

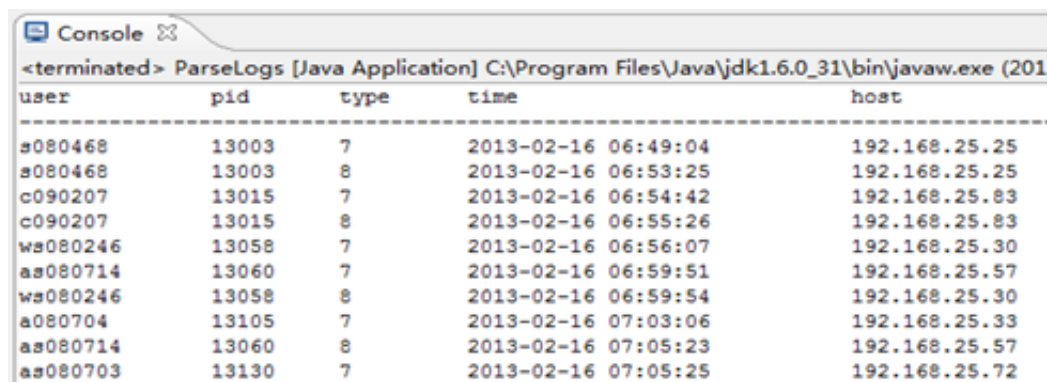
Java语言基础 Day06

1. [DMS V1.0 日志数据解析算法](#)
2. [DMS V1.0 日志数据匹配算法](#)
3. [DMS V1.0 登录数据查询算法](#)
4. [DMS V1.0 系统整合](#)

1 DMS V1.0 日志数据解析算法

1.1 问题

DMS 1.0 业务中需要完成日志数据的解析功能并显示解析后的记录。从日志数据源系统中得到原始数据，这些原始数据是日志数据源程序从Unix系统的日志文件中读取而来；解析原始记录中的用户登入和登出的数据。控制台输出情况如图 - 57所示：



user	pid	type	time	host
as080468	13003	7	2013-02-16 06:49:04	192.168.25.25
as080468	13003	8	2013-02-16 06:53:25	192.168.25.25
c090207	13015	7	2013-02-16 06:54:42	192.168.25.83
c090207	13015	8	2013-02-16 06:55:26	192.168.25.83
ws080246	13058	7	2013-02-16 06:56:07	192.168.25.30
as080714	13060	7	2013-02-16 06:59:51	192.168.25.57
ws080246	13058	8	2013-02-16 06:59:54	192.168.25.30
a080704	13105	7	2013-02-16 07:03:06	192.168.25.33
as080714	13060	8	2013-02-16 07:05:23	192.168.25.57
as080703	13130	7	2013-02-16 07:05:25	192.168.25.72

图-57

1.2 方案

从日志数据源程序中得到原始数据，该原始数据为字节数组：

```
01. byte[] logs = LogDataSource.getLogData();
```

每条登录记录包含 372 字节的数据，因此，登录的总记录数为：数组长度 (logs.length) /372

每条记录（每372字节）中需要采集的信息有：

- 登录的用户名
- 用户登录的进程 ID
- 用户登录/登出的时间
- 用户登录期间的在线时间
- 用户的终端IP

这些需要采集的数据信息在每 372个字节中的位置如表 - 1所示。

表 - 1 字节数组中需要采集的数据位置说明

位置范围	长度	数据含义
000-031	32	用户名，为文本类型数据（string）
068-071	4	进程 ID，为整数类型数据（int）
072-073	2	登录类型，值在 1-8 之间，但只处理 7 与 8 两种情况：7 为登入，8 为登出；为整数类型数据（short）
080-083	4	登入或者登出时刻，为整数类型数据（int）
114-371	257	用户登录的终端 IP，为文本类型数据（string）

解析原始日志字节数组的核心参考代码如下：

```

01.      /**
02.      * 解析日志数据logs 解析结果存储到5个数组中 解析规则 每372个字节数据为一
03.      *
04.      * @param logs
05.      *          原始日志数据，每372个字节为一个日志记录
06.      * @param users
07.      *          解析以后的登录用户名数组
08.      * @param pids
09.      *          解析以后的用户进程号数组
10.      * @param types
11.      *          存储解析以后登录类型的数组
12.      * @param times
13.      *          存储解析以后登录时间的数组
14.      * @param hosts
15.      *          存储解析以后用户主机名的数组
16.      */
17.      public static void parseLogs(byte[] logs, String[] users, int[] pids,
18.      short[] types, int[] times, String[] hosts) {
19.      for (int i = 0, start = 0; i < users.length; i++, start += 372)
20.      users[i] = toString(logs, start + 0, 32);
21.      pids[i] = toInt(logs, start + 68);
22.      types[i] = toShort(logs, start + 72);
23.      times[i] = toInt(logs, start + 80);
24.      hosts[i] = toString(logs, start + 114, 257);
25.      }
26.      }

```

1.3 实现

新建ParseLogs类，在类中添加toShort方法，将byte数组中从offset开始连续的2个byte数据转换为一个short类型数据。

1.连续读取两个byte数据到int的类型

2.使用8位掩码清理int数据: 将高24位清理为0, 低8位是从数组中读取的byte数据

3.向左移位int数据第一个整数的低八位向高位移动8次

4.拼接为一个int数据: 就是将两个整数相加使两个整数数据拼接为一个整数，强制类型转换为short类型返回

```
01.      /**
02.          * 将byte数组中连续的2个byte数据转换为一个short类型数据.
03.          *
04.          * @param bytes
05.          *          需要读取的元素数组
06.          * @param offset
07.          *          转换的起始偏移位置
08.          * @return 将连续两个byte转换为一个short数据
09.          */
10.      public static short toShort(byte[] bytes, int offset) {
11.          int d1 = bytes[offset + 0];
12.          int d2 = bytes[offset + 1];
13.          d1 &= 0xff;
14.          d2 &= 0xff;
15.          d1 <<= 8;
16.          d2 <<= 0;
17.          int i = d1 + d2;
18.          return (short) i;
19.      }
```

在ParseLogs类中添加toInt方法，将byte数组中从offset开始连续的4个byte数据转换为一个int类型数据。

1.连续读取4个byte数据到int的类型变量: d1, d2, d3, d4

2.使用8位掩码清理int数据: 将高24位清理为0, 低8位是从数组中读取的byte数据

3.向左移位int数据:

d1移动24位,

d2移动16位,

d3移动8位,

d4移动0位,

4.将d1,d2,d3,d4拼接为一个int数据: 就是将4个整数相加使两个整数数据拼接为一个整数i

5.返回整数i

```
01.      /**
02.      * 将byte数组中连续的4个byte数据转换为一个int类型数据.
03.      *
04.      * @param bytes
05.      *          需要读取的元素数组
06.      * @param offset
07.      *          转换的起始偏移位置
08.      * @return 将连续4个byte转换为一个int数据
09.      */
10.     public static int toInt(byte[] bytes, int offset) {
11.         int d1 = bytes[offset + 0] & 0xff;
12.         int d2 = bytes[offset + 1] & 0xff;
13.         int d3 = bytes[offset + 2] & 0xff;
14.         int d4 = bytes[offset + 3] & 0xff;
15.         int i = (d1 << 24) + (d2 << 16) + (d3 << 8) + d4;
16.         return i;
17.     }
```

在ParseLogs类中添加toString方法, 将byte数组中从offset开始连续的length个byte数据转换为一个String类型数据。

1.利用字符串的构造器new String(byte[] bytes) 实现转换

2.利用方法trim() 去除两端的空白字符

3.返回字符串

```
01.      /**
02.      * 将byte数组中从offset位置开始, 解析length个字节为String类型数据.
03.      *
04.      * @param bytes
05.      *          需要读取的元素数组
06.      * @param offset
07.      *          转换的起始偏移位置
08.      * @param length
09.      *          要解析的byte数
```

```

10.      * @return String类型数据
11.      */
12.      public static String toString(byte[] bytes, int offset, int length)
13.      {
14.          return new String(bytes, offset, length).trim();
15.      }

```

在ParseLogs类中添加parseLogs方法，解析日志数据logs 解析结果存储到5个数组中。算法过程如图-58所示：

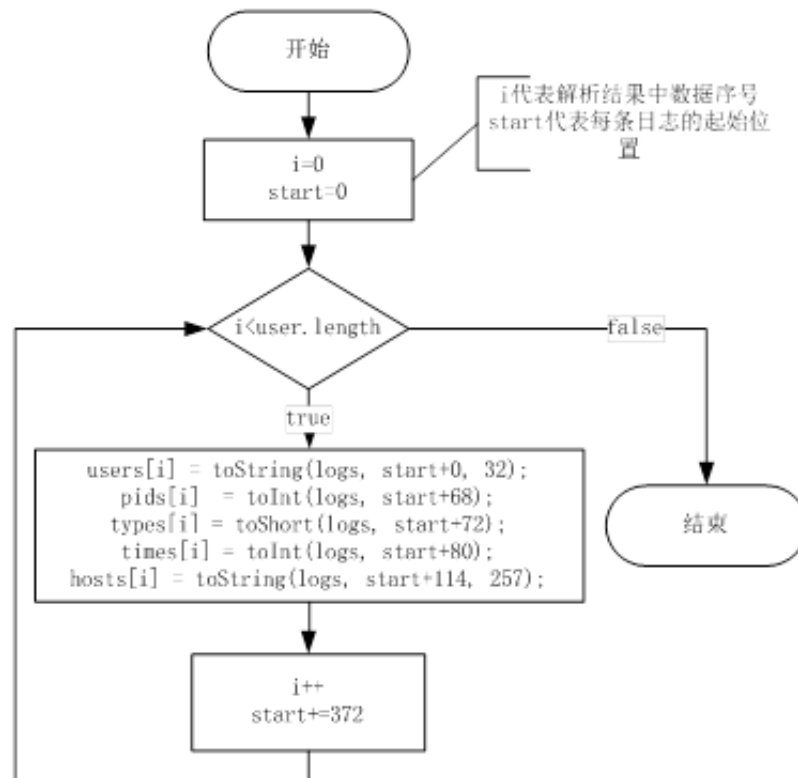


图-58

代码如下：

```

01.      /**
02.      * 解析日志数据logs 解析结果存储到5个数组中 解析规则 每372个字节数据为一
03.      *
04.      * @param logs
05.      *          原始日志数据，每372个字节为一个日志记录
06.      * @param users
07.      *          解析以后的登录用户名数组
08.      * @param pids
09.      *          解析以后的用户进程号数组
10.      * @param types

```

```

11.      *           存储解析以后登录类型的数组
12.      * @param times
13.      *           存储解析以后登录时间的数组
14.      * @param hosts
15.      *           存储解析以后用户主机名的数组
16.      */
17.      public static void parseLogs(byte[] logs, String[] users, int[] pids,
18.          short[] types, int[] times, String[] hosts) {
19.          for (int i = 0, start = 0; i < users.length; i++, start += 32)
20.          {
21.              users[i] = toString(logs, start + 0, 32);
22.              pids[i] = toInt(logs, start + 68);
23.              types[i] = toShort(logs, start + 72);
24.              times[i] = toInt(logs, start + 80);
25.              hosts[i] = toString(logs, start + 114, 257);
26.          }
27.      }

```

在ParseLogs类中添加printLogs方法，打印解析好日志数据。

```

01.      /**
02.      * 打印解析好日志数据
03.      * @param users
04.      *           解析以后的登录用户名数组
05.      * @param pids
06.      *           解析以后的用户进程号数组
07.      * @param types
08.      *           解析以后登录类型的数组
09.      * @param times
10.      *           解析以后登录时间的数组
11.      * @param hosts
12.      *           解析以后用户主机名的数组
13.      */
14.      public static void printLogs(String users[], int[] pids, short[] types,
15.          int[] pad = { 15, 10, 10, 30, 20 });
16.          System.out.println(StringUtils.rightPad("user", pad[0])
17.              + StringUtils.rightPad("pid", pad[1])

```

```

18.         + StringUtils.rightPad("type", pad[2])
19.         + StringUtils.rightPad("time", pad[3])
20.         + StringUtils.rightPad("host", pad[4]));
21.     System.out.println(StringUtils.repeat("-",
22. pad[0] + pad[1] + pad[2] + pad[3] + pad[4]));
23.     for (int i = 0; i < users.length; i++) {
24.         System.out.println(StringUtils.rightPad(users[i], pad[0])
25.             + StringUtils.rightPad("'" + pids[i], pad[1])
26.             + StringUtils.rightPad("'" + types[i], pad[2])
27.             + StringUtils.rightPad("'"
28. + DateUtils.formatDate(times[i]), pad[3])
29. + StringUtils.rightPad(hosts[i], pad[4]));
30.     }
31. }

```

在ParseLogs类中添加main方法，进行日志数据解析的测试。

```

01.     public static void main(String[] args) {
02.         byte[] logs = LogDataSource.getLogData(); //获取原始日志数据
03.
04.         int count = logs.length / 372; // 登录记录数量
05.         //存储 解析以后的登录用户名数组
06.         String[] users = new String[count];
07.         int[] pids = new int[count]; //存储解析以后的用户进程号数组
08.         short[] types = new short[count]; //存储解析以后登录类型的数组
09.         int[] times = new int[count]; //存储解析以后登录时间的数组
10.         //存储解析以后用户主机名的数组
11.         String[] hosts = new String[count];
12.         //解析原始日志数组
13.         parseLogs(logs, users, pids, types, times, hosts);
14.         //打印解析好的原始数据
15.         printLogs(users, pids, types, times, hosts);
16.
17.     }

```

ParseLogs类完整代码：

```

01.     public class ParseLogs {
02.         public static void main(String[] args) {
03.             byte[] logs = LogDataSource.getLogData(); //获取原始日志数据
04.
05.             int count = logs.length / 372; // 登录记录数量
06.             //存储 解析以后的登录用户名数组
07.             String[] users = new String[count];
08.             int[] pids = new int[count]; //存储解析以后的用户进程号数组
09.             short[] types = new short[count]; //存储解析以后登录类型的数组
10.             int[] times = new int[count]; //存储解析以后登录时间的数组
11.             //存储解析以后用户主机名的数组
12.             String[] hosts = new String[count];
13.             //解析原始日志数组
14.             parseLogs(logs, users, pids, types, times, hosts);
15.             //打印解析好的原始数据
16.             printLogs(users, pids, types, times, hosts);
17.
18.         }
19.
20.         /**
21.          * 解析日志数据logs 解析结果存储到5个数组中 解析规则 每372个字节数据为一
22.          *
23.          * @param logs
24.          *          原始日志数据，每372个字节为一个日志记录
25.          * @param users
26.          *          解析以后的登录用户名数组
27.          * @param pids
28.          *          解析以后的用户进程号数组
29.          * @param types
30.          *          存储解析以后登录类型的数组
31.          * @param times
32.          *          存储解析以后登录时间的数组
33.          * @param hosts
34.          *          存储解析以后用户主机名的数组
35.          */
36.         public static void parseLogs(byte[] logs, String[] users, int[] pids,
37.             short[] types, int[] times, String[] hosts) {
38.             for (int i = 0, start = 0; i < users.length; i++, start += 372)

```



```

39.         users[i] = toString(logs, start + 0, 32);
40.         pids[i] = toInt(logs, start + 68);
41.         types[i] = toShort(logs, start + 72);
42.         times[i] = toInt(logs, start + 80);
43.         hosts[i] = toString(logs, start + 114, 257);
44.     }
45. }
46.
47. /**
48.  * 打印解析好日志数据
49.  * @param users
50.  *      解析以后的登录用户名数组
51.  * @param pids
52.  *      解析以后的用户进程号数组
53.  * @param types
54.  *      解析以后登录类型的数组
55.  * @param times
56.  *      解析以后登录时间的数组
57.  * @param hosts
58.  *      解析以后用户主机名的数
59.  */
60. public static void printLogs(String users[], int[] pids, short[] ty
61.     int[] pad = { 15, 10, 10, 30, 20 };
62.     System.out.println(StringUtils.rightPad("user", pad[0])
63.         + StringUtils.rightPad("pid", pad[1])
64.         + StringUtils.rightPad("type", pad[2])
65.         + StringUtils.rightPad("time", pad[3])
66.         + StringUtils.rightPad("host", pad[4]));
67.     System.out.println(StringUtils.repeat("-",
68. pad[0] + pad[1] + pad[2] + pad[3] + pad[4]));
69.     for (int i = 0; i < users.length; i++) {
70.         System.out.println(StringUtils.rightPad(users[i], pad[0])
71.             + StringUtils.rightPad("" + pids[i], pad[1])
72.             + StringUtils.rightPad("" + types[i], pad[2])
73.             + StringUtils.rightPad(""
74. + DateUtils.formatDate(times[i]), pad[3])
75. + StringUtils.rightPad(hosts[i], pad[4]));
76.     }
77. }

```

```
78.
79.     /**
80.      * 将byte数组中连续的4个byte数据转换为一个int类型数据.
81.      *
82.      * @param bytes
83.      *      需要读取的元素数组
84.      * @param offset
85.      *      转换的起始偏移位置
86.      * @return 将连续4个byte转换为一个int数据
87.      */
88.     public static int toInt(byte[] bytes, int offset) {
89.         int d1 = bytes[offset + 0] & 0xff;
90.         int d2 = bytes[offset + 1] & 0xff;
91.         int d3 = bytes[offset + 2] & 0xff;
92.         int d4 = bytes[offset + 3] & 0xff;
93.         int i = (d1 << 24) + (d2 << 16) + (d3 << 8) + d4;
94.         return i;
95.     }
96.
97.     /**
98.      * 将byte数组中连续的2个byte数据转换为一个short类型数据.
99.      *
100.     * @param bytes
101.     *      需要读取的元素数组
102.     * @param offset
103.     *      转换的起始偏移位置
104.     * @return 将连续两个byte转换为一个short数据
105.     */
106.     public static short toShort(byte[] bytes, int offset) {
107.         int d1 = bytes[offset + 0];
108.         int d2 = bytes[offset + 1];
109.         d1 &= 0xff;
110.         d2 &= 0xff;
111.         d1 <= 8;
112.         d2 <= 0;
113.         int i = d1 + d2;
114.         return (short) i;
115.     }
116.
```

```

117.      /**
118.      * 将byte数组中从offset位置开始,解析length个字节为String类型数据.
119.      *
120.      * @param bytes
121.      *          需要读取的元素数组
122.      * @param offset
123.      *          转换的起始偏移位置
124.      * @param length
125.      *          要解析的byte数
126.      * @return String类型数据
127.      */
128.      public static String toString(byte[] bytes, int offset, int length)
129.      {
130.          return new String(bytes, offset, length).trim();
131.      }

```

隐藏

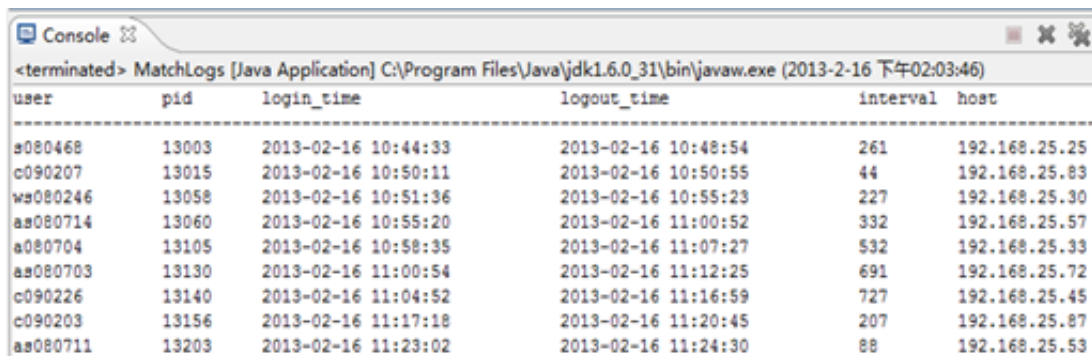
1.4 扩展

熟练掌握并完成DMSV1.0日志数据解析算法。

2 DMS V1.0 日志数据匹配算法

2.1 问题

DMS 1.0 业务中需要完成数据的匹配功能并显示匹配后的记录。将解析完成的记录成对匹配为完整的登录会话记录（一次登录会话包含一次登入记录和一次登出记录）。控制台显示数据如图-59所示：



user	pid	login_time	logout_time	interval	host
as080468	13003	2013-02-16 10:44:33	2013-02-16 10:48:54	261	192.168.25.25
c090207	13015	2013-02-16 10:50:11	2013-02-16 10:50:55	44	192.168.25.83
wa080246	13058	2013-02-16 10:51:36	2013-02-16 10:55:23	227	192.168.25.30
as080714	13060	2013-02-16 10:55:20	2013-02-16 11:00:52	332	192.168.25.57
as080704	13105	2013-02-16 10:58:35	2013-02-16 11:07:27	532	192.168.25.33
as080703	13130	2013-02-16 11:00:54	2013-02-16 11:12:25	691	192.168.25.72
c090226	13140	2013-02-16 11:04:52	2013-02-16 11:16:59	727	192.168.25.45
c090203	13156	2013-02-16 11:17:18	2013-02-16 11:20:45	207	192.168.25.87
as080711	13203	2013-02-16 11:23:02	2013-02-16 11:24:30	88	192.168.25.53

图-59

2.2 方案

匹配登录登出记录为一对登录数据序号相邻, 每对数据前一条为登录数据,后一条为登出数据 , users数组, pids数组, types数组 中保存的是原始数据, 匹配结果序号存储到entrys中.匹配过程如图-60所示：

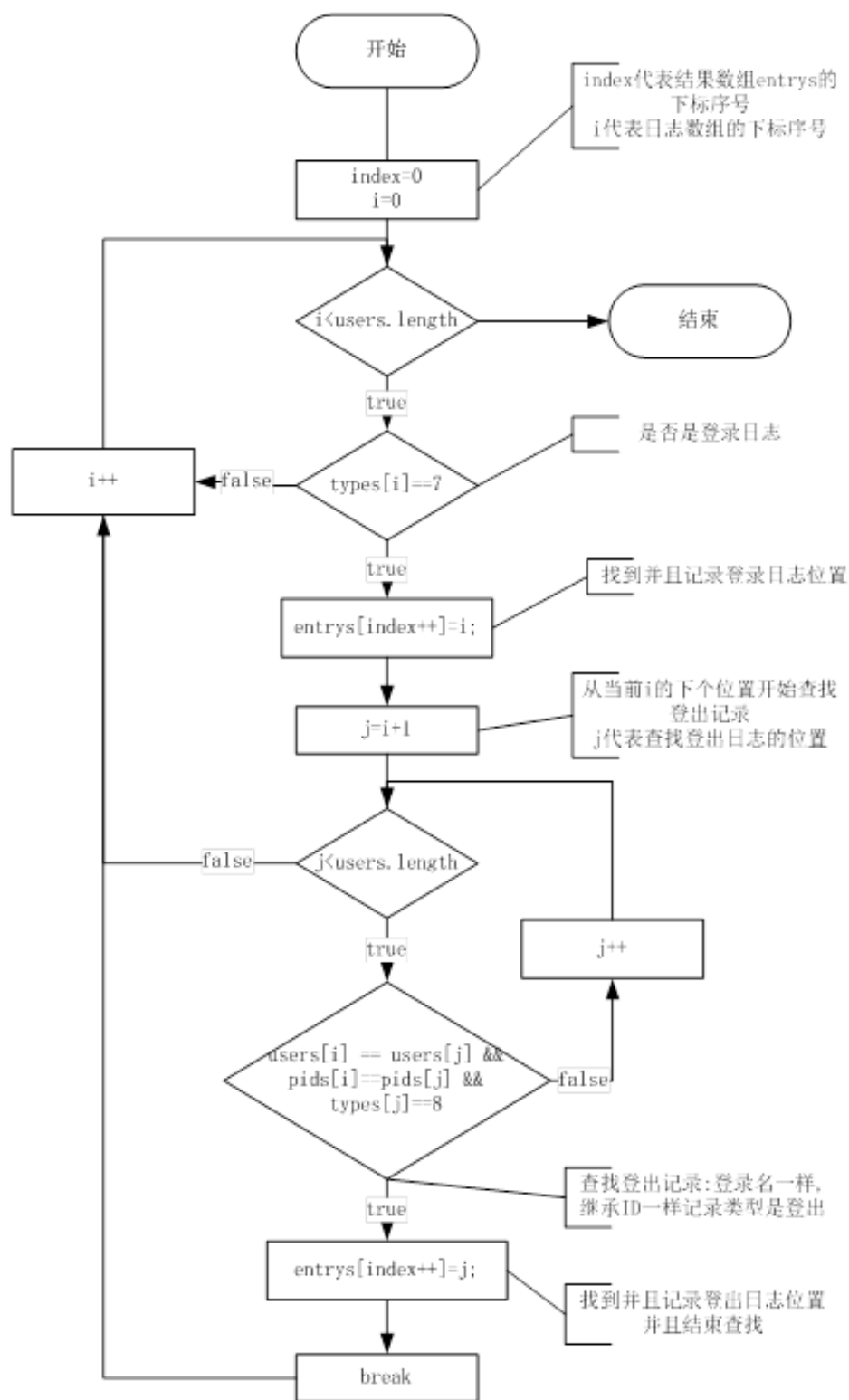


图-60

匹配过程的参考代码如下：

01. /**
02. * 匹配登录登出记录为一对登录数据，每对数据第一条为登录数据，第二条为登
03. * 出数据 users, pids, types 中保存的是原始数据，
04. * 匹配结果存储到entrys中。
05. * @param entrys 匹配结果数组

```

06.      * @param users  登录用户名数组
07.      * @param pids   用户进程号数组
08.      * @param types  登录类型的数组
09.      */
10.      public static void matchLogs(int[] entrys, String[] users, int[] pids) {
11.          int index = 0;
12.          for (int i = 0; i < users.length; i++) { // 迭代每个记录
13.              // 如果是登录记录(type==7) 就从当前位置开始向后查找登出记录
14.              if (types[i] == 7) {
15.                  entrys[index++] = i; // 先记录登录记录位置
16.                  // 从登录记录位置(i+1)开始向后查找登出记录
17.                  for (int j = i + 1; j < users.length; j++) {
18.                      // 查找登出记录的条件：用户名相同, 进程号相同,
19.                      // 记录类型为登出(type==8)
20.                      if (users[i].equals(users[j]) && pids[i] == pids[j]
21.                          && types[j] == 8) { // 找到logout记录
22.                          entrys[index++] = j; // 记录登出位置
23.                          break; // 找到就可以结束查找过程
24.                      }
25.                  }
26.              }
27.          }
28.      }

```

[隐藏](#)

2.3 实现

新建MatchLogs类，在类中添加matchLogs方法。

```

01.      /**
02.          * 匹配登录登出记录为一对登录数据，每对数据第一条为登录数据，第二条为登
03.          * 出数据 users, pids, types 中保存的是原始数据，
04.          * 匹配结果存储到entrys中。
05.          * @param entrys 匹配结果数组
06.          * @param users  登录用户名数组
07.          * @param pids   用户进程号数组
08.          * @param types  登录类型的数组
09.          */
10.      public static void matchLogs(int[] entrys, String[] users, int[] pids) {

```

```

11.         int index = 0;
12.         for (int i = 0; i < users.length; i++) { // 迭代每个记录
13.             // 如果是登录记录(type==7) 就从当前位置开始向后查找登出记录
14.             if (types[i] == 7) {
15.                 entrys[index++] = i; // 先记录登录记录位置
16.                 // 从登录记录位置(i+1)开始向后查找登出记录
17.                 for (int j = i + 1; j < users.length; j++) {
18.                     // 查找登出记录的条件: 用户名相同, 进程号相同,
19.                     // 记录类型为登出(type==8)
20.                     if (users[i].equals(users[j]) && pids[i] == pids[j]
21.                         && types[j] == 8) { // 找到logout记录
22.                         entrys[index++] = j; // 记录登出位置
23.                         break; // 找到就可以结束查找过程
24.                     }
25.                 }
26.             }
27.         }
28.     }

```

在MatchLogs类中添加printRecHeader方法。

```

01.     /**
02.     * 打印向控制台输出的头信息即第一行信息
03.     */
04.     public static void printRecHeader() {
05.         String[] headers = { "user", "pid", "login_time", "logout_time",
06.                               "interval", "host" };
07.         int[] pad = { 15, 10, 30, 30, 10, 20 };
08.         int total = 0;
09.         for (int i = 0; i < headers.length; i++) {
10.             System.out.print(StringUtils.rightPad(headers[i],
11. pad[i]));
12.             total += pad[i];
13.         }
14.         System.out.println();
15.         System.out.println(StringUtils.repeat("-", total));
16.     }

```

在MatchLogs类中添加printRecHeader方法。

```
01.      /**
02.      * 打印一条匹配后的登录会话
03.      * @param user 登录用户名
04.      * @param pid 用户进程号
05.      * @param loginTime 登录时间
06.      * @param logoutTime 登出时间
07.      * @param host 用户主机名
08.      */
09.      public static void printRec(String user, int pid, int loginTime,
10.          int logoutTime, String host) {
11.          int[] pad = { 15, 10, 30, 30, 10, 20 };
12.          int padIndex = 0;
13.          System.out.print(StringUtils.rightPad(user,
14. pad[padIndex++]));
15.          System.out.print(StringUtils.rightPad(pid + "",
16. pad[padIndex++]));
17.          System.out.print(StringUtils.rightPad(
18. DateUtils.formatDate(loginTime), pad[padIndex++]));
19.          System.out.print(StringUtils.rightPad(
20. DateUtils.formatDate(logoutTime), pad[padIndex++]));
21.          System.out.print(StringUtils.rightPad((
22. logoutTime - loginTime) + "", pad[padIndex++]));
23.          System.out.println(StringUtils.rightPad(
24. host, pad[padIndex++]));
25.      }
```

在MatchLogs类中添加list方法。

```
01.      /**
02.      * 打印全部匹配后的登录会话
03.      * @param entrys 匹配成功的日志对索引数组
04.      * @param users 解析以后的登录用户名数组
05.      * @param pids 解析以后的用户进程号数组
```

```

06.      * @param types 解析以后登录类型的数组
07.      * @param hosts 解析以后用户主机名的数组
08.      */
09.      public static void list(int[] entrys, String[] users, int[] pids,
10.          short[] types, int[] times, String[] hosts) {
11.          printRecHeader();
12.          for (int i = 0; i < entrys.length; i += 2) {
13.              int login = entrys[i];
14.              int logout = entrys[i + 1];
15.              printRec(users[login], pids[login],
16.                  times[login], times[logout], hosts[login]);
17.          }
18.      }

```

在MatchLogs类中添加Main方法，进行测试。

```

01.      public static void main(String[] args) {
02.          byte[] logs = LogDataSource.getLog();
03.
04.          int count = logs.length / 372; // 登录记录数量
05.
06.          String[] users = new String[count];
07.          int[] pids = new int[count];
08.          short[] types = new short[count];
09.          int[] times = new int[count];
10.          String[] hosts = new String[count];
11.
12.          ParseLogs.parseLogs(logs, users, pids, types, times, hosts);
13.          int[] entrys = new int[count];
14.          matchLogs(entrys, users, pids, types);
15.          list(entrys, users, pids, types, times, hosts);
16.
17.      }

```

MatchLogs类完成参考代码如下：


```

01.     public class MatchLogs {
02.         public static void main(String[] args) {
03.             byte[] logs = LogDataSource.getLogLogs();
04.
05.             int count = logs.length / 372; // 登录记录数量
06.
07.             String[] users = new String[count];
08.             int[] pids = new int[count];
09.             short[] types = new short[count];
10.             int[] times = new int[count];
11.             String[] hosts = new String[count];
12.
13.             ParseLogs.parseLogs(logs, users, pids, types, times, hosts);
14.             int[] entrys = new int[count];
15.             matchLogs(entrys, users, pids, types);
16.             list(entrys, users, pids, types, times, hosts);
17.
18.         }
19.
20.         /**
21.          * 匹配登录登出记录为一对登录数据, 每对数据第一条为登录数据, 第二条为登
22.          * 出数据 users, pids, types 中保存的是原始数据,
23.          * 匹配结果存储到entrys中.
24.          * @param entrys 匹配结果数组
25.          * @param users 登录用户名数组
26.          * @param pids 用户进程号数组
27.          * @param types 登录类型的数组
28.          */
29.         public static void matchLogs(int[] entrys, String[] users, int[] pi
30.             int index = 0;
31.             for (int i = 0; i < users.length; i++) { // 迭代每个记录
32.                 // 如果是登录记录(type==7) 就从当前位置开始向后查找登出记录
33.                 if (types[i] == 7) {
34.                     entrys[index++] = i; // 先记录登录记录位置
35.                     // 从登录记录位置(i+1)开始向后查找登出记录
36.                     for (int j = i + 1; j < users.length; j++) {
37.                         // 查找登出记录的条件: 用户名相同, 进程号相同,
38.                         // 记录类型为登出(type==8)

```

```

39.             if (users[i].equals(users[j]) && pids[i] == pids[j]
40.                 && types[j] == 8) { // 找到logout记录
41.                 entrys[index++] = j; // 记录登出位置
42.                 break; // 找到就可以结束查找过程
43.             }
44.         }
45.     }
46. }
47. }
48.
49.
50. /**
51.  * 打印全部匹配后的登录会话
52.  * @param entrys 匹配成功的日志对索引数组
53.  * @param users 解析以后的登录用户名数组
54.  * @param pids 解析以后的用户进程号数组
55.  * @param types 解析以后登录类型的数组
56.  * @param hosts 解析以后用户主机名的数组
57.  */
58. public static void list(int[] entrys, String[] users, int[] pids,
59.                         short[] types, int[] times, String[] hosts) {
60.     printRecHeader();
61.     for (int i = 0; i < entrys.length; i += 2) {
62.         int login = entrys[i];
63.         int logout = entrys[i + 1];
64.         printRec(users[login], pids[login],
65. times[login], times[logout], hosts[login]);
66.     }
67. }
68.
69. /**
70.  * 打印向控制台输出的头信息即第一行信息
71.  */
72. public static void printRecHeader() {
73.     String[] headers = { "user", "pid", "login_time", "logout_time",
74.         "interval", "host" };
75.     int[] pad = { 15, 10, 30, 30, 10, 20 };
76.     int total = 0;
77.     for (int i = 0; i < headers.length; i++) {

```

```

78.         System.out.print(StringUtils.rightPad(headers[i],
79.         pad[i]));
80.         total += pad[i];
81.     }
82.     System.out.println();
83.     System.out.println(StringUtils.repeat("-", total));
84. }
85.
86. /**
87.  * 打印一条匹配后的登录会话
88.  * @param user 登录用户名
89.  * @param pid 用户进程号
90.  * @param loginTime 登录时间
91.  * @param logoutTime 登出时间
92.  * @param host 用户主机名
93.  */
94.     public static void printRec(String user, int pid, int loginTime,
95.         int logoutTime, String host) {
96.         int[] pad = { 15, 10, 30, 30, 10, 20 };
97.         int padIndex = 0;
98.         System.out.print(StringUtils.rightPad(user,
99.         pad[padIndex++]));
100.        System.out.print(StringUtils.rightPad(pid + "",
101.        pad[padIndex++]));
102.        System.out.print(StringUtils.rightPad(
103.        DateUtils.formatDate(loginTime), pad[padIndex++]));
104.        System.out.print(StringUtils.rightPad(
105.        DateUtils.formatDate(logoutTime), pad[padIndex++]));
106.        System.out.print(StringUtils.rightPad((
107.        logoutTime - loginTime) + "", pad[padIndex++]));
108.        System.out.println(StringUtils.rightPad(
109.        host, pad[padIndex++]));
110.    }
111.
112. }

```

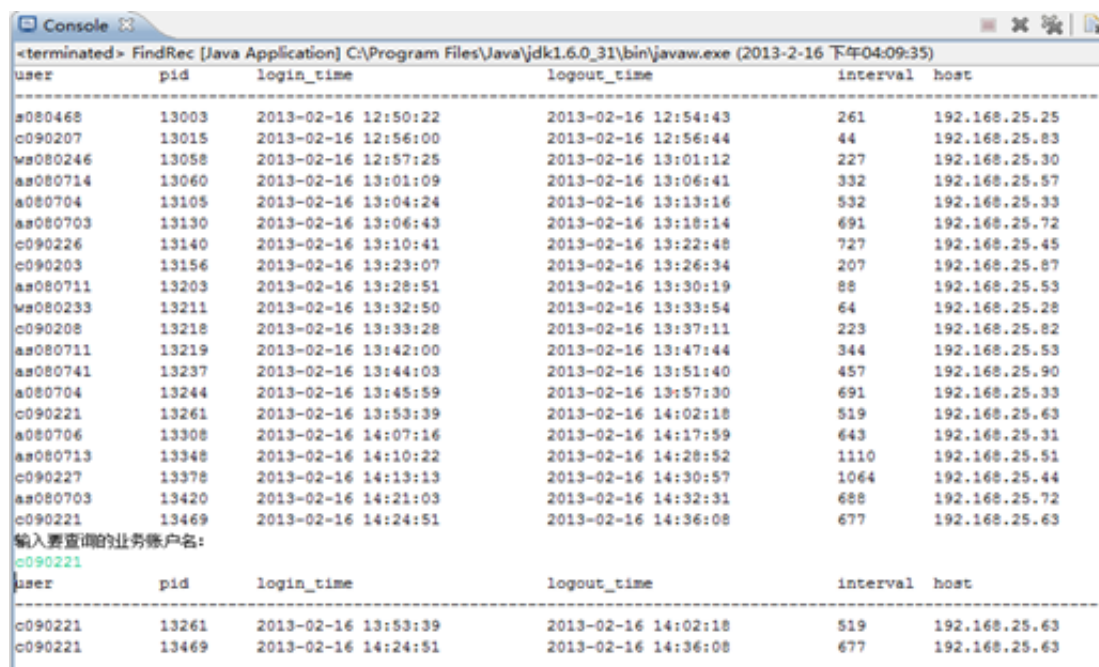
2.4 扩展

熟练掌握并完成DMSV1.0日志数据的匹配算法。

3 DMS V1.0 登录数据查询算法

3.1 问题

DMS 1.0 业务中需要根据用户名查询某用户的登录记录并打印查询结果。控制台输出情况如图-61所示：



user	pid	login_time	logout_time	interval	host
a080468	13003	2013-02-16 12:50:22	2013-02-16 12:54:43	261	192.168.25.25
c090207	13015	2013-02-16 12:56:00	2013-02-16 12:56:44	44	192.168.25.83
wa080246	13058	2013-02-16 12:57:25	2013-02-16 13:01:12	227	192.168.25.30
aa080714	13060	2013-02-16 13:01:09	2013-02-16 13:06:41	332	192.168.25.57
a080704	13105	2013-02-16 13:04:24	2013-02-16 13:13:16	532	192.168.25.33
aa080703	13130	2013-02-16 13:06:43	2013-02-16 13:18:14	691	192.168.25.72
c090226	13140	2013-02-16 13:10:41	2013-02-16 13:22:48	727	192.168.25.45
c090203	13156	2013-02-16 13:23:07	2013-02-16 13:26:34	207	192.168.25.87
aa080711	13203	2013-02-16 13:28:51	2013-02-16 13:30:19	88	192.168.25.53
wa080233	13211	2013-02-16 13:32:50	2013-02-16 13:33:54	64	192.168.25.28
c090208	13218	2013-02-16 13:33:28	2013-02-16 13:37:11	223	192.168.25.82
aa080711	13219	2013-02-16 13:42:00	2013-02-16 13:47:44	344	192.168.25.53
aa080741	13237	2013-02-16 13:44:03	2013-02-16 13:51:40	457	192.168.25.90
a080704	13244	2013-02-16 13:45:59	2013-02-16 13:57:30	691	192.168.25.33
c090221	13261	2013-02-16 13:53:39	2013-02-16 14:02:18	519	192.168.25.63
a080706	13308	2013-02-16 14:07:16	2013-02-16 14:17:59	643	192.168.25.31
aa080713	13348	2013-02-16 14:10:22	2013-02-16 14:28:52	1110	192.168.25.51
c090227	13378	2013-02-16 14:13:13	2013-02-16 14:30:57	1064	192.168.25.44
aa080703	13420	2013-02-16 14:21:03	2013-02-16 14:32:31	688	192.168.25.72
c090221	13469	2013-02-16 14:24:51	2013-02-16 14:36:08	677	192.168.25.63

输入要查询的业务账户名:
c090221

user	pid	login_time	logout_time	interval	host
c090221	13261	2013-02-16 13:53:39	2013-02-16 14:02:18	519	192.168.25.63
c090221	13469	2013-02-16 14:24:51	2013-02-16 14:36:08	677	192.168.25.63

图-61

3.2 方案

在匹配成功的记录中查找指定用户名的登录日志, 如果有多条登录日志记录, 就显示多条登录日志记录。算法过程如图-62所示。

图-62

3.3 实现

新建FindRec类, 在类中添加find方法。

```
01.      /**
02.      * 查询某用户全部的登录匹配结果.
03.      *
04.      * @param name
05.      *      查询用户名
06.      * @param entrys
07.      *      匹配结果索引数组
08.      * @param users
09.      *      日志数据 用户名
```

```

10.      * @param pids
11.      *          日志数据 进程ID
12.      * @param types
13.      *          日志类型
14.      * @param times
15.      *          日志数据, 登录时间
16.      * @param hosts
17.      *          日志数据, 登录主机名, 一般是IP
18.      */
19.      public static void find(String name, int[] entrys, String[] users,
20.          int[] pids, short[] types, int[] times, String[] hosts) {
21.          MatchLogs.printRecHeader();
22.          for (int i = 0; i < entrys.length; i += 2) {
23.              int login = entrys[i];
24.              int logout = entrys[i + 1];
25.              if (users[login].equals(name)) {
26.                  MatchLogs.printRec(users[login], pids[login],
27.                      times[login], times[logout], hosts[login]);
28.              }
29.          }
30.      }

```

在FindRec类中添加main方法,进行测试。

```

01.      public static void main(String[] args) {
02.          byte[] logs = LogDataSource.getLog();
03.
04.          int count = logs.length / 372; // 登录记录数量
05.
06.          String[] users = new String[count];
07.          int[] pids = new int[count];
08.          short[] types = new short[count];
09.          int[] times = new int[count];
10.          String[] hosts = new String[count];
11.
12.          ParseLogs.parseLogs(logs, users, pids, types, times, hosts);
13.          // ParseLogs.printLogs(users, pids, types, times, hosts);

```

```
14.
15.         int[] entrys = new int[count];
16.
17.         MatchLogs.matchLogs(entrys, users, pids, types);
18.         MatchLogs.list(entrys, users, pids, types, times, hosts);
19.
20.         System.out.println("输入要查询的业务账户名: ");
21.         Scanner scanner = new Scanner(System.in);
22.         String name = scanner.next().trim();
23.
24.         find(name, entrys, users, pids, types, times, hosts);
25.
26.         scanner.close();
27.
28.     }
```

FindRec类的完整参考代码如下：

```
01.     public class FindRec {
02.         public static void main(String[] args) {
03.             byte[] logs = LogDataSource.getLog();
04.
05.             int count = logs.length / 372; // 登录记录数量
06.
07.             String[] users = new String[count];
08.             int[] pids = new int[count];
09.             short[] types = new short[count];
10.             int[] times = new int[count];
11.             String[] hosts = new String[count];
12.
13.             ParseLogs.parseLogs(logs, users, pids, types, times, hosts);
14.             // ParseLogs.printLogs(users, pids, types, times, hosts);
15.
16.             int[] entrys = new int[count];
17.
18.             MatchLogs.matchLogs(entrys, users, pids, types);
19.             MatchLogs.list(entrys, users, pids, types, times, hosts);
```

```
20.
21.         System.out.println("输入要查询的业务账户名: ");
22.         Scanner scanner = new Scanner(System.in);
23.         String name = scanner.next().trim();
24.
25.         find(name, entrys, users, pids, types, times, hosts);
26.
27.         scanner.close();
28.
29.     }
30.
31.     /**
32.      * 查询某用户全部的登录匹配结果.
33.      *
34.      * @param name
35.      *          查询用户名
36.      * @param entrys
37.      *          匹配结果索引数组
38.      * @param users
39.      *          日志数据 用户名
40.      * @param pids
41.      *          日志数据 进程ID
42.      * @param types
43.      *          日志类型
44.      * @param times
45.      *          日志数据, 登录时间
46.      * @param hosts
47.      *          日志数据, 登录主机名, 一般是IP
48.      */
49.     public static void find(String name, int[] entrys, String[] users,
50.         int[] pids, short[] types, int[] times, String[] hosts) {
51.         MatchLogs.printRecHeader();
52.         for (int i = 0; i < entrys.length; i += 2) {
53.             int login = entrys[i];
54.             int logout = entrys[i + 1];
55.             if (users[login].equals(name)) {
56.                 MatchLogs.printRec(users[login],
57.                     pids[login], times[login],
58.                         times[logout], hosts[login]);
```

```

59.         }
60.     }
61. }
62. }

```

3.4 扩展

熟练掌握并完成DMSV1.0登录数据的查询算法。

4 DMS V1.0 系统整合

4.1 问题

DMS 1.0 系统提供数据浏览和查询功能。程序运行后，先显示所有的登录记录，用户可以选择后续操作。后续操作分为三种方式：

1.显示全部记录：界面录入1后，即选择打印所有的登录数据；

2.查询登录记录：界面录入2后，即选择查询某用户的登录数据，则需要输入用户名，然后界面会打印显示该用户的登录数据；

0.退出：界面录入0后，即选择退出，程序结束。

DMS 1.0 的界面效果如图 - 63所示。

```

<terminated> Dms [Java Application] C:\Program Files\Java\jdk1.6.0_31\bin\javaw.exe (2013-2-16 下午04:48:46)
as080714      13060      2013-02-16 13:40:20      2013-02-16 13:45:52      332      192.168.25.57
a080704      13105      2013-02-16 13:43:35      2013-02-16 13:52:27      532      192.168.25.33
as080703      13130      2013-02-16 13:45:54      2013-02-16 13:57:25      691      192.168.25.72
c090226      13140      2013-02-16 13:49:52      2013-02-16 14:01:59      727      192.168.25.45
c090203      13156      2013-02-16 14:02:18      2013-02-16 14:05:45      207      192.168.25.87
as080711      13203      2013-02-16 14:08:02      2013-02-16 14:09:30      88      192.168.25.53
ws080233      13211      2013-02-16 14:12:01      2013-02-16 14:13:05      64      192.168.25.28
c090208      13218      2013-02-16 14:12:39      2013-02-16 14:16:22      223      192.168.25.82
as080711      13219      2013-02-16 14:21:11      2013-02-16 14:26:55      344      192.168.25.53
as080741      13237      2013-02-16 14:23:14      2013-02-16 14:30:51      457      192.168.25.90
a080704      13244      2013-02-16 14:25:10      2013-02-16 14:36:41      691      192.168.25.33
c090221      13261      2013-02-16 14:32:50      2013-02-16 14:41:29      519      192.168.25.63
a080706      13308      2013-02-16 14:46:27      2013-02-16 14:57:10      643      192.168.25.31
as080713      13348      2013-02-16 14:49:33      2013-02-16 15:08:03      1110     192.168.25.51
c090227      13378      2013-02-16 14:52:24      2013-02-16 15:10:08      1064     192.168.25.44
as080703      13420      2013-02-16 15:00:14      2013-02-16 15:11:42      688      192.168.25.72
c090221      13469      2013-02-16 15:04:02      2013-02-16 15:15:19      677      192.168.25.63
DMS v01>请选择功能: 1—显示全部记录 2—查询登录记录 0—退出:2
DMS v01>输入用户名:c090221
user      pid      login_time      logout_time      interval      host
-----
c090221      13261      2013-02-16 14:32:50      2013-02-16 14:41:29      519      192.168.25.63
c090221      13469      2013-02-16 15:04:02      2013-02-16 15:15:19      677      192.168.25.63
DMS v01>请选择功能: 1—显示全部记录 2—查询登录记录 0—退出:0
DMS v01>再见!

```

图-63

4.2 方案

创建Dms类，整合ParseLogs类，MatchLogs类，FindRec类中的方法到Dms类中。

Dms类中main方法实现流程如图-64所示：

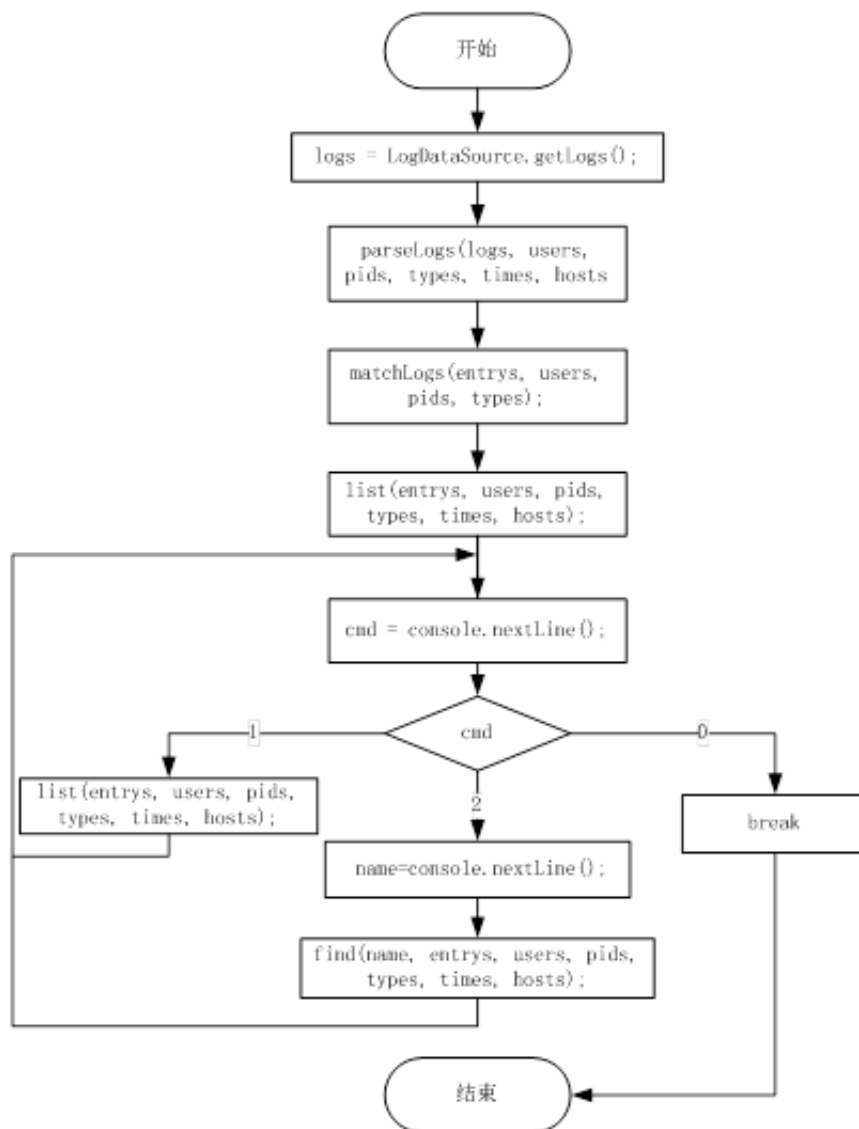


图-64

4.3 实现

新建Dms类，整合ParseLogs类中的parseLogs方法，toInt方法，toShort方法，toString方法到Dms类中。

```

01.     public class Dms {
02.         /**
03.          * 解析日志数据logs 解析结果存储到5个数组中 解析规则 每372个字节数据为一
04.          *
05.          * @param logs
06.          *          原始日志数据，每372个字节为一个日志记录
07.          * @param users
08.          *          解析以后的登录用户名数组
09.          * @param pids
10.          *          解析以后的用户进程号数组
11.          * @param types

```

```

12.      *           存储解析以后登录类型的数组
13.      * @param times
14.      *           存储解析以后登录时间的数组
15.      * @param hosts
16.      *           存储解析以后用户主机名的数组
17.      */
18.      public static void parseLogs(byte[] logs, String[] users, int[] pids,
19.          short[] types, int[] times, String[] hosts) {
20.          for (int i = 0, start = 0; i < users.length; i++, start += 372)
21.          {
22.              users[i] = toString(logs, start + 0, 32);
23.              pids[i] = toInt(logs, start + 68);
24.              types[i] = toShort(logs, start + 72);
25.              times[i] = toInt(logs, start + 80);
26.              hosts[i] = toString(logs, start + 114, 257);
27.          }
28.      }
29.      /**
30.      * 将byte数组中连续的4个byte数据转换为一个int类型数据.
31.      *
32.      * @param bytes
33.      *           需要读取的元素数组
34.      * @param offset
35.      *           转换的起始偏移位置
36.      * @return 将连续4个byte转换为一个int数据
37.      */
38.      public static int toInt(byte[] bytes, int offset) {
39.          int d1 = bytes[offset + 0] & 0xff;
40.          int d2 = bytes[offset + 1] & 0xff;
41.          int d3 = bytes[offset + 2] & 0xff;
42.          int d4 = bytes[offset + 3] & 0xff;
43.          int i = (d1 << 24) + (d2 << 16) + (d3 << 8) + d4;
44.          return i;
45.      }
46.
47.      /**
48.      * 将byte数组中连续的2个byte数据转换为一个short类型数据.
49.      *
50.      * @param bytes

```

```

51.      *          需要读取的元素数组
52.      * @param offset
53.      *          转换的起始偏移位置
54.      * @return 将连续两个byte转换为一个short数据
55.      */
56.      public static short toShort(byte[] bytes, int offset) {
57.          int d1 = bytes[offset + 0];
58.          int d2 = bytes[offset + 1];
59.          d1 &= 0xff;
60.          d2 &= 0xff;
61.          d1 <<= 8;
62.          d2 <<= 0;
63.          int i = d1 + d2;
64.          return (short) i;
65.      }
66.
67.      /**
68.       * 将byte数组中从offset位置开始, 解析length个字节为String类型数据.
69.       *
70.       * @param bytes
71.       *          需要读取的元素数组
72.       * @param offset
73.       *          转换的起始偏移位置
74.       * @param length
75.       *          连续的的byte数
76.       * @return String类型数据
77.       */
78.      public static String toString(byte[] bytes, int offset, int length)
79.      {
80.          return new String(bytes, offset, length).trim();
81.      }
82.  }

```

整合MatchLogs类中的matchLogs方法，list方法，printRecHeader方法，printRec方法到Dms类中。

```

01.      /**

```

```

02.      * 匹配登录登出记录为一对登录数据，每对数据第一条为登录数据，第二条为登出
03.      * 匹配结果存储到entrys中。
04.      * @param entrys 匹配结果数组
05.      * @param users 登录用户名数组
06.      * @param pids 用户进程号数组
07.      * @param types 登录类型的数组
08.      */
09.      public static void matchLogs(int[] entrys, String[] users, int[] pids,
10.          int index = 0;
11.          for (int i = 0; i < users.length; i++) { // 迭代每个记录
12.              // 如果是登录记录(type==7) 就从当前位置开始向后查找登出记录
13.              if (types[i] == 7) {
14.                  entrys[index++] = i; // 先记录登录记录位置
15.                  // 从登录记录位置(i+1)开始向后查找登出记录
16.                  for (int j = i + 1; j < users.length; j++) {
17.                      // 查找登出记录的条件：用户名相同，进程号相同，
18.                      // 记录类型为登出(type==8)
19.                      if (users[i].equals(users[j]) && pids[i] == pids[j]
20.                          && types[j] == 8) { // 找到logout记录
21.                          entrys[index++] = j; // 记录登出位置
22.                          break; // 找到就可以结束查找过程
23.                      }
24.                  }
25.              }
26.          }
27.      }
28.
29.      /**
30.      * 打印全部匹配后的登录会话
31.      * @param entrys 匹配成功的日志对索引数组
32.      * @param users 解析以后的登录用户名数组
33.      * @param pids 解析以后的用户进程号数组
34.      * @param types 解析以后登录类型的数组
35.      * @param hosts 解析以后用户主机名的数组
36.      */
37.      public static void list(int[] entrys, String[] users, int[] pids,
38.          short[] types, int[] times, String[] hosts) {
39.          printRecHeader();
40.          for (int i = 0; i < entrys.length; i += 2) {

```

```

41.         int login = entrys[i];
42.         int logout = entrys[i + 1];
43.         printRec(users[login], pids[login], times[login],
44. times[logout], hosts[login]);
45.     }
46. }
47.
48. /**
49.  * 打印向控制台输出的头信息即第一行信息
50.  */
51. public static void printRecHeader() {
52.     String[] headers = { "user", "pid", "login_time", "logout_time",
53.         "interval", "host" };
54.     int[] pad = { 15, 10, 30, 30, 10, 20 };
55.     int total = 0;
56.     for (int i = 0; i < headers.length; i++) {
57.         System.out.print(StringUtils.rightPad(headers[i],
58. pad[i]));
59.         total += pad[i];
60.     }
61.     System.out.println();
62.     System.out.println(StringUtils.repeat("-", total));
63. }
64.
65. /**
66.  * 打印一条匹配后的登录会话
67.  * @param user 登录用户名
68.  * @param pid 用户进程号
69.  * @param loginTime 登录时间
70.  * @param logoutTime 登出时间
71.  * @param host 用户主机名
72.  */
73. public static void printRec(String user, int pid, int loginTime,
74.     int logoutTime, String host) {
75.     int[] pad = { 15, 10, 30, 30, 10, 20 };
76.     int padIndex = 0;
77.     System.out.print(StringUtils.rightPad(user,
78. pad[padIndex++]));
79.     System.out.print(StringUtils.rightPad(pid

```

```

80.     + """, pad[padIndex++]));
81.         System.out.print(StringUtils.rightPad(
82.     DateUtils.formatDate(loginTime), pad[padIndex++]));
83.         System.out.print(StringUtils.rightPad(
84.     DateUtils.formatDate(logoutTime), pad[padIndex++]));
85.         System.out.print(StringUtils.rightPad((logoutTime - loginTime)
86.
87.     + """, pad[padIndex++]));
88.         System.out.println(StringUtils.rightPad(host,
89.     pad[padIndex++]));
90.     }

```

整合FindRec类中find方法到Dms类中。

```

01.     /**
02.     * 查询某用户全部的登录匹配结果.
03.     *
04.     * @param name
05.     *         查询用户名
06.     * @param entrys
07.     *         匹配结果索引数组
08.     * @param users
09.     *         日志数据 用户名
10.     * @param pids
11.     *         日志数据 进程ID
12.     * @param types
13.     *         日志类型
14.     * @param times
15.     *         日志数据, 登录时间
16.     * @param hosts
17.     *         日志数据, 登录主机名, 一般是IP
18.     */
19.     public static void find(String name, int[] entrys, String[] users,
20.         int[] pids, short[] types, int[] times, String[] hosts) {
21.         MatchLogs.printRecHeader();
22.         for (int i = 0; i < entrys.length; i += 2) {
23.             int login = entrys[i];

```

```

24.         int logout = entrys[i + 1];
25.         if (users[login].equals(name)) {
26.             MatchLogs.printRec(users[login], pids[login], times[login]
27.                 times[logout], hosts[login]);
28.         }
29.     }
30. }

```

[Top](#)

在Dms类中添加main方法。

```

01.     public static void main(String[] args) {
02.         // 从日志数据源获取日志数据
03.         byte[] logs = LogDataSource.getLog();
04.         // 计算日志记录数量
05.         int count = logs.length / 372; // 登录记录数量
06.         // 根据日志数量创建解析结果存储数组
07.         String[] users = new String[count];
08.         int[] pids = new int[count];
09.         short[] types = new short[count];
10.         int[] times = new int[count];
11.         String[] hosts = new String[count];
12.         // 解析日志，将结果保存到结果存储数组中
13.         parseLogs(logs, users, pids, types, times, hosts);
14.
15.         list(entrys, users, pids, types, times, hosts);
16.
17.         int[] entrys = new int[count];
18.         // 匹配日志(根据用户名，进程ID，日志类型)，将匹配结果存储到 entrys
19.         matchLogs(entrys, users, pids, types);
20.
21.         Scanner console = new Scanner(System.in);
22.         while (true) {
23.             System.out.print("DMS v01>请选择功能：1——显示全部记录 2——");
24.             // 从控制台，读取一条用户命令
25.             String cmd = console.nextLine();
26.             if (cmd.equals("1")) {
27.                 // 如果是命令是"1" 就列出全部匹配结果

```

```

28.         list(entrys, users, pids, types, times, hosts);
29.     } else if (cmd.equals("2")) {
30.         // 如果是命令是"2" 就根据用户名列出全部的用户登录记录
31.         System.out.print("DMS v01>输入用户名:");
32.         String name = console.nextLine();
33.         find(name, entrys, users, pids, types, times, hosts);
34.     } else if (cmd.equals("0")) {
35.         System.out.println("DMS v01>再见!");
36.         break;
37.     } else {
38.         System.out.println("DMS v01>不识别命令!");
39.     }
40. }
41. }

```

Dms类完整参考代码如下所示：

```

01.     public class Dms {
02.         public static void main(String[] args) {
03.             // 从日志数据源获取日志数据
04.             byte[] logs = LogDataSource.getLogLogs();
05.             // 计算日志记录数量
06.             int count = logs.length / 372; // 登录记录数量
07.             // 根据日志数量创建解析结果存储数组
08.             String[] users = new String[count];
09.             int[] pids = new int[count];
10.             short[] types = new short[count];
11.             int[] times = new int[count];
12.             String[] hosts = new String[count];
13.             // 解析日志，将结果保存到结果存储数组中
14.             parseLogs(logs, users, pids, types, times, hosts);
15.
16.             int[] entrys = new int[count];
17.             // 匹配日志(根据用户名，进程ID，日志类型)，将匹配结果存储到，entrys
18.             matchLogs(entrys, users, pids, types);
19.
20.             Scanner console = new Scanner(System.in);

```



```

21.         while (true) {
22.             System.out.print("DMS v01>请选择功能: 1——显示全部记录 2——查
23.             // 从控制台, 读取一条用户命令
24.             String cmd = console.nextLine();
25.             if (cmd.equals("1")) {
26.                 // 如果是命令是"1" 就列出全部匹配结果
27.                 list(entrys, users, pids, types, times, hosts);
28.             } else if (cmd.equals("2")) {
29.                 // 如果是命令是"2" 就根据用户名列出全部的用户登录记录
30.                 System.out.print("DMS v01>输入用户名:");
31.                 String name = console.nextLine();
32.                 find(name, entrys, users, pids, types, times, hosts);
33.             } else if (cmd.equals("0")) {
34.                 System.out.println("DMS v01>再见!");
35.                 break;
36.             } else {
37.                 System.out.println("DMS v01>不识别命令!");
38.             }
39.         }
40.     }
41.
42.     /**
43.     * 查询某用户全部的登录匹配结果.
44.     *
45.     * @param name
46.     *         查询用户名
47.     * @param entrys
48.     *         匹配结果索引数组
49.     * @param users
50.     *         日志数据 用户名
51.     * @param pids
52.     *         日志数据 进程ID
53.     * @param types
54.     *         日志类型
55.     * @param times
56.     *         日志数据, 登录时间
57.     * @param hosts
58.     *         日志数据, 登录主机名, 一般是IP
59.     */

```

```
60.     public static void find(String name, int[] entrys, String[] users,
61.         int[] pids, short[] types, int[] times, String[] hosts) {
62.         printRecHeader();
63.         for (int i = 0; i < entrys.length; i += 2) {
64.             int login = entrys[i];
65.             int logout = entrys[i + 1];
66.             if (users[login].equals(name)) {
67.                 printRec(users[login], pids[login], times[login],
68.                     times[logout], hosts[login]);
69.             }
70.         }
71.     }
72.     /**
73.      * 打印全部匹配后的登录会话
74.      * @param entrys 匹配成功的日志对索引数组
75.      * @param users 解析以后的登录用户名数组
76.      * @param pids 解析以后的用户进程号数组
77.      * @param types 解析以后登录类型的数组
78.      * @param hosts 解析以后用户主机名的数组
79.      */
80.     public static void list(int[] entrys, String[] users, int[] pids,
81.         short[] types, int[] times, String[] hosts) {
82.         printRecHeader();
83.         for (int i = 0; i < entrys.length; i += 2) {
84.             int login = entrys[i];
85.             int logout = entrys[i + 1];
86.             printRec(users[login], pids[login],
87. times[login], times[logout], hosts[login]);
88.         }
89.     }
90.
91.     /**
92.      * 匹配登录登出记录为一对登录数据，每对数据第一条为登录数据，第二条为登出
93.      * 匹配结果存储到entrys中。
94.      *
95.      * @param entrys
96.      * @param users
97.      * @param pids
98.      * @param types
```

```

99.         */
100.     public static void matchLogs(int[] entrys, String[] users, int[] pids,
101.        int index = 0;
102.        for (int i = 0; i < users.length; i++) { // 迭代每个记录
103.            // 如果是登录记录(type==7) 就从当前位置开始向后查找登出记录
104.            if (types[i] == 7) {
105.                entrys[index++] = i; // 先记录登录记录位置
106.                // 从登录记录位置(i+1)开始向后查找登出记录
107.                for (int j = i + 1; j < users.length; j++) {
108.                    // 查找登出记录的条件：用户名相同, 进程号相同,
109.                    // 记录类型为登出(type==8)
110.                    if (users[i].equals(users[j]) && pids[i] == pids[j]
111.                        && types[j] == 8) { // 找到logout记录
112.                        entrys[index++] = j; // 记录登出位置
113.                        break; // 找到就可以结束查找过程
114.                    }
115.                }
116.            }
117.        }
118.    }
119.
120.    /**
121.     * 解析日志数据logs 解析结果存储到5个数组中 解析规则 每372个字节数
122.     * 据为一个单元解析为一组数据.
123.     *
124.     * @param logs
125.     *         原始日志数据, 每372个字节为一个日志记录
126.     * @param users
127.     *         解析以后的登录用户名数组
128.     * @param pids
129.     *         解析以后的用户进程号数组
130.     * @param types
131.     *         存储解析以后登录类型的数组
132.     * @param times
133.     *         存储解析以后登录时间的数组
134.     * @param hosts
135.     *         存储解析以后用户主机名的数组
136.     */
137.    public static void parseLogs(byte[] logs, String[] users, int[] pids,

```

```

138.         short[] types, int[] times, String[] hosts) {
139.     for (int i = 0, start = 0; i < users.length; i++, start += 372)
140.     {
141.         users[i] = toString(logs, start + 0, 32);
142.         pids[i] = toInt(logs, start + 68);
143.         types[i] = toShort(logs, start + 72);
144.         times[i] = toInt(logs, start + 80);
145.         hosts[i] = toString(logs, start + 114, 257);
146.     }
147. }
148. /**
149.  * 将byte数组中连续的4个byte数据转换为一个int类型数据.
150.  *
151.  * @param bytes
152.  *         需要读取的元素数组
153.  * @param offset
154.  *         转换的起始偏移位置
155.  * @return 将连续4个byte转换为一个int数据
156.  */
157. public static int toInt(byte[] bytes, int offset) {
158.     int d1 = bytes[offset + 0] & 0xff;
159.     int d2 = bytes[offset + 1] & 0xff;
160.     int d3 = bytes[offset + 2] & 0xff;
161.     int d4 = bytes[offset + 3] & 0xff;
162.     int i = (d1 << 24) + (d2 << 16) + (d3 << 8) + d4;
163.     return i;
164. }
165.
166. /**
167.  * 将byte数组中连续的2个byte数据转换为一个short类型数据.
168.  *
169.  * @param bytes
170.  *         需要读取的元素数组
171.  * @param offset
172.  *         转换的起始偏移位置
173.  * @return 将连续两个byte转换为一个short数据
174.  */
175. public static short toShort(byte[] bytes, int offset) {
176.     int d1 = bytes[offset + 0];

```

```

177.         int d2 = bytes[offset + 1];
178.         d1 &= 0xff;
179.         d2 &= 0xff;
180.         d1 <<= 8;
181.         d2 <<= 0;
182.         int i = d1 + d2;
183.         return (short) i;
184.     }
185.     /**
186.      * 将byte数组中从offset位置开始,解析length个字节为String类型数
187.      * 据.
188.      *
189.      * @param bytes
190.      *         需要读取的元素数组
191.      * @param offset
192.      *         转换的起始偏移位置
193.      * @param length
194.      *         连续的的byte字节个数
195.      * @return String类型数据
196.      */
197.     public static String toString(byte[] bytes, int offset, int length)
198.     {
199.         return new String(bytes, offset, length).trim();
200.     }
201.
202.     /**
203.      * 打印向控制台输出的头信息即第一行信息
204.      */
205.     public static void printRecHeader() {
206.         String[] headers = { "user", "pid", "login_time", "logout_time",
207.             "interval", "host" };
208.         int[] pad = { 15, 10, 30, 30, 10, 20 };
209.         int total = 0;
210.         for (int i = 0; i < headers.length; i++) {
211.             System.out.print(StringUtils.rightPad(headers[i],
212.                 pad[i]));
213.             total += pad[i];
214.         }
215.         System.out.println();

```

```
216.         System.out.println(StringUtils.repeat("-", total));
217.     }
218.
219.     /**
220.      * 打印一条匹配后的登录会话
221.      * @param user 登录用户名
222.      * @param pid 用户进程号
223.      * @param loginTime 登录时间
224.      * @param logoutTime 登出时间
225.      * @param host 用户主机名
226.      */
227.     public static void printRec(String user, int pid, int loginTime,
228.         int logoutTime, String host) {
229.         int[] pad = { 15, 10, 30, 30, 10, 20 };
230.         int padIndex = 0;
231.         System.out.print(StringUtils.rightPad(user,
232. pad[padIndex++]));
233.         System.out.print(StringUtils.rightPad(pid
234. + "", pad[padIndex++]));
235.         System.out.print(StringUtils.rightPad(
236. DateUtils.formatDate(loginTime), pad[padIndex++]));
237.         System.out.print(StringUtils.rightPad(
238. DateUtils.formatDate(logoutTime), pad[padIndex++]));
239.         System.out.print(StringUtils.rightPad(
240. (logoutTime - loginTime) + "", pad[padIndex++]));
241.         System.out.println(StringUtils.rightPad(
242. host, pad[padIndex++]));
243.     }
244. }
```

4.4 扩展

熟练掌握并完成DMS V1.0系统的数据浏览和查询功能。