

该文档是关于实验一代码自动测试的说明文档，将说明如何对自动测试程序进行配置。

1. 指定被测测试源代码所在包

为了使得测试代码能顺利加载每个同学的 class，实验一通过 Javadoc API 规定了每个类的类名、每个类的数据成员的类型和变量名、每个方法的方法名、形参类型、返回类型必须一致。除此之外，还必须**统一包名**。

IDEA Java 工程里定义了如下 package，这些包的作用如下图所示：

程序包	说明
<code>hust.cs.javacourse.search.index</code>	<code>hust.cs.javacourse.search.index</code> 包里定义了和倒排索引数据结构相关的抽象类，以及和索引构建相关的抽象类和接口。
<code>hust.cs.javacourse.search.parse</code>	<code>hust.cs.javacourse.search.parse</code> 包里定义了文档解析、分词，单词过滤有关的抽象类。学生需要实现这些抽象类的具体子类。
<code>hust.cs.javacourse.search.query</code>	<code>hust.cs.javacourse.search.query</code> 包里定义了和搜索有关的抽象类和接口。学生需要实现这些抽象类和接口的具体子类。
<code>hust.cs.javacourse.search.run</code>	最后的程序运行入口类放在 <code>hust.cs.javacourse.search.run</code> 里。
<code>hust.cs.javacourse.search.util</code>	<code>hust.cs.javacourse.search.util</code> 包里实现了一些工具类，学生可以参考和直接使用。具体包括：Config：索引构建和搜索的配置信息；StopWords：停用词表；StringSplitter：将字符串分割成一个个的单词；FileUtil：读写文本文件。

另外工程里还定义了下面三个空的包

`hust.cs.javacourse.search.index.impl`：对 `hust.cs.javacourse.search.index` 包里定义的抽象类和接口的具体实现放在这个包里。

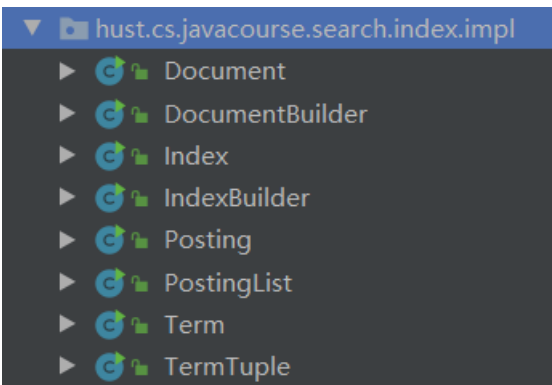
`hust.cs.javacourse.search.parse.impl`：对 `hust.cs.javacourse.search.parse` 包里定义的抽象类和接口的具体实现放在这个包里。

`hust.cs.javacourse.search.query.impl`：对 `hust.cs.javacourse.search.query` 包里定义的抽象类和接口的具体实现放在这个包里。

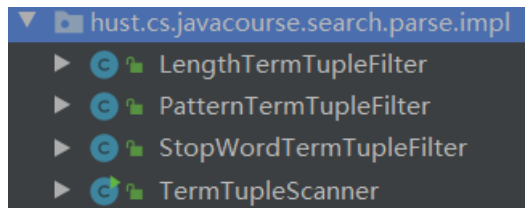
工程里还定义了 `hust.cs.javacourse.search.util` 包，里面包含了一些工具类，其中的类请不要做任何修改。

2. 指定具体类的类名

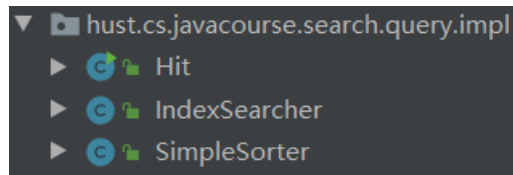
在实现工程里定义的抽象类的具体子类时，具体子类的类名为抽象父类名去掉 `Abstract` 得到名字，因此，`hust.cs.javacourse.search.index.impl` 包里的具体类应如下图所示：



`hust.cs.javacourse.search.parse.impl` 包里的具体类应如下图所示：



hust.cs.javacourse.search.query.impl 包里的具体类应如下图所示:



其中 SimpleSorter 是 Sort 接口的实现类。

3. 测试条件

在测试过程中需要保持一致的 **Config** 类的配置信息如下所示:

```
/**
 * 构建索引和检索时是否忽略单词大小写
 */
public static boolean IGNORE_CASE = true;

/**
 * <pre>
 * 将字符串切分成单词时所需的正则表达式。
 * 例如根据中英文的逗号, 分号, 句号, 问号, 冒号, 感叹号, 中文顿号, 空白分割符进行切分
 * </pre>
 */
public static String STRING_SPLITTER_REGEX = "[,|,|;|,|.|。|?| ?|:|:|!|!|、|\\s|\\\"|\"+]";

/**
 * <pre>
 * 单词过滤的正则表达式。
 * 例如正则表达式指定只保留由字母组成的term, 其他的term 全部过滤掉, 不写入倒排索引
 * </pre>
 */
public static String TERM_FILTER_PATTERN = "[a-zA-z]+";

/**
 * <pre>
 * 基于单词的最小长度过滤单词。
 * 例如指定最短单词长度为3, 长度小于3 的单词过滤掉, 不写入倒排索引
 * </pre>
 */
public static int TERM_FILTER_MINLENGTH = 3;
```

```
/**
 * <pre>
 * 基于单词的最小长度过滤单词.
 * 例如指定最长单词长度为20, 长度大于20的单词过滤掉, 不写入倒排索引
 * </pre>
 */
public static int TERM_FILTER_MAXLENGTH = 20;
```

除了上面的测试条件要保持一致外，抽象类 `AbstractIndex` 的数据成员 `docIdToDocPathMapping`、`termToPostingListMapping` 的访问控制权限要改成公有，如下所示：

```
public abstract class AbstractIndex implements FileSerializable{
    /**
     * 内存中的docId 和 docPath 的映射关系, key 为 docId, value 为对应的 docPath.
     * TreeMap 可以对键值排序
     */
    public Map<Integer, String> docIdToDocPathMapping = new TreeMap<>();

    /**
     * 内存中的倒排索引结构为HashMap, key 为Term 对象, value 为对应的PostingList 对象.
     */
    public Map<AbstractTerm, AbstractPostingList> termToPostingListMapping =
        new TreeMap<AbstractTerm, AbstractPostingList>();
```

改成公有的原因是这二个数据成员没有定义公有的 `getter` 和 `setter`，因此测试代码无法检测其中内容的正确性。

除此之外，在测试过程中发现 `hust.cs.javacourse.search.util` 包里的 `StringSplitter` 类有 bug，因此会发布这个类的新版本，请同学们更新这个类。

4. 测试代码的目录结构

测试代码的目录结构如下图所示：

Experiment1Test				
名称	修改日期	类型	大小	
betest	2020/4/10 10:39	文件夹		
lib	2020/4/10 10:52	文件夹		
test	2020/4/10 11:56	文件夹		
test-output	2020/4/10 11:05	文件夹		
test.bat	2020/4/10 11:38	Windows 批处理...	1 KB	
test.sh	2020/3/21 0:04	SH 文件	1 KB	
testng.xml	2020/4/10 11:58	XML 文档	10 KB	
实验—自动测试Readme.docx	2020/4/10 12:12	Microsoft Word ...	1,321 KB	

测试代码的根目录为 `Experiment1Test`。在该目录中，各文件和目录内容说明如下：

- `testng.xml` 为测试的配置信息；
- `test.bat` 是 Windows 运行测试程序的脚本文件，**大家需要修改**，修改方式和 11-13 章作业的测试脚本修改一样；

- test.sh 是 Linux/Mac 运行测试程序的脚本文件，**大家需要修改**，修改方式和 11-13 章作业的测试脚本修改一样；
- test-output 目录是测试结果报告的输出目录，**该目录名不要修改，目录也别删除**；
- lib 目录是测试程序运行所依赖的 jar 包，**该目录名不要修改，目录也别删除**；
- betest 目录是最后提交测试代码的目录，学生在本机测试时，可以不使用这个目录。但提交作业时，需要把作业的 class 文件拷贝到这个目录。具体说明见第 6 节。
- test 目录为测试代码以及测试需要的资源文件，**请不要对该目录做任何修改**。

请保持 Experiment1Test 下的目录结构不变，**该目录不要放在包含中文带空格的目录下**。大家进入到该目录，运行 test.bat 就可以启动测试程序。如果是在 Linux 下或者 Mac 下面，目录结构和要求一样。

5. 测试程序运行及查看测试结果

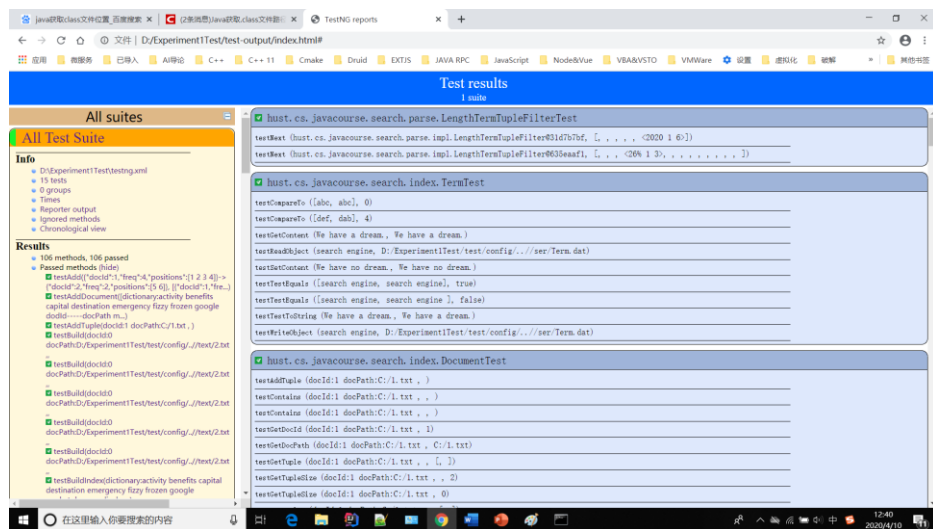
在命令行进入到 Experiment1Test 目录，运行 test.bat。运行结果如下图所示：

```
=====  
All Test Suite  
Total tests run: 106, Failures: 0, Skips: 0  
=====  
  
D:\Experiment1Test>_
```

这时进入到子目录 test-output，里面内容如下图所示：

Experiment1Test > test-output				
名称	修改日期	类型	大小	
All Test Suite	2020/4/10 12:39	文件夹		
junitreports	2020/4/10 12:39	文件夹		
old	2020/4/10 12:39	文件夹		
bullet_point.png	2020/4/10 12:39	PNG 文件	1 KB	
collapseall.gif	2020/4/10 12:39	GIF 文件	1 KB	
emailable-report.html	2020/4/10 12:39	Chrome HTML D...	101 KB	
failed.png	2020/4/10 12:39	PNG 文件	1 KB	
index.html	2020/4/10 12:39	Chrome HTML D...	238 KB	
jquery-1.7.1.min.js	2020/4/10 12:39	JavaScript 文件	92 KB	
navigator-bullet.png	2020/4/10 12:39	PNG 文件	1 KB	
passed.png	2020/4/10 12:39	PNG 文件	1 KB	
skipped.png	2020/4/10 12:39	PNG 文件	1 KB	
testng.css	2020/4/10 10:50	CSS 文件	1 KB	
testng-failed.xml	2020/4/10 11:05	XML 文档	7 KB	
testng-reports.css	2020/4/10 12:39	CSS 文件	6 KB	
testng-reports.js	2020/4/10 12:39	JavaScript 文件	4 KB	
testng-results.xml	2020/4/10 12:39	XML 文档	116 KB	

打开 index.html，可以看到测试结果，如下图所示：



可以看到一共有 106 个测试用例，106 个都通过。点击左边的链接，可以看到详细信息。

6. 提交测试包

为了方便作业提交后助教对学生的代码测试，学生需要把作业编译后的 class 文件拷贝到 Experiment1Test 下的 betest 目录下。即 betest 目录为学生代码的包的上一级目录，如下图所示：

Experiment1Test > betest				
名称	修改日期	类型	大小	
hust	2020/4/10 12:47	文件夹		

同时，将测试脚本里 TO_BE_TEST_CLASSPAH 的目录设置成相对目录。

对于 Windows 脚本 test.bat，相应语句改为：

```
set TO_BE_TEST_CLASSPAH=.\betest
```

对于 Linux/Mac 脚本 test.bat，相应语句改为：

```
TO_BE_TEST_CLASSPAH=./betest
```

注意斜杠前面有一个.，表示当前目录。

类似地，第三题作业拷贝到 **betest\homework\ch11_13\p4** 包下面。

最后把 Experiment1Test 打包，随同实验工程一起提交。在将 Experiment1Test 打包提交前，学生应该在本机运行修改后的脚本，确保测试代码可以测试 betest 下面的 class 文件。

7. 测试运行的 JDK 版本问题

由于测试代码是在控制台下运行而不是在 IDEA 里运行，**所以要保证从控制台运行时启动 java 程序的 JDK 版本为 JDK13**。验证方法是在控制台运行 `java -version`，如下图所示：

```
F:\课件\JavaCourse\2019-2020春季PPT\作业\AutoTest_update2>java -version
java version "13.0.2" 2020-01-14
Java(TM) SE Runtime Environment (build 13.0.2+8)
Java HotSpot(TM) 64-Bit Server VM (build 13.0.2+8, mixed mode, sharing)
```

8. test 目录

在 `test` 目录里除了包含测试代码外，还包含了一些资源文件。其中 `text` 开头的子目录包含了用来进行测试的最原始的 `txt` 文件；`index` 目录和 `ser` 目录存放序列化得到的二进制 `dat` 文件；`json` 目录包含了很多 `Json` 文件，是将 `Java` 运行的正确结果序列化为 `Json` 文件得到的。在测试学生代码时，测试代码再将这些 `Json` 文件反序列化成 `Java` 对象，然后作为正确结果对学生代码进行测试。

`test` 目录下的 `config` 包里也有一个 `Config` 类，但这个 `Config` 是对测试代码进行配置的，和学生代码的 `Config` 类无关。