软件复用课程设计

Client-Server 应用程序

复用文档

PM构件

小组： Team10

成员： 谢志杰 1352975

计鹏玥 1352914

王思尧 1352896

谢明玥 1352937

徐锦程 1353012

[1． 构件简介 1](#_Toc9816)

[2． 接口说明 1](#_Toc14573)

[2.1 CheckCount.java 1](#_Toc2368)

[(1) CheckCount(String logFilename) 1](#_Toc1730)

[(2) addCountType(String msg) 2](#_Toc28348)

[(3) addCount(String msg) 2](#_Toc20860)

[(4) getLock(String msg) 3](#_Toc25766)

[(5) setValue(String key, long value) 3](#_Toc5782)

[2.2 IOLog.java 3](#_Toc9583)

[(1) IOLog(String logAddr, boolean bAppend) 3](#_Toc5657)

[(2) IOWrite(String sWriten) 4](#_Toc2794)

[2.3 Logger.java 5](#_Toc15539)

[(1) Logger(String logFilename) 5](#_Toc2198)

[(2) setTime(int firstTime, int period) 5](#_Toc2919)

[(3) commence() 6](#_Toc28918)

[(4) terminate() 6](#_Toc20976)

[(5) addCountType(String msg) 6](#_Toc9983)

[(6) addCount(String msg) 7](#_Toc30989)

[(7) getLock(String msg) 7](#_Toc9125)

[3．Demo 8](#_Toc17481)

# 构件简介

|  |  |  |  |
| --- | --- | --- | --- |
|  | **计数与写日志 － CheckCount** | **IO写操作－IOLog** | **计时与写日志 － Logger** |
| 构件介绍 | 这一构件的主要功能是对功能要求的计数字段进行计数，并实现将相关字段以及计数结果写入日志的功能。 | 这一构件的主要功能是针对于频繁的IO写操作提供一个便利的接口使得IO写操作的具体实现对外不可见，用户仅仅需要提供文件地址并使用提供的方法即可简单完成文件写操作。 | 这一构件的主要功能是对用户在计时器时间范围内登录次数的计数，包括初次登录和系统重新登录。 |
| 功能说明 | 用户可以根据自己的需要添加所需的计数字段，计数字段一旦添加即不可更改，在写入文件时全部字段会写入日志。  该构件计数功能有相应接口，每次计数增加值为1，计数从0开始。 | 这一构件本质为一个方法类，用户使用构件时需要实例化一个类，并传入写文件地址、是否在文件尾写入作为参数，实例化后用户可以调用类中IOWrite的方法，传入写入的字符串作为参数，即可完成写操作的功能。 | 用户可以使用该构件实现对于登录次数的计数功能。 |

# 接口说明

## 2.1 CheckCount.java

注意：应先导入PM包。

### (1) CheckCount(String logFilename)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| logFilename | String | 写入的日志的地址 |

**作用描述：**

该方法用于指定写日志的地址，地址可以为绝对路径，也可以为相对路径。

**使用方法：**

构造函数，可以指定写日志的地址。该方法只有一个string类型的参数，代表写日志的文件地址，形如：

CheckCount checkCount = new CheckCount(“server.log”);

**注意事项：**

A. 该方法实际上为该构件的构造函数，通过该方法可以实例化一个CheckCount对象。

B. 该方法的参数代表日志地址，可以为相对路径，也可以为绝对路径。

### (2) addCountType(String msg)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| msg | String | 计数字段 |

**作用描述：**

该方法用于添加计数字段，添加之后可以利用后面的方法计数。

**使用方法：**

使用该方法添加计数字段需要传递一个string类型的参数，该参数代表计数字段，添加后即可真对该计数字段计数。形如：

checkCount.addCountType(valid\_login\_per\_min);

**注意事项：**

该方法可以生成一个新的计数字段，计数字段一旦添加不可删除，计数字段从0开始计数。

### (3) addCount(String msg)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| msg | String | 需要增加的计数字段 |

**作用描述：**

该方法用于针对指定的计数字段进行计数，调用该方法后。可以在指定的计数字段上加一。

**使用方法：**

使用该方法对某一指定的计数字段计数需要传递一个string类型的参数，参数代表需要增加的计数字段，形如：

checkCount.addCount(received\_msg);

**注意事项：**

该方法用于对指定的计数字段执行增加操作，每次调用只增加一。

### (4) getLock(String msg)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| msg | String | 获取对象锁的计数字段 |

**作用描述：**

该方法用于获取指定计数字段的锁，用于在多线程中保证上述增加方法的同步，避免造成多个线程同时调用上述方法导致的问题。

**使用方法：**

使用该方法获取某一指定的计数字段锁需要传递一个string类型的参数，参数代表需要获取对象锁的计数字段，形如：

checkCount.getLock(received\_msg);

**注意事项：**

使用该方法需要传递计数字段作为参数，获取的是与传递的计数字段对应的锁。

### (5) setValue(String key, long value)

**作用描述：**

该方法用于将全部的计数字段以及各个计数字段所对应的计数结果存入在构造函数中指定的日志文件中，该方法实质为TimerTask中的run方法，可以用于定时器操作。

**使用方法：**

该方法一般用于定时器，形如：

loginTimer.schedule(checkCount, 0, 60000);

该方法也可以直接调用。

**注意事项：**

该方法一般用于定时器，即本质为TimerTask中的run方法。

## 2.2 IOLog.java

注意：应先导入PM包。

### (1) IOLog(String logAddr, boolean bAppend)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| logAddr | String | 写文件地址 |
| bAppend | boolean | 是否在文件为写入 |

**作用描述：**

该方法实质为方法类的构造函数，用来指定写文件的地址以及是否在文件末尾执行写操作，使用该方法实例化方法类后可以使用后述的方法进行文件的写操作。

**使用方法：**

利用该方法实例化方法类，传入写文件地址、是否在文件尾写入作为参数，形如：

IOLog ioLog = new IOLog(“server.log”,true);

**注意事项：**

A. 在指定是否在文件末尾执行写操作时，传入true表示在文件末尾执行写操作，传入false表示覆盖原文件执行写操作。

B. 文件路径可以使用绝对路径也可以使用相对路径。

### (2) IOWrite(String sWriten)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| sWriten | String | 要写入的字符串 |

**作用描述：**

该方法用于将传入的字符串写入文件，通过上述构造函数产生的实例，调用该方法并传入想要写入的字符串作为参数，即可完成对于文件的写操作。

**使用方法：**

首先需要通过上述的方法实例化一个方法类，在上述方法中传入文件路径、是否在文件尾写入的参数后，即可调用该方法，该方法有一个string类型的参数，即传入想要写入指定文件的字符串，形如：

ioLog.IOWrite("forwarded message\n”);

**注意事项：**

只能向指定的文件执行写操作，而且只能以指定的方式（是否在文件尾写）执行写操作。

## 2.3 Logger.java

注意：应先导入PM包。

### (1) Logger(String logFilename)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| logFilename | String | 将要创建日志的文件名 |

**作用描述：**

新建登录日志，实例化计时器。

**使用方法：**

对于一个实例化的Logger对象，可以直接调用该方法，该方法输入的字符串参数表示将要创建日志的日志名，没有返回值，形如：

String logger = username;

Logger logger = new Logger(logger + ”.log”);

**注意事项：**

此方法在实例化Logger对象时被调用。

### (2) setTime(int firstTime, int period)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| firstTime | int | 计时器的开始时间 |
| period | int | 计时器的周期 |

**作用描述：**

该方法用于设置登录计数的开始时间和周期长度。

**使用方法：**

对于一个实例化的Logger对象，可以直接调用该方法，该方法有两个int类型的参数，分别代表checkCount开始执行的时间和其执行周期长度，没有返回值，形如：

logger.setTime(0, 60000);

### (3) commence()

**作用描述：**

该方法用于开启登录计数的计时器。

**使用方法：**

对于一个实例化的Logger对象，可以直接调用该方法，该方法没有输入参数，返回值为空，形如：

logger.commence();

**注意事项：**

必须先对firstTime和period进行初始化，并对checkCount进行实例化才可调用。

### (4) terminate()

**作用描述：**

关闭计时器。

**使用方法：**

对于一个实例化的Logger对象，可以直接调用该方法，该方法既没有输入参数，也没有返回值，形如：

logger.terminate();

**注意事项：**

只有在实例化计时器之后才可以将计时器关闭。

### (5) addCountType(String msg)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| msg | String | 要计数的消息 |

**作用描述：**

该方法用于给消息器设置计数类型，包括成功登录次数、失败登录次数。

**使用方法：**

对于一个实例化的Logger对象，可以直接调用该方法，该方法的输入参数为string类型，表示要计数的消息，没有返回值，形如：

logger.addCountType(“valid\_login\_per\_minute”);

**注意事项：**

该方法表示的计数类型在server端只有五种，分别是vaild\_login\_per\_minute，invaild\_login\_per\_minute，received\_msg，ignored\_msg，forwarded\_msg。

### (6) addCount(String msg)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| msg | String | 消息的内容 |

**作用描述：**

该方法将消息数进行加一。

**使用方法：**

对于一个实例化的Logger对象，可以直接调用该方法，该方法的输入参数为string类型，表示要计数的消息，没有返回值。形如：

logger.addCount(“valid\_login\_per\_minute”);

**注意事项：**

传入的参数message必须为string类型。

### (7) getLock(String msg)

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| msg | String | 要计数的消息 |

**作用描述：**

该方法获得具有同步锁的消息。

**使用方法：**

对于一个实例化的Logger对象，可以直接调用该方法，该方法的输入参数为string类型，表示要计数的消息，没有返回值，形如：

logger.getLock(“valid\_login\_per\_minute”);

**注意事项：**

传入的参数message必须为string类型。

# 3．Demo

public class Server {  
 private boolean withLog = true;  
 private String logFile;  
  
 private Logger logger;  
  
 public String valid\_login\_per\_min = "valid login per min: ";  
 public String invalid\_login\_per\_min = "invalid login per min: ";  
 public String received\_msg = "received message: ";  
 public String ignored\_msg = "ignored message: ";  
 public String forwarded\_msg = "forwarded message: ";  
  
 public void work() throws IOException {  
 if (withLog) {  
 System.out.println("PM into " + this.logFile);  
 // 用日志文件名初始化Logger.  
 logger = new Logger(this.logFile);  
 // 以字符串标示需要记录的内容。  
 logger.addCountType(valid\_login\_per\_min);  
 logger.addCountType(invalid\_login\_per\_min);  
 logger.addCountType(received\_msg);  
 logger.addCountType(ignored\_msg);  
 logger.addCountType(forwarded\_msg);  
 // 设定第一次记录延迟，以及以后每次间隔。  
 logger.setTime(0, 60000);  
 // 开始记录。  
 logger.commence();  
 }  
  
 while(true) {  
 if (2 > 1) {  
 if (license.checkMsgInSecond()) {  
 // 将以该字符串标示的指标加一。  
 logger.addCount(received\_msg);  
 } else {  
 // 将以该字符串标示的指标加一。  
 logger.addCount(ignored\_msg);  
 }  
 }  
 }  
 // 终止日志。  
 logger.terminate();  
 }  
}