**Java&J2EE应用与开发**

学号： 201692454

姓名： 高江

大作业得分点

文档质量，程序注释，

编码规范及易读性（10分）：

功能实现（10分）：

异常处理（10分）：

类设计，线程同步（10分）：

大作业成绩（满分40分）：

平时作业成绩（满分60分）：

期末总成绩（满分100分）：

**消息传递机制**

**1.1 消息传递与解析机制**

消息与内容之间用空格隔开以利于消息的Responder进行解析。例如上线消息是”Online A”当Responder收到该消息时就会知道节点名称为A的点已经上线。

**1.2 消息类型**

|  |  |  |
| --- | --- | --- |
| 消息头 | 内容 | 备注 |
| Online | 节点名 | 结点上线时发送的消息 |
| Ack | 节点名 | 新上线节点能够通过该消息知道哪些节点依然在线 |
| Offline | 节点名 | 离线消息 |
| Check | Null | 检查邻居是非在线 |
| Response | 节点名 | 邻居返回的消息类型 |
| ShutDown | 节点名 | 发送邻居的掉线消息 |
| Request | 路径+空格+cost | 发送已经学习到的路径 |
| NeibourNode | 路径+空格+JSON | 回应Request请求 |
| CheckCmd | Null | 用于指定结点输出路径 |
| ReCheckCmd | 最短路径信息 | 用于回复CheckCmd消息 |

表1-2消息类型

**静态成员**

**2.1静态数据**

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 备注 |
| port | int | 端口号 |
| Servicer | DatagramSocket | 代表节点与互联网交换信息的Socket |
| name | String | 节点名 |
| RunningSteady | boolean | 表示节点已经稳定运行 |
| Configs | ArrayList | Config文件中读取的信息 |
| Neibours | ArrayList | 一跳之内能到达的在线的节点的相关信息 |
| Reach | ArrayList | 经过若干跳能够到达的节点的相关信息（包含路径） |
| ShortestPaths | ArrayList | 到达点的最短路径(包含路径) |
| ResponseStaticNodes | HashMap | 备用变量，在Heart类中使用。 |

表2-1 静态数据

StaticNode类内的数据和构造函数：

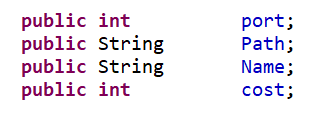
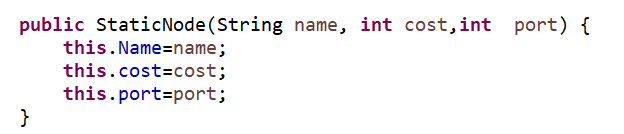
 

图2-1 StaticNode类内的数据和构造函数

**2.2静态函数**

**2.2.1字符串为通信的基本单位**





第一个函数会将Head和Body中间加一个空格拼到一起。并且将数据转换为一个可以经过Socket传递到目的端口号的数据包。到达目的地后，可以使用第二个函数进行解包。解包过程包含了对结果的校验。返回Kind+” ”+Messege字符串。使用以下函数可以方便解析数据：





在我们的程序中，经常要将形如”str1 str2 str3”这样的按照空格隔开的字符串进行分离。取名叫空格串





第一个函数用于获取空格串中包含的子串的个数

第二个函数用于获取空格串s中 第index号串

**2.2.2 类和字符串的相互转化**





如意思所示。

类似的还有





**2.2.3 其他**

这一类函数增强了代码的可读性。



直达邻居的花费。



根据节点名称获得端口号



某个节点已经被计算过最路径

**类实现**

**3.1 主类Node**

用于构造和启动所有线程，拥有大部分的静态数据。

**3.2 Responser**

Responder类继承了Thread，用于无限接收和解析来自各个节点的消息。

收到一个数据包后Responder会做相应的操作，提取出DatagramPacket数据包中的内容。data=Node.*getPacketContent*(dp);

head=Node.*getHead*(data);

body=Node.*getBody*(data);

根据head的值，Responder会将消息进行分类。例如：

**if**(head.equals("Online")) {

…………………

**continue**;

}

通过这个类对数据包进行接收、转发和回应

**3.3 Djikstra**

涉及数据类型：StaticNode类。

涉及消息类型：Request、NodeNeibour。

涉及静态变量：Neibours(ArrayList)，ShortestPath(ArrayList)，Reach(ArrayList)



图3-1 整体逻辑

**的消息请求和返回机制：**

**请求数据**（Request）：比如A经过B向C请求数据，消息格式就是” Request ABCCBA 0”其中ABC代表请求路径，CBA代表返回路径，0代表当前花费的累加这里记为(int)cost。当A将数据包发送给B的时候，A会在cost中加上A到B的cost，同样的B到C也是。到C时，C会收到一个“Request CCBA （int）cost”的数据包。这个时候C会做两件事，第一是将Neibours数组复制一份，然后再将收到的cost加到每一个的cost之中。第二就是将包之中的路径信息倒转，赋值到Path String之中。做完这两件事后C就构造好了一个数组(ArrayList) al，C会将返回路径(“CBA”)加上一个空格在加上al解析成字符串回传。

对于B来说，他就是一个负责转发的节点。对于C来说，他最终发出的信息就会是”NeibourNode CBA {D ABCD 6 2003}”

对于每个点来说，数据包要么是发给自己的，要么是发给邻居的。如果这个包时发给邻居的，那么数据包就负责解析和转发，就比如B收到” Request BCCBA (int)cost”时他就会将自己去掉，将剩下的信息转发给下一个节点” Request CCBA (int)cost”。

如果是发给自己的那么就如同C那样。该过程大部分工作在Responser中完成。

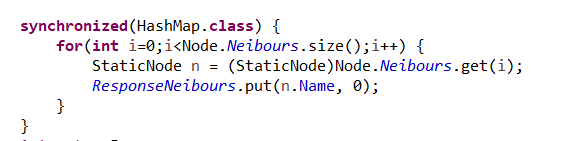
**返回数据**(NeibourNode)：对于返回的数据，这里存在有三种情况。第一就是发出。如上文的C那样。第二就是路过如同上文中的B。第三就是到达，当到达目的地时，目的节点会将字符串解析回数组对象记为al。并且与Reach数组一一对应。当al中有和Reach中同名的节点并且al的cost低于Reach的cost，Reach中的节点就会被替换为al中的节点。同样的大部分工作在Responser中完成。

**3.4 Heart类**

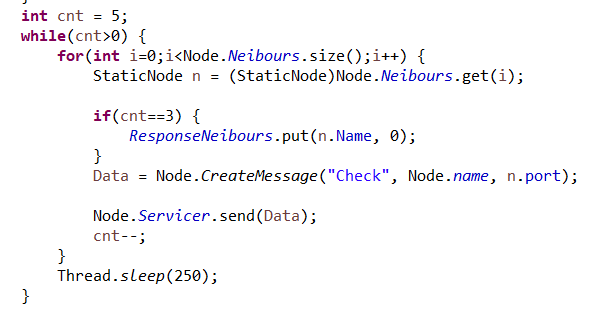
数据成员：一个集合映射



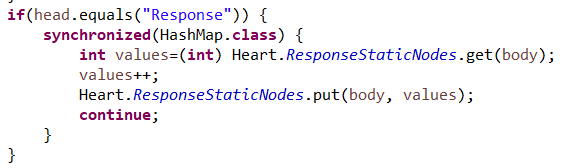
初始化ReponseStaticNodes,由于访问了Node类的共有变量，所以这里要加上互斥锁



接着连续向邻居发包五次



邻居收到Check包后，在Responser类里面对统计心跳的集合进行计数增加

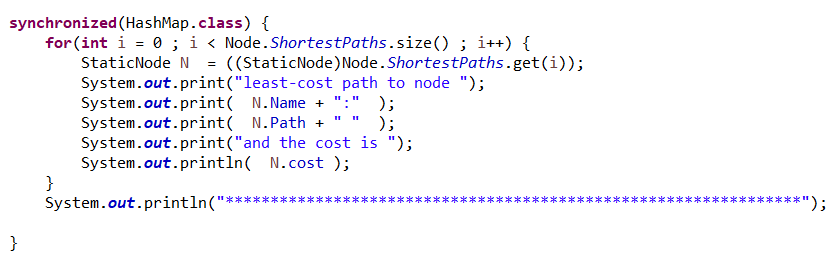


之后回到heart里面进行判断，若发现统计结果中有一个邻居为0，则证明该节点已经死了，之后向邻居广播ShutDown包，从而从网络中删除该节点。

**3.5 Printer类**

此类负责一件事情，就是每隔一段时间（20s）,输出自己的路径消息，即输出ShortestPaths中的信息。

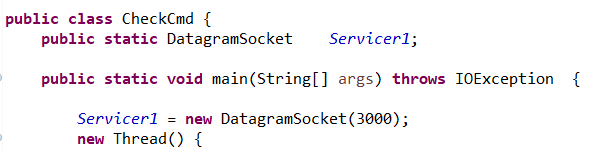
由于同样涉及到了共享变量，也要进行加锁保护。



**3.6 CheckCmd类**

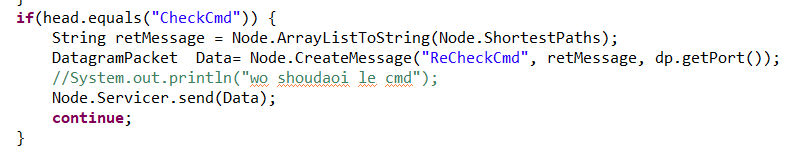
由于六个进程是在同步执行的，每个进程在运行的时候又会有不定时的输出，所以不便于对命令进行输入，为此，我们专门开启了一个进程负责接收命令行的参数，就是要询问的结点信息，消息头记为CheckCmd

，当结点收到这个消息的时候将自己的路径消息打包发送给询问他的进程，之后再询问的进程进行输出，这样就解决了命令行输出与输入冲突的问题。



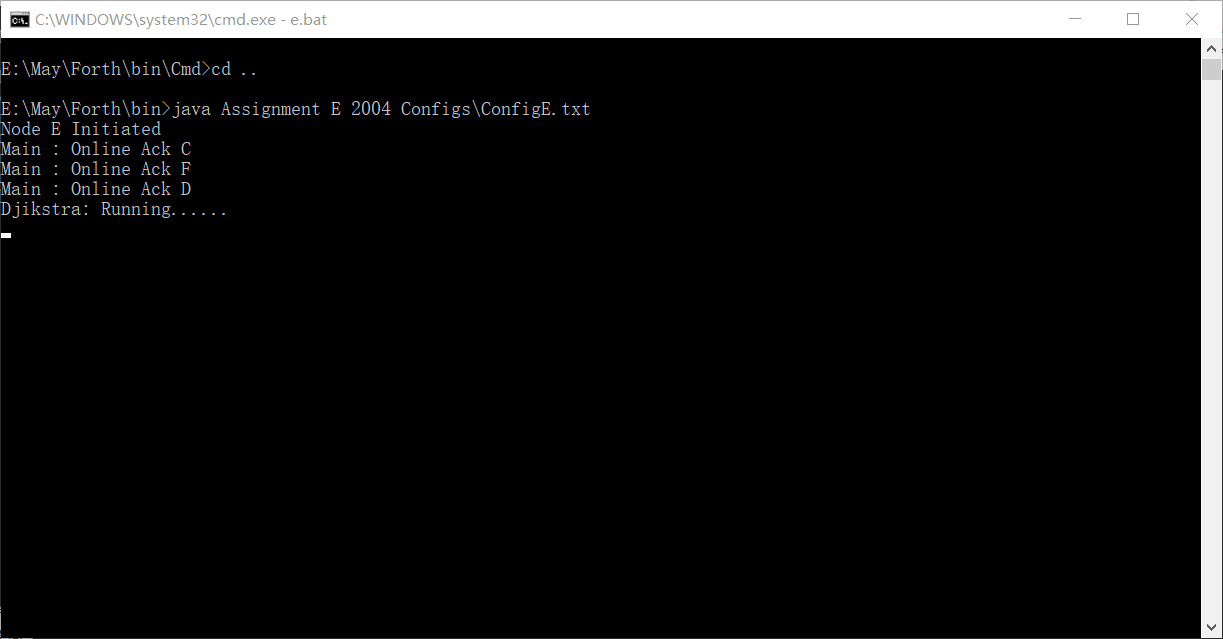
CheckCmd类打开3000端口，通过这个端口向其他结点发送询问消息。

当一个进程收到CheckCmd消息的时候，将自己的信息打包：发送回去。

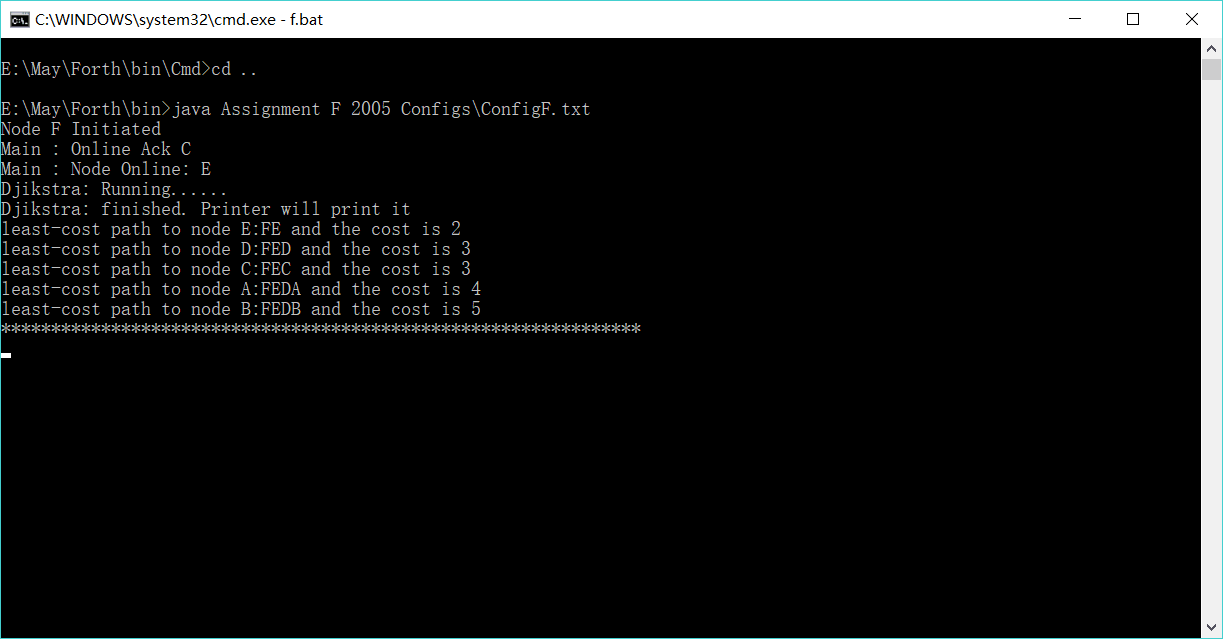


**功能演示**

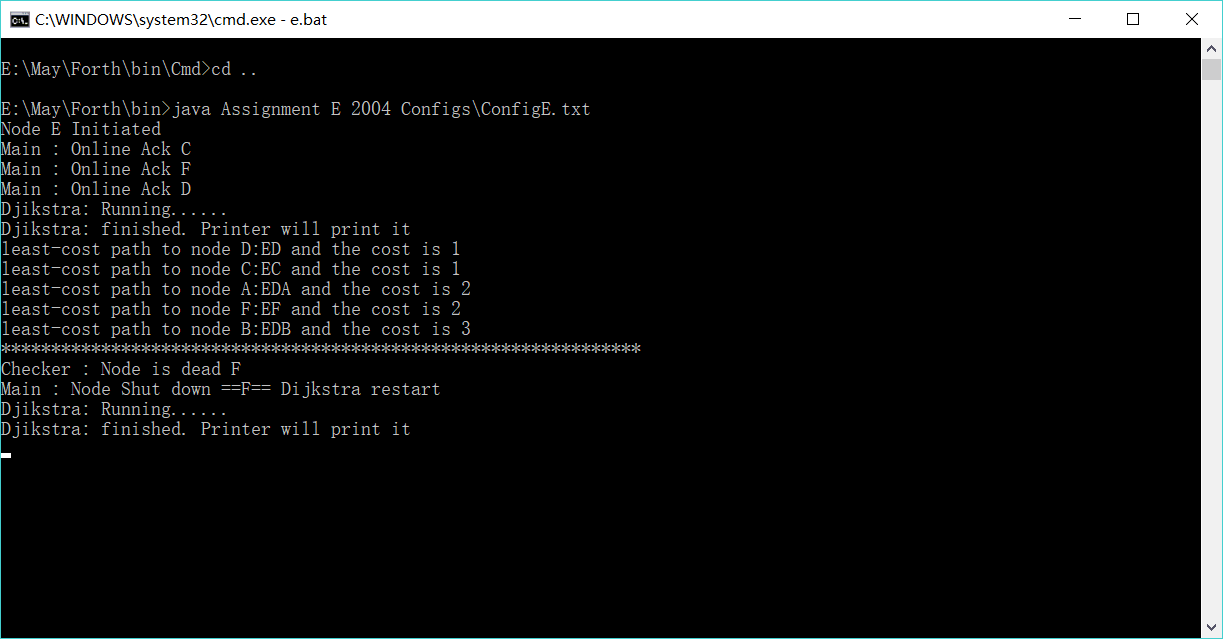
**4.1初始化**



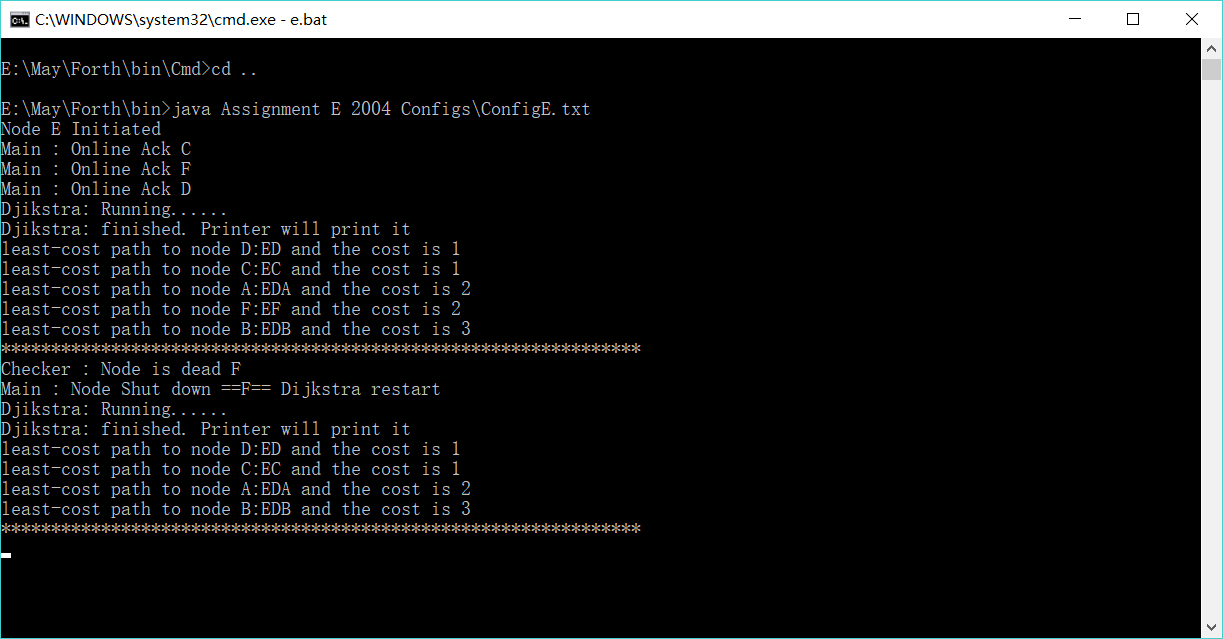
**4.2迪杰斯特拉开始运行**



**4.3结点掉线**



**4.4重启迪杰斯特拉算法**



**4.5询问A结点的信息**

