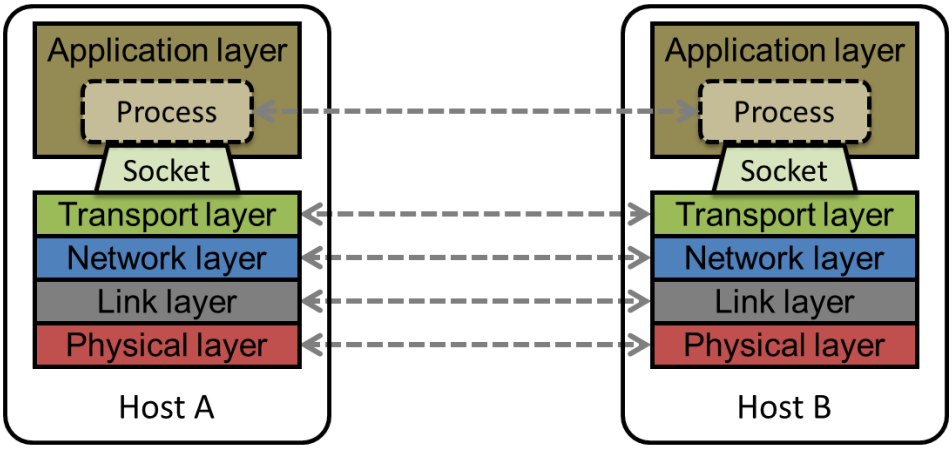
Java Socket/多线程/线程池编程讲义

西安电子科技大学　李龙海

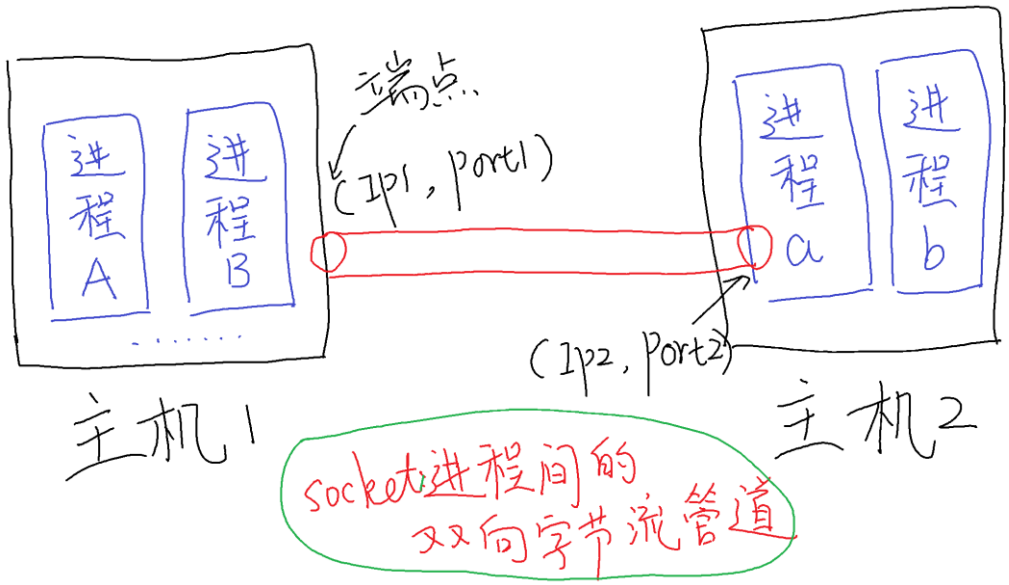
一. Socket 基本概念

1. **什么是 Socket：传输层和网络层提供给应用层的标准化服务接口（或称为编程接口）。**应用程序利用 Socket 接口，可以实现不同主机中不同进程之间的点对点通信。应用程序不用关心底层网络通信细节，比如链路中经过了几个路由器、有线网络还是无线网络等。应用程序只是感觉在跨主机的两个进程之间搭建了一个**端到端的虚拟双向通信管道**。



2. Socket 分类：

（1）流式 Socket：对 TCP 面向连接传输服务的封装。创建一个流式 Socket，相当于在跨主机的两个进程之间搭建了一个端到端的虚拟双向通信管道。



一个流式 Socket 的一个端点用一个二元组表示（IP 地址，端口号）。

三次握手：

滑动窗口协议：

流式 Socket 的发送函数向应用层报告某个消息发送成功之后，该消息就一定达到了接收方的接收缓冲区。（但不保证对方的应用程序一定成功接收了该消息。）

（2）**数据报 Socket：**对 UDP 传输服务的封装。数据报 Socket 的发送函数向应用层报告某个消息完毕之后，该消息有可能会在传输过程中丢失。

3. 客户端与服务器端角色

（1） **服务器端：**必须先于客户端运行，而且要长久保持运行。被动等待客户端发起三次握手，等待客户端的请求，并为客户端服务。

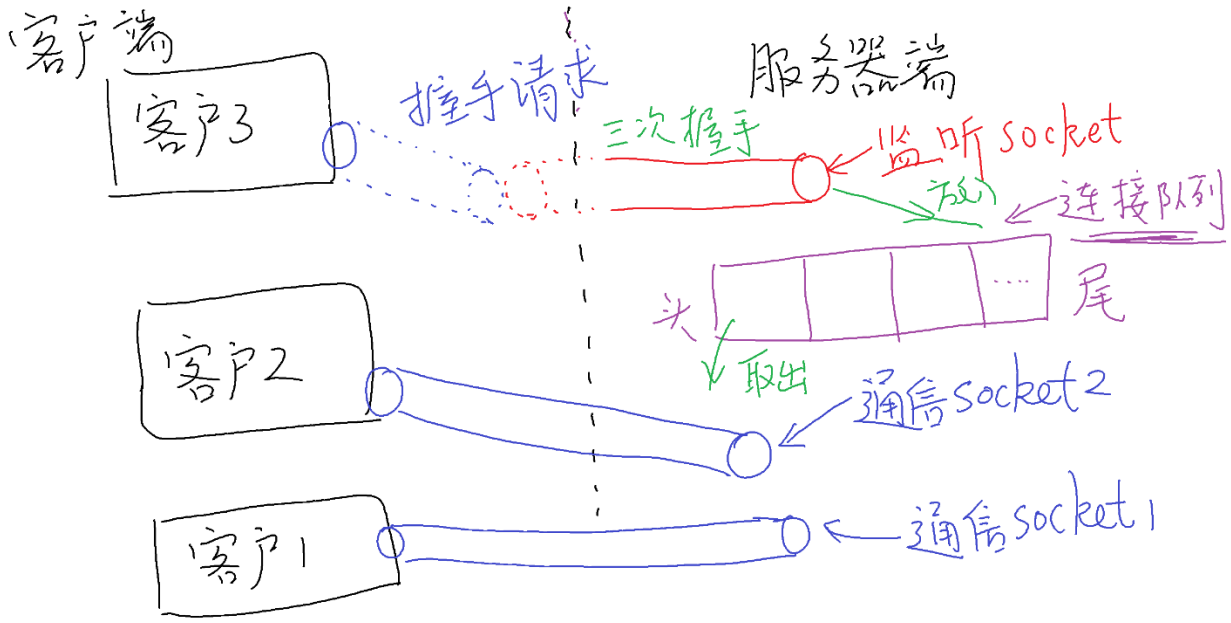
（2） **客户端：**只在有需求时才启动。主动向服务器端发出三次握手请求，连接建立之后发送服务请求，并等待服务结果。

4. Socket 服务器端的几个重要概念：

（1） **监听 Socket 对象：**负责在指定的 IP 地址和指定的端口上监听来自于不同客户端的连接请求，并实现三次握手。握手成功之后将已经建立的 TCP 连接记录放入“连接队列”。三次握手、与客户端建立 TCP 连接、将 TCP 连接信息放入“连接队列”以及维护“连接队列”的工作都是**由操作系统内核模块自动完成的**。可以把这个内核模块想象成一个在后台默默工作的线程。

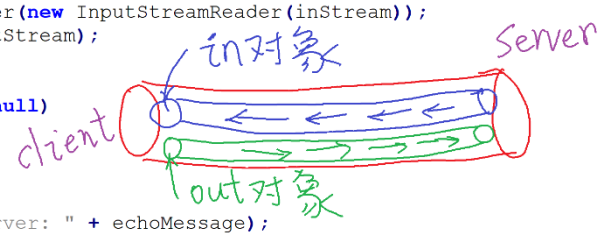
（2） **连接队列：**存放 TCP 连接信息的队列。由操作系统自己维护。

（3） **通信 Socket 对象：**操作系统内核模块从“连接队列”中每取出一个 TCP 连接记录，就关于该 TCP 连接创建一个新的 Socket 与客户端对接到一起。该 Socket 专门负责客户端与服务器端的实际数据通信。



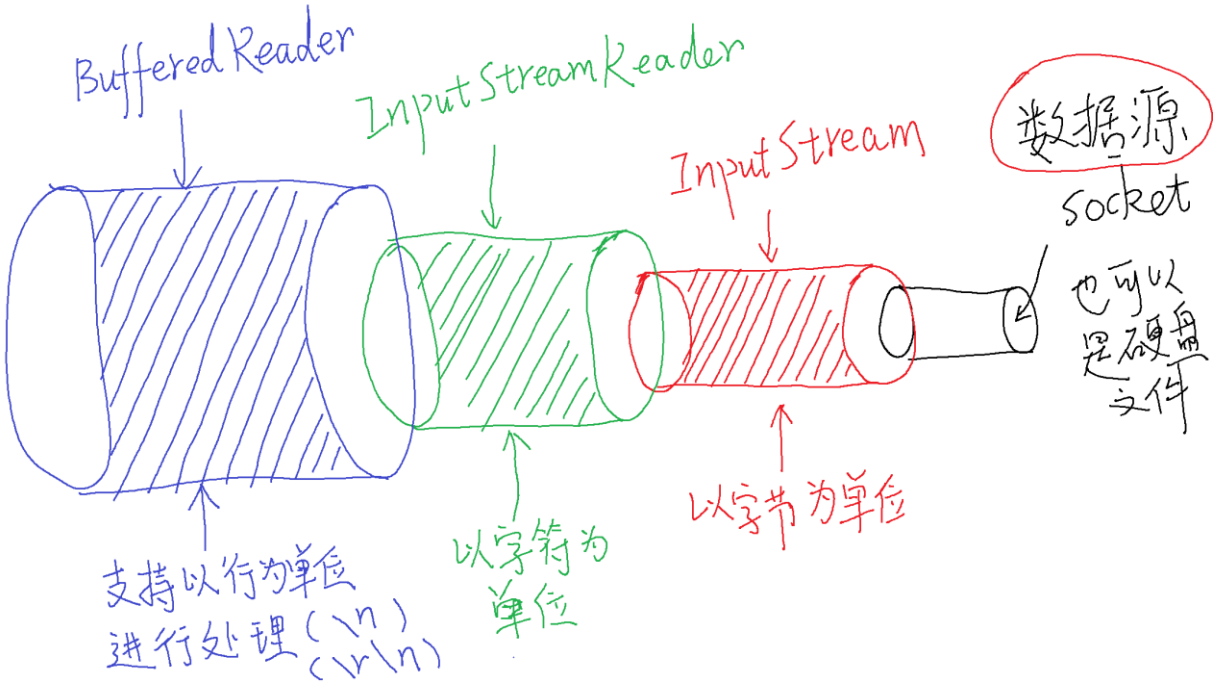
二. 客户端源码分析

```java
import java.io.*;
import java.net.*;

public class EchoClient {

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

        String userInput = null;
        String echoMessage = null;

        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));

        Socket socket = new Socket("127.0.0.1", 8189);
        System.out.println("Connected to Server");

        InputStream inStream = socket.getInputStream();
        OutputStream outStream = socket.getOutputStream();
        BufferedReader in = new BufferedReader(new InputStreamReader(inStream));
        PrintWriter out = new PrintWriter(outStream);


        while((userInput=stdIn.readLine())!=null)
        {
            out.println(userInput);
            out.flush();
            echoMessage = in.readLine();
            System.out.println("Echo from server: " + echoMessage);
        }

        socket.close();

    }
}
```



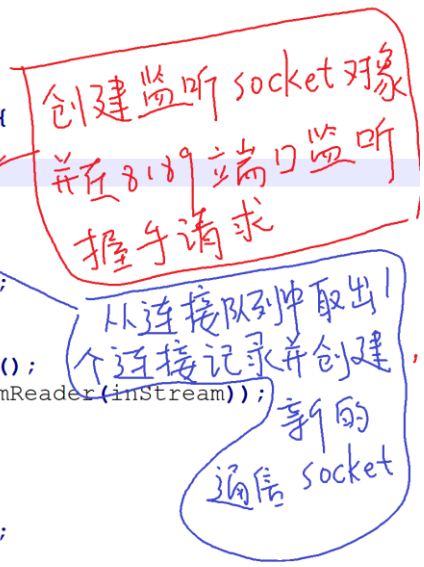
InputStream 、 InputStreamReader 、 BufferedReader 的区别

提问：字节与字符有什么区别？



三. 服务器端源码分析

```java
import java.io.*;
import java.net.*;

public class EchoServer {
    public static void main(String[] args) throws Exception {
        Socket clientSocket = null;
        ServerSocket listenSocket = new ServerSocket(8189);

        System.out.println("Server listening at 8189");
        clientSocket = listenSocket.accept();
        System.out.println("Accepted connection from client");

        InputStream inStream = clientSocket.getInputStream();
        OutputStream outStream = clientSocket.getOutputStream();
        BufferedReader in = new BufferedReader(new InputStreamReader(inStream));
        PrintWriter out = new PrintWriter(outStream);

        String line = null;
        while((line=in.readLine())!=null) {
            System.out.println("Message from client:" + line);
            out.println(line);
            out.flush();
        }
        clientSocket.close();
        listenSocket.close();
    }
}
```



以上源码中，listenSocket 是监听 Socket，clientSocket 是通信 Socket。

四. 多线程

1. **线程的概念：**
   a) 一个独立的执行线路。
   b) 不同的线程在逻辑上相互独立地并行地执行。
   c) 属于同一个进程的多个线程共享该进程的内存地址空间。
   d) **每一个线程都有自己的“主函数”**。线程启动后，其主函数开始执行；线程主函数执行完毕，线程的生命周期就结束了。
2. **Java 的线程对象：**
   a) Java 线程对象是对操作系统线程对象的一个封装
   b) 一个 Java 线程对象一般都是 Thread 类（JDK 实现的基类）的子类。子类需要重载 run 函数，作为自己的线程主函数。
   c) Java 线程对象的构造函数（下面源程序中 MyThread(int t)函数）可以接收一些初始化状态或初始化数据。
3. **多线程编程举例：**

```java
class MyThread extends Thread{
    private int ticket;

    MyThread(int t) {
        ticket = t;
    }

    public void run(){
        for(int i=0;i<ticket;i++){
            if(ticket>0){
                System.out.println(this.getName()+" sold ticket"+ [redacted]);
            }
        }
    }
};

public class ThreadTest {
    public static void main(String[] args) {
        MyThread t1=new MyThread(10);
        MyThread t2=new MyThread(15);
        MyThread t3=new MyThread(30);
        t1.start();
        t2.start();
        t3.start();
    }
}
```

线程主函数.

i

3个售票员并行售票。

不同售票员最开始时拿到的票数不同

10, 15, 30

五. 多线程实现 Socket 服务器端

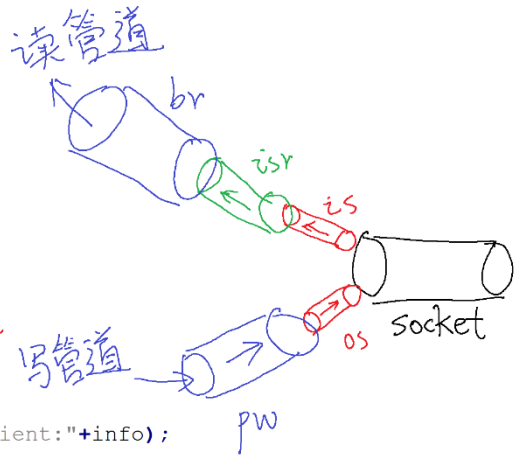
```java
import java.io.*;
import java.net.*;

public class MultiThreadEchoServer {
    public static void main(String[] args) throws Exception {
        ServerSocket listenSocket=new ServerSocket(8189);
        Socket socket=null;

        int count=0;
        System.out.println("Server listening at 8189");

        while(true){
            socket=listenSocket.accept();
            count++;
            System.out.println("The total number of clients is " + count + ".");
            ServerThread serverThread=new ServerThread(socket);
            serverThread.start();
        }
    }
}
```

创建监听socket，开始监听

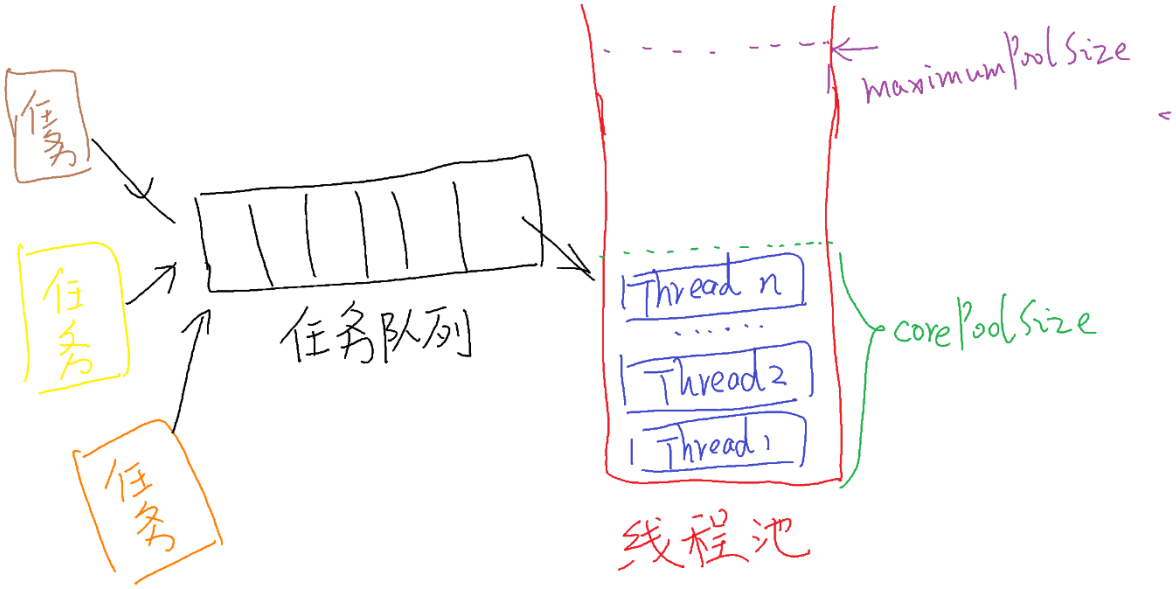
从连接队列中取出一个连接记录创建一个新的通信socket存放到socket变量中

创建一个新的线程，以上面新创建的通信socket作为该线程的初始状态。

```java
import java.io.*;
import java.net.*;

public class ServerThread extends Thread {

    Socket socket = null;

    public ServerThread(Socket socket) {
        this.socket = socket;
    }

    public void run(){
        InputStream is=null;
        InputStreamReader isr=null;
        BufferedReader br=null;
        OutputStream os=null;
        PrintWriter pw=null;
        try {
            is = socket.getInputStream();
            isr = new InputStreamReader(is);
            br = new BufferedReader(isr);
            os = socket.getOutputStream();
            pw = new PrintWriter(os);
            String info=null;
            while((info=br.readLine())!=null){
                System.out.println("Message from client:"+info);
                pw.println(info);
                pw.flush();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }finally{
            try {
                if(pw!=null)
                    pw.close();
                if(os!=null)
                    os.close();
                if(br!=null)
                    br.close();
                if(isr!=null)
                    isr.close();
                if(is!=null)
                    is.close();
                if(socket!=null)
                    socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```



六.线程池



ThreadPoolExecutor 参数解析

ThreadPoolExecutor 主要有以下几个参数:

```
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                          BlockingQueue<Runnable> workQueue,
                          ThreadFactory threadFactory,
                          RejectedExecutionHandler handler)
```

参数说明

（1）corePoolSize 核心线程数量

即使没有任务执行，核心线程也会一直存活

线程数小于核心线程时，即使有空闲线程，线程池也会创建新线程执行任务

设置 allowCoreThreadTimeout=true 时，核心线程会超时关闭

（2）maximumPoolSize 最大线程数

当所有核心线程都在执行任务，且任务队列已满时，线程池会创建新线程执行任务。

当线程数=maxPoolSize, 且任务队列已满，此时添加任务时会触发 RejectedExecutionHandler 进行处理。

（3）keepAliveTime TimeUnit 线程空闲时间

如果线程数>corePoolSize，且有线程空闲时间达到 keepAliveTime 时，线程会销毁，直到线程数量=corePoolSize

如果设置 allowCoreThreadTimeout=true 时，核心线程执行完任务也会销毁直到数量=0

（4） workQueue 任务队列

```java
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.RejectedExecutionHandler;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

public class ThreadPoolTest {
    public static void main(String[] args) {
        ThreadPoolExecutor executor = new ThreadPoolExecutor(5, 10, 200, TimeUnit.
        MILLISECONDS, new ArrayBlockingQueue<Runnable>(5));

        for(int i=0;i<15;i++){
            MyTask myTask = new MyTask(i);
            executor.execute(myTask);
            System.out.println("The number of threads in the ThreadPool:"+executor.
            getPoolSize());
            System.out.println("The number of tasks in the Queue:" + executor.getQueue().
            size());
            System.out.println("The number of tasks completed:"+executor.
            getCompletedTaskCount());
        }
        executor.shutdown();
    }
}

class MyTask implements Runnable {
    private int taskNum;

    public MyTask(int num) {
        this.taskNum = num;
    }

    @Override
    public void run() {
        int sum = 0;

        System.out.println("Task"+taskNum+"is running!");

        try {
            for(int i=0; i<15; i++) {
                sum += i;
            }
            Thread.currentThread().sleep(4000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Task "+taskNum+" has been done!");
    }
}
```