链滴

jvm-demo 系列一：解析命令行工具类

作者：shyjiang233
原文链接：https://ld246.com/article/1682331309940
来源网站：链滴

参考资料：《深入理解java虚拟机》

《自己动手写java虚拟机》

zachaxy手写jvm系列

先自己动手编写一个类似java命令行的工具，用它启动虚拟机。Java虚拟机的工作是运行Java应用程。和其他类型的应用程序一样，Java应用程序也需要一个入口点，这个入口点就是我们熟知的main）方法。如果一个类包含main（）方法，这个类就可以用来启动Java应用程序，我们把这个类叫作类。最简单的Java程序是

只有一个main（）方法的类，如著名的HelloWorld程序。那么Java虚拟机如何知道我们要从哪个类动应用程序呢？对此，Java虚拟机规范没有明确规定。也就是说，是由虚拟机实现自行决定的。比如Oracle的Java虚拟机实现是通过java命令来启动的，主类名由命令行参数指定。java命令有如下形式：

```
java [-options] class [args]

java [-options] -jar jarfile [args]

javaw [-options] class [args]
```

这里我们不先考虑完整的java命令用法，打算实现几个比较基础的命令

```
    java -version
    java -?
    java -help
    java -cp your/classpath yourClassName
```

先用一个结构体表示命令行

```
public class Cmd{
   Boolean helpFlag;//是否是help命令
   Boolean versionFlag;//是否是查看版本命令
   String cpOption;//classPath路径
   String  clazz;//
   String[]  args;//执行clazz文件需要的参数
}
```

先对输入的命令进行正则表达式处理，判断是否符合java命令用法，不符合直接返回错误。

```
public class CmdValidator {
   private static final Pattern CMD_PATTERN = Pattern.compile("^java\\s+(-version|-\\?|-help|
cp|-classpath)\\s*(.*)$");
    public  static  boolean isValidCmd(String cmdLine){
       Matcher matcher=CMD_PATTERN.matcher(cmdLine);
       return matcher.matches();
    }
}
```

一开始我是想去用if,else的逻辑去判断命令行处理，类似于这样

```
 public void parseCmd(String[] args){
      if("-help".equals(args[1]) || "-?".equals(args[1])){
         System.out.println("java help");
      }else if("-verson".equals(args[1])){
```

```
            System.out.println("java version");
        }else if("-cp".equals(args[1])){
            System.out.println("java  -cp");
            System.out.println(args[2]+"  "+args[3]);
        }
```

代码看起来太丑陋了，而且这种代码只是我们实现的命令少，实现的命令多就会非常冗杂，所以我们责任链模式优化一下。

我们定义了一个抽象类 CommandHandler，它代表责任链中的一个处理步骤。每个具体的处理者类如 HelpHandler、VersionHandler和 ClasspathHandler）都负责检查它是否可以处理给定的命令参数。如果它不能处理，就将处理委托给责任链中的下一个处理者。

```
public abstract class CommandHandler {
    protected CommandHandler successor;

    public void setSuccessor(CommandHandler successor) {
        this.successor = successor;
    }

    public abstract void handleCommand(String[] args);
}

public class HelpHandler extends CommandHandler {
    @Override
    public void handleCommand(String[] args) {
        if ("-help".equals(args[1]) || "-?".equals(args[1])) {
            System.out.println("java help");
        } else if (successor != null) {
            successor.handleCommand(args);
        }
    }
}

public class VersionHandler extends CommandHandler {
    @Override
    public void handleCommand(String[] args) {
        if ("-version".equals(args[1])) {
            System.out.println("java version");
        } else if (successor != null) {
            successor.handleCommand(args);
        }
    }
}

public class ClassPathHandler extends CommadHandler{
    @Override
    public void handleCommand(String[] args) {
     if("-cp".equals(args[1])){
         System.out.println("successful");
         System.out.println("class文件的路径"+args[2]+"class文件"+args[3]+"交给下一节处理");
     } else if (successor != null) {
         successor.handleCommand(args);
     }
```

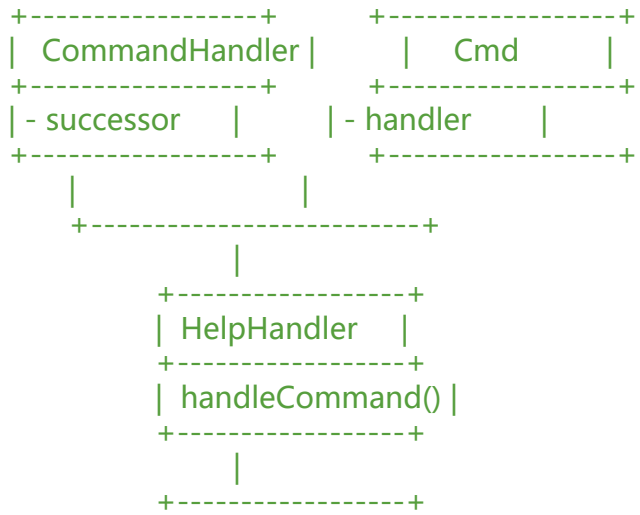
```
    }
}

public class Cmd {
    Boolean helpFlag;//是否是help命令
    Boolean versionFlag;//是否是查看版本命令
    String cpOption;//classPath路径
    String[]  args;//执行clazz文件需要的参数
    private CommandHandler handler;

   public  Cmd(String cmdLine){
        //解析命令行参数,以单个或者多个空格分开,这种方式目前不支持,因为如果输入的 路径名 中间
空格会导致下面解析失败
        if(!CmdValidator.isValidCmd(cmdLine)){
            System.out.println("Unrecognized option: -ewf\n" +
                    "Error: Could not create the Java Virtual Machine.\n" +
                    "Error: A fatal exception has occurred. Program will exit.");
            return;
        }
        //初始化责任链
        CommadHandler helpHandler=new HelperHandler();
        CommadHandler vesionHandler=new VersionHandler();
        CommadHandler classpathHandler=new ClassPathHandler();
        helpHandler.setSuccessor(vesionHandler);
        vesionHandler.setSuccessor(classpathHandler);
        handler=helpHandler;
        String[] args = cmdLine.trim().split("\\s+");
        parseCmd(args);
    }

    public void parseCmd(String[] args) {
        handler.handleCommand(args);
    }
}
```

在 Cmd类中，我们初始化了责任链，然后调用 handler.handleCommand(args)来开始处理命令行数。



这个逻辑图展示了这几个类之间的关系。CommandHandler类是一个抽象类，它定义了一个 successor属性，该属性表示责任链中的下一个处理者。具体的处理者类（HelpHandler、VersionHandler和 CasspathHandler）继承自 CommandHandler类，并实现了 handleCommand方法来处理命令行参。Cmd类初始化了责任链，并调用责任链的第一个处理者（HelpHandler）来处理命令行参数。

责任链模式的使用让代码的扩展性变好了。