ClassFileReader

1. 实验背景

如果搜索关键字"安装java"那么绝大部分的教程都会说明一件事——在安装完之后需要设置环境变量。
那么什么是环境变量，为什么一定要设置它?

我们先从如何在命令行中启动一个java程序讲起。
想要在命令行里启动java程序，通常会借助java命令，你可以在JDK/jre/bin这个目录下找到它。
java命令就是上一次作业中提到过的一个命令行工具，它是Java虚拟机的启动器。
官方文档中是这样描述的

```
The java command starts a Java application.
It does this by starting the Java Runtime Environment (JRE),
loading the specified class, and calling that class's main() method.
The method must be declared public and static,
it must not return any value, and it must accept a String array as a
parameter.
 The method declaration has has the following form:

public static void main(String[] args)
```

它可以通过下面这样的指令来执行一个类或者一个Jar包中的main方法。
-cp是classpath的缩写，它带有一个必选参数。

```
    java [options] -cp classpath className [args]
java [options] -cp classpath -jar jarName [args] //中括号代表可选
```

以第一条指令为例，className是除了选项和其带有的参数这些之外的第一个参数。
java会默认把第一个参数作为要查找的类。
首先，每一个类都是以一个.class文件的形式存放在磁盘中。
现在java拿到了类名，那么想要从磁盘里读出数据，显然还需要一个路径。
因为路径+文件名才唯一确定了这个文件。
简单来说，环境变量的作用就是提供了一条默认的路径。
java命令会搜索这个路径下面的文件或路径下某个子路径的文件来匹配是否存在要找的类，并把它读出来。
设置环境变量这一步其实并不是必须的，因为java里也有-Xbootstrap -Xjre -cp这些命令可以修改默认查找的路径，是可以在输入参数时指定的。

在编写java程序的时候往往会借助一些别人写好的类，这些类大部分来自官方标准库。
当使用到这些类时，只需要一句import，而程序是如何知道该怎么用这些类的呢?
回到环境变量，设定JAVA_HOME的目的就是指定了去哪里找到这些标准库，这些标准库就是以.class文件的形式存在在磁盘上的。
而-cp classpath的作用与JAVA_HOME雷同，只不过它是告诉程序去哪里寻找你自己写的类，而JAVA_HOME找的是标准库中的类。

命令行执行"java [options] -cp classpath className"时候经常会出现“Could not find or load main class”这个报错。
这个报错一种可能的情况是路径正确，找到了对应的类，但是类中没有一个"public static void main(String[] args)"
另一种可能的情况是，类是正确的(类名正确，磁盘上存在文件且有一个标准的main方法)，而路径是错误的。

在官方规范中，java命令可以接收四种格式的classpath：

- 相对路径： 某个单独的路径 例如**dir/subdir**
- 归档文件： 某个jar包或zip包 例如**dir/subdir/target.jar**
- 通配符 ： 某个路径下的全部jar文件 例如**dir/subdir/***
- 复合型 ：使用系统路径分隔符（windows下为分号，unix下为冒号。可以从Java类库里`File.pathSeparator`获取）混合上述三种的路径表达 例如**dir/subdir;dir/subdir/*;dir/target.jar**

我们本次作业的任务是编写一个工具，它能够从指定目录读取指定的class文件内容。

2. 实验要求

在动手开始实现之前，**请仔细阅读文档中的每一个要求和说明**
不用太着急，耐心读完下面的手册内容你很快就可以实现一个正确的版本。

2.1 实验输入

程序需要对外暴露的唯一接口是ClassFileReader类中的readClassFile方法。
这意味着这个方法的方法签名**绝对不能够被修改**，而其余的方法都可以自由修改实现。

```
1  public static byte[] readClassFile(String classpath,String className) throws
   ClassNotFoundException
```

readClassFile有两个String类型的参数，分别指定classpath和className，需要注意以下几点：

1. classpath使用的是相对路径，它一定符合上述四种规范中某一种，测试用例中不会使用不存在的路径
2. classpath和className中均不需要考虑双星号通配符的情况，例如将**/lang/Object匹配为java/lang/Object
3. className的格式是全限定名，即类似 java/lang/Object 这样使用"/"将包名和类隔开的字符串

2.2 实验输出

程序的合法输出包括

- 读取出来的class文件内容，以 byte[] 的形式返回
- 在指定cp下没有发现class时抛出ClassNotFoundException

2.3 实验要求

1. 对于dir/*这样的路径，只需要考虑dir目录下的jar文件，不需要考虑dir/subdir目录的jar，即无需递归查找
2. 在通过系统路径分隔符组合起来的路径中，各个路径具有先后顺序，位于前面的路径中一旦查找到了对应的文件则读出来返回
3. 考虑参数的所有可能的情况，避免Null Pointer Exception的情况
4. zip和jar文件后缀名均有两种形式，全大写和全小写，"zip ZIP" "jar JAR"

2.4 代码指导

我们在src/test目录下放置了用于测试的文件，使用tree命令的打印结果如下。
首先，抛开之前的实验相关需求，我们不妨先回答一个问题——对于某个合法但未知的路径和一个确定为 java/lang/Object 的类名，这个目录下面哪些文件的内容可能被读出来?

```
testfilepath
     └dir
         |  Object.class
         |
         ├java
         |  └lang
         |          Object.class
         |
         └subdir
                 empty.jar //里面没有东西
                 rt.JAR //里面有一个java/lang/Object.class
```

答案是有两个，一个是dir/java/lang目录下的Object.class，另一个是subdir目录下的rt.JAR

那么路径和类名到底做了什么，它们是如何组合的?
(某些名词如果不清楚含义请自行查阅解释，例如什么是绝对路径和相对路径，什么是zip/jar包)

第一种，相对路径：与类名直接组合，然后获取绝对路径读取文件
第二种，归档文件：遍历寻找匹配文件
第三种，通配符：一个jar包与无关文件的集合
第四种，复合型：以上所有组合形式的集合

在作业中，我们提供了一种推荐的实现模式(当然你也可以自己实现)
定义一个Entry类，它是ClassFileReader中真正负责读数据的类。Entry类是一个抽象类，它拥有一个抽象的readClassFile方法。

```
public abstract class Entry {
    public final String PATH_SEPARATOR = File.pathSeparator;
    public final String FILE_SEPARATOR = File.separator;
    public String classpath;

    public Entry(String classpath){
        this.classpath = classpath;
    }

    public abstract byte[] readClassFile(String className) throws
IOException;
}
```

然后我们定义四个子类，继承Entry并实现readClassFile方法。

```

public class DirEntry extends Entry //相对路径
public class ArchivedEntry extends Entry //归档文件
public class WildEntry extends Entry //通配符
public class CompositeEntry extends Entry //复合型
```

前面两种比较好理解，平时也应该做过相应的练习。
而后两者其实是前两者的集合，对于通配符类型，可以将目录中的jar文件全部取出，然后用路径分隔符连接。
这样就从第三种转变成为了第四种复合型。

Tips: 从代码复用的角度出发来思考如何实现后两者

我们在ClassFileReader中声明了这样一个方法，它的作用是根据classpath来判断该使用哪一种Entry的实现来读取文件。为了防止错误的读取，每一种类型都有其各自的特征，如果用if-else来实现这个方法，应该注意一定的顺序。例如，在一开始就判断classpath是否以.zip结尾，如果是就使用ArchivedEntry来读取。这种做法在面临a.zip:b.zip:c.zip 这样的路径时就会出错。此外还需要当心zip文件和jar文件的大小写问题。

```
1 public static Entry chooseEntryType(String classpath)
```

我们还提供了什么?
一个IOUtil工具类，总是写输入输出流读文件是很烦的，所以我们提供了一个正确实现的版本。
在这个工具类里还实现了一个字符串转换的方法，也许你会在处理系统路径时需要用到它。

2.5 关于测试

1. 我们对每一种路径的类型都设计了一正一反的用例。但是，在看测试用例之前你应该先思考如何正确实现这个工具。一个不错的思路是，如果你面对之前的树状图，你会如何设计用例？你会怎么样在给定条件内尽可能去刁难你的代码
2. 如果你认为自己完成了正确的实现但是还没能通过全部测试用例，不要怀疑人生，机器永远是对的。请再看一眼2.3实验要求中的内容，检查是否遗漏了某个要求
3. 测试用例里唯一会catch的异常只有ClassNotFound，在助教自己写的版本中，测试用例中的路径以及读文件的工具类不会导致IO异常。如果你出现了相关的报错，请确认是否是系统分隔符引起的

其他

如果你在使用时发现了作业设计中的bug，请直接QQ群中猛戳助教。
也欢迎其他关于文档的建议和反馈。

我们会尽量快速及时地回复大家的问题，在这之前，推荐同学们通过阅读实验手册、官方文档的方式来增加对实验要求和使用场景的理解。对于具体的报错信息，可以尝试使用Bing(英文)、百度(中文)进行搜索。
我们推荐在提问的时候贴出自己搜索的结果，这样解决问题的同时也能提升大家搜索信息的能力。