软件复用课程设计

Client-Server 应用程序

复用文档

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

文档变更记录

|  |  |  |
| --- | --- | --- |
| 更改人 | 日期 | 更改内容 |
| 王思尧 | 2016年3月28日 | 创建文档 |
| 计鹏玥，谢明玥 | 2016年4月2日 | 添加可复用构件 |
| 计鹏玥，谢明玥 | 2016年4月8日 | 完成可复用构件 |
| 计鹏玥，谢明玥 | 2016年4月11日 | 添加demo |
| 计鹏玥，谢明玥 | 2016年4月14日 | 接口变更 |

[文档变更记录 1](#_Toc32058)

[1．应用程序简介 1](#_Toc8740)

[2．冗余检查及标示可复用构件 1](#_Toc23029)

[2.1冗余检查 1](#_Toc11717)

[2.1.1 IO写操作 1](#_Toc13244)

[2.1.2 JSON相关操作 1](#_Toc1043)

[2.1.3 数据库操作 1](#_Toc30986)

[2.1.4 计数及日志 1](#_Toc540)

[2.1.5 消息计数 2](#_Toc8499)

[2.1.6 登录验证 2](#_Toc32379)

[2.1.7 计时与写日志 2](#_Toc15019)

[2.1.8 配置信息 2](#_Toc16113)

[2.1.9 许可证 2](#_Toc10444)

[2.1.10 消息发送 2](#_Toc25509)

[2.2 可复用构件列表 3](#_Toc29834)

[3. 可复用构件 3](#_Toc15741)

[3.1 Authentication构件 3](#_Toc28653)

[3.1.1 身份验证 - Authentication 3](#_Toc28723)

[3.1.2 用户登录验证 － Verification 5](#_Toc10835)

[3.2 CM构件 8](#_Toc1547)

[3.2.1 获得配置信息 － GetConfiguration 8](#_Toc25542)

[3.3 DataSource构件 12](#_Toc2656)

[3.3.1 数据库 － DataSource 12](#_Toc24546)

[3.4 License构件 14](#_Toc10586)

[3.4.1 许可证 － License 14](#_Toc15657)

[3.4.2 消息计数 － MessageCount 18](#_Toc1673)

[3.5 MessageUtils构件 21](#_Toc23475)

[3.5.1 消息 － Message 21](#_Toc18755)

[3.5.2 消息发送 － MessageDeparturer 27](#_Toc30363)

[3.6 PM构件 29](#_Toc5357)

[3.6.1 计数与写日志 － CheckCount 29](#_Toc28058)

[3.6.2 IO写操作－IOLog 31](#_Toc5275)

[3.6.3 计时与写日志 － Logger 32](#_Toc26795)

[4．可复用构件Demo 35](#_Toc15537)

[4.1 Authentication构件 35](#_Toc3508)

[4.2 CM构件 35](#_Toc14939)

[4.3 DataSource构件 37](#_Toc130)

[4.4 License构件 37](#_Toc2087)

[4.5 MessgaeUtils构件 38](#_Toc26525)

[4.6 PM构件 40](#_Toc6142)

# 1．应用程序简介

Client-Server 应用程序是一个JAVA语言开发的客户端服务器连接应用程序，具备登录、客户端消息发送、服务端消息验证及消息转发、消息数量限制、日志记录等基本功能。

# 2．冗余检查及标示可复用构件

## 2.1冗余检查

### 2.1.1 IO写操作

由于应用程序中无论是客户端还是服务端均涉及到诸多的日志记录需求，频繁的io写操作会造成一定的代码冗余，因此在本应用程序中将io写操作标示为可复用构件。

### 2.1.2 JSON相关操作

由于应用程序需要在客户端与服务器之间传递消息，消息采用JSON格式，因而程序中涉及到很多对于JSON格式的操作，例如JSON的生成与解析，因此在本应用程序中将JSON相关操作标示为可复用构件。

### 2.1.3 数据库操作

由于应用程序涉及到数据库的存取操作，对于数据库的频繁查找操作是本应用程序的一个基本特点，因此在本应用程序中将数据库操作标示为可复用构件。

### 2.1.4 计数及日志

由于应用程序在客户端与服务端均涉及到计数并存取日志的功能，因此在本应用程序中将该功能模块标示为可复用构件。

### 2.1.5 消息计数

由于应用程序中服务端需要对客户端的消息进行流量控制，因此对于客户端发送的消息需要在服务端计数，本应用程序中将消息计数标示为可复用构件。

### 2.1.6 登录验证

由于应用程序中服务端涉及到多次用户登录验证操作，包括初次登录验证以及登录后用户发送消息超过一百条后的重新登录验证，因此本应用程序中将该功能模块标示为可复用构件。

### 2.1.7 计时与写日志

由于应用程序中会多次重新登录，包括超时登录和将登录成功与失败的次数写入文件，因此本应用程序中将该功能模块标示为可复用构件。

### 2.1.8 配置信息

由于应用程序中会将消息发送限制条件与数据库的名称和密码进行配置，因此本应用程序中将该功能模块标示为可复用构件。

### 2.1.9 许可证

由于应用程序中对消息进行了一定的限制，并每次要对发送的消息进行一定的处理，因此本应用程序中将这些信息写入许可证，将该功能模块标示为可复用构件。

### 2.1.10 消息发送

由于应用程序中涉及到server端与client端进行相互的消息发送，包括登录消息及普通消息，因此本应用程序中将消息发送模块标示为可复用构件。

## 2.2 可复用构件列表

|  |  |
| --- | --- |
| 身份验证构件 | Authentication |
| 用户登录验证构件 | Verification |
| 获得配置信息构件 | GetConfiguration |
| 数据库构件 | DataSource |
| 许可证构件 | License |
| 消息计数构件 | MessageCount |
| 消息构件 | Message |
| 消息发送构件 | MessageDeparturer |
| 计数与写日志构件 | CheckCount |
| IO写操作构件 | IOLog |
| 计时与写日志构件 | Logger |

# 3. 可复用构件

## 3.1 Authentication构件

### 3.1.1 身份验证 - Authentication

(1) 构件介绍

这一构件的主要功能是对用户登录身份进行验证，该构件依赖于数据库构件和消息构件。

(2) 功能说明

用户可以使用该构件实现对登录用户身份的验证，该构件确保客户端用户以合法的身份与密码登录系统。

(3) 接口说明

1）getUsername()方法：

作用描述：

该方法用于获取用户的登录名。

使用方法：

对于一个实例化的Authentication对象，可以直接调用该方法，该方法没有参数，返回值为string类型，代表用户的登录名，形如：

String username = authentication.getUsername();

注意事项：

该方法返回一个string类型的参数，代表用户名，获取有效用户名首先需要确保用户已经登录，即用户已经通过该构件的下述authentication方法进行身份的验证。

2）getPassword()方法：

作用描述：

该方法用于获取用户的密码。

使用方法：

对于一个实例化的Authentication对象，可以直接调用该方法，该方法没有参数，返回值为string类型，代表用户的密码，形如：

String password = authentication.getPassword();

注意事项：

该方法返回一个string类型的参数，代表密码，获取有效密码首先需要确保用户已经登录，即用户已经通过该构件的下述authentication方法进行身份的验证。

3）authenticate(DataSource dataSource, String username, String password)方法：

作用描述：

该方法用于通过数据库查询得到用户登录的身份是否有效。

使用方法：

对于一个实例化的Authentication对象，可以直接调用该方法，该方法有三个输入参数，分别代表数据源、用户名、密码，输出时，若密码与数据库中信息匹配，则返回true，否则就返回false，形如：

boolean pswAuthenticate = authentication.authenticate(dataSource, “user1”, “123456”);

注意事项：

该方法只能判断用户是否通过了身份验证，但是无法判断未通过身份验证的具体原因。

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。

### 3.1.2 用户登录验证 － Verification

(1) 构件介绍

这一构件的主要功能是对于服务端用户登录模块功能的集成，包括用户登录的验证以及用户登录统计的日志记录，该构件依赖于数据库构件、消息构件以及计数与写日志构件。

(2) 功能说明

用户可以使用该构件实现用户登录验证以及客户端用户登录统计日志记录等与客户端用户登录有关的功能，该构件会确保客户端用户以合法的身份与密码登录系统。

(3) 接口说明

1）getUsername()方法：

作用描述：

该方法用于获取用户的登录名。

使用方法：

对于一个实例化的Verification对象，可以直接调用该方法，该方法没有参数，返回值为string类型，代表用户的登录名，形如：

String username = verification.getUsername();

注意事项：

该方法返回一个string类型的参数，代表用户名，获取有效用户名首先需要确保用户已经登录，即已经通过该构件的下述login方法登录系统。

2）getPassword()方法：

作用描述：

该方法用于获取用户的密码。

使用方法：

对于一个实例化的Verification对象，可以直接调用该方法，该方法没有参数，返回值为string类型，代表用户的密码，形如：

String password = verification.getPassword();

注意事项：

该方法返回一个string类型的参数，代表密码，获取有效密码首先需要确保用户已经登录，即已经通过该构件的下述login方法登录系统。

3）login(BufferedReader in, PrintWriter out, DataSource dataSource, Logger logger, String valid\_login\_per\_min, String invalid\_login\_per\_min, long threadId)方法：

作用描述：

该方法用于通过数据库查询得到用户登录的确认以及用户登录统计的日志记录。

使用方法：

对于一个实例化的Verification对象，可以直接调用该方法，该方法有多个参数，具体如下：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| dataSource | DataSource | 数据库构件，用于获取数据库连接以及密码查询等数据库操作。 |
| logger | Logger | 计数与写日志构件，用于指定字段的计数以及将计数结果写入日志等操作。 |
| valid\_login\_per\_min | String | 统计每分钟合法登录的计数字段。 |
| invalid\_login\_per\_min | String | 统计每分钟非法登录的计数字段。 |
| threadId | long | 线程标识符，标示登录用户在服务端所属线程。 |

使用方式形如：

verification.login(in, out, dataSource, logger, valid\_login\_per\_min, invalid\_login\_per\_min, this.getId());

注意事项：

该方法会持续，直到客户端用户输入正确的用户名和密码为止，在此过程中会持续对登录进行统计并计入日志。

4）csLogin(BufferedReader in, PrintWriter out, DataSource dataSource, Logger logger, String valid\_login\_per\_min, String invalid\_login\_per\_min)方法：

作用描述：

该方法用于通过数据库查询得到用户登录的确认以及用户登录统计的日志记录。

使用方法：

对于一个实例化的Verification对象，可以直接调用该方法，该方法有多个参数，具体如下：

|  |  |  |
| --- | --- | --- |
| 参数 | 类型 | 含义 |
| dataSource | DataSource | 数据库构件，用于获取数据库连接以及密码查询等数据库操作。 |
| logger | Logger | 计数与写日志构件，用于指定字段的计数以及将计数结果写入日志等操作。 |
| valid\_login\_per\_min | String | 统计每分钟合法登录的计数字段。 |
| invalid\_login\_per\_min | String | 统计每分钟非法登录的计数字段。 |

使用方式形如：

verification.csLogin(in, out, dataSource, logger, valid\_login\_per\_min, invalid\_login\_per\_min);

注意事项：

该方法会持续，直到客户端用户输入正确的用户名和密码为止，在此过程中会持续对登录进行统计并计入日志。

## 3.2 CM构件

### 3.2.1 获得配置信息 － GetConfiguration

(1) 构件介绍

这一构件的主要功能是通过读写文件来对配置信息进行操作，为服务端与客户端的连接、消息发送限制及数据库中存储用户名和密码提供相关信息。

(2) 功能说明

用户可以使用该构件实现对配置文件的读写。

(3) 接口说明

1）GetConfiguration()方法：

作用描述：

实例化获取到的配置文件的对象。

使用方法：

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

GetConfiguration getConfig = new GetConfiguration();

Private String SERVER\_IP = getConfig.getSERVER\_IP();

注意事项：

此方法是一个构造函数，在调用该方法前必须先有一个非空的配置文件。

2）ReadJSONFile(String Path)方法：

作用描述：

该方法读取配置文件，将其内容以字符串的形式返回。

使用方法：

传入一个配置文件的路径，返回该配置文件的内容，形如：

Sring content = config.ReadJSONFile(“D:\\”);

注意事项：

必须先有一个实际存在的对应路径的配置文件。

3）writeJSONFile(String path,String[] key,String[] value)方法：

作用描述：

该方法用于将指定的配置信息写入指定路径下的配置文件中。

使用方法：

传入文件路径及将要写入的string数组中的配置信息，形如：

getConfig.writeJSONFile(“D:\\”, key, value);

注意事项：

写入配置信息的配置文件的路径必须有效，且key与value需要先进行初始化。

4）getStringByKey(String key)方法和getIntByKey(String key)方法：

作用描述：

为配置管理提供查询接口，根据key值查询对应的value，返回值分别为字符串类型和整型。

使用方法：

对于实例化的JSON对象，可以直接调用该方法，输入key值，可以获取到对应的value值，该value值可以是字符串类型或是整型，形如：

String server\_ip = GetConfig.getStringByKey(“SERVER\_IP”);

或Int max\_msg\_per\_sec = GetConfig.getIntByKey(“MAX\_MESSAGE\_PER\_SECOND”);

5）load()方法：

作用描述：

动态加载配置信息。

使用方法：

通过ReadJSONFile()方法，将读取到的配置文件内容存储到一个JSON对象当中，并可对mutable的JSON对象进行修改，形如：

getConfig.load();

注意事项：

1. 只有具有可更改权限的内容才可进行修改，immutable的内容只是可读的。
2. 可更改的信息分别为MAX\_MESSAGE\_PER\_SECOND和MAX\_MESSAGE\_FOR\_TOTAL。

6）loadData()方法：

作用描述：

每30秒动态加载一次配置信息。

使用方法：

创建计时器，每30秒就在LoadFileTimerTask类中进行一次load操作，形如：

getConfig.loadData();

7）getSERVER\_IP()方法：

作用描述：

获取服务器的IP地址。

使用方法：

对于一个实例化的GetConfiguration对象，可以直接调用该方法，该方法没有输入参数，返回值为string类型的服务器IP地址，形如：

String ip = getConfig.getSERVER\_IP();

注意事项：

SERVER\_IP需要先进行初始化。

8）getSERVER\_PORT()方法：

作用描述：

获取服务器的端口号。

使用方法：

对于一个实例化的GetConfiguration对象，可以直接调用该方法，该方法没有输入参数，返回值为string类型的服务器端口号，形如：

String port= getConfig.getSERVER\_PORT();

注意事项：

SERVER\_PORT需要先进行初始化。

9）getMAX\_MESSAGE\_PER\_SECOND()方法：

作用描述：

获取每分钟可以发送的最大消息数。

使用方法：

对于一个实例化的GetConfiguration对象，可以直接调用该方法，该方法没有输入参数，返回值为int类型的MAX\_MESSAGE\_PER\_SECOND，形如：

int max\_msg\_per\_second = getConfig.getMAX\_MESSAGE\_PER\_SECOND();

注意事项：

MAX\_MESSAGE\_PER\_SECOND需要先进行初始化。

10）getMAX\_MESSAGE\_FOR\_TOTAL()方法：

作用描述：

获取可以进行收发的最大消息数。

使用方法：

对于一个实例化的GetConfiguration对象，可以直接调用该方法，该方法没有输入参数，返回值为int类型的MAX\_MESSAGE\_FOR\_TOTAL，形如：

int max\_msg\_for\_total= getConfig.getMAX\_MESSAGE\_FOR\_TOTAL();

注意事项：

MAX\_MESSAGE\_FOR\_TOTAL需要先进行初始化。

11）getDBUSER()方法：

作用描述：

获取数据库名称。

使用方法：

对于一个实例化的GetConfiguration对象，可以直接调用该方法，该方法没有输入参数，返回值为string类型的数据库名称，形如：

string DB\_User = getConfig.getDBUSER();

注意事项：

DBUSERD需要先进行初始化。

12）getDBPW()方法：

作用描述：

获取数据库名称。

使用方法：

对于一个实例化的GetConfiguration对象，可以直接调用该方法，该方法没有输入参数，返回值为string类型的数据库密码，形如：

string DB\_PW = getConfig.getDBPW();

注意事项：

DBPW需要先进行初始化。

## 3.3 DataSource构件

### 3.3.1 数据库 － DataSource

(1) 构件介绍

这一构件的主要功能是对于数据库的查找操作，包括密码查找等基本数据库查找功能。

(2) 功能说明

用户可以根据自己的实际需要选择不同的数据库，并配置好数据库的用户名和密码，即可通过该构件连接到数据库。

使用密码查找功能需要在数据库中建立tb\_user表，表中包含username及password两个属性，配置好相应数据库表后即可使用该构件提供的密码查找功能。

(3) 接口说明

1）DataSource(BufferedReader in, PrintWriter out)方法：

作用描述：

对数据流进行读写。

使用方法：

在实例化DataSource时，输入参数in和out，无返回值，形如：

DataSource dataSrc = new DataSouce(in, out);

注意事项：

In和out的类型必须分别为BufferedReader和PrintWriter，且需要事先进行实例化。

2）DataSource(String dbuser, String dbpw)方法：

作用描述：

该方法实质为构造函数，用以构建数据库的连接。

使用方法：

使用该方法构建数据库的连接需要传递两个string类型的参数，第一个参数代表数据库用户名，第二个参数代表数据库密码，形如：

DataSource dataSource = new DataSource(“root”, “12345”);

注意事项：

该方法尽可以配置用户名与密码，默认为mysql数据库，ip为localhost，端口号3306，schema为”reusable”。

3）getType()方法：

作用描述：

获取当前对象的资源类型。

使用方法：

对于实例化的DataSource对象，可以直接调用该方法，无输入参数，返回值为当前对象的soueceType，形如：

String type = source.getType();

注意事项：

应先实例化DataSource。

4）getPasswordDB(String username)方法：

作用描述：

该方法用于根据指定的用户名在相应的数据库中查找用户名对应的密码，即执行相应的SQL语句对数据库中对应的表执行select操作。

使用方法：

对于一个实例化的DataSource对象，可以直接调用该方法，该方法有一个string类型的参数，即想要查找的用户名，返回类型为string 类型，即输入的用户名对应的密码，形如：

DataSource dataSource = new DataSource();

String username = “Tom”;

String password = dataSource.getPassword(username);

注意事项：

A. 该方法需要确保数据库中存在tb\_user表，并且表中存在username以及password属性。

B. 若输入的用户在数据库tb\_user表中存在，则返回该用户对应的密码，若输入的用户不存在，则返回空字符串。

5）getPasswordResponse(String msg)方法：

作用描述：

每当用户输入密码后，对该密码的匹配与否进行答复。

使用方法：

对于实例化的DataSource对象，可以直接调用该方法，输入参数为string类型的想要答复的信息内容，返回值为string类型的response。

## 3.4 License构件

### 3.4.1 许可证 － License

(1) 构件介绍

这一构件的主要功能是对消息的收发设置限制条件。

(2) 功能说明

用户可以使用该构件来设置每分钟消息收发最大数目、登录后收发的消息总数、在计数器规定范围内收发的消息数目等。

(3) 接口说明

1）License(int max\_msg\_in\_second, int max\_num\_message, int firstTime, int period)方法：

作用描述：

这个方法标识了许可证中所包含的信息内容，其中有每分钟消息最大值、总消息最大值、计时器开启时间、计时器周期。

使用方法：

对于一个实例化的License对象，可以直接调用该方法，该方法输入四个int类型的参数，没有返回值，形如：

License license = new License(5, 100, 0, 60000);

注意事项：

在实例化License对象的时候，应传入四个整型的参数。

2）setTime(int firstTime, int period)方法：

作用描述：

该方法用于设置消息计数的开始时间和周期长度。

使用方法：

对于一个实例化的License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messageCount开始执行的时间和其执行周期长度，没有返回值，形如：

license.setTime(0, 60000);

3）commence()方法：

作用描述：

该方法用于开启消息计数的计时器。

使用方法：

对于一个实例化的License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：

license.commence();

注意事项：

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

4）cancel()方法：

作用描述：

关闭计时器。

使用方法：

对于一个实例化的License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：

license.cancel();

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）setMax(int a)方法：

作用描述：

该方法用于设置总收发消息的最大数。

使用方法：

对于一个实例化的License对象，可以直接调用该方法，该方法的输入一个int类型的参数，即可赋值，没有返回值，形如：

license.setMax(100);

注意事项：

该方法的输入参数必须为非负整数。

6）setPerSec(int a)方法：

作用描述：

该方法用于设置每分钟收发消息最大数。

使用方法：

对于一个实例化的License对象，可以直接调用该方法，该方法的输入一个int类型的参数，即可赋值，没有返回值，形如：

license.setPerSec(5);

注意事项：

该方法的输入参数必须为非负整数。

7）increaseMsg()方法：

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License对象，可以直接调用该方法，该方法的无输入参数，也没有返回值，形如：

license.increaseMsg();

注意事项：

必须先对MAX\_MSG\_IN\_SECOND和MAX\_NUM\_MESSAGE进行初始化。

8）reset()方法：

作用描述：

该方法将消息数重置为0。

使用方法：

对于一个实例化的License对象，可以直接调用该方法，该方法的没有输入参数和返回值，形如：

license.reset();

注意事项：

必须先对messageCount进行实例化。

9）checkMsgInSecond()方法：

作用描述：

该方法判断消息计数器每分钟获取的消息数是否超过了MAX\_MSG\_IN\_SECOND。

使用方法：

对于一个实例化的License对象，可以直接调用该方法，该方法的没有输入参数，返回值为布尔值，形如：

boolean msgInSecond = license.checkMsgInSecond();

注意事项：

必须先对messageCount进行实例化。

10）checkTotalMsg()方法：

作用描述：

该方法断消息计数器获取的消息总数是否超过了MAX\_NUM\_MESSAGE。

使用方法：

对于一个实例化的License对象，可以直接调用该方法，该方法的没有输入参数，返回值为布尔值，形如：

boolean msgTotal = license.checkMsgInSecond();

注意事项：

必须先对messageCount进行实例化。

### 3.4.2 消息计数 － MessageCount

(1) 构件介绍

这一构件的主要功能是对于消息的计数功能，包括对于每分钟消息的计数以及对于消息总数的计数。

(2) 功能说明

用户可以使用该构件实现对于消息的计数功能，计数包括每分钟消息的计数以及消息总数的计数。

(3) 接口说明

1）MessageCount()方法：

作用描述：

将每秒钟的消息数和总消息数都初始化为0。

使用方法：

对于一个实例化的DataSource对象，可以直接调用该方法，该方法没有参数和返回值，形如：

MessageCount messageCount = new MessageCount();

注意事项：

此方法在实例化MessageCount对象时被调用。

2）getMsgInSecond()方法：

作用描述：

该方法用于获取每分钟消息的计数数额。

使用方法：

对于一个实例化的DataSource对象，可以直接调用该方法，该方法没有参数，返回值为int类型，代表每分钟消息的总数，形如：

messageCount.getMsgInSecond();

注意事项：

该方法返回一个int类型的参数，代表每分钟消息的计数数值。

3）getTotalMsg()方法：

作用描述：

该方法用于获取消息的计数数额。

使用方法：

对于一个实例化的DataSource对象，可以直接调用该方法，该方法没有参数，返回值为int类型，代表消息的计数总额，形如：

messageCount.getTotalMsg();

注意事项：

该方法返回一个int类型的参数，代表消息的计数数值。

4）increaseMsg()方法：

作用描述：

该方法用于将消息计数加1，包括每分钟消息计数与总消息计数。

使用方法：

对于一个实例化的DataSource对象，可以直接调用该方法，该方法没有参数，也没有返回值，形如：

messageCount.increaseMsg();

注意事项：

该方法会对每分钟消息计数与总消息计数均加1。

5）reset()方法：

作用描述：

该方法用于将消息计数归零，包括每分钟消息计数与总消息计数。

使用方法：

对于一个实例化的DataSource对象，可以直接调用该方法，该方法没有参数，也没有返回值，形如：

messageCount.reset();

注意事项：

该方法会对每分钟消息计数与总消息计数均归零。

6）run()方法：

作用描述：

该方法继承于TimerTask，可以用在定时器中，主要功能为将每分钟消息计数归零。

使用方法：

该方法一般用于定时器中，启用定时器后，可以实现每单位时间每分钟消息计数归零，形如：

timer.schedule(messageCount, 0, 1000);

注意事项：

该方法一般用于定时器，如上所示，也可以单独使用，单独使用时该方法没有参数也没有返回值。

## 3.5 MessageUtils构件

### 3.5.1 消息 － Message

(1) 构件介绍

这一构件的主要功能是构件与读取客户端与服务端通信时传递的消息，本质上是对于JSON生成与解析的方法的封装，并添加了线程标识符，用以表示消息的所属关系。

(2) 功能说明

用户可以通过该构件实现对消息的构造、消息所属线程的指定、消息内容的添加以及消息内容的解析等操作。

用户可以通过构造方法 Message指定消息所属的线程，同时也可以通过构造方法传递JSON格式的字符串用来生成消息的内容。用户也可以通过setValue方法添加消息内容，setValue方法通过重载可以接受不同类型的参数。用户可以通过getValue方法实现消息的解析。getOwner方法用于获取消息所属的线程，toString用于将JSONObject对象转换为JSON格式的字符串。

(3) 接口说明

1. Message(String msg, long ownerThread)方法：

作用简述：

该方法实质为方法类的构造函数，通过该方法可以实例化Message对象，并且可以指定消息所属的线程以及消息的内容。

使用方法：

利用该方法实例化Message方法类，其中需要传递两个参数，第一个参数为string类型，为JSON格式的字符串，即消息的内容，第二个参数为long类型，为消息所属线程的标识符，形如：

Message msg = new Message("{}", this.getId());

注意事项：

该方法中指定的消息的内容可以为空的JSON，即形如”{}”的字符串，但该参数不可为空。

2）getOwner()方法：

作用描述：

该方法用于获取消息所属消息的标识符，通过调用一个实例化的Message对象的该方法，即可获得这个实例化的Message对象的标识符。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法无需传递任何参数，返回类型为long型，即线程标识符，形如：

Long ownerThread = msg.getOwner();

注意事项：

该方法返回的是其所属消息的线程的标识符，该标识符在上述构造函数中指定。

3）getValue(String key)方法：

作用描述：

该方法用于获取消息内容，通过传入key值，可以获取到对应的value值。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法唯一的参数是string类型的key值，返回类型为string类型的value值，形如：

String sValue = msg.getValue(“target”);

注意事项：

尽管上述setValue方法实现了重载，value类型可以接受不同类型的参数，包括string类型与long类型，但是本方法返回的value类型仅为string类型。

4）setValue(String key, String value)， setValue(String key, long value)方法：

作用描述：

该方法用于添加消息内容，通过传递两个参数作为key和value值，可以将这个键值对加入到消息内容中。该方法进行重载，支持不同类型的value值，包括string类型和long类型。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法有两个参数，第一个参数为string类型的key，第二个参数为string类型或者long类型的value，调用该方法后，传入的键值对会添加到实例化的消息的内容中，形如：

msg.setValue("event", “login");

注意事项：

该方法通过重载可以接受两种不同类型的value值，即第二个参数可以接受不同类型的值，包括string类型与long类型。

5） toString()方法：

作用描述：

该方法用于将JSONObject对象转换为JSON格式的字符串。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法没有输入参数，返回类型为string类型，即JSON格式的字符串，形如：

String jsonMessage = msg.toString();

注意事项：

该方法会将实例对象中的内容转换为JSON格式的字符串，转换后若继续通过setValue方法添加内容，则需要在此调用本方法才能得到添加后的JSON格式的字符串。

6）reset(String msg)方法：

作用描述：

实例化一个JSON对象，可以传入字符串类型的参数。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法有一个string类型的输入参数，没有返回值，形如：

String msg = msg.rest(“Hello World!”);

注意事项：

传入参数必须是string类型。

7）clear()方法：

作用描述：

实例化一个空的JSON字符串。

使用方法：

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

msg.clear()

注意事项：

必须要有一个实例化的Message方法类来调用clear()方法。

8）init(String queue\_name, String hostName)方法：

作用描述：

使用了消息中间件RabitMQ，来实例化一个factory，并设置hostName，利用factory创建connection，并为connection创建信道。这种设置可利用与但发送多接收的场景，保证消息发送的可靠性，不丢失消息，将消息持久化。还设置了当前的队列名，用于存放消息。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法有两个string类型的输入参数，第一个为消息队列的名称，第二个为建立连接的主机名，没有返回值，形如：

msg.init(“msgQueue”, “host1”)

注意事项：

A. 两个参数都是string类型。

B. RabbitMQ建议客户端线程之间不要共用Channel，至少要保证共用Channel的线程发送消息必须是串行的，但是建议尽量共用Connection。

9）bindTo(String exchangeName, String routerKey)方法：

作用描述：

现有的信道将转发器设置为direct类型， 将消息队列与一个特定的routerKey绑定，并要求二者完全匹配才可进行转发。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法有两个string类型的输入参数，第一个为交换机的名称，第二个是路由键的名称，没有返回值，形如：

msg.bindTo(“exchangeName”, “myRouter”)

注意事项：

两个输入参数均为string类型。

10）getQueueName()方法：

作用描述：

获取消息队列的名称。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法没有传入参数，返回值为当前Messge对象的消息队列的名称，形如：

String queueName = msg.getQueueName();

注意事项：

接收返回值的必须为string类型。

11）getChannel()

作用描述：

获取信道对象。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法没有传入参数，返回值为当前Meaage对象的消息队列的名称，形如：

Channel channel = msg.getChannel();

注意事项：

接收返回值的必须为channel类型。

12）publishToOne(String exchangeName, String routerKey)方法：

作用描述：

将信道中的消息只发送给匹配routerKey的特定接收端。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法有两个string类型的输入参数，第一个为交换机名称，第二个为路由键，形如：

msg.publishToOne(“myExchange”, “myRouter”);

注意事项：

A. 两个传入参数均为string类型。

B. 该方法的调用只在server端进行。

13）publishToAll(String exchangeName)方法：

作用描述：

将信道中的消息发送给交换机对应的所有接收方。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法有一个string类型的输入参数，代表交换机名称，形如：

msg.publishToAll(“myExchange”);

注意事项：

A. 传入参数为string类型。

B. 该方法的调用只在server端进行。

14）publishToOthers(String exchangeName, String routerKey)方法：

作用描述：

将信道中的消息发送给所有不匹配routerKey的接收端。

使用方法：

对于一个实例化的Message对象，可以直接调用该方法，该方法有两个string类型的输入参数，第一个为交换机名称，第二个为路由键，形如：

msg.publishToOthers(“myExchange”, “myRouter”);

注意事项：

A. 两个传入参数均为string类型。

B. 该方法的调用只在server端进行。

15）terminate()

作用描述：

关闭信道和连接。

使用方法：

对于一个实例化的Message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：

msg.terminate();

注意事项：

该方法只可以在server端进行调用，且在调用之后，所有与信道或连接相关的操作都无法继续进行。

### 3.5.2 消息发送 － MessageDeparturer

(1) 构件介绍

这一构件的主要功能是对消息进行消耗，即发送消息。

(2) 功能说明

用户可以使用该构件来设置消息的接收方和想要进行发送的消息。

(3) 接口说明

1）MessageConsumer(Message msg, PrintWriter out, Logger logger, String logKey)方法：

作用描述：

这个方法获取消息信道，同时为多个登录成功的用户发送消息。

使用方法：

对于一个实例化的MessageConsumer对象，可以直接调用该方法，该方法输入四个参数，分别代表要发送的消息、输出流、登录计数器、登录关键字，没有返回值，形如：

MessageConsumer msgConsumer = new MessageConsumer(“Hello World!”, printWriter, logger, “login\_success”);

注意事项：

在实例化MessageConsumer对象的时候，应传入四个参数。

2）sendMessage(String msg)方法：

作用描述：

该方法通过输出流来发送消息。

使用方法：

对于一个实例化的MessageConsumer对象，可以直接调用该方法，该方法有一个string类型的输入参数，代表要发送的消息，没有返回值，形如：

msgConsumer.sendMessage(“Hello World!”);

注意事项：

传入的参数必须为string类型。

3）handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body)方法：

作用描述：

当消费者接收到消息后会自动调用该方法，对消息数进行加一。

使用方法：

对于一个实例化的MessageConsumer对象，可以直接调用该方法，输入四个参数，返回值为空，形如：

msgConsumer.handleDelivery(consumerTag, envelop, properties, body);

注意事项：

消费者接受消息后会自动调用该方法，无需再特意进行。

4）MessageDeparturer(Message msg, PrintWriter out, Logger logger, String logKey)方法：

作用描述：

当server给消息的consumer发送消息之后，将消息从队列中清除。

使用方法：

对于一个实例化的MessageDeparturer对象，可以直接调用该方法，输入四个参数，返回值为空，形如：

MessageDeparturer msgDeparturer = MessageDeparturer(“Hello World!”, printWriter, logger, forwarded\_message);

注意事项：

注意四个参数的输入格式。

## 3.6 PM构件

### 3.6.1 计数与写日志 － CheckCount

(1) 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。

(2) 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。

(3) 接口说明

1）CheckCount(String logFilename)方法：

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)方法：

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)方法：

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)方法：

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）run()方法：

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。

### 3.6.2 IO写操作－IOLog

(1) 构件介绍

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。

(2) 功能说明

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。

(3) 接口说明

1）IOLog(String logAddr, boolean bAppend)方法：

作用简述：

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)方法：

作用简述：

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。

### 3.6.3 计时与写日志 － Logger

(1) 构件介绍

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。

(2) 功能说明

用户可以使用该构件实现对于登录次数的计数功能。

(3) 接口说明

1）Logger(String logFilename)方法：

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)方法：

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()方法：

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()方法：

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)方法：

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)方法：

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)方法：

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。

# 4．可复用构件Demo

## 4.1 Authentication构件

这一构件的主要功

## 4.2 CM构件

public class getConfigurationDemo extends ServerSocket{  
  
  
 public getConfigurationDemo(int SERVER\_PORT) throws IOException {  
 super(SERVER\_PORT);  
 // *TODO Auto-generated constructor stub* }  
  
 public void run() throws IOException{  
 try {  
 while(true){//监听客户端请求，启动线程处理  
 Socket socket = accept();  
 new ServerThread(socket);  
 }  
 }catch (Exception e) {  
 }finally{  
 close();  
 }  
 }  
  
 public class ServerThread extends Thread{  
 //将从Json获取配置信息  
 private int MAX\_MESSAGE\_PER\_SECOND;  
 private int MAX\_MESSAGE\_FOR\_TOTAL;  
 public GetConfiguration getConfiguration;  
 ServerThread(Socket socket){  
 getConfiguration = new GetConfiguration();  
 //动态配置，配置文件信息更改时将自动读入  
 getConfiguration.loadData();  
 //打印动态配置信息  
 System.out.println("MAX\_MESSAGE\_PER\_SECOND:"+getConfiguration.getMAX\_MESSAGE\_PER\_SECOND());  
 System.out.println("MAX\_MESSAGE\_FOR\_TOTAL:"+getConfiguration.getMAX\_MESSAGE\_FOR\_TOTAL());  
  
 }  
  
 @Override  
 public void run() {  
 //执行线程run方法  
 }  
 }  
  
}

## 4.3 DataSource构件

这一构件的主要功

## 4.4 License构件

class ServerThread {  
 private int MAX\_MESSAGE\_PER\_SECOND = 5;  
 private int MAX\_MESSAGE\_FOR\_TOTAL = 10;  
 private License license;  
  
 public void work() {  
 license = new License(1,1,1,1);  
 try {  
 // 设置指标。设置第一次执行延迟时间，以及以后每次执行间隔。  
 license.setMax(MAX\_MESSAGE\_PER\_SECOND, MAX\_MESSAGE\_FOR\_TOTAL);  
 license.setTime(0, 60000);  
 // 使用前重置License。  
 license.reset();  
 // 启动内部计时器。  
 license.commence();  
  
 String line = in.readLine();  
 while(true) {  
 // 检查每秒信息量是否超标。  
 if (license.checkMsgInSecond()) {  
 // 未超标，计数加一。  
 license.increaseMsg();  
 ...  
 } else {  
 ...  
 }  
 // 检查总信息量是否超标。注意取反。  
 if (!license.checkTotalMsg()) {  
 ...  
 // 完成工作后，重置计数。  
 license.reset();  
 }  
 }  
 line = in.readLine();  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 // 停止内部计时器。  
 license.cancel();  
 }  
}  
}

## 4.5 MessgaeUtils构件

class ServerThread extends Thread {  
 private Socket client;  
 private PrintWriter out;  
 private BufferedReader in;  
 public Message msg;  
 public MessageDeparturer messageDeparturer;  
 String exchangeName = "test";  
 String queueName;  
  
 public ServerThread(Socket s)throws IOException {  
 this.client = s;  
 out = new PrintWriter(client.getOutputStream(),true);  
 in = new BufferedReader(new InputStreamReader(client.getInputStream()));  
 start();  
 }  
  
 @Override  
 public void run() {  
 try {  
  
 // 以线程ID标示queue。  
 queueName = String.valueOf(this.getId());  
 // 初始化。  
 msg = new Message("{}", this.getId());  
 // 设置字段。  
 msg.setValue("username", username);  
 msg.setValue("target", "others");  
 // 绑定指定频道。  
 msg.init(queueName, "localhost");  
 msg.bindTo(exchangeName, queueName);  
 // 为这个message对象建立departurer，信息会由其自动分发。  
 messageDeparturer = new MessageDeparturer(msg, out, logger, forwarded\_msg);  
 msg.setValue("event", "logedin");  
 // 将信息分发给同一exchange中的其她所有人。  
 msg.publishToAll(exchangeName);  
  
 String line = in.readLine();  
 msg.reset(line);  
 while(!"logout".equals(msg.getValue("event"))) {  
 //查看在线用户列表  
 if ("showuser".equals(msg.getValue("event"))) {  
 out.println(listOnlineUsers());  
 } else if ("message".equals(msg.getValue("event"))) {  
 msg.setValue("username", username);  
 msg.setValue("event", "message");  
 msg.setValue("target", "others");  
 msg.publishToAll(exchangeName);  
 }  
 line = in.readLine();  
 msg.reset(line);  
 }  
 msg.setValue("target", "all");  
 msg.setValue("event", "quit");  
 msg.publishToAll(exchangeName);  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 client.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

## 4.6 PM构件

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();  
 }  
}