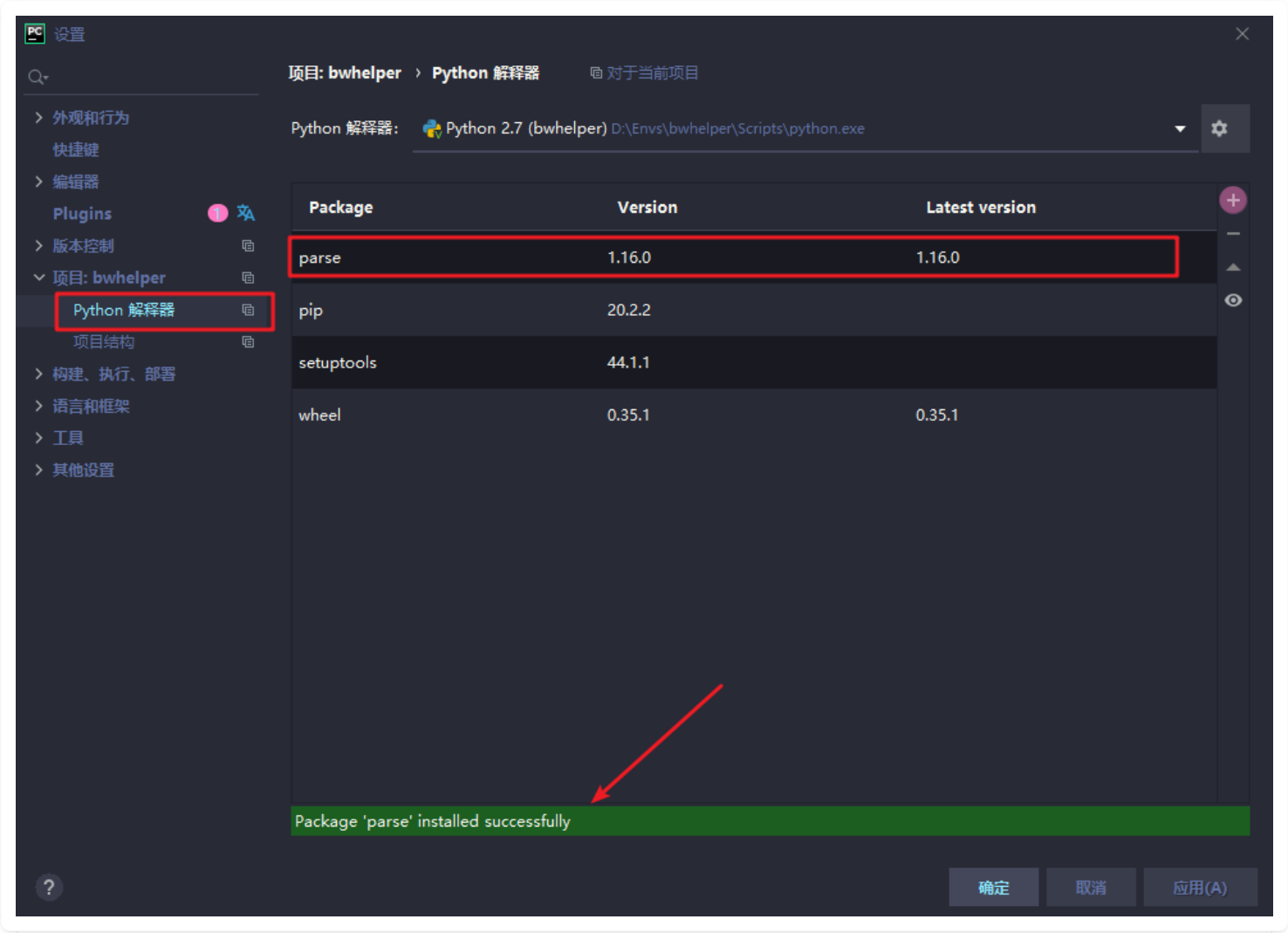
点击 `+` 按钮



搜索 `parse`，并点击右下角进行安装



退回包管理界面，已经安装成功了。



9.10 【必学技巧 10】快速移动/拷贝文件：F6/F5

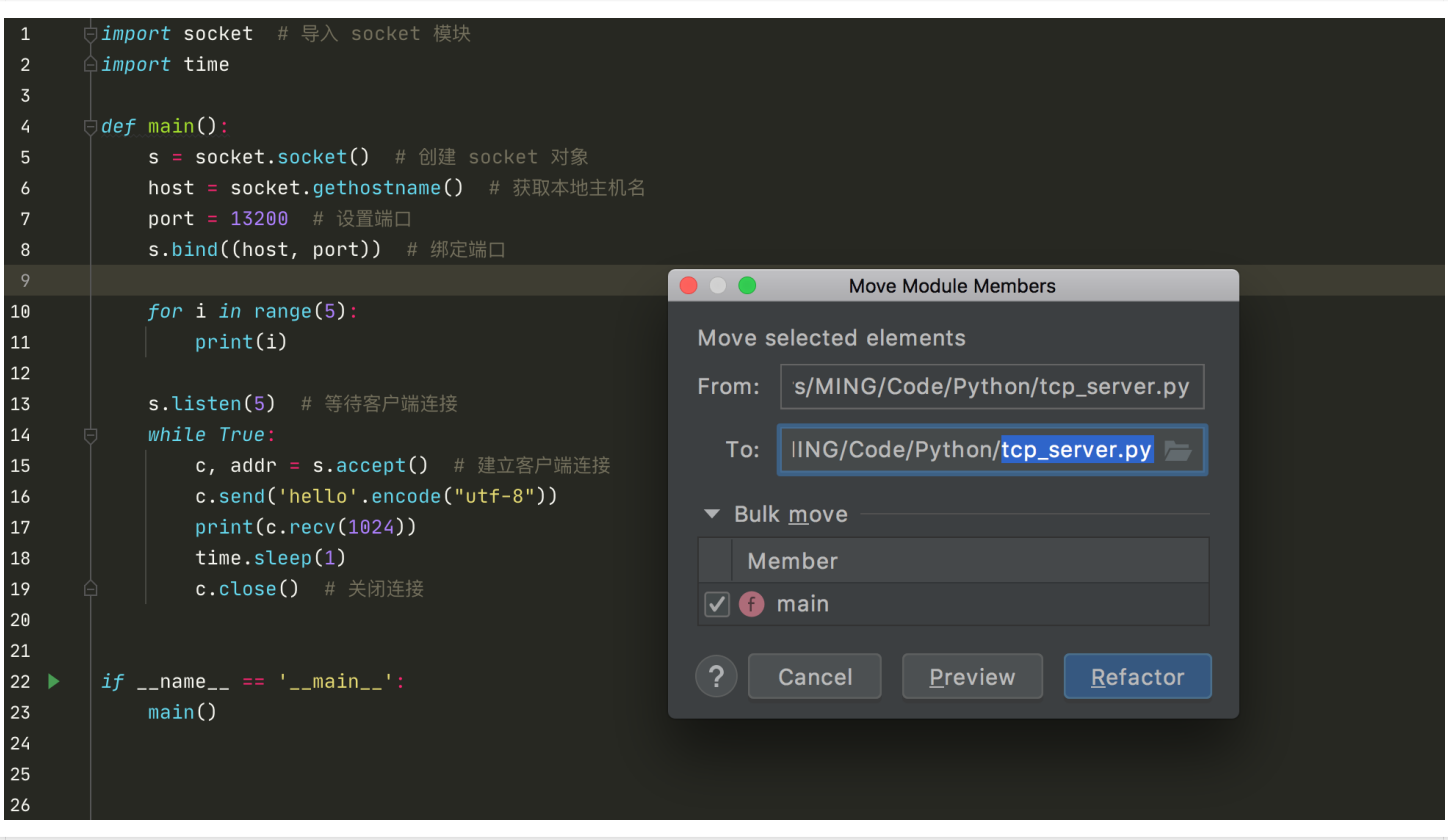
当你要把一个文件拖动到另外一个目录的时候，正常人的操作有两种：

1、直接拖拽过去(个人感觉这种才是最方便的)

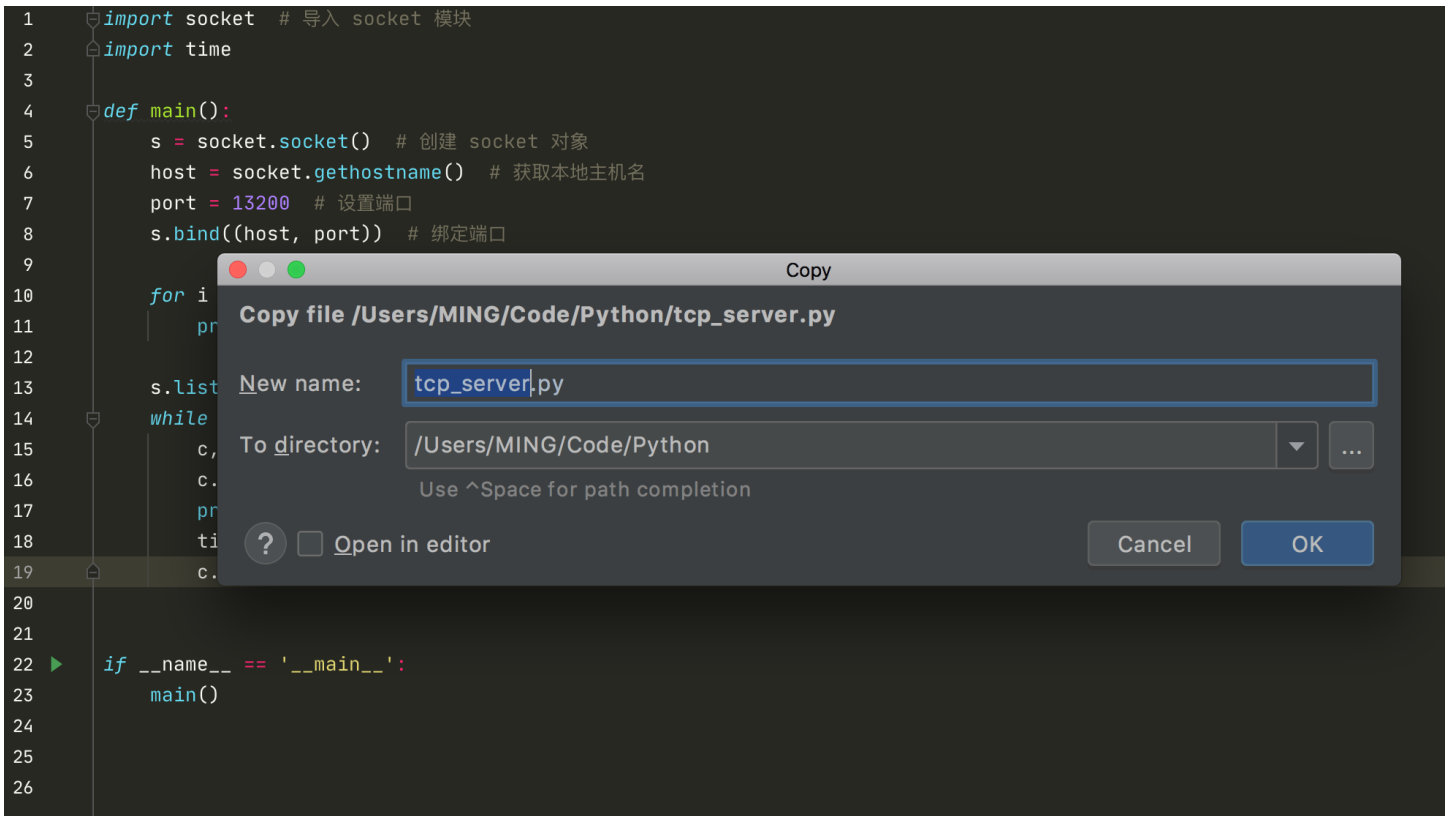
2、先剪切，再粘贴

PyCharm 对于这种重构操作，有更方便的入口。

只要按住 F6 就会弹出一个 Move Module 的窗口，直接选目标目录就可以。



除了移动之外， 拷贝也是可以的。快捷键变成了 F5

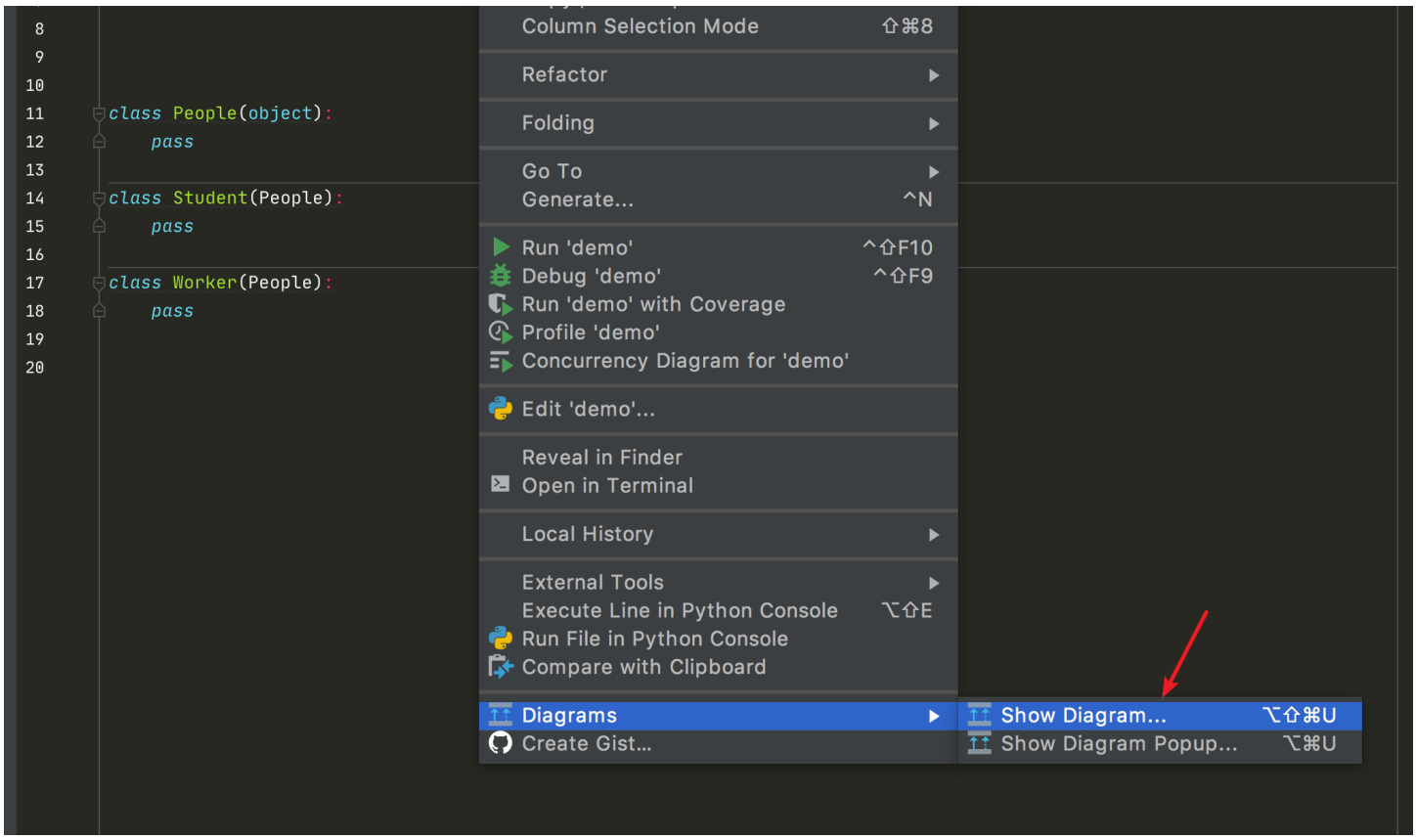


9.11 【必学技巧 11】显示类继承关系图：Show Diagrams

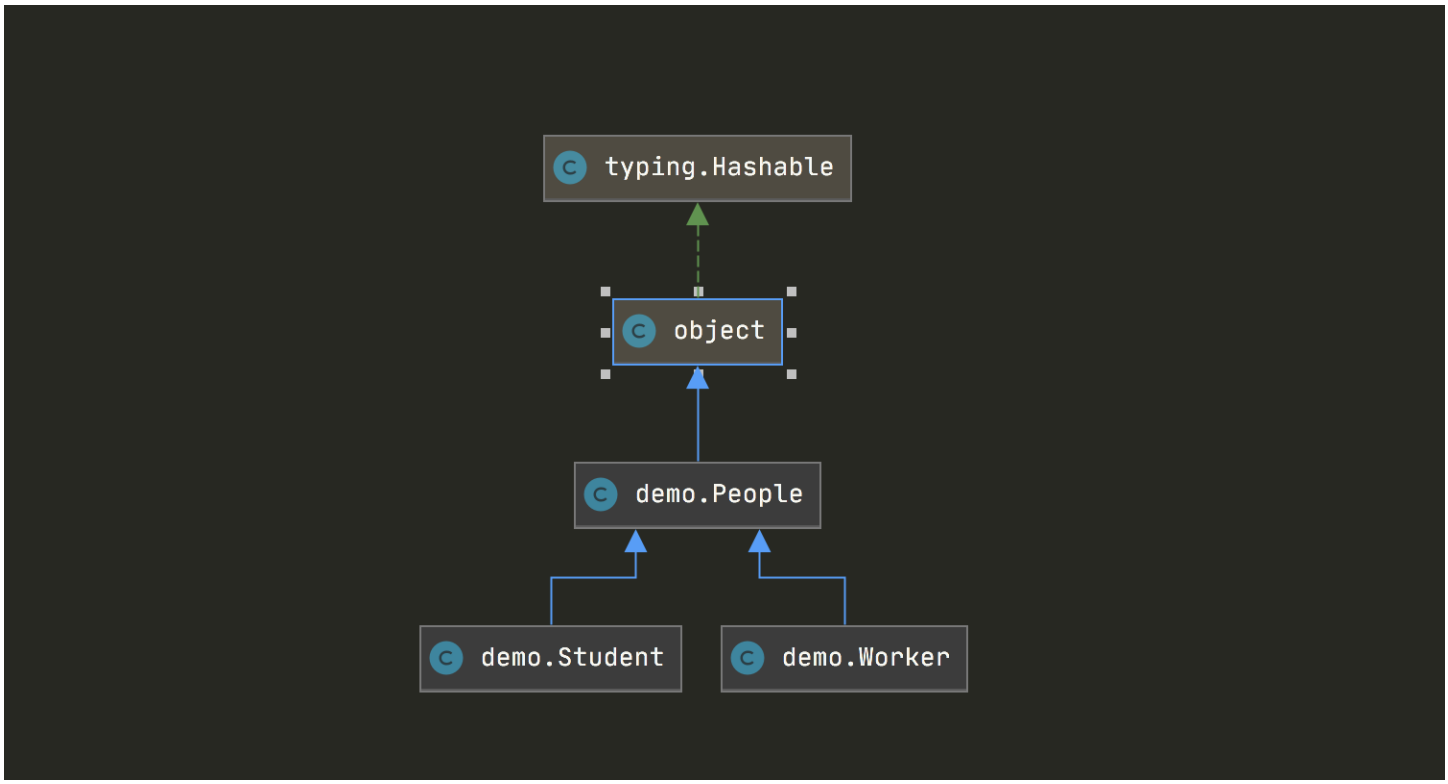
在阅读一些比较庞大的项目时， 如果类的继承关系比较复杂， 会给我们阅读源码带来不小的阻碍。

面对这种情况，本篇的这个技巧就能派上用场了。

在你想查看继承关系的 类 中，右键选择 `Diagrams` -> `Show Diagram`



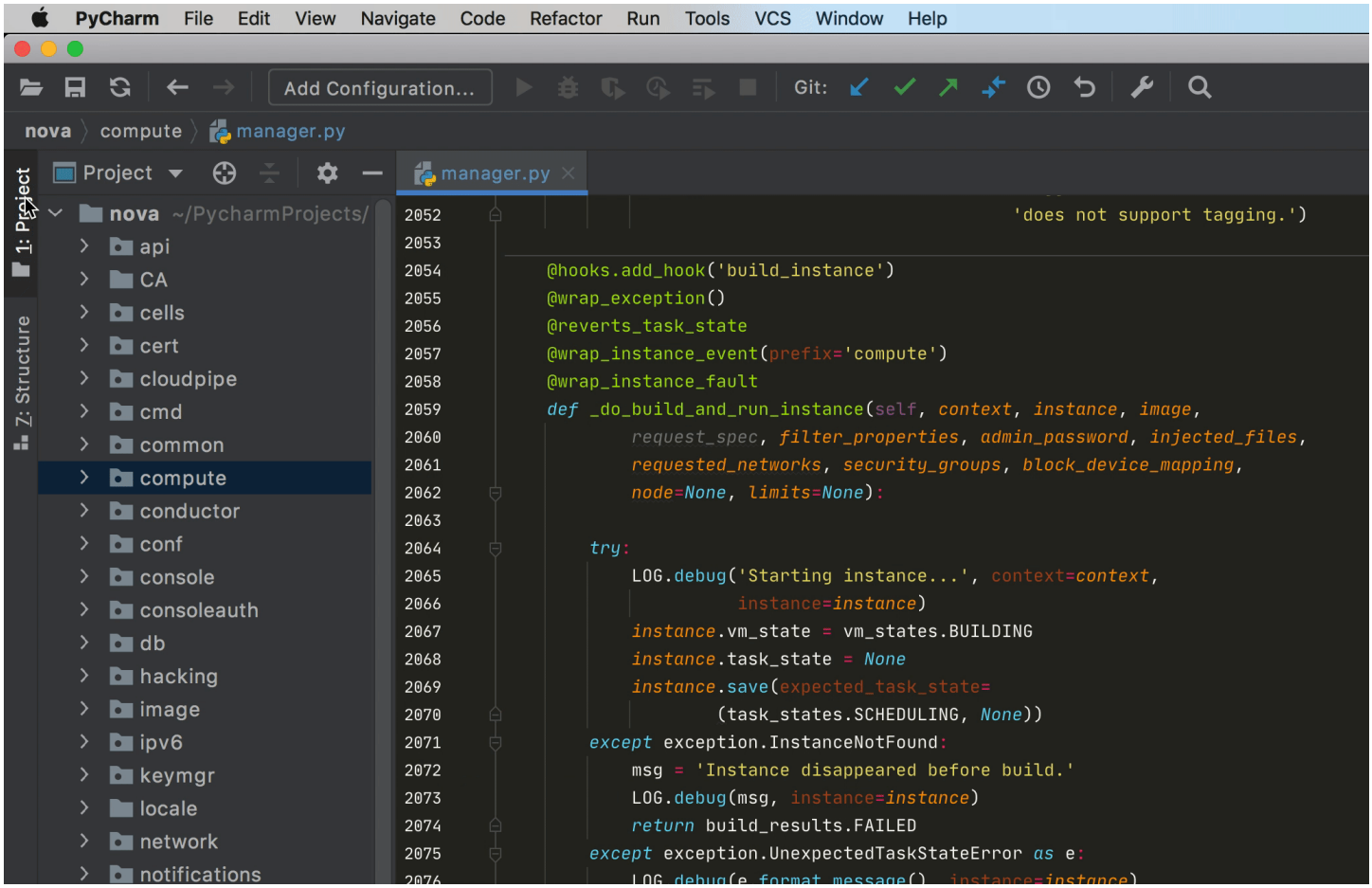
就会新增一个窗口，使用 UML 为你展示该类的继承关系。



9.12 【必学技巧 12】快速隐藏项目树

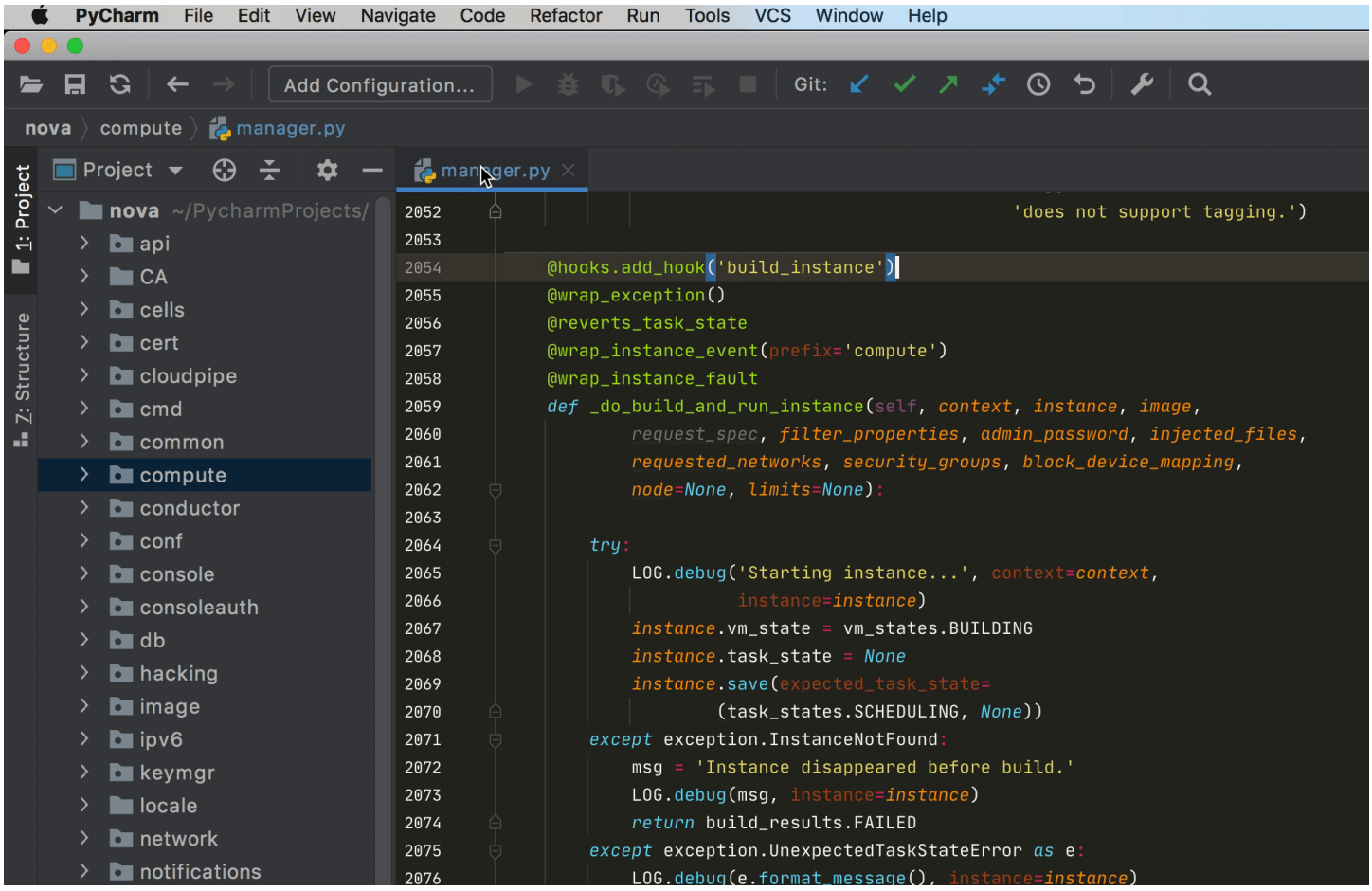
当你使用笔记本的小屏幕写代码时，左边的项目树就会显示特别的占空间。

通常人都会手动操起鼠标，去点击最左边的按钮或者点那个最小化的按钮。就像这样



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

但是其实还有其他更好的方法，双击标签页，就可以把它隐藏起来。

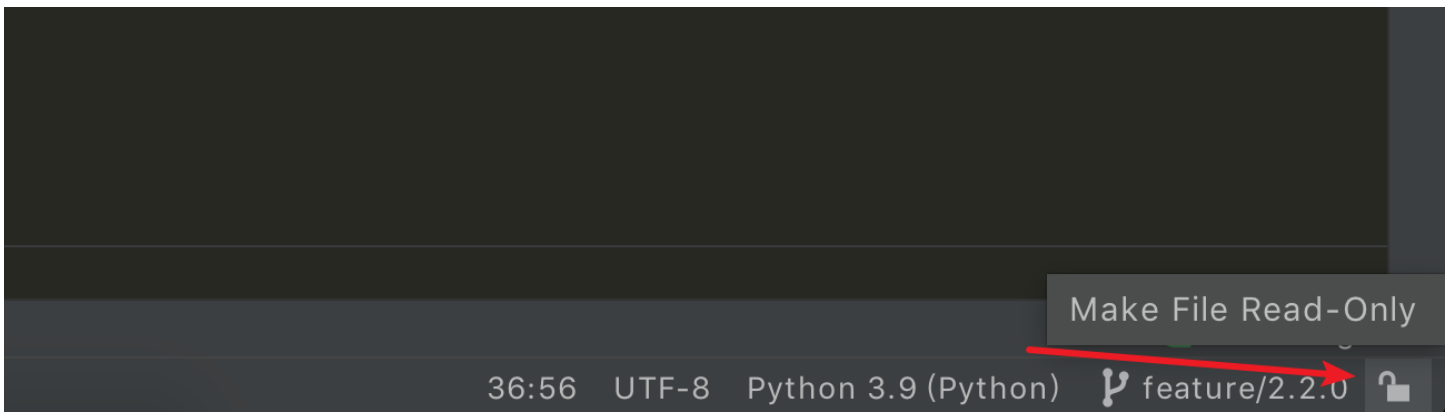


> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

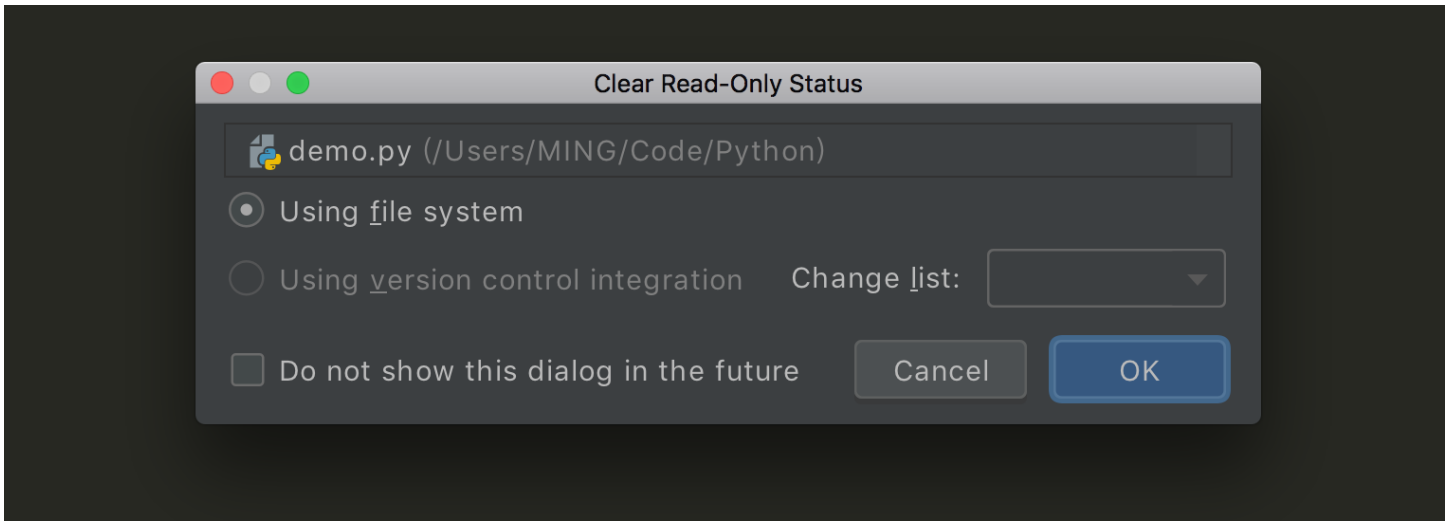
9.13 【必学技巧 13】把文件设置为只读：Read-Only

如果担心代码因为被自己不小心修改到，可以把该文件设置为只读。

方法很简单，只要点击右下角的 小锁 锁上该文件就可以。



锁上后，想编辑时，就会弹出下面的窗口。



只要你点击上面的 OK ， 就可以编辑了。

因为这个功能有限，只能作一个提醒，没办法作强制约束。

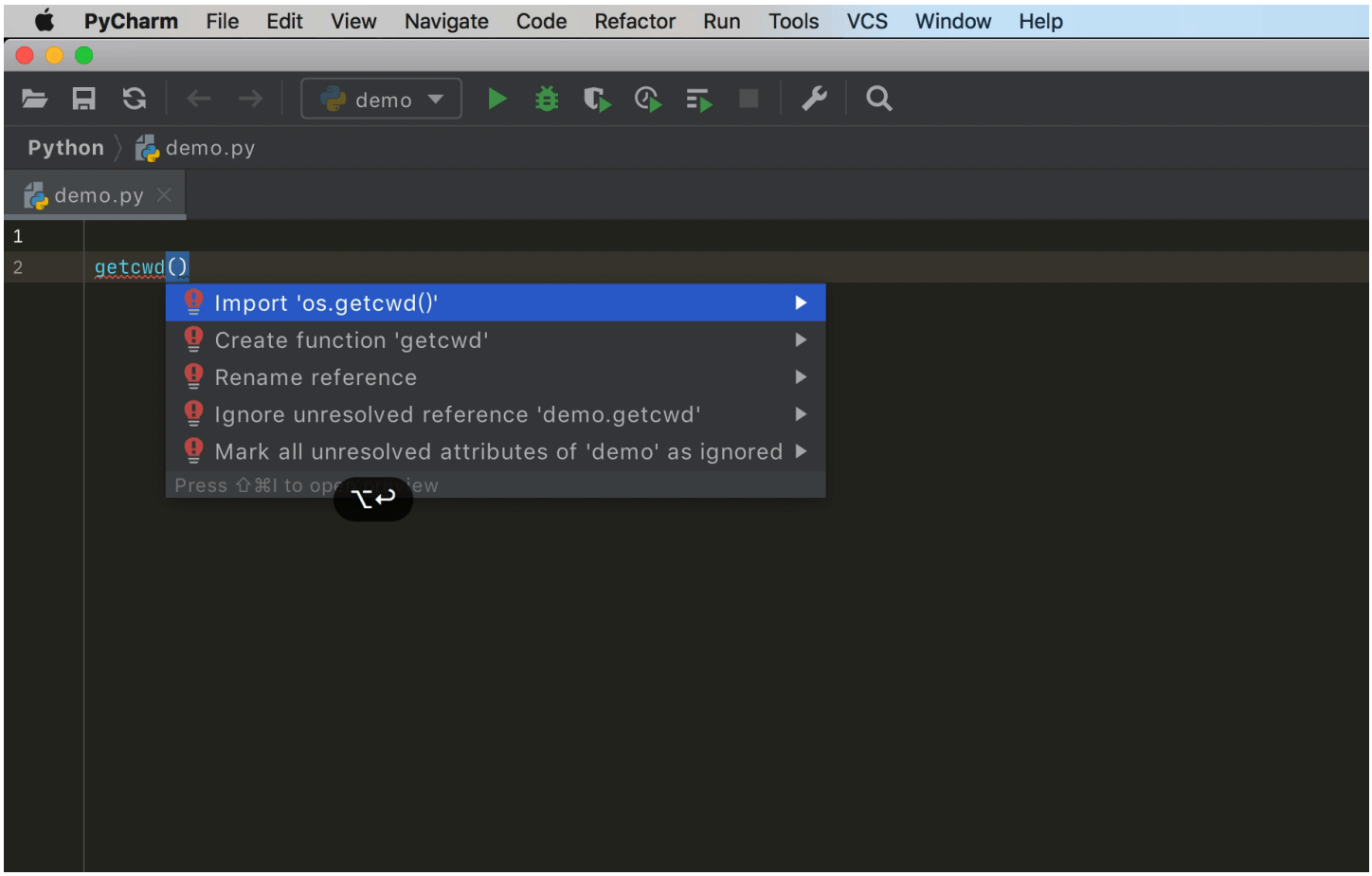
9.14 【必学技巧 14】自动导入解决依赖：Alt+Enter

当你使用某个包的某个函数，一般都是需要先导入该包，才能使用的。

有了 PyCharm 后，有多省心呢，它能够根据你的函数，自动查找该函数可能属于的包，经过你的确认后，便能自动导入。

查找的快捷键是 Option + Enter （Windows 上是 Alt + Enter）。

最终的使用效果如下：

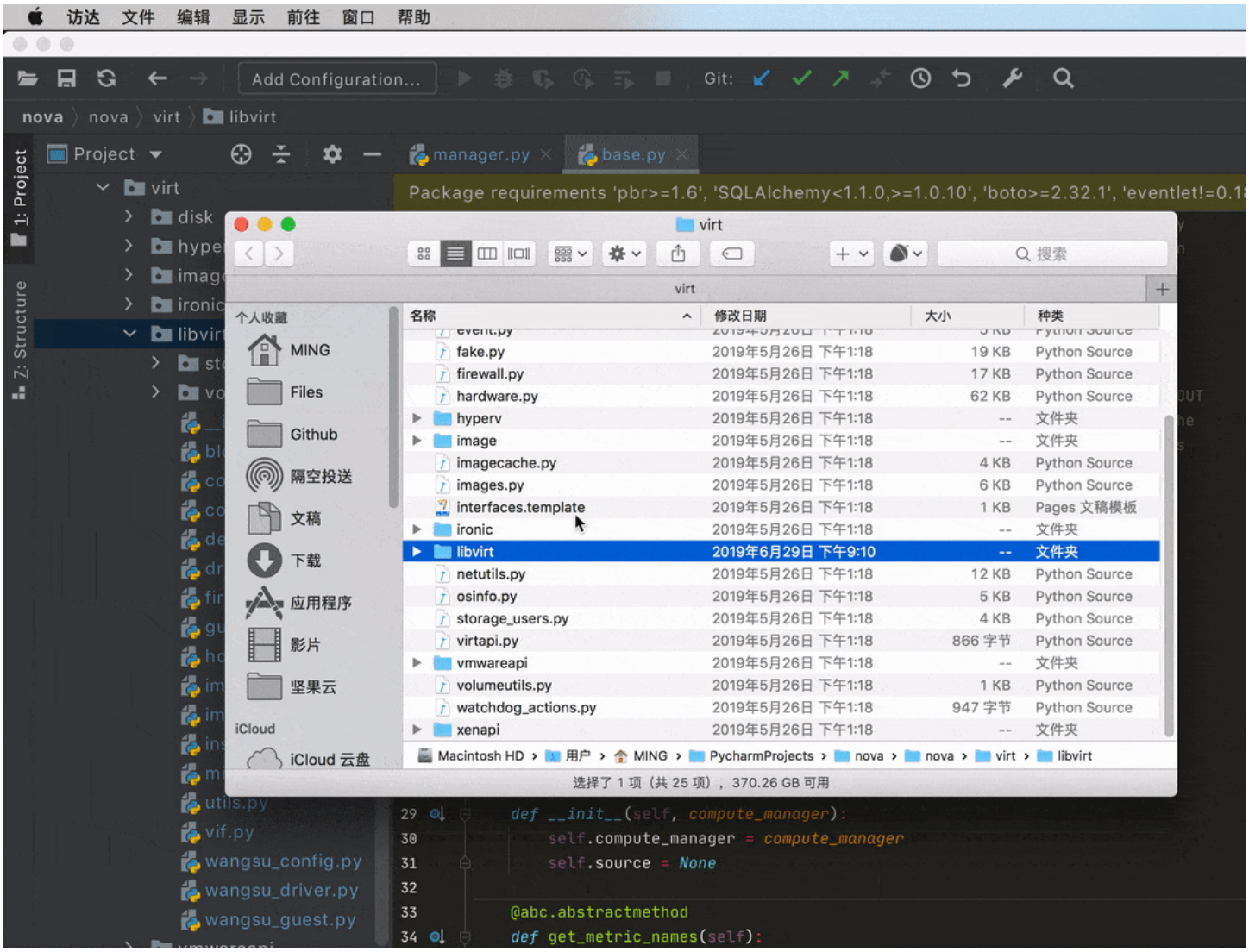


> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

9.15 【必学技巧 15】在文件管理器/Finder 中打开文件夹的三种方法

第一种方法

点击右键，选择 `Reveal in Finder` （Mac 下）或者 `Show in Explorer`



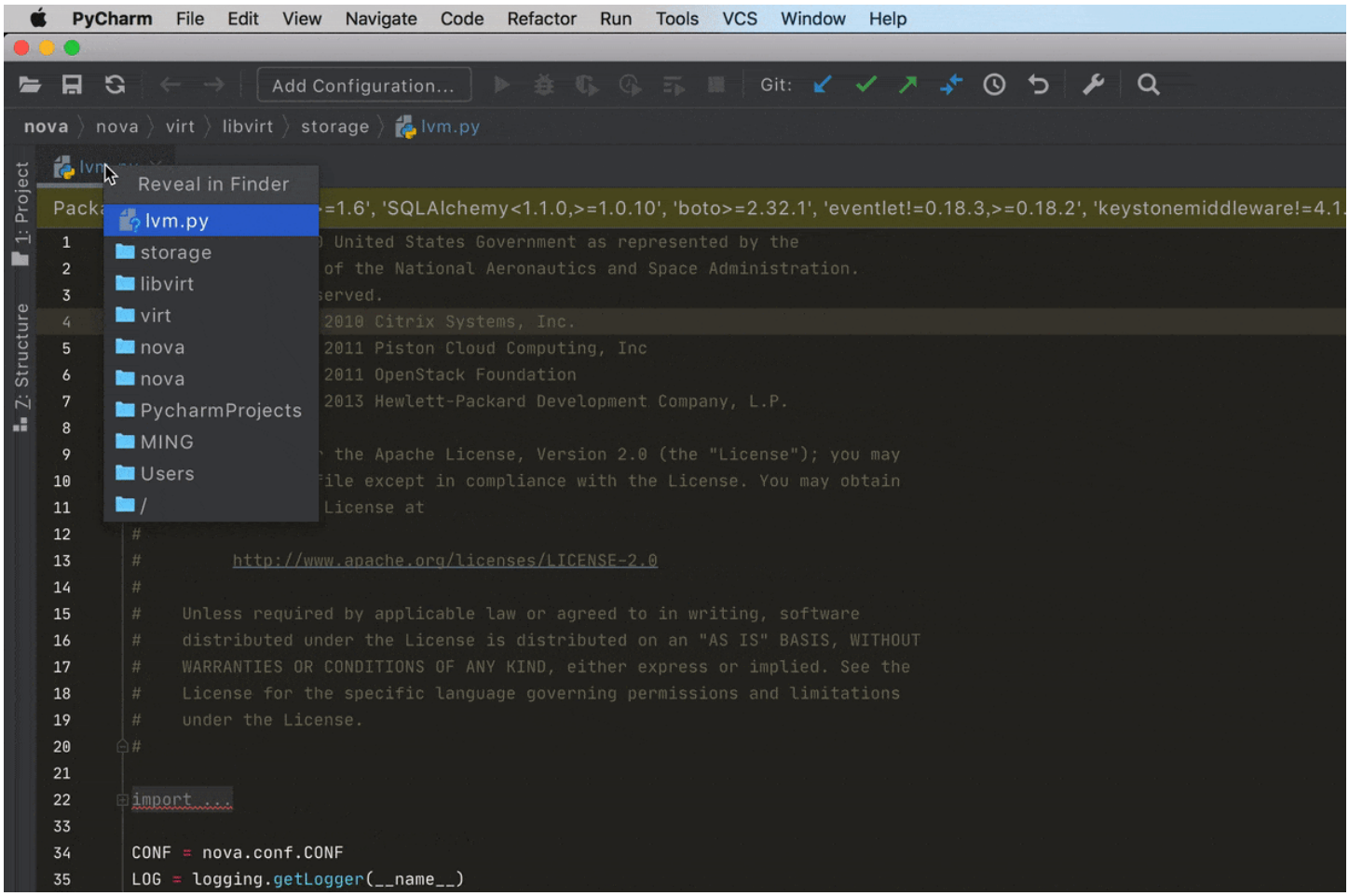
> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

第二种方法

当你使用的是 ⌘ + ⇧ + N 打开的文件时，你会发现左边的项目树中，该文件的目录并没有被展开，这时候你想使用 `第一种方法` 去在资源管理器或者 Finder 中打开，就必须先打开项目树侧边栏，一层一层的点开该文件的目录，然后再 `Reveal in Finder`。

实际上，面对这种尴尬的场景，有更好的解决方法。

请看下图：在左侧边栏未打开情况下 先 ⌘ + ⇧ + N 查找文件并打开，然后按住 ⌘ 再用鼠标单击，就会出现该文件的所有父级目录，使用 ↑ 或 ↓ 进行选择，最后调击回车就能在 Finder 或 资源管理器中打开。

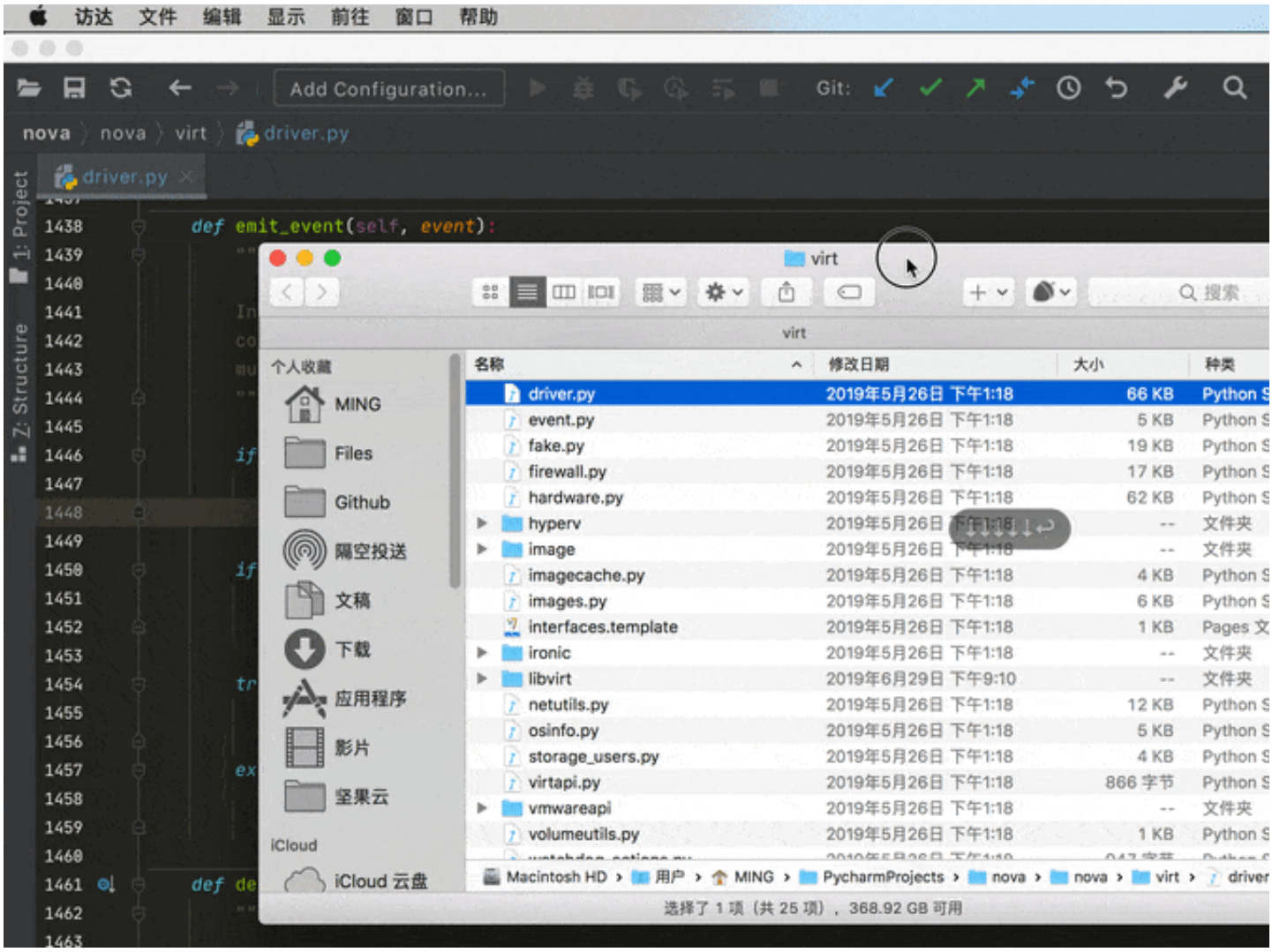


> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

第三种方法

是最快速，最简单，而且还是唯一一种不用鼠标参与的方法。

使用快捷键：⌥ + F1



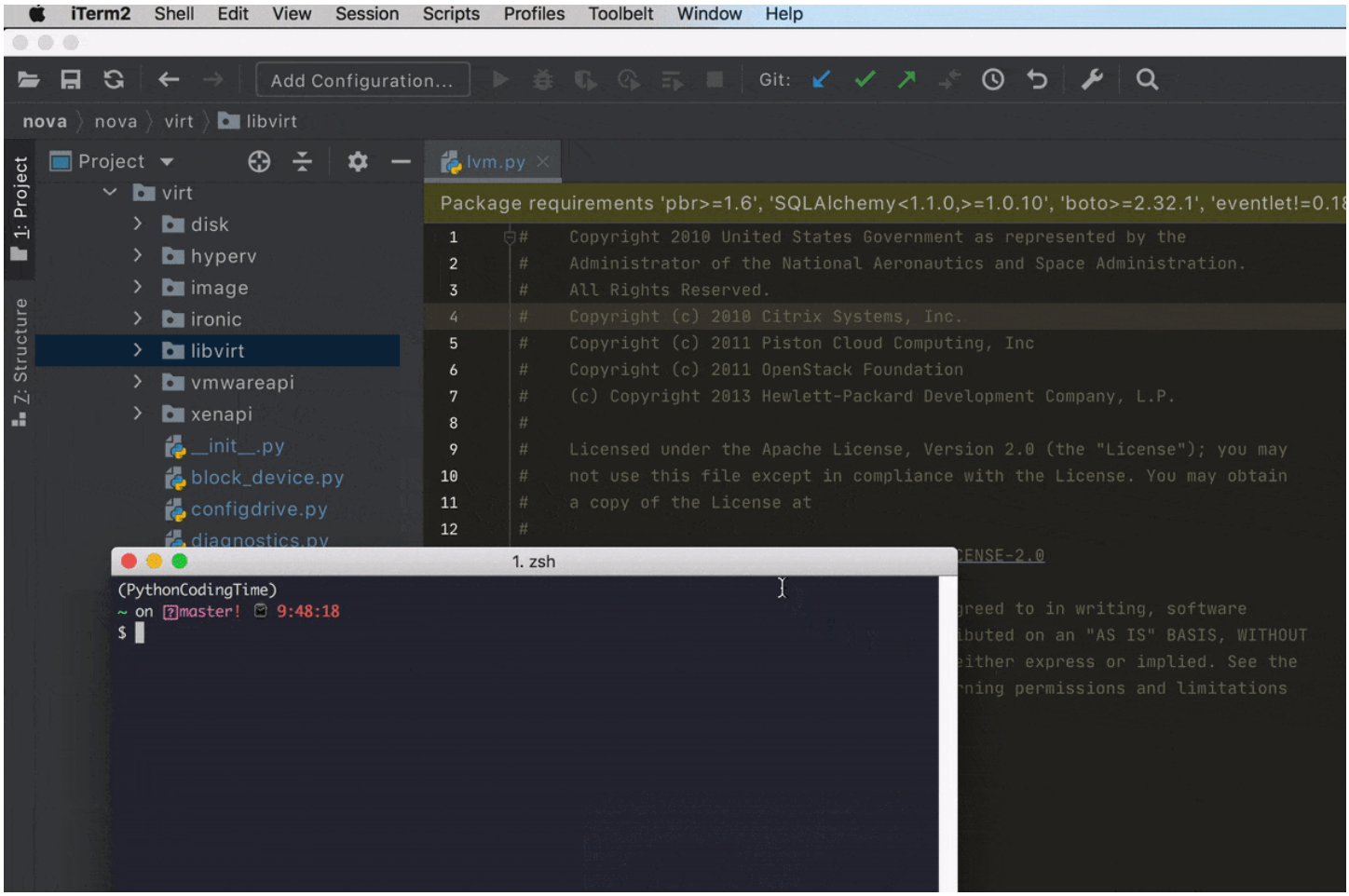
> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

9.16 【必学技巧 15】在Terminal 中打开文件夹

当你想在 Terminal 中打开项目的文件夹时，也许你会这样操作

第一种方法

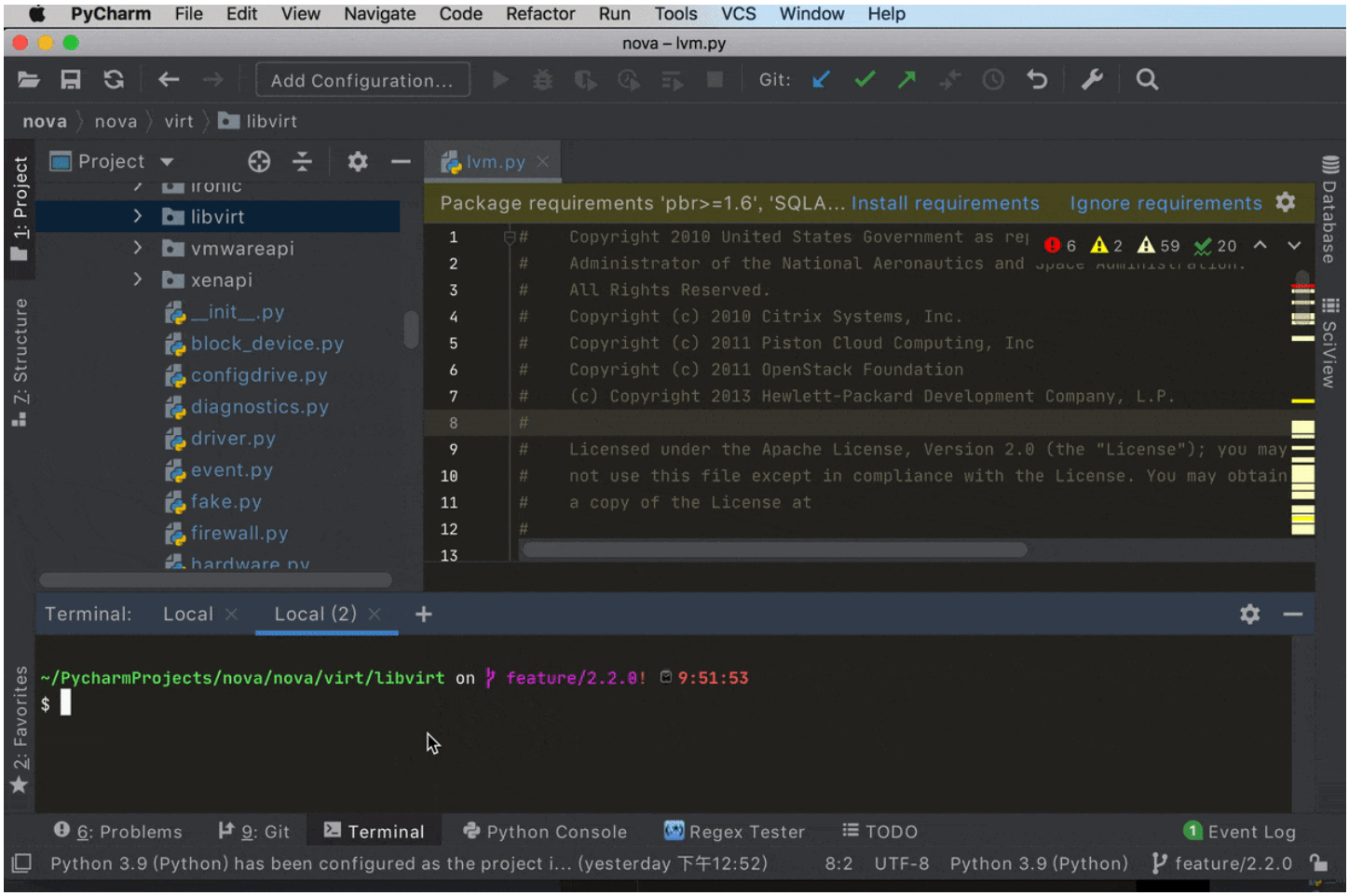
先拷贝绝对路径，再打开 terminal 粘贴 路径进入



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

第二种方法

PyCharm 直接为我们集成了 Terminal，我们可以直接点击进入



> PDF 无法查看 GIF 动图，请前往 http//:pycharm.iswbm.com 查看效果图

作者：王炳明
版本：v1.0
发布时间：2020年08月30日
微信公众号：Python编程时光
联系邮箱：wongbingming@163.com
项目主页：http://pycharm.iswbm.com
Github：https://github.com/iswbm/pycharm-guide

👆👆👆

回复 "**pycharm**"，获取最新版 PDF