Web服务器测试工具Gulu——基于JSCH实现分布式进程管理&分布式文件管理工具

JSCH是Java Secure Channel的缩写，是一个SSH2的纯Java实现，允许用户连接到SSH服务器上并进行X11转发、SFTP文件传输、基于端口转发等等操作。

本篇博文将从分布式文件管理介绍、分布式进程管理介绍、鉴权设计及远程连接建立流程，这三个方面介绍如何基于JSCH实现分布式进程管理&分布式文件管理。

【作者】 公远/陈震（gongyuan.cz$taobao.com/ 100continue$sina.com）

【源码下载】

github地址：https://github.com/100Continue/Gulu--web-server-test-tools

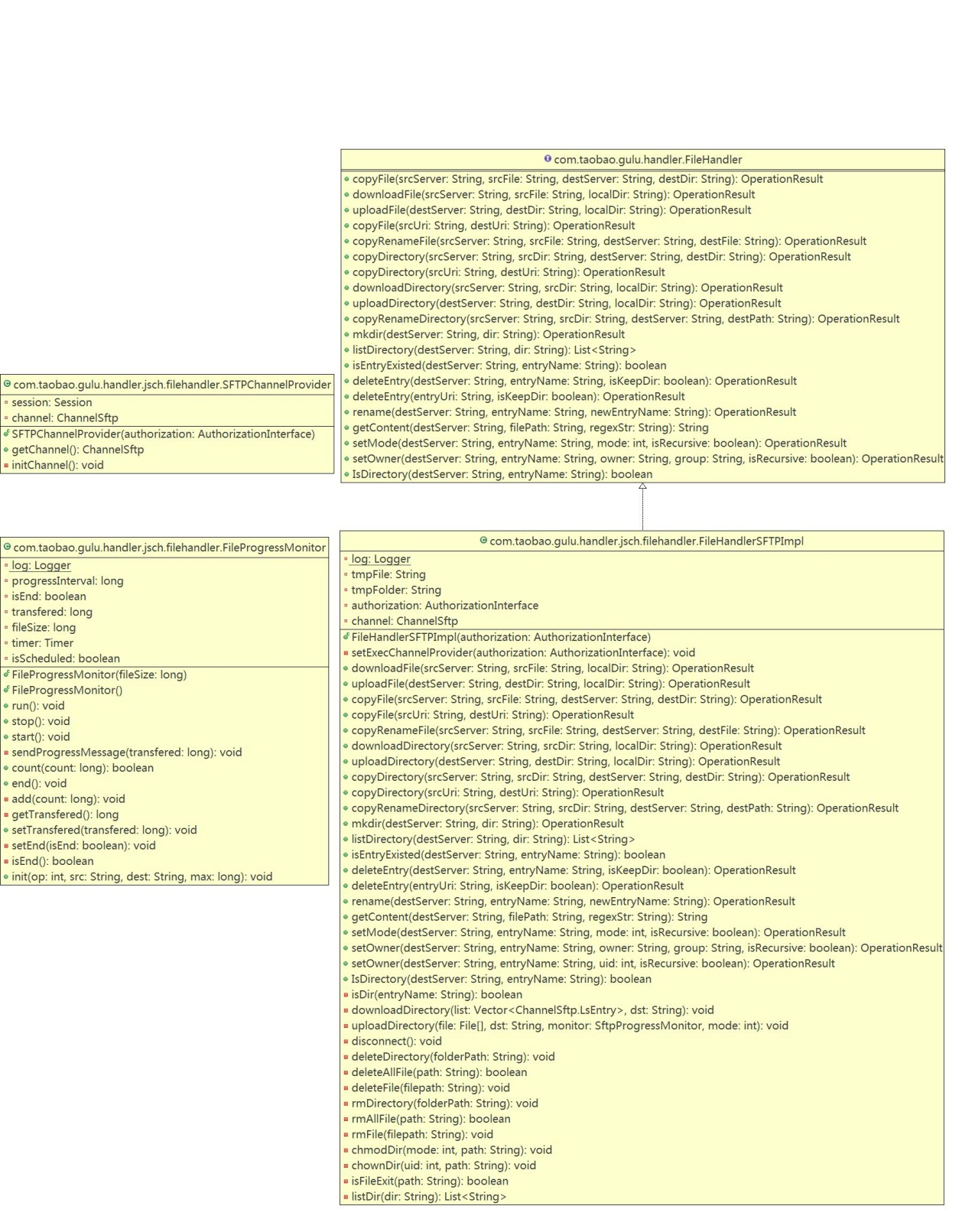
【扩展阅读】

web服务器测试工具Gulu——简介

【功能介绍】

1. 分布式文件管理介绍
   1. 需求：
      1. 设计要求：
         1. 支持不同操作系统之间的文件管理；
         2. 支持文件上传下载的进度展现；
         3. 接口设计简洁且易扩展维护，代码应用快捷方便；
      2. 功能要求：
         1. 支持两台远程服务器之间的文件/文件夹互相拷贝及重命名保存
         2. 支持从远程服务器上下载文件/文件夹到本地服务器的指定地址
         3. 支持将本地服务器的文件/文件夹上传到远程服务器的指定地址
         4. 支持删除远程服务器上的文件/文件夹
         5. 支持获取远程文件内符合特定关键字或正则匹配的文本数据
         6. 支持验证文件路径是否是目录
         7. 支持验证文件路径的文件/文件夹是否存在
         8. 支持将文件路径内的所有文件名输出成list对象
         9. 支持在远程服务器上创建目录
         10. 支持在远程服务器上对文件/文件夹进行重命名
         11. 支持在远程服务器上设置文件/文件夹的文件属性及权限
         12. 支持在远程服务器上设置文件/文件夹的用户组及归属用户
   2. 具体实现：

结合上述需求及保障可扩展性，可维护性的要求。分布式文件管理设计参见下幅UML图：



FileHandler作为分布式文件管理的接口类，不仅具备接口设计简洁，代码应用快捷的特点，而且具备可扩展可维护性，方便后续扩展其他实现方式（如STAF、手动设置信任关系等）；

FileHandlerSFTPImpl就是采用JSCH的SFTP方式对FileHandler接口进行实现。其中为了实现不同操作系统之间的文件/文件夹互相拷贝操作，将命令发起机器定义为中转站。如拷贝A机器的文件/文件夹到B机器上，则默认先将A机器的文件/文件夹下载到命令发起机器（中转站）的临时文件/文件夹内，再从命令发起机器（中转站）将临时文件/文件夹的A机器数据上传到B机器的相应位置；

FileProgressMonitor主要用于展现文件上传下载的进度展现，方便用户了解当前操作的进度；

SFTPChannelProvider主要用于鉴权、通讯建立、SFTP通道建立，该实现细节将在第三部分鉴权设计及远程连接建立流程中做介绍；

* 1. 代码应用：

// 采用提供用户名及加密密码的方式进行鉴权（第三部分介绍其他几种鉴权方式）

PasswordAuthorization passwords = new PasswordAuthorization(

“username”, "ab6sA7Eo7InqWEnnnFFqw788FvvOCZ7H");

// FileHandler 对象

FileHandlerSFTPImpl fileHandler = new FileHandlerSFTPImpl(

passwords);

// 两台服务器之间文件拷贝及重命名

OperationResult result = fileHandler.copyFile(“xx.xxx.x.29”,

"/home/admin/log.log", “xx.xxx.x.31”,

"/home/admin/copyfile\_log");

// 两台服务器之间文件夹拷贝及重命名

OperationResult result = fileHandler.copyDirectory(“xx.xxx.x.29”,

"/home/admin/sql/", “xx.xxx.x.31”,

"/home/admin/copyfolder");

// List对象保存文件夹内文件列表

List list = fileHandler.listDirectory(“xx.xxx.x.29”,

"/home/admin/sql/");

// 文件夹重命名

fileHandler.rename(“xx.xxx.x.29”,

"/home/admin/copyfolder", "/home/admin/ttttdd/");

// 修改文件夹使用权限

fileHandler.setMode(“xx.xxx.x.29”,

"/home/admin/xx.jpg", 7777, true);

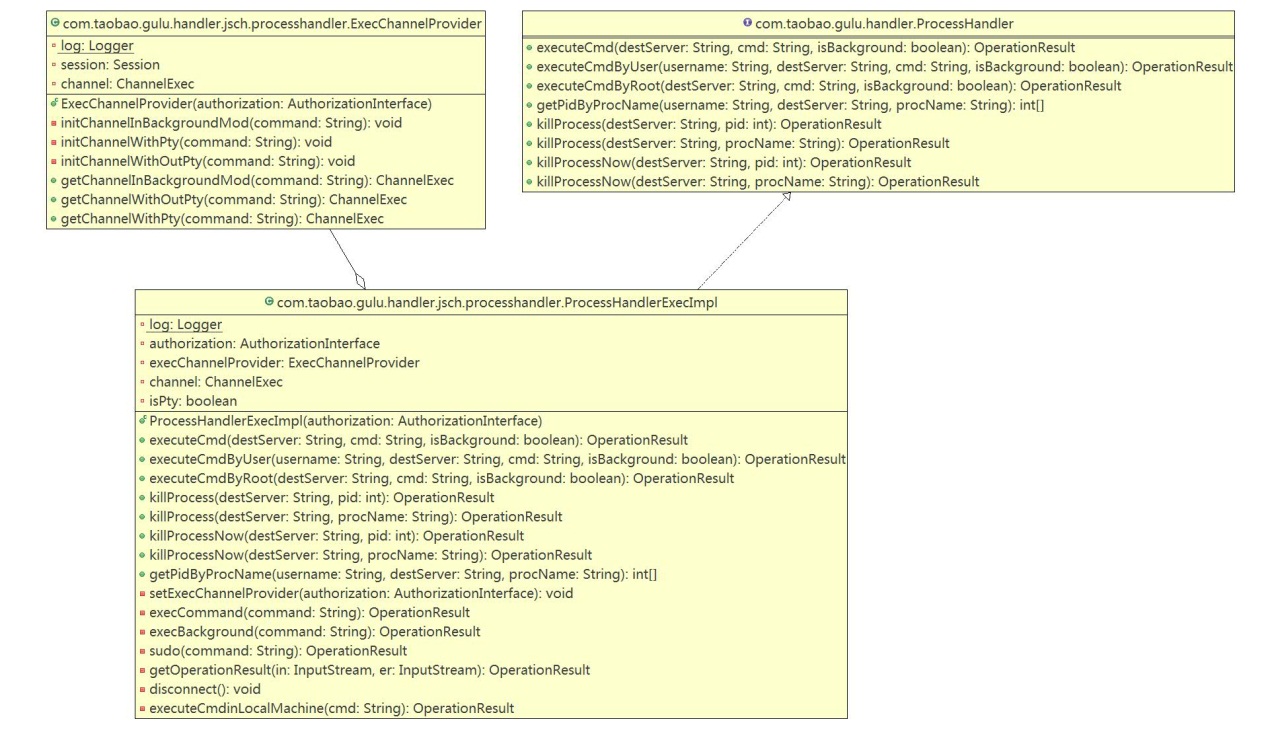
// 修改文件夹所有者

fileHandler.setOwner(“xx.xxx.x.29”,

"/home/admin/ttttdd", 0, true);

1. 分布式进程管理介绍
   1. 需求：
      1. 设计要求：
         1. 支持对所有能够建立SSH连接的机器进行远程进程管理；
         2. 接口设计简洁且易扩展维护，代码应用快捷方便
      2. 功能要求：
         1. 支持远程命令执行及指定用户的远程命令执行（如root权限、指定user的用户权限）；
         2. 支持通过进程名获取远程机器的进程ID列表；
         3. 支持通过进程名或进程ID来KILL进程；
   2. 具体实现：

结合上述需求及保障可扩展性，可维护性的要求。分布式进程管理设计参见下幅UML图：



ProcessHandler作为分布式进程管理的接口类，不仅具备接口设计简洁，代码应用快捷的特点，而且具备可扩展可维护性，方便后续扩展其他实现方式（如STAF、手动设置信任关系等）；

ProcessHandlerExecImpl就是采用JSCH的EXEC通道连接方式对ProcessHandler接口进行实现。支持本地机器及远程机器的进程管理；

ExecChannelProvider主要用于鉴权、通讯建立、EXEC通道建立，该实现细节将在第三部分鉴权设计及远程连接建立流程中做介绍；

* 1. 代码应用：

// 采用提供用户名及加密密码的方式进行鉴权（第三部分介绍其他几种鉴权方式）

PasswordAuthorization passwords = new PasswordAuthorization(

“username”, "ab6sA7Eo7InqWEnnnFFqw788FvvOCZ7H");

// processhandler 对象

ProcessHandlerExecImpl processhandler = new ProcessHandlerExecImpl(

passwords);

// 所提供用户名的用户权限下的命令执行

OperationResult result = processhandler.executeCmd(“xx.xxx.x.29”,

"./sleep.sh ", true);

// root权限下的命令执行

OperationResult result = processhandler.executeCmdByRoot(“xx.xxx.x.29”, "ls",

false);

// 指定用户权限的命令执行

OperationResult result = processhandler.executeCmdByUser("admin", “xx.xxx.x.29”,

"ls", false);

// 通过进程名及进程所属用户信息获取该进程ID列表

int[] id = processhandler.getPidByProcName(“username”, “xx.xxx.x.29”,

"./sleep.sh");

// 通过进程ID或进程名杀进程

OperationResult result = processhandler.killProcess(“xx.xxx.x.29”, "sleep.sh");

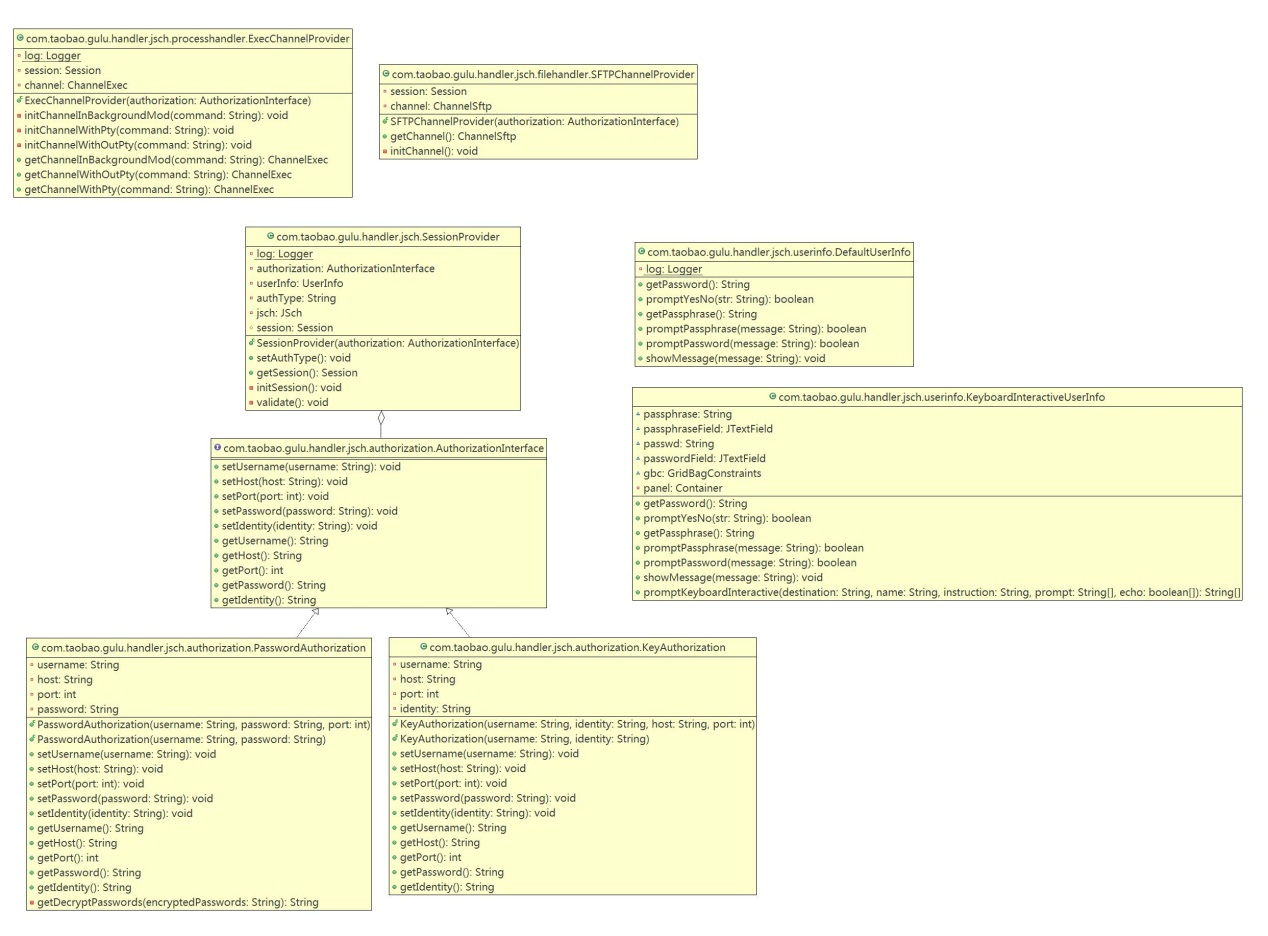
OperationResult result = processhandler.killProcess(“xx.xxx.x.29”, 10294);

OperationResult result = processhandler.killProcessNow(“xx.xxx.x.29”, "sleep.sh");

OperationResult result = processhandler.killProcessNow (“xx.xxx.x.29”, 10294);

1. 鉴权设计及远程连接建立流程
   1. 需求：
      1. 支持3种认证机制:
         1. 基于用户名密码的认证机制；
         2. 基于DSA/RSA的认证机制；
         3. 交互输入用户名密码的认证机制；
      2. 鉴权及连接建立过程对用户透明，保障代码编写便捷；
      3. 接口设计易于扩展与维护，方便后续实现其他认证机制；
   2. 具体实现

结合上述需求及保障可扩展性，可维护性的要求。分布式进程管理设计参见下幅UML图：



AuthorizationInterface作为鉴权的接口支持1. 基于用户名密码的认证机制；2. 基于DSA/RSA的认证机制；3. 交互输入用户名密码的认证机制；同时由于接口的易扩展性，后续可在该接口上扩展其他鉴权机制；

PasswordAuthorization和KeyAuthorization分别对应基于用户名密码的认证机制和基于DSA/RSA认证机制的接口实现；用于存储不同认证机制所需要的鉴权数据；

SessionProvider根据所传入的鉴权数据对象，区分用户采用哪种鉴权认证机制，并根据相应的鉴权认证机制进行数据校验和生成对应的Session对象与远程机器连接鉴权。如果鉴权机制为用户名密码的认证机制，则设置UserInfo对象为DefaultUserInfo类型；如果鉴权机制为交互输入用户密码的认证机制，则设置UserInfo对象为KeyboardInteractiveUserInfo，该对象将提供与用户交互的输入窗口；

最后将生成的Session对象传递给ExecChannelProvider用于生成应用于分布式进程管理的EXEC通道，通过该通道可以进行分布式进程管理操作；或者将生成的Session对象传递给SFTPChannelProvider用于生成应用于分布式文件管理的SFTP通道，通过该通道可以进行分布式文件管理操作。

注：目前JSCH只支持OpenSSH和SSH 1生成的private/public key所以当你们发现不能通过private/public key认证的时候，不是JSCH的问题，而是不能识别的问题；

* 1. 代码应用

// 基于用户名密码的认证机制生成进程管理和文件管理对象

PasswordAuthorization passwords = new PasswordAuthorization(

“username”, "ab6sA7Eo7InqWEnnnFFqw788FvvOCZ7H");

ProcessHandlerExecImpl processHandler = new ProcessHandlerExecImpl(

passwords);

FileHandlerSFTPImpl fileHandler = new FileHandlerSFTPImpl(

passwords);

// 基于DSA/RSA的认证机制生成进程管理和文件管理对象

KeyAuthorization keyAuth = new KeyAuthorization (

“username”, "id\_dsa\_filepath");

ProcessHandlerExecImpl processHandler = new ProcessHandlerExecImpl(

keyAuth);

FileHandlerSFTPImpl fileHandler = new FileHandlerSFTPImpl(

keyAuth);

// 交互输入用户名密码的认证机制生成进程管理和文件管理对象

PasswordAuthorization passwords = new PasswordAuthorization(

“username”, ""); //不传入密码，将采用交互输入方式进行认证

ProcessHandlerExecImpl processHandler = new ProcessHandlerExecImpl(

passwords);

FileHandlerSFTPImpl fileHandler = new FileHandlerSFTPImpl(

passwords);

// 对本地进程进行管理的设置方式

PasswordAuthorization passwords = new PasswordAuthorization(

"", ""); //不传入用户名密码，将允许对本地进行进行管理

ProcessHandlerExecImpl processHandler = new ProcessHandlerExecImpl(

passwords);

OperationResult result = processHandler.executeCmd("",

"ipconfig /all", false);