软件工程 1201 班
王靖伟
20125041

1. 书 P97-P98 程序实现

DataSrever.java

```
1  import java.io.*;
2  import java.net.*;
3
4  public class DataSrever {
5
6      public static void main(String[] args) {
7          try {
8              ServerSocket sock = new ServerSocket(6013);
9              while (true) {
10                 Socket client = sock.accept();
11                 PrintWriter pout = new PrintWriter(client.getOutputStream(),
12                         true);
13                 pout.println(new java.util.Date().toString());
14                 client.close();
15             }
16         } catch (IOException ioe) {
17             System.err.println(ioe);
18         }
19     }
20 }
21
```

DataClient.java

```
1  import java.io.*;
2  import java.net.*;
3
4  public class DataClient {
5
6      public static void main(String[] args) {
7          try {
8              Socket sock = new Socket("127.0.0.1", 6013);//书上代码有误
9              InputStream in = sock.getInputStream();//书上代码有误
10             BufferedReader bin = new BufferedReader(new InputStreamReader(in));
11             String line;
12             while ((line = bin.readLine()) != null)
13                 System.out.println(line);
14             sock.close();
15         } catch (IOException ioe) {
16             System.err.println(ioe);
17         }
18     }
19 }
```

DataSrever.java 和 DataClient.java 在 Eclipse 中的包资源管理器位置如图 1 所示：

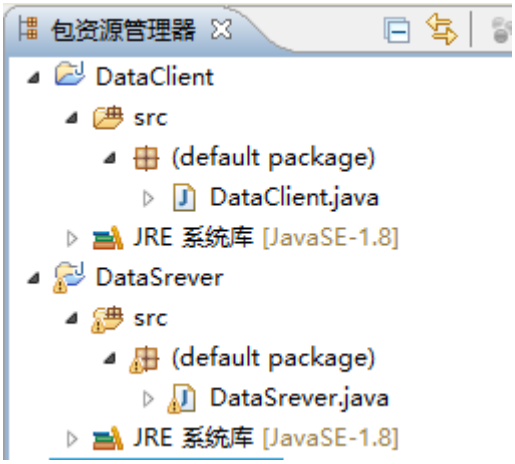

图 1. 包资源管理器中的内容

第一次运行 DataSrever.java 时，控制台无任何显示，如图 2 所示：

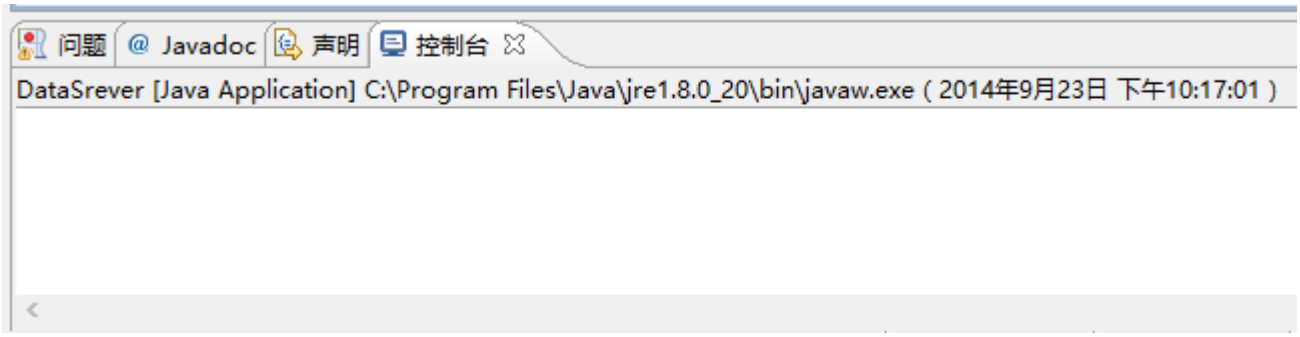

图 2. 第一次运行 DataSrever.java

之后运行 DataClient.java，控制台显示如图 3 所示：

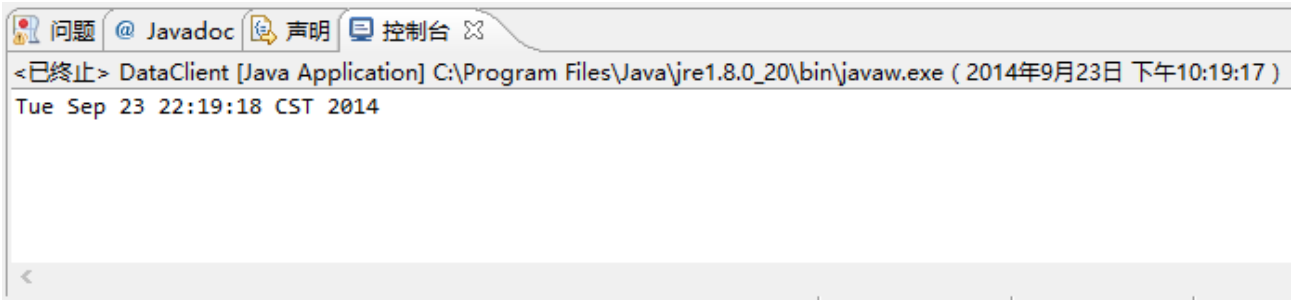

图 3.运行 DataClient.java 的结果

再次运行 DataSrever.java 时，控制台报错，如图 4 所示：

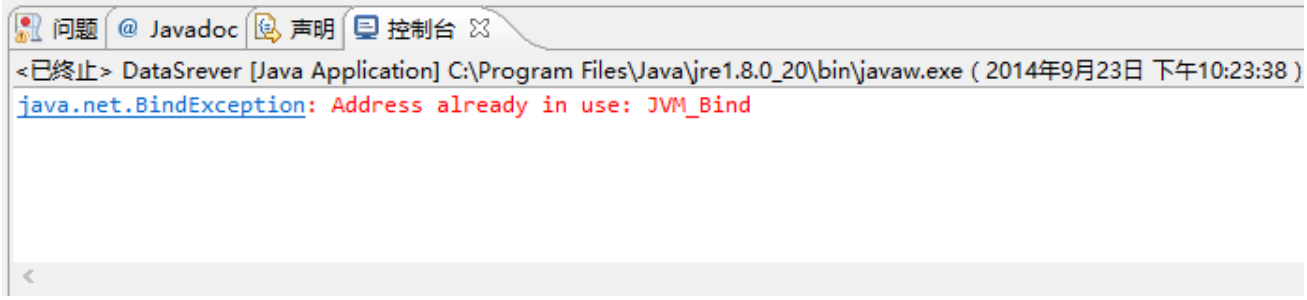

图 4.再次运行 DataSrever.java 的结果

对于图 4 的原因是运行服务端后，服务器端通过 accept()进行死循环监听端口，sock 从没有关闭，所以错误信息显示地址被 JVM_Bind 使用（主要原因是 6013 端口被占用没有释放）。

2. 书 P87 生产者/消费者进程改进

```
item nextProduced

while(true){
    while(((in+1)%BUFFER_SIZE)==out)
    ;
    buffer[in]=nextProduced;
    in=(in+1)%BUFFER_SIZE;
}
```

书 P87 生产者进程

```
item nextConsumed

while(true){
    while(in==out)
    ;
    nextConsumed=buffer[out];
    out=(out+1)%BUFFER_SIZE;
}
```

书 P87 消费者进程

对于上面程序的改进，参考书第六章 P166-P167，解决同时访问共享内存的问题。原理为增加一个整数变量 counter，并初始化为 0。每当向缓冲区增加一项时，counter 就递增；每当从缓冲区移走一项时，counter 就递减。修改后的代码如下：

```
while(true){
    while(counter==BUFFER_SIZE)
    ;
    buffer[in]=nextProduced
    in=(in+1)%BUFFER_SIZE;
    counter++;
}
```

改进后的生产者进程

```
while(true){
    while(counter==0)
    ;
    nextConsumed=buffer[out];
    out=(out+1)%BUFFER_SIZE;
    counter--;
}
```

改进后的消费者进程

3. 书 3.5 小结实例 Windows XP 的阅读总结

Windows XP 采用了共享内存作为提供特定类型消息传递的机制。Windows XP 的消息传递工具称为本地过程调用（LPC）工具。它在位于同一机器的两个进程之间进行通信，类似于 RPC，但是在 XP 中进行了优化，WinXP 使用了端口对象以建立和维护两进程之间的连接。调用子系统的每个客户需要一个通信频道，由端口对象提供且不能继承。通信过程如下：

1. 客户机打开系统的连接端口对象的句柄。
2. 客户机发送连接请求。
3. 服务器创建两个私有通信端口，并返回其中之一的句柄给客户机。
4. 客户机和服务器使用相应端口句柄以发送消息或回调，并等待回答。

如果客户机需要发送更大的消息，可以通过区段对象（构建共享内存）来传递消息。运用这种方法，避免了数据复制。