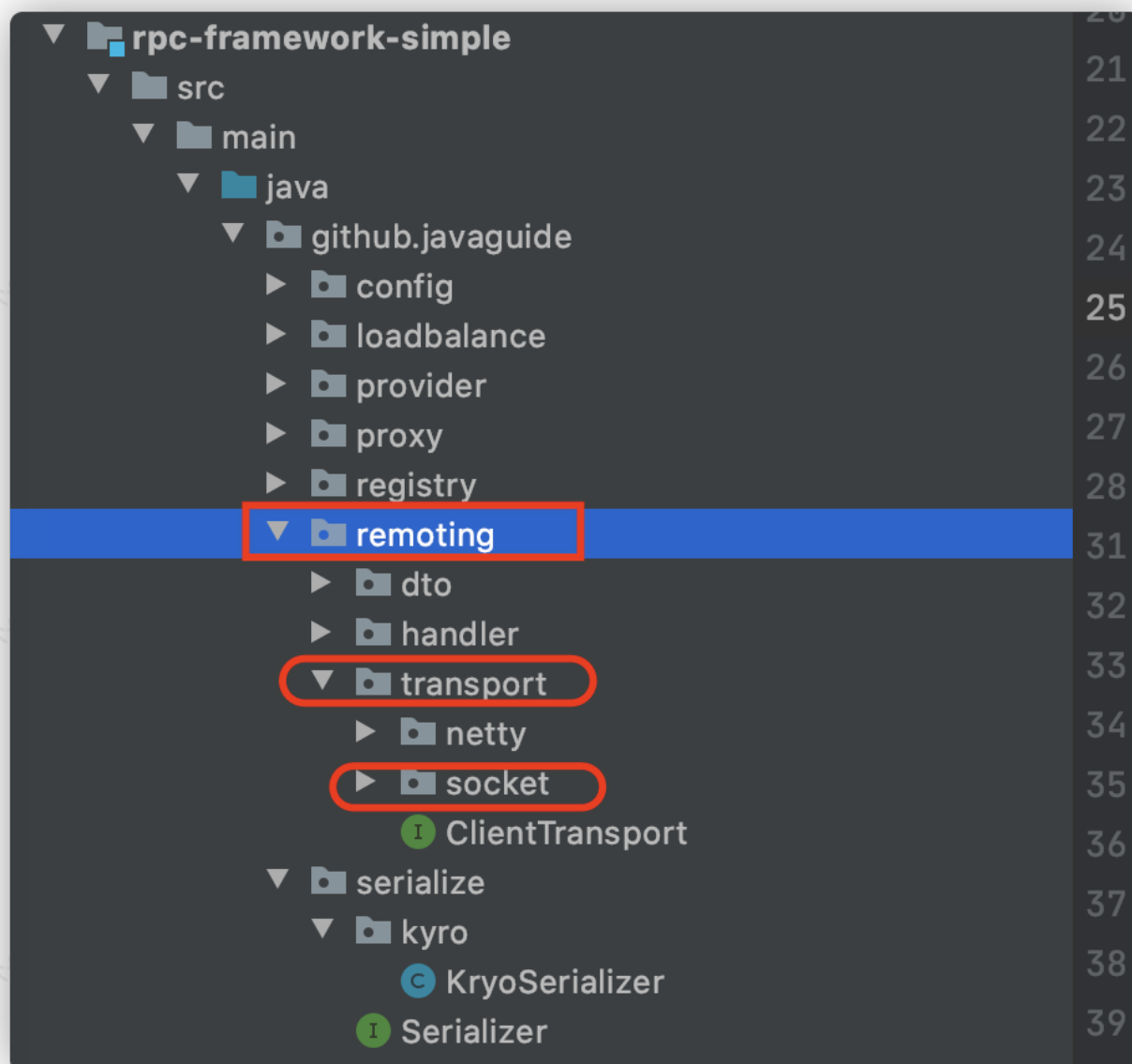


05 Socket 网络通信实战

05 Socket 网络通信实战

1. 前言

[guide-rpc-framework <https://github.com/Snailclimb/guide-rpc-framework>](https://github.com/Snailclimb/guide-rpc-framework) 的第一版使用的是 JDK 提供了 socket 进行网络编程。为了搞懂具体原理，我们首先要学会使用 Socket 进行网络通信。



2. 什么是 Socket(套接字)

Socket 是一个抽象概念，应用程序可以通过它发送或接收数据。在使用 Socket 进行网络通信的时候，通过 Socket 就可以让我们的数据在网络中传输。操作套接字的时候，和我们读写文件很像。套接字是 IP 地址与端口的组合，套接字 Socket=（IP 地址：端口号）。

要通过互联网进行通信，至少需要一对套接字：

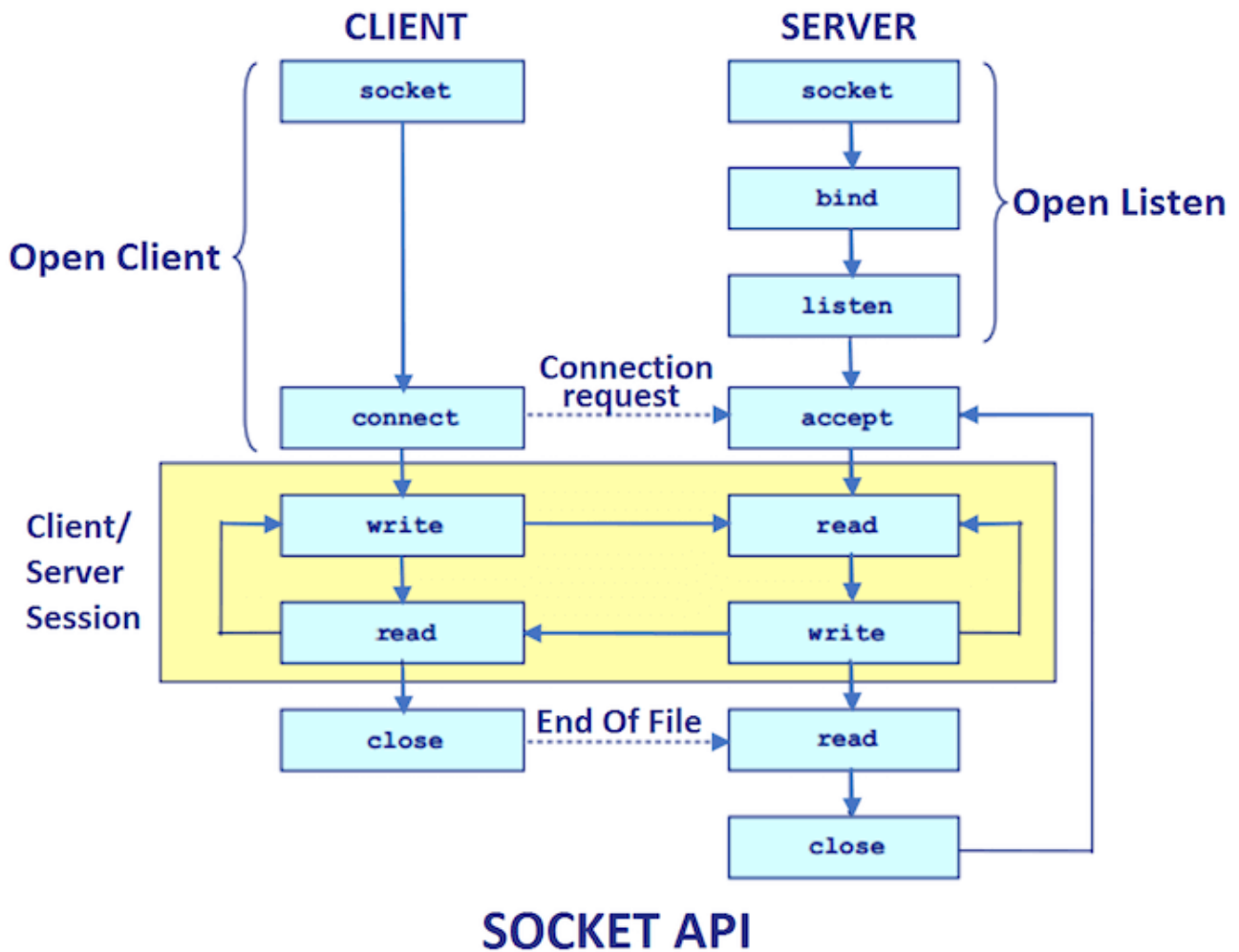
1. 运行于服务器端的 Server Socket。
2. 运行于客户机端的 Client Socket

在 Java 开发中使用 Socket 时会常用到两个类，都在 `java.net` 包中：

1. `Socket`：一般用于客户端
2. `ServerSocket`：用于服务端

3. Socket 网络通信过程

Socket 网络通信过程如下图所示：



<https://www.javatpoint.com/socket-programming>

Socket 网络通信过程简单来说分为下面 4 步：

1. 建立服务端并且监听客户端请求
2. 客户端请求，服务端和客户端建立连接
3. 两端之间可以传递数据
4. 关闭资源

对应到服务端和客户端的话，是下面这样的。

服务器端：

1. 创建 `ServerSocket` 对象并且绑定地址 (ip) 和端口号(port): `server.bind(new InetSocketAddress(host, port))`
2. 通过 `accept()` 方法监听客户端请求
3. 连接建立后，通过输入流读取客户端发送的请求信息
4. 通过输出流向客户端发送响应信息
5. 关闭相关资源

客户端：

1. 创建 `Socket` 对象并且连接指定的服务器的地址 (ip) 和端口号(port):
`socket.connect(inetSocketAddress)`
2. 连接建立后，通过输出流向服务器端发送请求信息
3. 通过输入流获取服务器响应的信息
4. 关闭相关资源

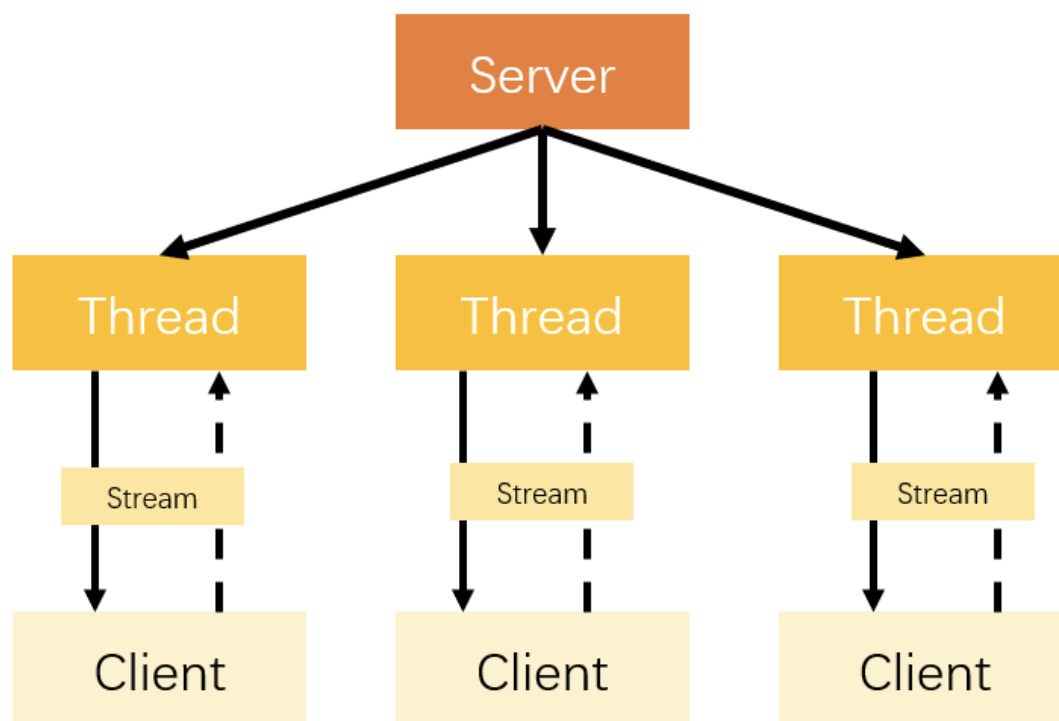
4. Socket 网络通信实战

4.1. 服务端

```
1  public class HelloServer {
2      private static final Logger logger = LoggerFactory.getLogger(HelloServer.class);
3
4      public void start(int port) {
5          //1.创建 ServerSocket 对象并且绑定一个端口
6          try (ServerSocket server = new ServerSocket(port);) {
7              Socket socket;
8              //2.通过 accept()方法监听客户端请求
9              while ((socket = server.accept()) != null) {
10                 logger.info("client connected");
11                 try (ObjectInputStream objectInputStream = new ObjectInputStream(socket.getInputStream());
12                     ObjectOutputStream objectOutputStream = new ObjectOutputStream(socket.getOutputStream());) {
13                     //3.通过输入流读取客户端发送的请求信息
14                     Message message = (Message) objectInputStream.readObject();
15                     logger.info("server receive message:" + message.getContent());
16                     message.setContent("new content");
17                     //4.通过输出流向客户端发送响应信息
18                     objectOutputStream.writeObject(message);
19                     objectOutputStream.flush();
20                 } catch (IOException | ClassNotFoundException e) {
21                     logger.error("occur exception:", e);
22                 }
23             }
24         } catch (IOException e) {
25             logger.error("occur IOException:", e);
26         }
27     }
28
29     public static void main(String[] args) {
30         HelloServer helloServer = new HelloServer();
31         helloServer.start(6666);
32     }
33 }
```

`ServerSocket` 的 `accept ()` 方法是阻塞方法，也就是说 `ServerSocket` 在调用 `accept ()` 等待客户端的连接请求时会阻塞，直到收到客户端发送的连接请求才会继续往下执行代码。

很明显，我上面演示的代码片段有一个很严重的问题：**只能同时处理一个客户端的连接，如果需要管理多个客户端的话，就需要为我们请求的客户端单独创建一个线程。**如下图所示：



公众号JavaGuide

对应的 Java 代码可能是下面这样的：

公众号JavaGuide

```
1 new Thread(() -> {  
2     // 创建 socket 连接  
3 }).start();
```

Java | Copy

但是，这样会导致一个很严重的问题：**资源浪费**。

我们知道线程是很宝贵的资源，如果我们为每一次连接都用一个线程处理的话，就会导致线程越来越多，最后达到了极限之后，就无法再创建线程处理请求了。处理的不好，甚至可能直接就宕机掉了。

公众号JavaGuide

很多人就会问了：那有没有改进的方法呢？

公众号JavaGuide

当然有！**比较简单并且实际的改进方法就是使用线程池**。线程池还可以让线程的创建和回收成本相对较低，并且我们可以指定线程池的可创建线程的最大数量，这样就不会导致线程创建过多，机器资源被不合理消耗。

公众号JavaGuide

公众号JavaGuide

```
1 ThreadFactory threadFactory = Executors.defaultThreadFactory();
2 ExecutorService threadPool = new ThreadPoolExecutor(10, 100, 1, TimeUnit.
3 threadPool.execute(() -> {
4     // 创建 socket 连接
5 });
```

但是，即使你再怎么优化和改变。也改变不了它的底层仍然是同步阻塞的 BIO 模型的事实，因此无法从根本上解决问题。

为了解决上述的问题，Java 1.4 中引入了 NIO，一种同步非阻塞的 I/O 模型。由于使用同步非阻塞的 I/O 模型 NIO 来进行网络编程真的太麻烦了。你可以使用基于 NIO 的网络编程框架 Netty，它将是你的最好的选择（前面的章节提到过，后面的章节会详细讲解如何使用 Netty 进行网络编程）！

4.2. 客户端

```
1  /**
2   * @author shuang.kou
3   * @createTime 2020年05月11日 16:56:00
4   */
5  public class HelloClient {
6
7      private static final Logger logger = LoggerFactory.getLogger(HelloClient.class);
8
9      public Object send(Message message, String host, int port) {
10         //1. 创建Socket对象并且指定服务器的地址和端口号
11         try (Socket socket = new Socket(host, port)) {
12             ObjectOutputStream objectOutputStream = new ObjectOutputStream(socket.getOutputStream());
13             //2. 通过输出流向服务器端发送请求信息
14             objectOutputStream.writeObject(message);
15             //3. 通过输入流获取服务器响应的信息
16             ObjectInputStream objectInputStream = new ObjectInputStream(socket.getInputStream());
17             return objectInputStream.readObject();
18         } catch (IOException | ClassNotFoundException e) {
19             logger.error("occur exception:", e);
20         }
21         return null;
22     }
23
24     public static void main(String[] args) {
25         HelloClient helloClient = new HelloClient();
26         helloClient.send(new Message("content from client"), "127.0.0.1", 8080);
27         System.out.println("client receive message:" + message.getContent());
28     }
29 }
```

发送的消息实体类：

```
1  /**
2   * @author shuang.kou
3   * @createTime 2020年05月11日 17:02:00
4   */
5  @Data
6  @AllArgsConstructor
7  public class Message implements Serializable {
8
9      private String content;
10 }
```

首先运行服务端，然后再运行客户端，控制台输出如下：

服务端:

Bash |  Copy

```
1  [main] INFO github.javaguide.socket.HelloServer - client connected
2  [main] INFO github.javaguide.socket.HelloServer - server receive message
```

客户端:

Bash |  Copy

```
1  client receive message:new content
```

好的！我们的第一个使用 Socket 进行网络编程的案例已经完成了。

下一篇我们来看看如何使用 Netty 进行网络编程。