Atitit 同步获取阻塞式命令行输出

费阻塞式样命令行执行完毕后退出了，可以直接获取其输出

InputStream is = **new** StreamGobbler(session.getStderr());// 获得标准输出流

BufferedReader brs = **new** BufferedReader(**new** InputStreamReader(is));

**for** (String line = brs.readLine(); line != **null**; line = brs.readLine()) {

result.add(line);

}

但比如redis这类祖塞师 ，就不能这么做了

需要式样异步线程读取stdstream，然后输出，，可以配合sleep模拟同步模式

/redisJsDriver/src/sshdemo.java

**import** ch.ethz.ssh2.Connection;

**import** ch.ethz.ssh2.Session;

**import** ch.ethz.ssh2.StreamGobbler;

**public** **class** sshdemo {

**public** **static** **void** main(String[] args) **throws** Exception {

Connection con = **new** Connection("10 11", 22);

con.connect();

**boolean** isAuthed;

isAuthed = con.authenticateWithPassword("root", " 21");

Session session = con.openSession();

String redisCmd = "redis-cli -h .11 -p 63790 -a ttredis$ ";

System.***out***.println(redisCmd);

session.execCommand(redisCmd);

List<String> result\_tmp = **new** ArrayList<>();

OutputStream oStream = session.getStdin();

String auth = "auth ttre 24";

IOUtilsStreamUtil. *execV2*(oStream, auth);

result\_tmp =IOUtilsStreamUtil. *readLines*(session.getStdout());

System.***out***.println(result\_tmp);

// String get ="get access\_token";// + "\n\n ";

System.***out***.println("----------------------\r\n");

result\_tmp.clear();

IOUtilsStreamUtil. *execV2*(oStream,"get access\_token");

Thread.*sleep*(500);

result\_tmp =IOUtilsStreamUtil. *readLines*(session.getStdout());

System.***out***.println(result\_tmp);

// get access\_token

System.***out***.println("----------------------\r\n");

result\_tmp.clear();

IOUtilsStreamUtil. *execV2*(oStream, "smembers 300348232050020352\_2019\_04\_02");

result\_tmp =IOUtilsStreamUtil. *readLines*(session.getStdout());

System.***out***.println(PrettyUtilV2t33.*showListObjV2*(result\_tmp)); ;

System.***out***.println("--f");

}

// new Thread(new Runnable() {

//

// @Override

// public void run() {

// try {

// out2li(result, session);

// } catch (IOException e) {

// // **TODO** Auto-generated catch block

// e.printStackTrace();

// }

//

// }

// }).start();

//

// new Thread(new Runnable() {

//

// @Override

// public void run() {

// try {

// InputStream is = new StreamGobbler(session.getStderr());// 获得标准输出流

// BufferedReader brs = new BufferedReader(new InputStreamReader(is));

// for (String line = brs.readLine(); line != null; line = brs.readLine()) {

// System.out.println("errStream:>" + line);

// }

//

// } catch (IOException e) {

// // **TODO** Auto-generated catch block

// e.printStackTrace();

// }

//

// }

// }).start();

// logger.info(" coll err out ok");

}

**package** com.attilax.io;

**import** java.io.BufferedReader;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.InputStreamReader;

**import** java.io.OutputStream;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.function.Consumer;

**import** com.google.common.collect.Maps;

**import** ch.ethz.ssh2.StreamGobbler;

**public** **class** IOUtilsStreamUtil {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

}

**static** Map<InputStream, Map<String, Object>> *inputStreamTable* = Maps.*newConcurrentMap*();

**public** **synchronized** **static** List<String> IOUtils\_readLines(InputStream stdout, Consumer Consumer1) **throws** Exception {

Map map = Maps.*newConcurrentMap*();

**if** (!*inputStreamTable*.containsKey((stdout))) { // first

List<String> result = **new** ArrayList<>();

Thread thread = **new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

**try** {

InputStream is = **new** StreamGobbler(stdout);// 获得标准输出流

BufferedReader brs = **new** BufferedReader(**new** InputStreamReader(is));

map.put("br", brs);

**for** (String line = brs.readLine(); line != **null**; line = brs.readLine()) {

Consumer1.accept(line);

result.add(line);

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

});

thread.start();

Map recMapLine = Maps.*newConcurrentMap*();

recMapLine.put("thd", thread);

recMapLine.put("rzt", result);

*inputStreamTable*.put(stdout, recMapLine);

Thread.*sleep*(500);

**return** result;

} **else** {

Thread.*sleep*(500);

List<String> result = (List<String>) *inputStreamTable*.get(stdout).get("rzt");

**return** result;

}

}

**public** **static** **void** execV2(OutputStream oStream, String get) **throws** IOException, InterruptedException {

System.***out***.println(get);

get = get + "\r\n";

oStream.write(get.getBytes());

oStream.flush();

// Thread.sleep(500);

}

**public** **static** **void** exec(OutputStream oStream, String get) **throws** IOException, InterruptedException {

System.***out***.println(get);

get = get + "\r\n";

oStream.write(get.getBytes());

oStream.flush();

Thread.*sleep*(500);

}

**public** **synchronized** **static** List<String> readLines(InputStream stdout) **throws** Exception {

**return** *IOUtils\_readLines*(stdout, **new** Consumer() {

@Override

**public** **void** accept(Object line) {

System.***out***.println("xStream:>" + line);

}

});

// BufferedReader brs=(BufferedReader) map.get("br");

// brs.close();

// return result;

}

// private static void out2li(List<String> result, Session session) throws IOException {

// InputStream is = new StreamGobbler(session.getStdout());// 获得标准输出流

// BufferedReader brs = new BufferedReader(new InputStreamReader(is));

// for (String line = brs.readLine(); line != null; line = brs.readLine()) {

// result.add(line);

// System.out.println("stdStream:>" + line);

// }

// }

}