

OO工具链介绍

author: 张少昂 牛雅哲 葛毅飞

OO工具链介绍

- 版本控制工具Git

 - Git安装

 - 在Windows上安装Git

 - 在Linux上安装Git

 - 配置Git

 - Git使用

- JDK安装配置指南

 - 下载JDK

 - 下载版本的选择

 - 对于Windows用户

 - 对于Linux或MacOS用户

 - 安装与配置

 - 安装JDK

 - 配置环境变量

 - JAVA_HOME

 - CLASSPATH

 - PATH

 - 其他

 - 常见疑问的解答

 - 如何知道我操作系统（Windows）的字长？

 - 如何知道是否已经完成了安装呢？

 - 系统评测机使用的Java版本是什么呢？

 - 是否必须使用官方版本的JDK呢？OpenJDK是否可以呢？

- 关于IDEA的那些事

 - Jetbrains的那些事

 - Jetbrains

 - Idea

 - Idea的那些事

 - 下载安装Idea

 - 开始使用

 - 代码风格配置检查

 - IDEA的特性

 - 代码风格

 - 高度智能的联想

 - import自动添加

 - 批量修改

 - 代码快速查看

 - 快速寻找我想要的功能

 - javadoc

 - Git

 - 插件

 - MetricsReloaded

 - Statistic

 - .ignore

 - Markdown support

 - Background Image Plus

- Markdown

 - Typora安装

 - Windows安装

在我们的oo课程中，需要涉及到一系列的工具链，下面我们对需要使用到的工具进行一一介绍。

版本控制工具Git

Git安装

Git是一个开源的分布式版本控制系统，可以高效处理人和或大或小的项目。

何为版本控制？想必大家都遇到过一个文件需要反复修改完善的场景，大量的文件副本不仅浪费大量空间，也会对我们造成一定的困扰。哪一个版本是最终版本？我们这一版相对上一版作了什么修改了？我们要消除修改，又应该回退到哪一个文件？文档如此，代码也如此，因此我们需要借助版本控制工具对文件进行控制。在的OO课程中，我们采用Git作为版本控制工具提交和管理项目。言归正传，下面开始Git的安装教程！

在Windows上安装Git

首先，我们可以到Git官网（<http://git-scm.com/downloads>）上直接下载安装程序，然后按照默认选项安装即可。

安装完成后，在开始菜单中找到“Git”->“Git Bash”，蹦出一个类似命令行窗口的东西，就说明Git安装成功！

在Linux上安装Git

如果使用 Debian 或 Ubuntu，在终端敲入命令 `sudo apt-get install git` 即可完成git安装。

如果使用 Centos 或 RedHat，安装命令为 `yum -y install git`

配置Git

安装完成后，我们需要配置用户名密码，命令行窗口中输入以下内容配置git

```
git config --global user.name "Your Name"
git config--global user.email "email@example.com"
```

Git使用

在此，我们推荐大家使用廖雪峰的官方网站学习Git

网站链接：<https://www.liaoxuefeng.com/wiki/896043488029600>

JDK安装配置指南

JDK是Java Development Kit 的缩写，中文称为Java开发工具包，由SUN公司提供。它为Java程序开发提供了编译和运行环境，所有的Java程序的编写都依赖于它。

在我们开始使用Java编写、运行程序之前，我们首先需要手动配置JDK。下面是JDK的一般配置过程。

下载JDK

首先，我们需要在官方网站上下载DK安装包。

网站链接: <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>。或者百度搜索JDK,也可以找到官网链接。(注意:推荐在官方网站上下载。)

下载版本的选择

对于Windows用户

- 如果你的操作系统字长为64位,则选择Windows x64版本。
- 如果你的操作系统字长为32位,则选择Windows x86版本。

对于Linux或MacOS用户

对于非Windows的用户,则需要按照自己操作系统的类型自行进行选择。

安装与配置

安装JDK

直接运行安装下载下来的安装包文件即可。(Linux和MacOS操作类似)

配置环境变量

安装完毕后,我的电脑右键属性,点击左边的 高级系统设置, 点击 高级 --> 环境变量。

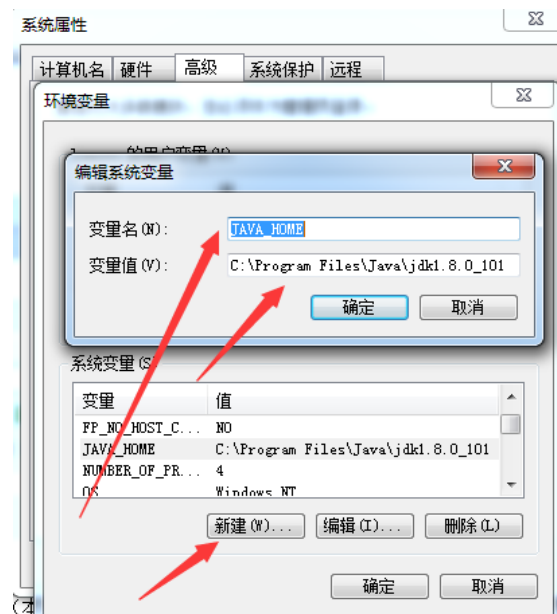
(本部分将围绕windows进行讲解, Linux和MacOS的配置方式可以自行了解)

JAVA_HOME

在下方系统变量栏中,新建环境变量 JAVA_HOME。

变量值为 C:\Program Files\Java\jdk1.8.0_101 (具体路径因安装路径而异,不要包含 bin)。

如下图所示



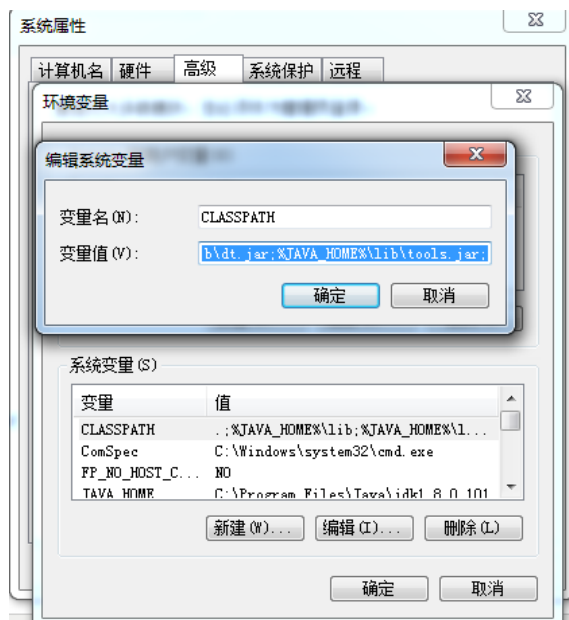
CLASSPATH

在系统变量栏中,新建变量 CLASSPATH, 变量值

为 .;%JAVA_HOME%\lib;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;。

(注意,一定要留有前面的 . 符号)

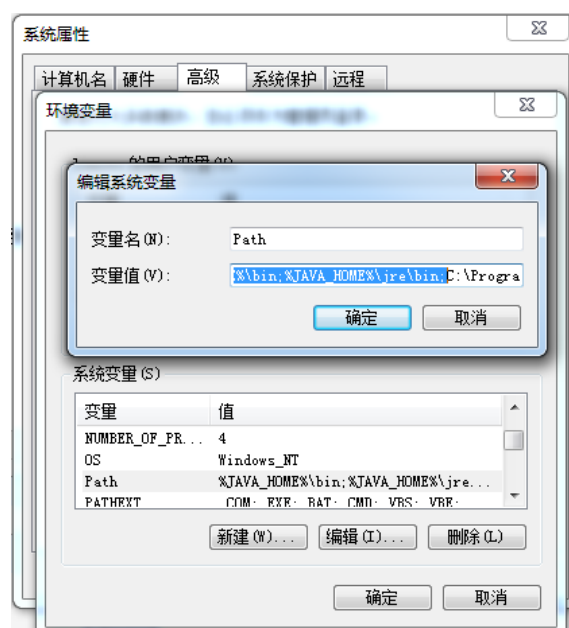
如下所示:



PATH

在系统环境变量中，设置 `PATH`，变量值 `%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin;`（不要覆盖掉原本的内容，将这个值加入到 `PATH` 的最前面）。

如下图所示：



其他

常见疑问的解答

如何知道我操作系统（Windows）的字长？

对于Windows10系统，可以右键我的电脑，进入属性，查看即可查看操作系统类型。（具体细节可能略有差异，更多细节可以百度）

对于Linux和MacOS系统，一般也有类似的通过GUI或者命令行的方式查看系统信息，具体可以自行百度。

如何知道是否已经完成了安装呢？

打开 `cmd`（Linux或MacOS下的终端），输入 `java`，应该出现类似这样的内容。

```
命令提示符
E:\tongpao-dev>java
用法: java [-options] class [args...]
           (执行类)
或 java [-options] -jar jarfile [args...]
           (执行 jar 文件)
其中选项包括:
  -d32          使用 32 位数据模型 (如果可用)
  -d64          使用 64 位数据模型 (如果可用)
  -server       选择 "server" VM
                 默认 VM 是 server.

  -cp <目录和 zip/jar 文件的类搜索路径>
  -classpath <目录和 zip/jar 文件的类搜索路径>
               用 : 分隔的目录, JAR 档案
               和 ZIP 档案列表, 用于搜索类文件。

  -D<名称>=<值> 设置系统属性
  -verbose:[class|gc|jni]
                 启用详细输出
  -version      输出产品版本并退出
  -version:<值> 警告: 此功能已过时, 将在
                 未来发行版中删除。
                 需要指定的版本才能运行
  -showversion  输出产品版本并继续
  -jre-restrict-search | -no-jre-restrict-search
                 警告: 此功能已过时, 将在
                 未来发行版中删除。
                 在版本搜索中包括/排除用户专用 JRE
```

系统评测机使用的Java版本是什么呢？

Java版本: `java -version`

```
java version "1.8.0_101"
Java(TM) SE Runtime Environment (build 1.8.0_101-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)
```

Javac版本: `javac -version`

```
javac 1.8.0_101
```

是否必须使用官方版本的JDK呢？OpenJDK是否可以呢？

推荐使用官方版本的JDK（不过具体的版本号不需要严格一致，保证是 `java 1.8.0` 即可）

OpenJDK的话，在绝大部分场合下问题不大，但是课程组没有就所有的细节做过相关的具体测试，故不保证不会出现因为OpenJDK版本问题导致的评测异常，还请大家自行选择判断。（另：如果你是Mac或者Linux用户的话，OpenJDK安装会远比Oracle JDK方便）

关于IDEA的那些事

配置好了Java的编译和运行环境之后，我们还需要安装Java的集成开发环境（IDE），说到Java的IDE，似乎eclipse和Idea是目前的主流。然而，OO的课程组却一直在推荐使用eclipse，于是很多人就这样错过了Idea这样强大的IDE工具。本文将会对于Idea和Idea的一些常见（实际上，很多是Jetbrains系列IDE的代表性操作）操作进行一些介绍。

Jetbrains的那些事

Jetbrains

Jetbrains是捷克的一家企业（[Jetbrains官网](#)），目前其主打产品是各个现代主流语言的IDE，包含 `Python`、`Ruby`、`PHP`、`SQL` 等语言（对于企业用户还提供一些teamwork管理工具）。其IDE用过的人都知道，颇具现代感，很多功能解决了令不少程序猿们头疼多年的难题（后面将会详细讲到）。

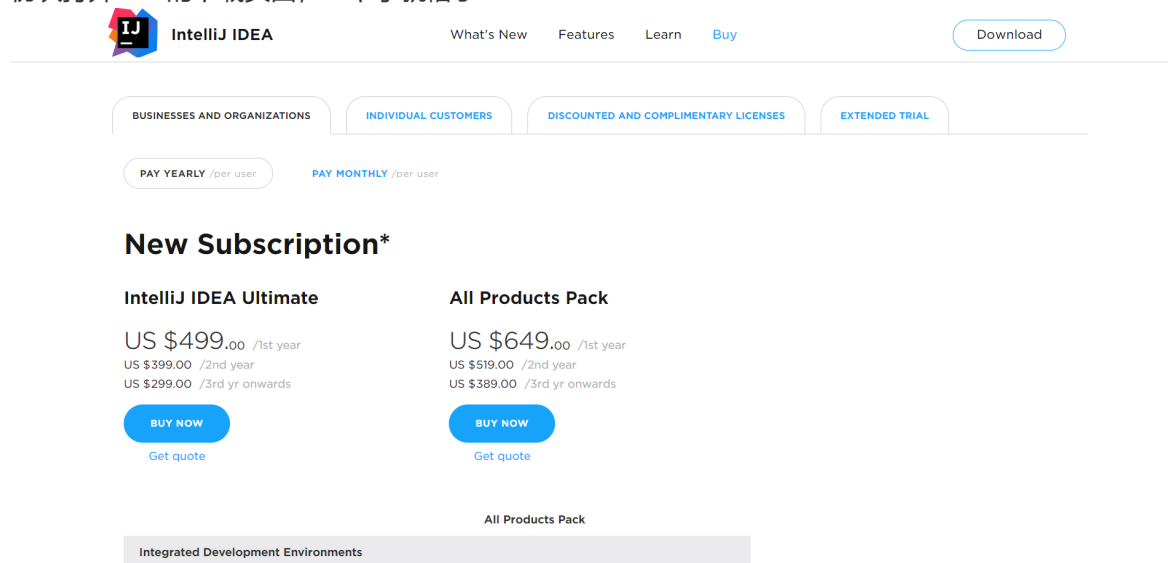
Idea

Idea则是Jetbrains全家桶的一员（[Idea官网](#)），其除了Jetbrains一些共性的王牌功能之外，还针对java这门语言的一些特性进行了进一步的用户体验优化。（后文也将详细阐述）

Idea的那些事

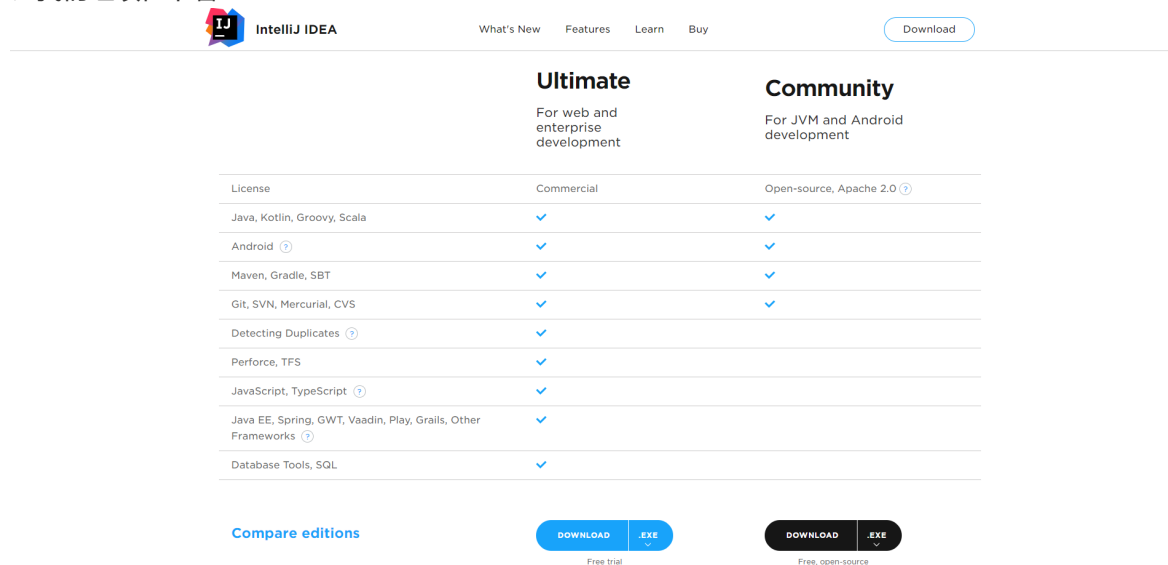
下载安装Idea

初次打开Idea的下载页面，一下子就懵了：



499刀一年。。。看的有些肾疼。那是不是我们Idea之旅就要就此止步了呢？Of course, **NO!**

让我们继续往下看：



果然，IntelliJ Idea 和 Pycharm 一样，都提供了**完全免费的社区版**，可以直接下载使用。

然而，对于本科生，我们依然**可以通过注册学生账号的方式来免费使用Ultimate版**（准确的说，Jetbrains大礼包里面除了完全面向企业的团队工具之外，所有的专业版工具都可以凭学生优惠免费下载使用）

大家可以自己去[按照官网的引导](#)或者网上的教程等进行认证操作和许可证的免费申请，本文中不再赘述。

安装的过程也并不复杂，也没有什么坑点，大家可以自行完成。

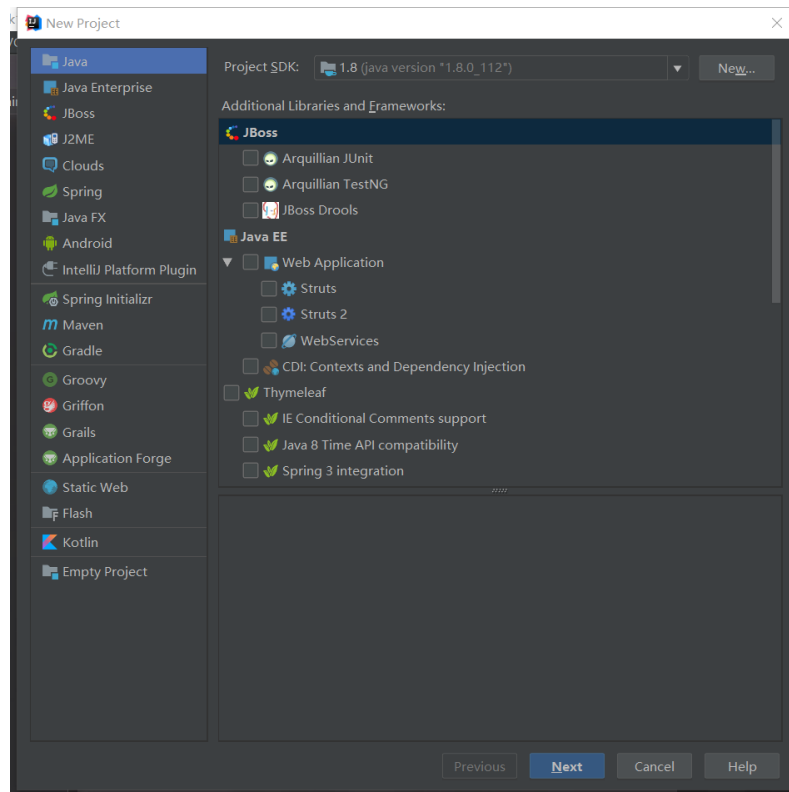
（注：本文中接下来的图片示范均以Ultimate版为例）

开始使用

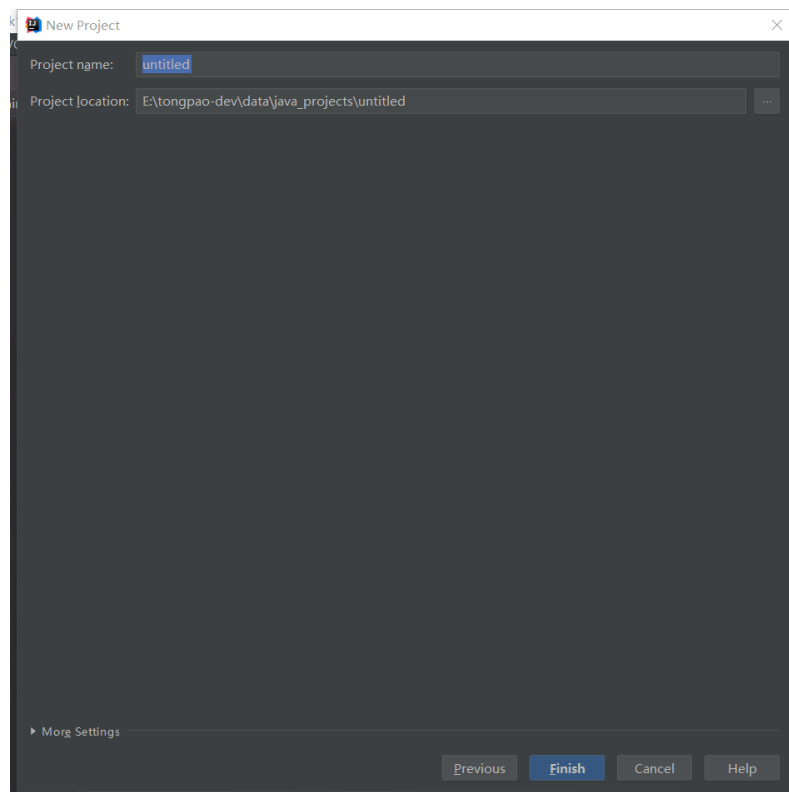
(在开始本节之前, 请一定保证你已经妥善完成了JDK的安装与环境配置)

初次打开软件时, 可能会需要你设置一些风格主题一类的东西, 这个按照自己的喜好设置就好 (笔者本人力荐 Darcula 黑色风格, 通宵爆肝护眼必备), 而且在进入软件之后, 也可以在Setting中继续进行设置。

创建新工程时, 我们只需要按照一般的套路来: `File-->New-->Project...`

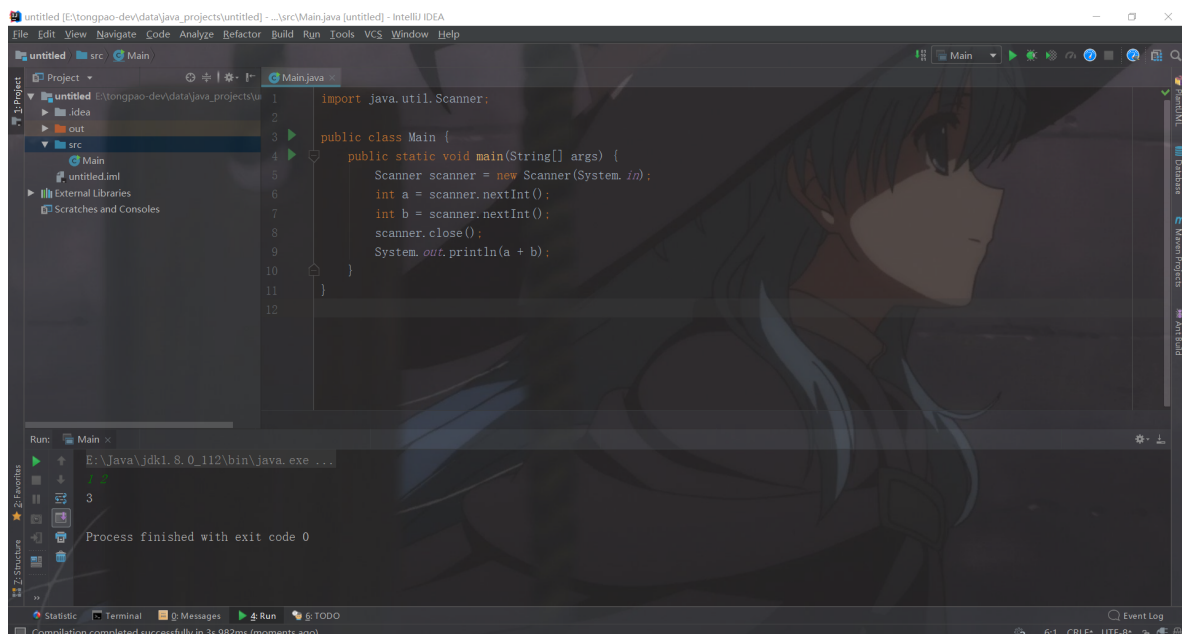


在我们的作业项目中, 我们不使用框架, 使用原生Java。所以一路Next到最后一步。



在这里, 我们需要设置一下项目名, 然后点击 `Finish`, 即可完成项目创建。

接下来, 只需要在 `src` 路径下创建程序文件, 并运行即可, 如下图所示。



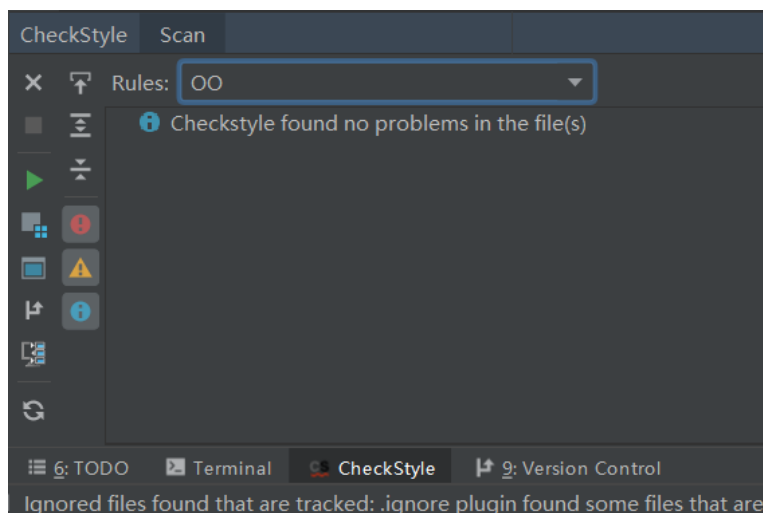
代码风格配置检查

研读过[阿里巴巴java开发代码规范手册](#)的同学们应该知道，在真正的工程代码中，处于**代码可维护性**和**提高团队合作效率**的考量，会有很多代码规范性的要求。因此在每一次oo作业中，我们都会使用一个叫做 `checkstyle` 的java代码风格检查工具进行代码风格的检查。其中配置文件使用本仓库内提供的 `config.xml` 文件，作为代码风格检查的依据。

如何在本地配置代码风格检查插件，检查java代码是否符合规范？我们通过下列操作来进行。

在IDEA中，我们依次点击 `File->Settings->Plugins` 下载安装 `Checkstyle-IDEA` 插件，对ide内的代码进行实时风格检测。安装好插件后通过 `File->Settings->Other Settings->CheckStyle` 将我们提供的 `config.xml` 载入即可。

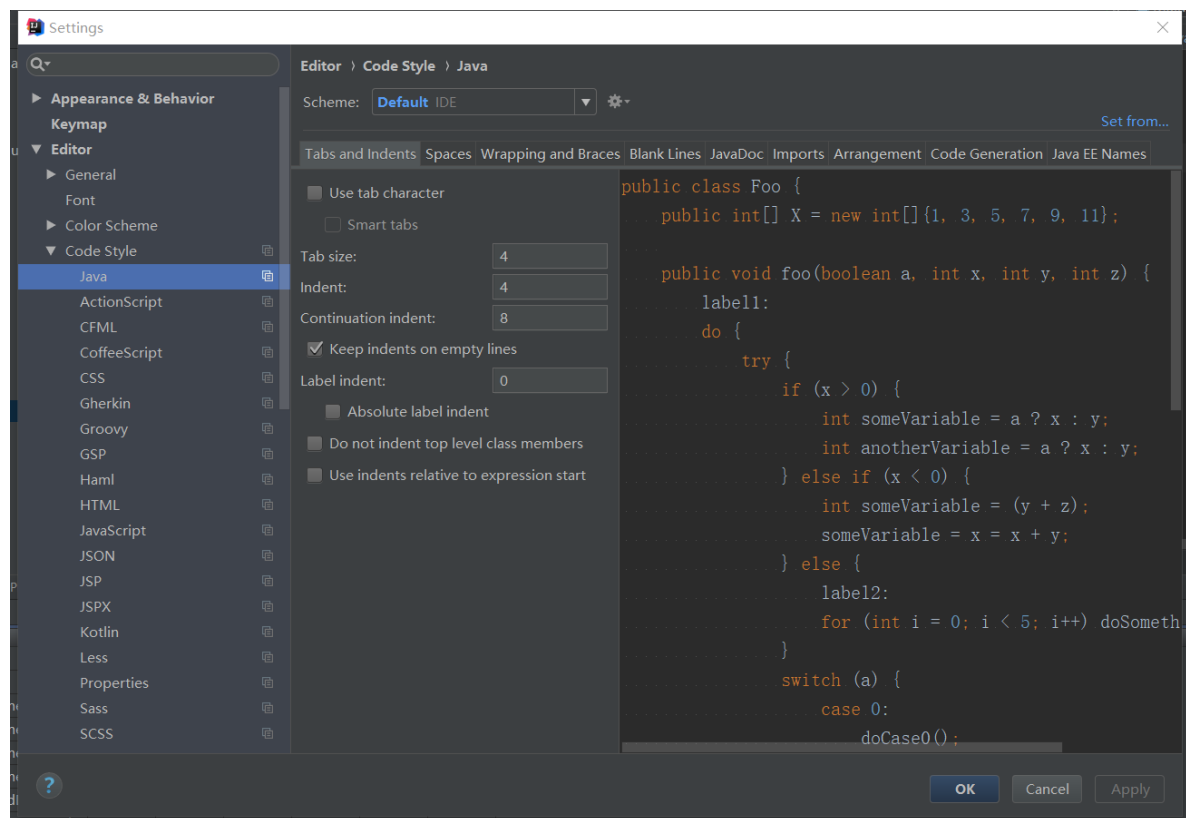
在窗口左下角点入CheckStyle选取相应的规则（即我们导入的配置文件）进行代码风格检查



IDEA的特性

代码风格

可能不少同学已经写了规模不小的代码，而且从未参照过代码规范。不必担心，jetbrains给我们提供了很方便的代码风格工具：



可以看到，使用 tab 还是空格缩进，以及缩进几格都是可以自由调整的（实际上，**一般企业的代码工程规范是使用4个空格作为缩进**）。此外，在别的标签页下，还有很多可以调整的代码风格相关的东西（包括你们圣战了无数年的大括号换行不换行问题）。

而这样的代码习惯调整，只需要**Menu -> File -> Settings -> Editor -> Code Style -> Java**即可找到并调整（可以看到，除了java还有非常多种的语言。没错，一般的jetbrains IDE都支持多种语言的编辑，如果你有同时使用多种语言的需求的话，可以在其他语言对应的区域进行编辑。）

在我们调整好了之后，我们在代码位置按下 **Ctrl+Alt+L**（Pycharm中是 **Alt+F8**）即可**完成代码规范化**（或者**Menu -> Code -> Reformat Code**），效果如下：

```
public static void test(int n) {
    for (int i=1;i<=n;i++)
    {
        int y = i * i + 2; System.out.println(y * y);
    }
    for (int i=1;i<=n;i++) {
        int x=i*i;
        System.out.println(x + x);
    }
}
```

只需要按下 **Ctrl+Alt+L**，代码立刻就变成了这样：

```
public static void test(int n) {
    for (int i = 1; i <= n; i++) {
        int y = i * i + 2;
        System.out.println(y * y);
    }
    for (int i = 1; i <= n; i++) {
        int x = i * i;
        System.out.println(x + x);
    }
}
```

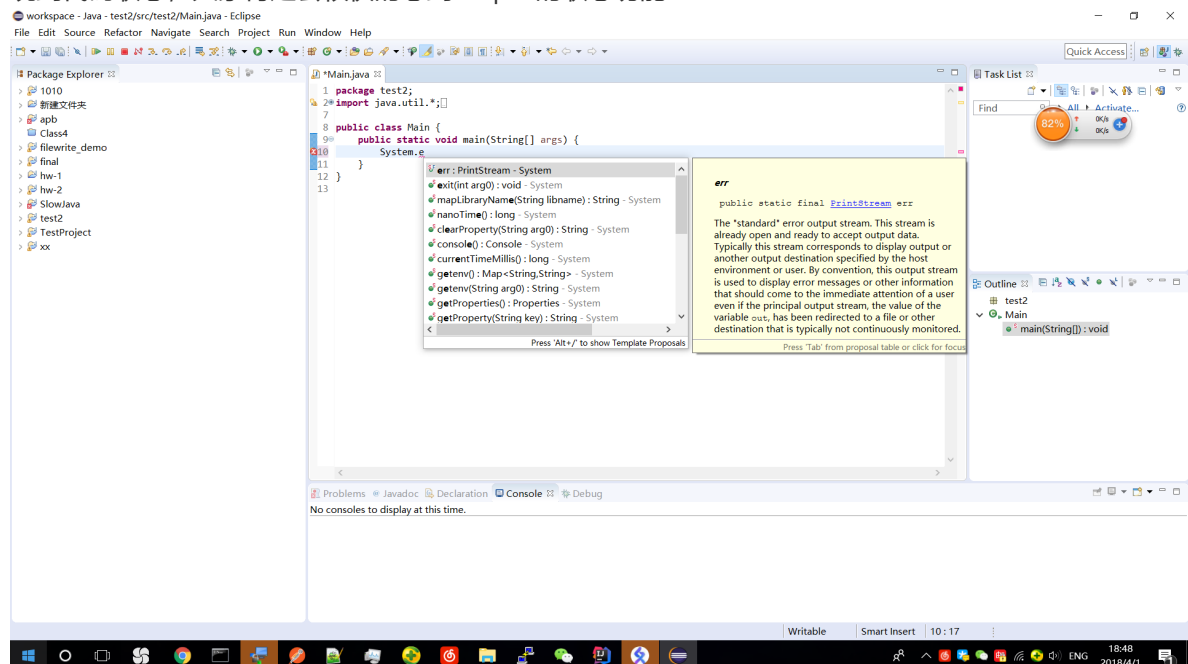
代码瞬间变得干净整洁，清清爽爽。

高度智能的联想

说到代码联想，大家可能对这一概念并不陌生。事实上很多的IDE也都已经支持这一功能了。

但是，等你一用idea的代码联想功能，你就会再也放不下来了。

说到代码联想，大家肯定会很快的想到eclipse的联想功能：

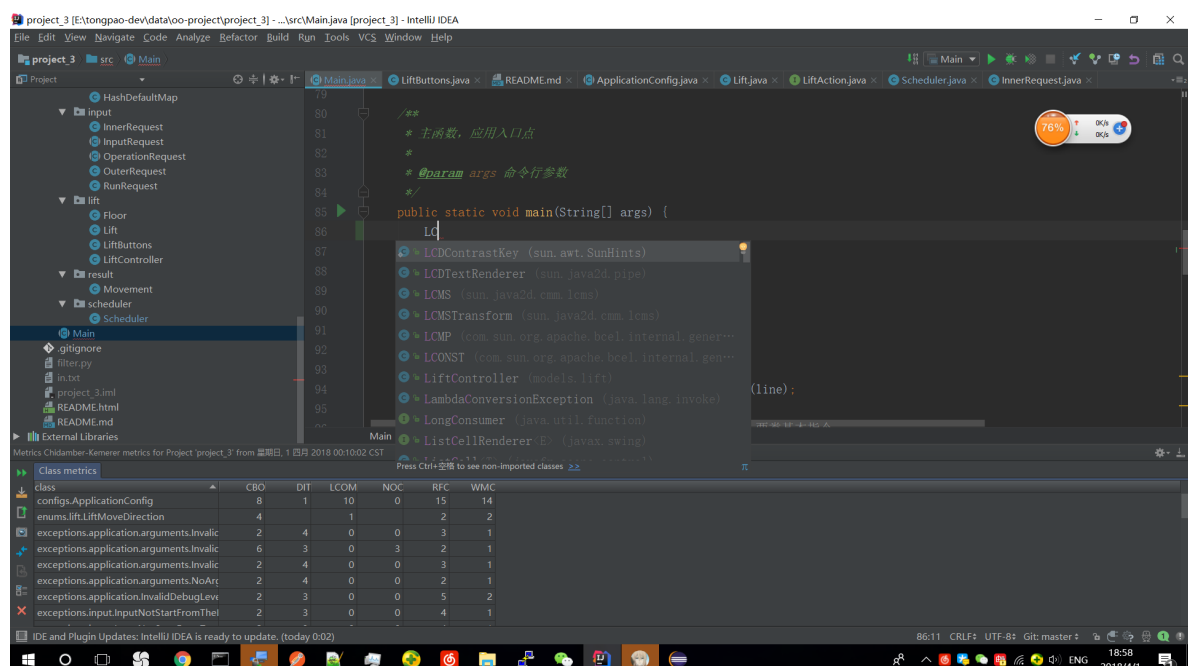


然而，eclipse的代码联想实际上存在一些局限性（以及其他很多的IDE也是这样）：

- **写类名的时候没有联想** 例如，开始写System这个类时，整个过程不会出现任何的联想
- **联想出来的方法快捷键操作不便** 例如，当System按下.之后输入e，联想到了exit，但总还是需要一些比较不优雅的操作（比如鼠标操作，比如并不符合人类直觉的一些其他操作）来快速输入

这意味着什么呢？这意味着，**当你对一门语言或者某些类不够熟悉（甚至根本不知道它们的存在）时，你连自我尝试和探索的可能性都没有**，只能去翻阅冗长且并不友好的java文档，这显然不符合程序猿的探索精神。以及，如此不优雅的快速输入，多年的码农表示怎么用怎么觉得别扭。

然而在idea中，这些问题都得到了极大地改善：



- **从输入类名的第一个字起，就可以进行智能的联想** 仔细观察上图的话还可以发现一件有趣的事情，输入 `LC` 后，连我们的 `LiftController` 类都联想到了。是的，idea的代码联想完全支持英文音序联想。
- **根据用户近期使用的情况来智能调节联想顺序** 这是idea代码联想另一个很神奇很贴心的feature，如果你近期频繁使用 `LC` 来输入 `LiftController` 类的话，你会发现 `LiftController` 类会在列表中越来越靠前，最多两三次过后就跑到了顶部。
- **可以直接按上下键和回车来进行快速键入** 这一点相比eclipse等其他ide有了非常大的改善，整个过程非常符合一般人的操作直觉，且全过程**不依赖任何键盘以外的操作**。

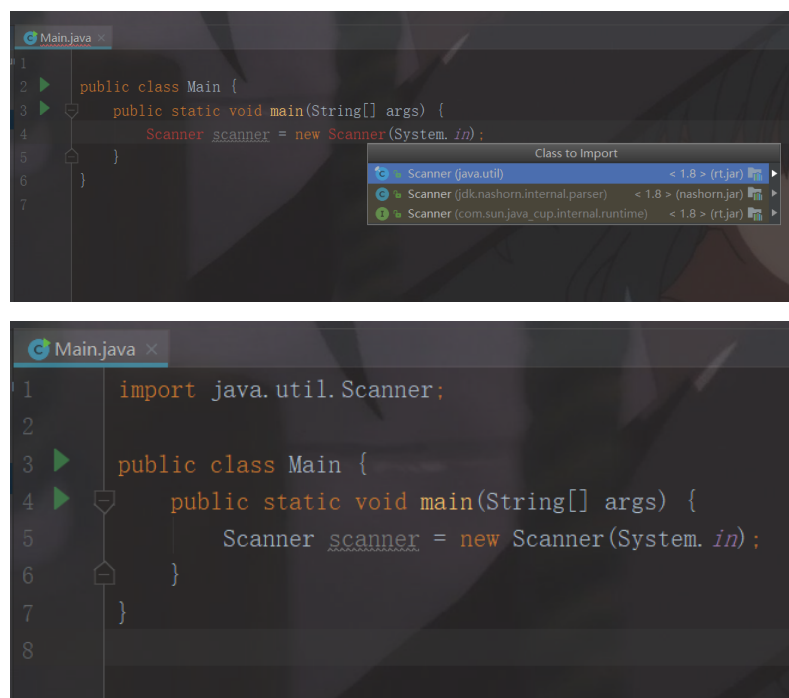
有了idea强大的代码联想功能（准确的说，jetbrains全家桶IDE都有这些特性），我们的代码产出速度可以大幅度提升。不仅如此，我们还可以**对于并不熟悉或者未知的一些代码或库进行大胆的尝试**，不得不说，这一功能还很适合进行自我探索。

import自动添加

在Idea中，我们实际上不需要一条一条在顶上打 `import` 语句。

我们只需要想到了某个类或者某个方法，直接在程序中写上去，然后ide就会提示该类未导入，就像这样

我们只需要将鼠标或者光标移至该类名上，并使用 `Alt + Enter` 快捷键，从提供的类中选择需要的，完成自动添加。（有时可能只有一个类可供选择，此时将不会出现选择直接完成添加）



批量修改

不知道大家有没有遇到过这样的尴尬状况：

```

public class Schedue1r {
    // something inside
}

public abstract class Main {
    public static void main(String[] args) {
        schedue1r s = new Schedue1r(); // execution of the constructor method
        Schedue1r.someStaticMethod(); // execution of the static method
    }
}

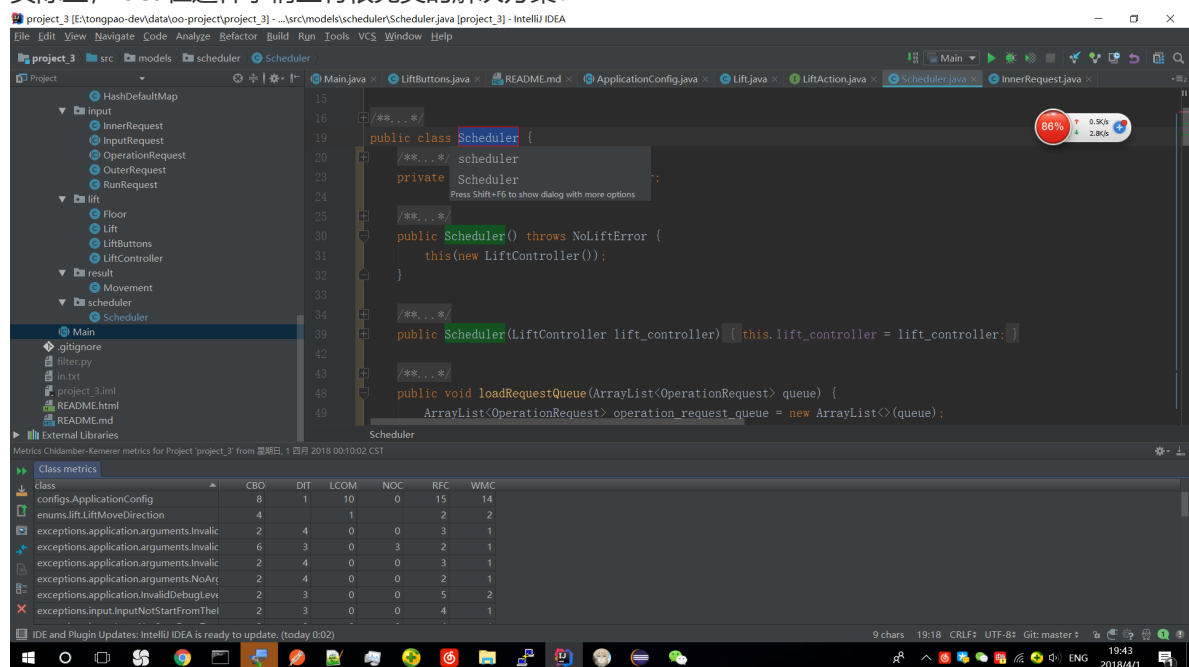
```

没错，细心的你应该已经发现了问题所在——`Schedue1r` 类名的拼写是错误的，应该是 `Scheduler`。

按照一般的代码规范，这样的拼写错误绝对是不可以容忍的（就算可以容忍，这样的东西也会导致笔者强迫症大犯 -_-||）。

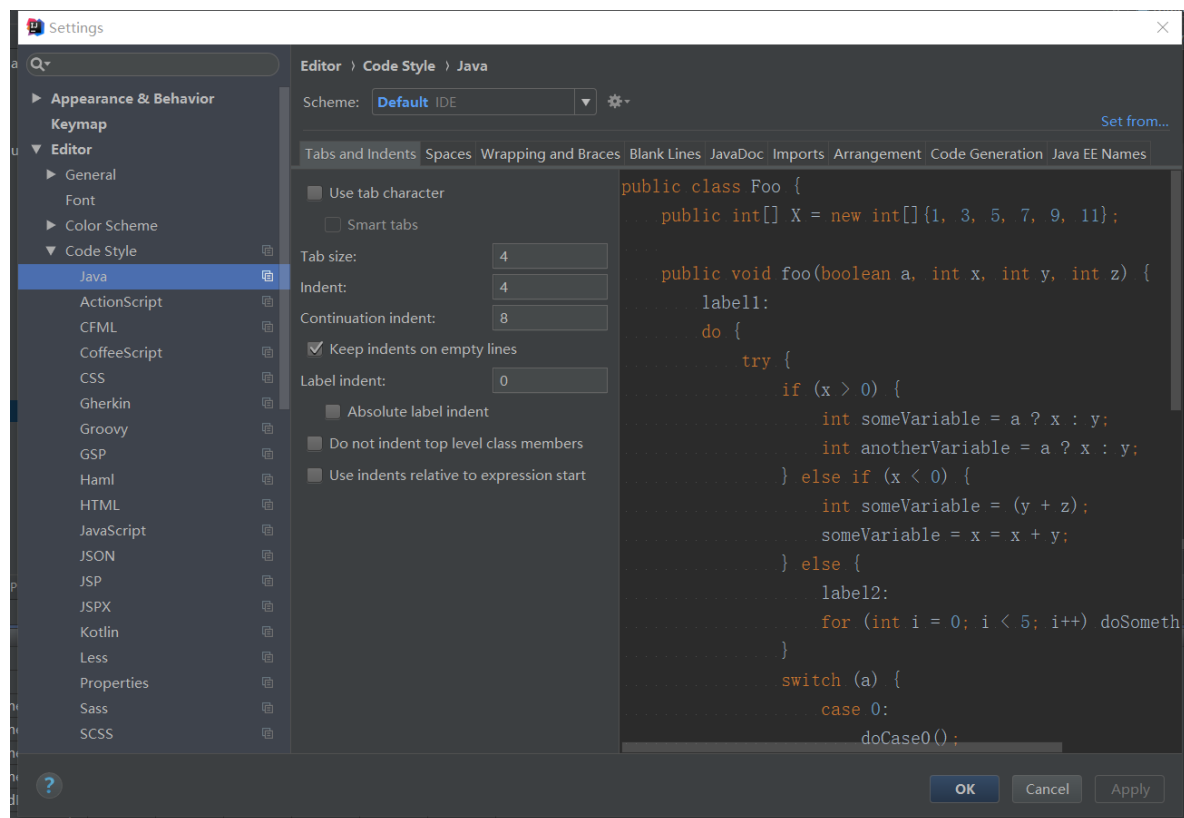
然而，再一看，可能已经有无数的地方已经在用着这个拼写错误的类名调用。想改？烦得很，而且还很容易错改和漏改。不改？强迫症使我面目全非o(╥╰╣╣)o。于是，相信很多人最终的选择还是一——不改，宁可被自己代码恶心一遍遍也不能有bug。

实际上，idea在这件事情上有很完美的解决方案：



只需要在类名（实际上方法名、变量名、甚至文件名等也都可以这么做）上右键->Refactor->Rename，或者直接 `Shift + F6`，即可直接修改名字，而且整个工程中相关的地方也都会一起随之改动。

更有趣的是，笔者做了一个实验：



在这样的一个函数中，将第一个for循环内的 `x` 值进行rename操作，效果如下：

```
public static void test() {  
    for (int i = 1; i <= 10; i++) {  
        int y = i * i + 2;  
        System.out.println(y * y);  
    }  
    for (int i = 1; i <= 10; i++) {  
        int x = i * i;  
        System.out.println(x + x);  
    }  
}
```

可以看出，idea的rename功能完全**不会误伤到不同作用域类的同名实体**，可以说是做到了精确打击。

此外，Refactor中还提供了Safe delete等人性化的功能，等待大家去尝试（Safe delete是在删除类、方法、变量时，检测是否依然在别的地方对该实体存在依赖，以达到安全删除的目的）。

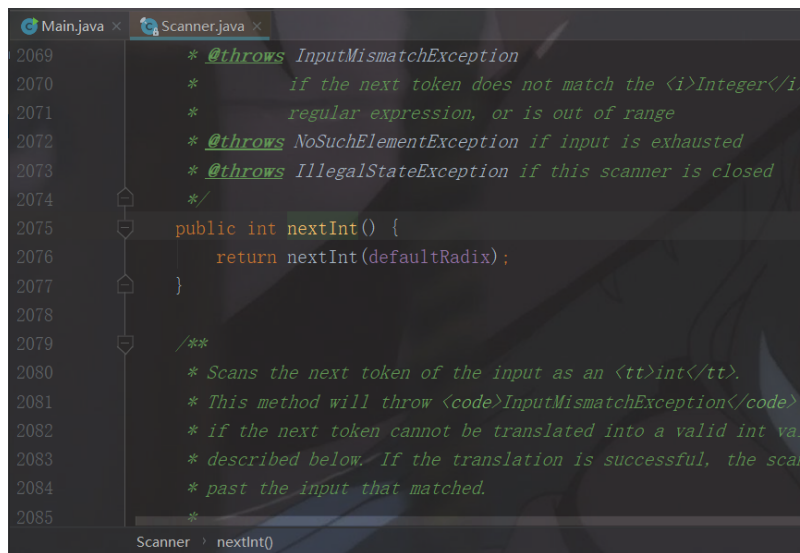
此外，**IDEA中还提供自动的单词拼写检查**，如果出现错误的单词拼写，则会像word一样第一时间显示出来并提示编程者尽快修复，或者将该单词（因为的确可能有存在专有名词等非常规词汇的情况）添加进工程字典内。

代码快速查看

有的时候，代码一旦变得复杂，我们就会需要经常在方法和类之间来回跳跃查找代码。

然而在IDEA中，这一麻烦不复存在，我们可以直接在类名、方法名、变量名上进行 **Ctrl + 鼠标左键**，一键跳跃到想看的类或方法的代码实现位置上。（甚至可以跳到java的底层源代码上，源码党们的福音）

就像在上面的程序的 `nextInt` 方法上点击 **Ctrl + 鼠标左键**，结果如下：

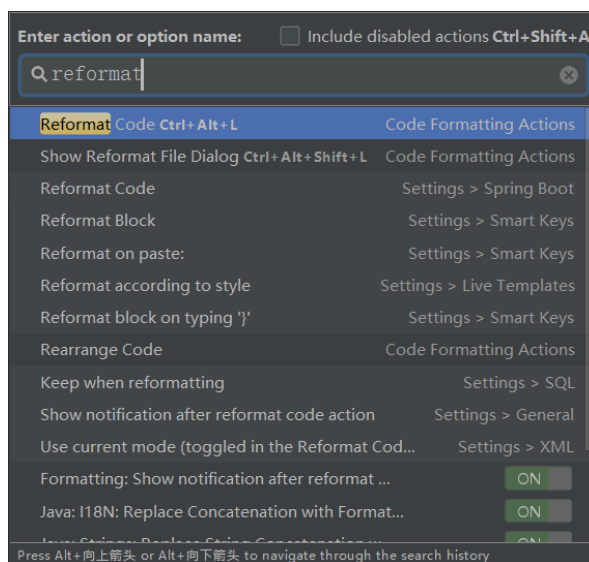


我们成功跳到了Java底层对 `Scanner` 类 `nextInt` 方法的源码实现上。

快速寻找我想要的功能

IDEA作为一款真真正正的现代化ide，和其他一些古代编辑器相比，另一大亮点就是可以快速寻找想要的功能。

在菜单栏，点击 `Help`--> `Find Action`（快捷键为 `Ctrl+Shift+A`）



可以在这里直接搜索想要找的功能或者设置。（实际上Jetbrains的Find Action功能提供非常精细的查找，不仅仅支持菜单栏的查找，连内部设置的细节甚至都可以找得到）

javadoc

在正规的工程代码规范中，还有一项很重要的要求——写文档。

然而，这个文档也是有很严格的规范的，不是很多人认为的那样，随便注释一点就可以当做文档。而这种符合java工程规范的文档形式就称之为javadoc（类似的代码注释规范还有phpdoc等，更多的规则等细节可以自行查阅代码规范手册或者百度，本文中不作过多讲述）

比如，我们再次来到之前写的 `test` 方法上，打入 `/**`、`*`、`*`，再按下回车：

```

/**
 *
 * @param n
 */
public static void test(int n) {
    for (int i = 1; i <= n; i++) {
        int y = i * i + 2;
        System.out.println(y * y);
    }
    for (int i = 1; i <= n; i++) {
        int x = i * i;
        System.out.println(x + x);
    }
}

```

然后我们按照规定的格式来补齐这个javadoc框架：

```

/**
 * 我是一个测试函数2333
 *
 * @param n 要输入的N值
 */
public static void test(int n) {
    for (int i = 1; i <= n; i++) {
        int y = i * i + 2;
        System.out.println(y * y);
    }
    for (int i = 1; i <= n; i++) {
        int x = i * i;
        System.out.println(x + x);
    }
}

```

一个格式规范且很清晰的方法文档就这样生成了。

除此之外，javadoc规范另外一个很重要的用途就是可以一键生成html页面版项目文档。点击**Menu -> Tools -> Generate Javadoc**，即可自动生成完整的javadoc网页版文档（[具体操作可参考此教程](#)）

Git

除此之外，Idea实际上也像eclipse一样对于git有完美的图形化支持。然而笔者一直使用git命令行进行所有git相关操作，对这一块暂不是很熟悉。所以还请各位搜集资料并自行探索。

插件

实际上，在这个网络化体系化作战的时代，jetbrains也有**很多的在线插件支持**。

我们只需要进入**Menu -> File -> Settings -> Plugins**，再点击 **Install JetBrains plugin...**，即可**搜索插件**并直接进行**在线安装**。（实际上，由于大陆内一些神奇的不可言表的原因，经常会出现连接失败或者下载速度极慢的情况。这时候请自行**设置代理**，本文不再赘述）

接下来笔者来安利几款比较好的插件：

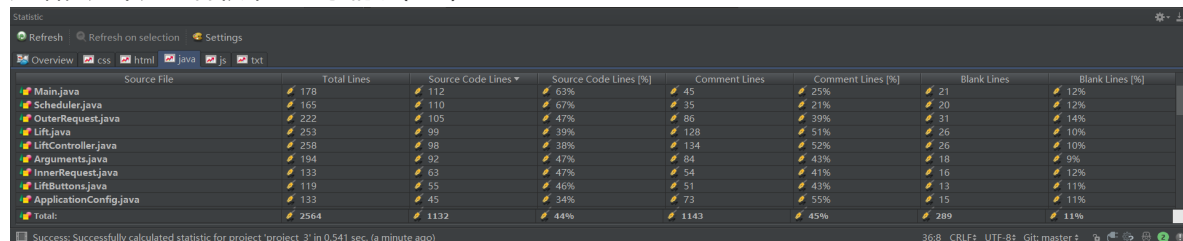
MetricsReloaded

对于之后的博客作业，我们需要用到的代码分析插件。

Class metrics						
class	CBO	DIT	LCOM	NOC	RFC	WMC
configs.ApplicationConfig	8	1	10	0	15	14
enums.lift.LiftMoveDirection	4		1		2	2
exceptions.application.arguments.Invalid	2	4	0	0	3	1
exceptions.application.arguments.Invalid	6	3	0	3	2	1
exceptions.application.arguments.Invalid	2	4	0	0	3	1
exceptions.application.arguments.NoArg	2	4	0	0	2	1
exceptions.application.InvalidDebugLeve	2	3	0	0	5	2
exceptions.input.InputNotStartFromTheI	2	3	0	0	4	1

Statistic

这个插件没啥别的功能，就是统计代码行数。那意义何在呢？嘿嘿，试想写着代码，看着代码行数不断递增，是不是一件很带感的事情呢(^▽^)。



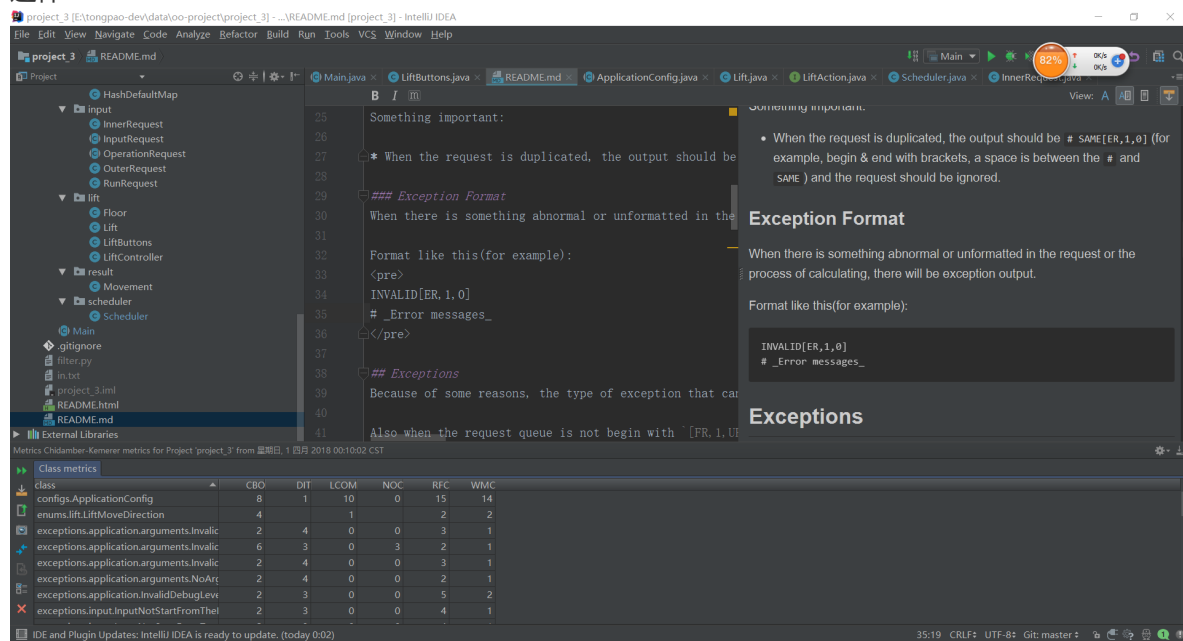
Source File	Total Lines	Source Code Lines	Source Code Lines (%)	Comment Lines	Comment Lines (%)	Blank Lines	Blank Lines (%)
Main.java	178	112	63%	45	25%	21	12%
Scheduler.java	165	110	67%	35	21%	20	12%
OuterRequest.java	222	105	47%	86	39%	31	14%
Lift.java	253	99	39%	128	51%	26	10%
LiftController.java	258	98	38%	134	52%	26	10%
Arguments.java	194	92	47%	84	43%	18	9%
InnerRequest.java	133	63	47%	54	41%	16	12%
LiftButtons.java	119	55	46%	51	43%	13	11%
ApplicationConfig.java	133	45	34%	73	55%	15	11%
Total:	2564	1132	44%	1143	45%	289	11%

.ignore

这是个管理.gitignore的插件，可以用颜色标记出当前项目下所有文件的git状态（包括Ignored, **Untracked**, **Unmodified**, **Modified**）

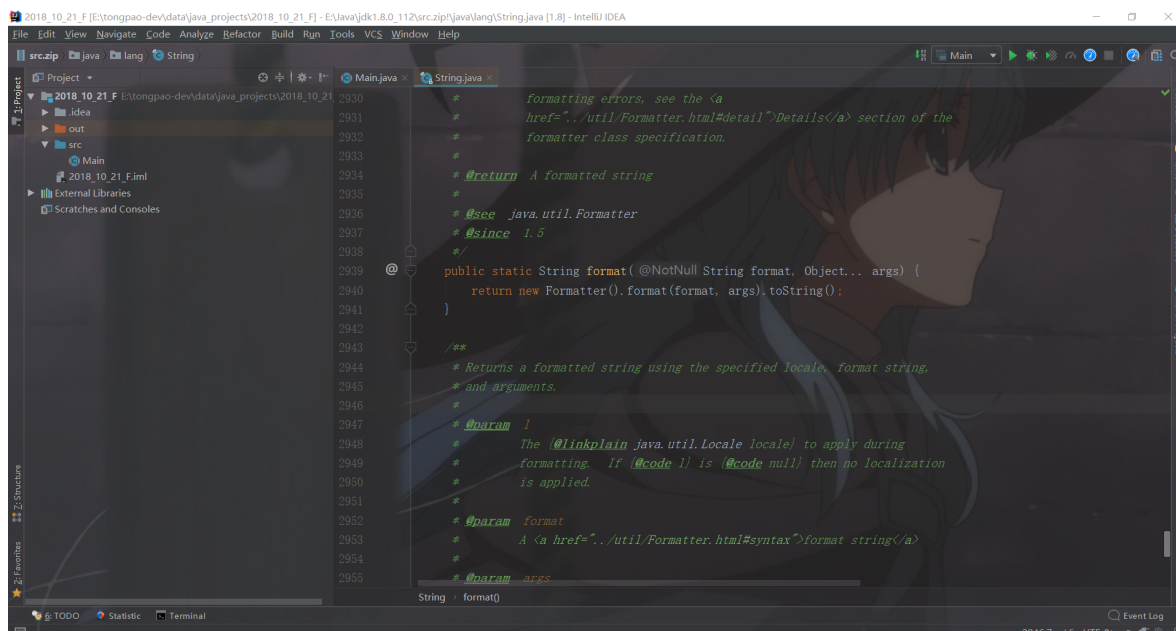
Markdown support

对于使用Markdown书写文档的同学来说，能有一款优雅可视的内置插件当然是一件很爽的事情。就像这样



Background Image Plus

对于想要给自己的ide设置壁纸的同学们，可以安装这一插件，并进行相关的配置，就像这样。



Markdown

Markdown 是一种用来写作的轻量级标记语言，它用简洁的语法代替排版，使普通文本内容具有一定的格式。对比于Word文档一般需要大量的排版、字体设置，Markdown语言可以通过很简单的几行代码就准确控制好文章的格式。在之后，大家写的单元总结以及代码Readme文档都用到这种语言书写文档。

那么，我们要如何利用Markdown语言书写文档？这就需要借助Markdown编辑器了。Typora编辑器是一种即时预览型编辑器，具有很快地流畅度和反应速度。因此，我们推荐大家使用Typora编辑器。

Typora安装

Typora 官网：<https://typora.io/>

Windows安装

在官网上下载相应exe文件，按照指引进行安装。

Linux安装

使用Linux系统的同学可以移步官方教程，在终端敲入相应命令完成安装。

官方教程链接：<https://support.typora.io/Typora-on-Linux/>

Markdown教程

我们推荐同学们通过菜鸟教程学习Markdown基本语法

链接：<https://www.runoob.com/markdown/md-tutorial.html>