


MongoDB

- 1. 基础
- 2. MongoDB单机、集群和分片模式的配置和模拟
 - 2.1 单机模式
 - 2.2 集群模式
 - 2.3 分片模式
- 3. MongoDB OPS Manager
- 4. 封装的linux命令执行器
- 5. 使用linux命令创建mongodb服务
 - 5.1 启动mongo服务
 - 5.2 关于keyfile和security配置的使用流程
 - 5.3 配置和启动集群
 - 5.4 关于mongodb的一些命令
 - 5.5 关于使用Java Runtime去执行linux shell cmd
 - 5.5.1 nohup命令
 - 5.5.2 ps命令

1. 基础

windows下安装最新版的mongodb还需要安装一个mongosh，用来进入shell

 [MongoDB入门（一） | Laravel China 社区](#)

 [MongoDB Tutorials](#)

- 1. mongodb中的集合类似于关系数据库中的表
- 2. 使用类似于 `mongosh "mongodb://127.0.0.1:27017" --eval db.isMaster()` 的命令可以直接执行mongodb命令。

2. MongoDB单机、集群和分片模式的配置和模拟

可以使用Studio 3T来进行数据可视化的展示，使用docker来搭建mongodb环境。

2.1 单机模式

2.2 集群模式

2.3 分片模式

准备3个configure server, 3个 shard1, 3个 shard2, 1个mongos, 目录结构如下 (需要在本地创建)

```
1  - configsvr_1/  
2    - conf/  
3      - mongod.conf  
4    - db/  
5    - log/  
6  - configsvr_2/  
7    - conf/  
8      - mongod.conf  
9    - db/  
10   - log/  
11 - configsvr_3/  
12   - conf/  
13     - mongod.conf  
14   - db/  
15   - log/  
16  
17 -----  
18 - shard1_1/  
19   - conf/  
20     - mongod.conf  
21   - db/  
22   - log/  
23 - shard1_2/  
24   - conf/  
25     - mongod.conf  
26   - db/  
27   - log/  
28 - shard1_3/  
29   - conf/  
30     - mongod.conf  
31   - db/  
32   - log/  
33  
34 -----  
35 - shard2_1/  
36   - conf/  
37     - mongod.conf  
38   - db/  
39   - log/  
40 - shard2_2/  
41   - conf/  
42     - mongod.conf  
43   - db/  
44   - log/  
45 - shard2_3/
```

```

46     - conf/
47     - mongod.conf
48     - db/
49     - log/
50
51     -----
52     - mongos/
53     - log/
54     - mongos.conf

```

1. `docker pull mongo:<version>` (https://hub.docker.com/_/mongo/tags)
2. 设置容器网桥, 网段定义在: 10.20.0.0/24范围内: `docker network create --subnet=10.20.0.0/24 mongodbnnet`
3. 启动config server
 - a. config server的mongod.conf配置信息如下 (三个一样), 参数 `-v` (volumes) 映射宿主机目录与容器内部目录, 参数 `-p` (port) 映射宿主机端口与容器内服务端口, 用来使用外部工具登录, 参数 `-f` 制定启动mongo服务所需要的配置文件(容器内部路径/data/conf 被映射到 ~/mongo_sharding/configsvr_1/conf)

```

1 net:
2   bindIp: 0.0.0.0
3   port: 27017
4 processManagement:
5   timeZoneInfo: /usr/share/zoneinfo
6 replication:
7   replSetName: cfg
8 sharding:
9   clusterRole: configsvr
10 storage:
11   dbPath: /data/db
12   engine: wiredTiger
13   journal:
14     enabled: true
15 systemLog:
16   destination: file
17   path: /data/log/mongod.log

```

- b. 启动容器: `docker run --rm -d --name=mongo_cfg1 --network=mongodbnnet --ip=10.20.0.2 -v ~/mongo_sharding/configsvr_1/db (替换成本地db的路径) :/data/db -v ~/mongo_sharding/configsvr_1/conf:/data/conf -v ~/mongo_sha`

```
rding/configsvr_1/log:/data/log -p 27017:27017 mongo:<version> -f /data/conf/mongod.conf
```

 (另外两个ip对应的就是10.20.0.3和10.20.0.4)

c. 设置主从模式:

- i. `docker exec -it mongo_cfg1 mong`
- ii. `rs.initiate({ _id : "cfg", members:[{ _id:0, host: "10.20.0.2:27017", priority:5}, { _id:1, host: "10.20.0.3:27017", priority:3}, { _id:2, host: "10.20.0.4:27017", priority:2}] })`
- iii. `show databases`

4. 启动shard

a. shard的mongod.conf配置信息如下

```
1  ## mongod.conf 内容
2  net:
3    bindIp: 0.0.0.0
4    port: 27017
5  processManagement:
6    timeZoneInfo: /usr/share/zoneinfo
7  replication:
8    replSetName: shard2
9  sharding:
10   clusterRole: shardsvr
11  storage:
12   dbPath: /data/db
13   engine: wiredTiger
14   journal:
15     enabled: true
16  systemLog:
17   destination: file
18   path: /data/log/mongod.log
```

- b. 启动容器: `docker run --rm -d --name=mongo_shard2_1 --network=mongodbnet --ip=10.20.0.8 -v ~/mongo_sharding/shard2_1/db:/data/db -v ~/mongo_sharding/shard2_1/conf:/data/conf -v ~/mongo_sharding/shard2_1/log:/data/log -p 27023:27017 mongo:<version> -f /data/conf/mongod.conf` (另外两个ip对应的就是10.20.0.9和10.20.0.10)

c. 设置主从模式

- i. `docker exec -it mongo_shard2_1 mongo`
- ii. `rs.initiate({ _id : "shard2", members: [{ _id: 0, host: "10.20.0.8:27017", priority:5},{ _id:`

```
1, host: "10.20.0.9:27017", priority:3},{_id: 2, host: "10.20.0.10:27017", priority:2}}))
```

iii. `show databases`

5. 启动mongos

a. `mongos.conf`配置文件内容如下

```
1 net:
2   bindIp: 0.0.0.0
3   port: 27020
4 processManagement:
5   fork: true
6   timeZoneInfo: /usr/share/zoneinfo
7 sharding:
8   configDB: cfg/10.20.0.2:27017,10.20.0.3:27017,10.20.0.4:27017
9 systemLog:
10  destination: file
11  path: /data/log/mongos.log
```

b. 启动容器: `docker run --rm -d --name=mongos --network=mongodbnnet --ip=10.20.0.11 -p 27030:27020 -v ~/mongo_sharding/mongos:/data mongo:<version>`

c. 启动mongos: `docker exec -it mongos bash`

d. 增加shard1和shard2

```
1 sh.addShard("shard1/10.20.0.5:27017,10.20.0.6:27017,10.20.0.7:27017") ##
# mongos内增加分片配置 shard1
2 sh.addShard("shard2/10.20.0.8:27017,10.20.0.9:27017,10.20.0.10:27017") ###
# mongos内增加分片配置 shard2
```

3. MongoDB OPS Manager

<https://www.mongodb.com/docs/ops-manager/current/application/>

MongoDB OPS Manager介绍

MongoDB OPS Manager是随MongoDB Enterprise版本发布的MongoDB管理、监控、备份系统(但是也可以用于community版本), 是MMS(MongoDB Management Service)的内网版本。

MMS工作原理如下:

1. 在MMS服务器上配置你的MongoDB信息 (Host, Port, User, Passwd等)
2. 在一台能够访问你MongoDB服务的内网机器上运行其提供的Agent
3. Agent脚本从MMS服务器获取到你配置的MongoDB信息
4. Agent脚本连接到相应的MongoDB获取必要的监控数据
5. Agent脚本将监控数据上传到MMS的服务器
6. 登录MMS网站查看整理过后的监控数据图表了

Ops Manager主要功能：

1. 监控：实时的数据可视化展示，关键数据库指标监控报警
2. 管理：自动部署，修改配置，启停等
3. 备份：数据备份、恢复，支持scheduled snapshots and point-in-time recovery

The Ops Manager Application provides the user interface and the HTTP services the MongoDB Agent uses to transmit data to and from Ops Manager.

感觉主要的能力还是由MongoDB Agent提供的，他会进行备份、监控、自动化管理等，然后通过Ops Manager的数据交换，将数据汇总到Ops Manager应用的界面上。

Ops Manager可以配置和维护MongoDB节点和集群，MongoDB Agent和MongoDB是一一对应的关系，在每个 MongoDB 主机上使用自动化的 MongoDB 代理可以维护该 MongoDB 部署

4. 封装的linux命令执行器

 [java:执行linux sudo命令_10km的博客-程序员信息网 - 程序员信息网](#)

```
5 import java.io.InputStream;
6 import java.io.LineNumberReader;
7 import java.util.ArrayList;
8 import java.util.Arrays;
9 import java.util.Collections;
10 import java.util.List;
11 import java.util.logging.Logger;
12
13 /**
14  * linux命令行执行器
15  *
16  * @author guyadong
17  */
18 public class CmdExecutor {
19
20     private static final Logger logger = Logger.getLogger(CmdExecutor.class.getSimpleName());
21     private static final String SUDO_CMD = "sudo";
22     private static final String SHELL_NAME = "/bin/bash";
23     private static final String SHELL_PARAM = "-c";
24     private static final String REDIRECT = "2>&1";
25
26     /**
27      * 执行 sudo 的密码
28      */
29     private final String sudoPassword;
30
31     /**
32      * 是否显示命令内容及输出
33      */
34     private boolean verbose = true;
35
36     /**
37      * 是否错误输出重定向
38      */
39     private boolean errRedirect = true;
40
41     /**
42      * 是否同步，主线程是否等待命令执行结束
43      */
44     private boolean sync = true;
45
46     /**
47      * 执行多条命令时的命令分隔符
48      */
49     private String cmdSeparator = " && ";
50     private List<String> cmds = new ArrayList<String>(16);
51
52     public static CmdExecutor builder() {
53         return new CmdExecutor();
54     }
55 }
```



```

50
51     public static CmdExecutor builder(String sudoPassword) {
52         return new CmdExecutor(sudoPassword);
53     }
54
55     protected CmdExecutor() {
56         this(null);
57     }
58
59     protected CmdExecutor(String sudoPassword) {
60         this.sudoPassword = sudoPassword;
61     }
62
63     public CmdExecutor verbose(boolean verbose) {
64         this.verbose = verbose;
65         return this;
66     }
67
68     public CmdExecutor errRedirect(boolean errRedirect) {
69         this.errRedirect = errRedirect;
70         return this;
71     }
72
73     public CmdExecutor sync(boolean sync) {
74         this.sync = sync;
75         return this;
76     }
77
78     public CmdExecutor cmdSeparator(String cmdSeparator) {
79         if (null != cmdSeparator && !cmdSeparator.isEmpty()) {
80             this.cmdSeparator = cmdSeparator;
81         }
82         return this;
83     }
84
85     private String getRedirect() {
86         return errRedirect ? REDIRECT : "";
87     }
88
89     /**
90     * 添加一条需要sudo执行的命令
91     *
92     * @param cmd 要执行的命令(字符串中不需要有sudo)
93     * @return
94     */
95     public CmdExecutor sudoCmd(String cmd) {
96         if (null != cmd && 0 != cmd.length()) {
97             if (null == sudoPassword) {

```

```

98         cmds.add(String.format("%s %s %s", SUDO_CMD, cmd, getRedi
99     rect()));
100     } else {
101         cmds.add(String.format("echo '%s' | %s %s %s", sudoPasswo
102     rd, SUDO_CMD, cmd, getRedirect()));
103     }
104     return this;
105 }
106
107 /**
108  * 添加一条普通命令
109  *
110  * @param cmd
111  * @return
112  */
113 public CmdExecutor cmd(String cmd) {
114     if (null != cmd && 0 != cmd.length()) {
115         cmds.add(String.format("%s %s", cmd, getRedirect()));
116     }
117     return this;
118 }
119
120 private List<String> build() {
121     return cmds.isEmpty()
122         ? Collections.<String>emptyList()
123         : Arrays.asList(SHELL_NAME, SHELL_PARAM, join(cmds, cmdSepara
124     tor));
125 }
126
127 private static String join(List<String> str, String separator) {
128     StringBuffer buffer = new StringBuffer();
129     for (int i = 0; i < str.size(); ++i) {
130         if (i > 0) {
131             buffer.append(separator);
132         }
133         buffer.append(str.get(i));
134     }
135     return buffer.toString();
136 }
137
138 /**
139  * 将{@link InputStream}中所有内容输出到{@link StringBuffer}
140  *
141  * @param in
142  * @return
143  * @throws IOException
144  */

```

```

143     private static void toBuffer(InputStream in, StringBuffer buffer) thr
144     ws IOException {
145         if (null == in || null == buffer) {
146             return;
147         }
148         InputStreamReader ir = new InputStreamReader(in);
149         LineNumberReader input = new LineNumberReader(ir);
150         try {
151             String line;
152             while ((line = input.readLine()) != null) {
153                 buffer.append(line).append("\n");
154             }
155         } finally {
156             input.close();
157         }
158     }
159
160     /**
161     * 调用{@link Runtime#exec(String[])}执行命令
162     *
163     * @return 返回输出结果
164     */
165     public String exec() throws IOException {
166         StringBuffer outBuffer = new StringBuffer();
167         exec(outBuffer, null);
168         return outBuffer.toString();
169     }
170
171     /**
172     * 调用{@link Runtime#exec(String[])}执行命令
173     *
174     * @param outBuffer 标准输出
175     * @param errBuffer 错误信息输出
176     * @throws IOException
177     */
178     public void exec(StringBuffer outBuffer, StringBuffer errBuffer) thro
179     ws IOException {
180         List<String> cmdlist = build();
181         if (!cmdlist.isEmpty()) {
182             if (verbose) {
183                 logger.info(join(cmdlist, " "));
184             }
185
186             Process process = Runtime.getRuntime().exec(cmdlist.toArray(n
187             ew String[cmdlist.size()]));
188             if (sync) {
189                 try {
190                     process.waitFor();

```

```

188         } catch (InterruptedException e) {
189             }
190     }
191     toBuffer(process.getInputStream(), outBuffer);
192     toBuffer(process.getErrorStream(), errBuffer);
193 }
194 }

```

调用方式如下：

```

Java | 复制代码
1 CmdExecutor.builder("password")
2   .cmd("ls -l")
3   .sudoCmd("chmod 777 /usr/local")
4   .exec();


```

5. 使用linux命令创建mongodb服务


[2021年12月2日 Linux安装MongoDB – Deed's博客 – Java 技术 分享 学习](#)

[MongoDB shell 使用 mongosh 替换 mongo](#)

 [Install mongosh](#)

 [How to fix 'Sessions collection is not set up' error when trying to convert existing mongodb instance to replica set](#)

 [MongoDB TLS证书文件权限](#)

 [Calling a Linux command via java.lang.Runtime.exec](#) (nohup xxx &命令使用runtime无法正确运行，需要使用processBuilder)

5.1 启动mongo服务

以创建三个节点的集群模式为例，使用到的配置文件如下

```
1 net:
2   bindIp: 0.0.0.0 # mongo服务绑定的ip地址
3   port: 27017 # mongo服务绑定的端口
4 processManagement:
5   timeZoneInfo: /usr/share/zoneinfo
6 replication:
7   replSetName: xxxx # 集群名称
8 sharding:
9   clusterRole: shardsvr
10 storage:
11   dbPath: /mongodb/data/db # 需要自己新建该目录
12   engine: wiredTiger
13   journal:
14     enabled: true
15 systemLog:
16   destination: file
17   logAppend: true
18   logRotate: reopen
19   path: /mongodb/log/mongodb.log # 需要自己新建对应的目录和日志文件
20 # security:
21 #   authorization: enabled
22 #   keyFile: /mongodb/key/mongodb.key # keyFile的路径, 开启安全认证
```

需要使用到的linux命令如下:

```
1  mkdir ~/mongodb # 在用户目录下新建mongodb目录，存放mongodb的配置文件等
2  wget -q -P ~/ https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-ubuntu
   2004-5.0.12.tgz # 下载mongodb到指定的目录下
3  tar -zxvf ~/mongodb-linux-x86_64-ubuntu2004-5.0.12.tgz -C ~/mongodb # 解压缩
   mongodb软件包到mongodb目录下
4  # 新建一些配置mongodb需要的目录和文件
5  mkdir ~/mongodb/data
6  mkdir ~/mongodb/data/db
7  mkdir ~/mongodb/log
8  touch ~/mongodb/log/mongodb.log
9  mkdir ~/mongodb/conf
10 # 已经事先在test目录下预存了mongodb.conf配置文件，复制到指定目录即可，也可以在~/mong
   odb/conf/目录下进行创建并写入配置内容，配置内容见上
11 cp ~/test/mongodb.conf ~/mongodb/conf/mongodb.conf
12 mkdir ~/mongodb/key
13 # 新建keyFile文件并写入对应的key用来进行安全验证
14 touch ~/mongodb/key/mongodb.key
15 # 修改keyFile的权限，防止启动mongod时too open和permission denied的报错出现
16 chmod 600 ~/mongodb/key/mongodb.key
17 # 启动mongod，也就是启动mongo服务（若出现报错，可以去mongodb.log日志下查看报错内容）
18 nohup ~/mongodb/mongodb-linux-x86_64-ubuntu2004-5.0.12/bin/./mongod --conf
   ig ~/mongodb/conf/mongodb.conf &
19 mkdir ~/mongosh
20 # 下载mongosh到指定目录
21 wget -q -P ~/ https://downloads.mongodb.com/compass/mongosh-1.5.4-linux-x6
   4.tgz
22 # 解压缩mongosh
23 tar -zxvf ~/mongosh-1.5.4-linux-x64.tgz -C ~/mongosh
```

可以通过 `~/mongosh/mongosh-1.5.4-linux-x64/bin/mongosh --port 27017` 来在指定的端口启动mongosh（端口对应的就是配置文件中的端口，也可以不加）

完整的目录结构如下：

```
1  |— mongodb
2  |   |— conf
3  |   |   |— mongodb.conf
4  |   |— data
5  |   |   |— db
6  |   |— key
7  |   |   |— mongodb.key
8  |   |— log
9  |   |   |— mongodb.log
10 |   |— mongodb-linux-x86_64-ubuntu2004-5.0.12
11 |   |   |— LICENSE-Community.txt
12 |   |   |— MPL-2
13 |   |   |— README
14 |   |   |— THIRD-PARTY-NOTICES
15 |   |   |— bin
16 |   |       |— install_compass
17 |   |       |— mongo
18 |   |       |— mongod
19 |   |       |— mongos
20 |— mongodb-linux-x86_64-ubuntu2004-5.0.12.tgz
21 |— mongosh
22 |   |— mongosh-1.5.4-linux-x64
23 |   |   |— LICENSE-crypt-library
24 |   |   |— LICENSE-mongosh
25 |   |   |— README
26 |   |   |— THIRD_PARTY_NOTICES
27 |   |   |— bin
28 |   |       |— mongosh
29 |   |       |— mongosh_crypt_v1.so
30 |   |— mongosh.1.gz
31 |— mongosh-1.5.4-linux-x64.tgz
```

5.2 关于keyfile和security配置的使用流程

1. 先使用不带security配置的config配置文件启动mongodb
2. 初始化mongodb集群
3. 创建一个root用户
4. 在config文件中添加security配置
5. 重新启动mongodb服务

如果一开始就添加security配置启动mongodb服务，可能会导致没有权限去操作数据库，必须先创建一个root角色的用户，后面通过root用户才可以对mongodb数据库做一系列的操作（DDL、DML、privilege.....）。

5.3 配置和启动集群

1. 为了模拟出在多个不同机器上启动mongo服务并配置集群模式的效果，按照上一节的方法，在一台机器上启动三个mongo服务，分别运行mongo1、mongo2、mongo3目录下，对应不同的端口：27017、27018、27019，将27017当成primary节点
2. 普通登录主节点 `~/mongo1/mongosh/mongosh-1.5.4-linux-x64/bin/mongosh --port 27017`
 - a. 初始化集群 `rs.initiate({ _id : "shard1", members: [{ _id: 0, host: "127.0.0.1:27017", priority:5},{ _id: 1, host: "127.0.0.1:27018", priority:3},{ _id: 2, host: "127.0.0.1:27019", priority:2}] })`
 - b. `use admin`
 - c. 创建admin用户 `db.createUser({user:"admin", pwd:"admin", roles:[{role:"userAdminAnyDatabase", db:"admin" }]})`
3. 退出后再以admin登录主节点 `~/mongo1/mongosh/mongosh-1.5.4-linux-x64/bin/mongosh --port 27017 -u "admin" -p "admin"`
 - a. `use admin`
 - b. 创建test用户 `db.createUser({user:'test',pwd:'test',roles:[{role:'readWrite',db:'test'}]})`
4. 退出后再以test登录主节点 `~/mongo1/mongosh/mongosh-1.5.4-linux-x64/bin/mongosh --port 27017 -u "test" -p "test"`
 - a. `use test`
 - b. 在主节点插入数据 `db.col.insertOne({name:"klein"})`
5. 以test身份登录从节点 `~/mongo2/mongosh/mongosh-1.5.4-linux-x64/bin/mongosh --port 27018 -u "test" -p "test"`
 - a. `use test`
 - b. `db.getMongo().setReadPref('secondary')`
 - c. `db.col.find()` 查看数据，会发现和主节点数据一致，已经实现主从复制，集群模式配置成功
6. 以admin身份登录主节点 `~/mongo1/mongosh/mongosh-1.5.4-linux-x64/bin/mongosh --port 27017 -u "admin" -p "admin"`

- a. `use admin`
- b. 创建root用户 `db.createUser({ user: 'root', pwd: 'root', roles: [{ role: 'root', db: 'admin' }] });`
7. 退出后以root身份登录主节点 `~/mongo1/mongosh/mongosh-1.5.4-linux-x64/bin/mongosh --port 27017 -u "root" -p "root"`
 - a. 可以查看 `rs.status()` , 查看节点信息

RestfulToolkitX一个比较好用的类似于postman的插件

5.4 关于mongodb的一些命令

假设在27017、27018、27019端口各部署了一个mongodb服务

1. 未初始化集群时只能通过一个指定的端口去连接特定端口上的数据库 `mongosh mongodb://127.0.0.1:27017`
2. 使用 `mongosh mongodb://127.0.0.1:27017 --eval 'rs.initiate({_id:"shard1", members:[{"_id":0,"host":"127.0.0.1:27017","priority":1},{_id":1,"host":"127.0.0.1:27018","priority":1},{_id":2,"host":"127.0.0.127019","priority":1}])}'` 命令可以初始化集群, 其中priority表示这个节点有多大概率被选为集群中的主节点。
3. 初始化集群完成后就可以通过 `mongosh mongodb://127.0.0.1:27017,127.0.0.1:27018,127.0.0.1:27019` 去登录mongodb了, 不需要指定一个特定的端口, 可以把集群中所有节点的ip全部写在一起, 会自动选择一个主节点登入进去。

5.5 关于使用Java Runtime去执行linux shell cmd

java runtime没有办法去正确识别一些转义字符等, 直接执行命令字符串会报错, 一般可以将命令拆成字符串数组再通过如下的方式去执行。

```
Java | 复制代码
1 String[] cmdList;
2 Process process = Runtime.getRuntime().exec(cmdList);
```

5.5.1 nohup命令

例如 `nohup /home/klein/mongodb/mongodb-linux-x86_64-ubuntu2004-5.0.12/bin/. /mongodb --config /home/klein/mongodb/conf/mongodb.conf &` 命令，直接执行会报错，需要使用如下方式执行

```
Java | 复制代码
1 String cmd="nohup /home/klein/mongodb/mongodb-linux-x86_64-ubuntu2004-5.0.12/bin/. /mongodb --config /home/klein/mongodb/conf/mongodb.conf &";
2 String[] cmdList = cmd.split(" ");
3 process = new ProcessBuilder(cmdList[1],cmdList[2],cmdList[3])
4     .redirectInput(new File("/dev/null"))
5     .redirectOutput(new File("nohup.out"))
6     .redirectError(new File("nohup.out"))
7     .start();
```

5.5.2 ps命令

通过 `ps -ef |grep xxxx` 想要得到指定进程的进程号时，可以通过如下方式执行

```
1 String targetProcessName="xxxxx";
2 Process p1 = Runtime.getRuntime().exec(new String[]{"ps", "-ef"});
3 InputStream input = p1.getInputStream();
4 Process p2 = Runtime.getRuntime().exec(new String[]{"grep", targetProcessName});
5 OutputStream output = p2.getOutputStream();
6 IOUtils.copy(input, output);
7 output.close(); // signals grep to finish
8 List<String> result = IOUtils.readLines(p2.getInputStream());
9 String res = result.get(0);
10 // res是字符串的形式（具体可以打印出来看一下结果再做解析），需要解析得到其中的pid
11 // parse result to get pid
12 int index = 0;
13 while (res.charAt(index) != ' ') {
14     index++;
15 }
16 while (res.charAt(index) == ' ') {
17     index++;
18 }
19 int start = index;
20 while (res.charAt(index) != ' ') {
21     index++;
22 }
23 String pid = res.substring(start, index);
24 System.out.println(pid);
```