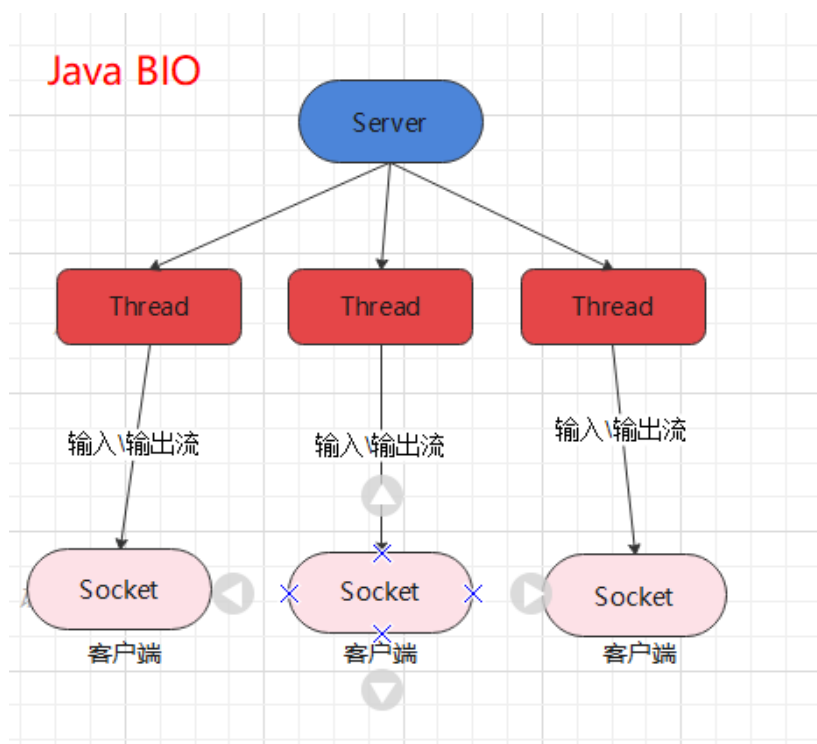


# BIO

## BIO (blocking IO)

### a. BIO 工作机制：

同步并阻塞(传统阻塞型)，服务器实现模式为一个 socket 连接一个线程，即客户端有连接请求时服务器端就需要启动一个线程进行处理，如果这个连接不做任何事情会造成不必要的线程开销，可以通过线程池机制改善(实现多个客户连接服务器)。



BIO 方式适用于连接数目比较小且固定的架构，这种方式对服务器资源要求比较高，并发局限于应用中，JDK1.4 以前的唯一选择，但程序简单易理解。基于 BIO 模式下的通信，客户端 - 服务端是完全同步，完全耦合的。

对 BIO 编程流程的梳理：

- 1) 服务器端启动一个 **ServerSocket**，注册端口，调用 `accept` 方法监听客户端的 Socket 连接。
- 2) 客户端启动 **Socket** 对服务器进行通信，默认情况下服务器端需要对每个客户建立一个线程与之通讯



## b. 传统通信实现：

```

1 //服务端
2 package com.IO.BIO.demo1;
3 import java.io.*;
4 import java.net.ServerSocket;
5 import java.net.Socket;
6 /**
7  * @Author: 22936
8  * @CreateTime: 2021-05-03 20:10
9  * @Description: 服务端
10  * 目标：客户端发送消息，服务端接受消息
11  */
12 public class Server {
13     public static void main(String[] args) {
14         try {
15             ServerSocket ss = new ServerSocket(9999);
16             Socket socket = ss.accept();
17             //从 socket 中得到字节输入流
18             InputStream is = socket.getInputStream();
19             //把字节输入流包装成缓冲字符输入流，采用这样的方法会使效率更高
20             BufferedReader br = new BufferedReader(new
21                 InputStreamReader(is));
22             String msg;
23             while((msg = br.readLine()) != null){
24                 System.out.println("服务端接收到：" + msg);
25             }
26         } catch (Exception e) {
27             e.printStackTrace();
28         }
29     }
30 }
  
```

```

1 //客户端
2 package com.IO.BIO.demo1;
3 import java.io.IOException;
  
```

```

4 import java.io.OutputStream;
5 import java.io.PrintStream;
6 import java.net.Socket;
7 /**
8  * @Author: 22936
9  * @CreateTime: 2021-05-03 20:10
10  * @Description: 客户端
11  */
12 public class Client {
13     public static void main(String[] args) throws Exception {
14         Socket socket = new Socket("127.0.0.1",9999);
15         OutputStream os = socket.getOutputStream();
16         PrintStream ps = new PrintStream(os);
17         ps.println("hello world!"); //如果不是 println，服务端会因为客户端挂掉
而连接重置，抛出异常信息
18         ps.flush();
19     }
20 }

```

### c. 客户端和服务端多发多收机制：

```

1 package com.IO.BIO.demo2;
2 import java.io.BufferedReader;
3 import java.io.InputStream;
4 import java.io.InputStreamReader;
5 import java.net.ServerSocket;
6 import java.net.Socket;
7 /**
8  * @Author: 22936
9  * @CreateTime: 2021-05-03 20:10
10  * @Description: 服务端不断的接受消息
11  *
12  */
13 public class Server {
14     public static void main(String[] args) {
15         try {
16             ServerSocket ss = new ServerSocket(9999);
17             System.out.println("++服务端启动++");
18             Socket socket = ss.accept();
19             //从 socket 中得到字节输入流
20             InputStream is = socket.getInputStream();
21             //把字节输入流包装成缓冲字符输入流
22             BufferedReader br = new BufferedReader(new
InputStreamReader(is));
23             String msg;
24             while((msg = br.readLine()) != null){
25                 System.out.println("服务端接收到：" + msg);
26             }
27
28         } catch (Exception e) {

```

```

29         e.printStackTrace();
30     }
31 }
32 }
33

```

```

1  package com.IO.BIO.demo2;
2  import java.io.OutputStream;
3  import java.io.PrintStream;
4  import java.net.Socket;
5  import java.util.Scanner;
6  /**
7   * @Author: 22936
8   * @CreateTime: 2021-05-03 20:10
9   * @Description: 客户端可以反复发送消息
10  */
11  public class Client {
12      public static void main(String[] args) throws Exception {
13          Socket socket = new Socket("127.0.0.1",9999);
14          OutputStream os = socket.getOutputStream();
15          PrintStream ps = new PrintStream(os);
16          Scanner sc = new Scanner(System.in);
17          while(true){
18              System.out.print("请输入 :");
19              String msg = sc.nextLine();
20              ps.println(msg);
21              ps.flush();
22          }
23      }
24  }
25

```

#### d. 服务端接受多个客户端:

如果服务端需要处理很多个客户端的消息通信请求应该如何处理呢，此时我们就需要在服务端引入线程了，也就是说客户端每发起一个请求，服务端就创建一个新的线程来处理这个客户端的请求。

特点:

- 1.每接收到一个 Socket，都会创建一个线程，线程的竞争、切换上下文影响性能；
- 2.每个线程都会占用栈空间和 CPU 资源；
- 3.并不是每个 socket 都进行 IO 操作，无意义的线程处理；
- 4.客户端的并发访问增加时。服务端将呈现 1:1 的线程开销，访问量越大，系统将发生线程栈溢出，线程创建失败，最终导致进程宕机或者僵死，从而不能对外提供服务。

```

1  package com.IO.BIO.demo3;
2  import java.io.BufferedReader;
3  import java.io.IOException;

```

```

4 import java.io.InputStream;
5 import java.io.InputStreamReader;
6 import java.net.ServerSocket;
7 import java.net.Socket;
8 /**
9  * @Author: 22936
10  * @CreateTime: 2021-05-03 20:10
11  * @Description: 服务端实现同时接受多个客户端的 socket 通信需求
12  *                每接收到一个不同 socket 就创建一个新的线程处理
13  *
14  */
15 public class Server {
16     public static void main(String[] args) {
17         try {
18             ServerSocket ss = new ServerSocket(9999);
19             // 不断接受客户端的 socket 连接请求
20             while(true){
21                 Socket socket = ss.accept();
22                 new ServerThreadReader(socket).start();
23             }
24         } catch (Exception e) {
25             e.printStackTrace();
26         }
27     }
28 }
29

```

```

1 package com.IO.BIO.demo3;
2 import java.io.*;
3 import java.net.Socket;
4 /**
5  * @Author: 22936
6  * @CreateTime: 2021-05-03 20:51
7  * @Description: 每接受一个 socket , 创建一个线程
8  */
9 public class ServerThreadReader extends Thread {
10     private Socket socket;
11     public ServerThreadReader(Socket socket) {
12         this.socket = socket;
13     }
14     @Override
15     public void run() {
16         try {
17             InputStream is = socket.getInputStream();
18             BufferedReader br = new BufferedReader(new
19             InputStreamReader(is));
20             String msg;
21             while((msg = br.readLine())!= null){
22
23             }
24         } catch (IOException e) {
25             e.printStackTrace();
26         }
27     }
28 }
29

```

```

21         System.out.println("服务器收到：" + msg);
22     }
23     } catch (Exception e) {
24         e.printStackTrace();
25     }
26 }
27 }

```

```

1 package com.IO.BIO.demo3;
2 import java.io.OutputStream;
3 import java.io.PrintStream;
4 import java.net.Socket;
5 import java.util.Scanner;
6 /**
7  * @Author: 22936
8  * @CreateTime: 2021-05-03 20:10
9  * @Description: 客户端可以反复发送消息
10 */
11 public class Client {
12     public static void main(String[] args) throws Exception {
13         Socket socket = new Socket("127.0.0.1", 9999);
14         OutputStream os = socket.getOutputStream();
15         PrintStream ps = new PrintStream(os);
16         Scanner sc = new Scanner(System.in);
17         while(true){
18             System.out.print("请输入：");
19             String msg = sc.nextLine();
20             ps.println(msg);
21             ps.flush();
22         }
23     }
24 }
25

```

## e. 伪异步 IO 编程

在上述案例中：客户端的并发访问增加时。服务端将呈现 1:1 的线程开销，访问量越大，系统将发生线程栈溢出，线程创建失败，最终导致进程宕机或者僵死，从而不能对外提供服务。

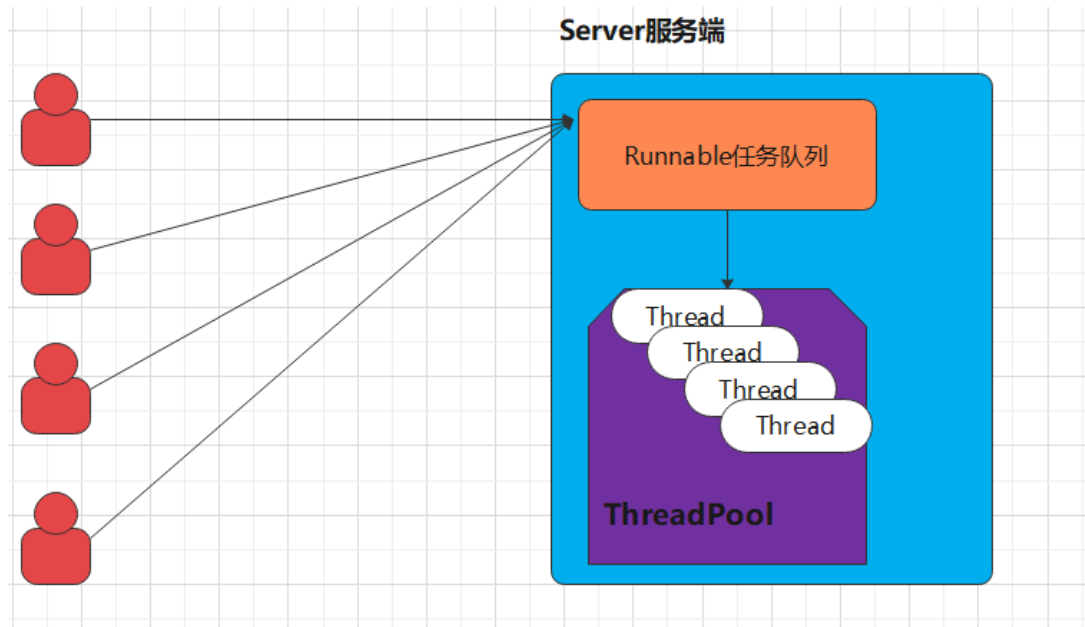
接下来我们采用一个伪异步 I/O 的通信框架，采用线程池和任务队列实现，当客户端接入时，将客户端的 Socket 封装成一个 Task(该任务实现 java.lang.Runnable 线程任务接口)交给后端的线程池中进行处理。JDK 的线程池维护一个消息队列和 N 个活跃的线程，对消息队列中 Socket 任务进行处理，由于线程池可以设置消息队列的大小和最大线程数，因此，它的资源占用是可控的，无论多少个客户端并发访问，都不会导致资源的耗尽和宕机。

特点：

- 伪异步 io 采用了线程池实现，因此避免了为每个请求创建一个独立线程造成线程资源耗尽的问题，但由于底层依然是采用的同步阻塞模型，因此无法从根本上解

决问题。

- 如果单个消息处理的缓慢，或者服务器线程池中的全部线程都被阻塞，那么后续 socket 的 i/o 消息都将在队列中排队。新的 Socket 请求将被拒绝，客户端会发生大量连接超时。



```
1 package com.IO.BIO.demo4;
2 import java.io.IOException;
3 import java.net.ServerSocket;
4 import java.net.Socket;
5 /**
6  * @Author: 22936
7  * @CreateTime: 2021-05-03 21:20
8  * @Description: 开发实现伪异步通信架构
9  */
10 public class Server {
11     public static void main(String[] args) {
12         try {
13             ServerSocket ss = new ServerSocket(9999);
14             HandlerSocketServerPool pool = new
15 HandlerSocketServerPool(6,10);
16             while(true){
17                 Socket socket = ss.accept();
18                 // 把 socket 封装成任务对象再交给线程池处理
19                 Runnable target = new ServerRunnableTarget(socket);
20                 pool.execute(target);
21             }
22         } catch (Exception e) {
23             e.printStackTrace();
24         }
25     }
26 }
```

```

1 package com.IO.BIO.demo4;
2 import java.util.concurrent.ArrayBlockingQueue;
3 import java.util.concurrent.ExecutorService;
4 import java.util.concurrent.ThreadPoolExecutor;
5 import java.util.concurrent.TimeUnit;
6 /**
7  * @Author: 22936
8  * @CreateTime: 2021-05-03 21:23
9  * @Description: 创建线程池
10 */
11 public class HandlerSocketServerPool {
12     private ExecutorService executorService;
13     //初始化线程池对象
14     public HandlerSocketServerPool(int maxThreadNum,int queueSize) {
15         executorService = new ThreadPoolExecutor(3,maxThreadNum,
16             120, TimeUnit.SECONDS,new ArrayBlockingQueue<Runnable>
17             (queueSize));
18     }
19     // 提供一个方法来提交任务给线程池的任务队列进行处理，等待线程池来处理
20     public void execute(Runnable target){
21         executorService.execute(target);
22     }
23 }

```

```

1 package com.IO.BIO.demo4;
2 import java.io.BufferedReader;
3 import java.io.InputStream;
4 import java.io.InputStreamReader;
5 import java.net.ServerSocket;
6 import java.net.Socket;
7 /**
8  * @Author: 22936
9  * @CreateTime: 2021-05-03 21:29
10 * @Description: 变为线程的任务对象
11 */
12 public class ServerRunnableTarget implements Runnable{
13     private Socket socket;
14     public ServerRunnableTarget(Socket socket) {
15         this.socket = socket;
16     }
17     @Override
18     public void run() {
19         try {
20             System.out.println("==服务端启动==");
21             //从 socket 中得到字节输入流
22             InputStream is = socket.getInputStream();
23             //把字节输入流包装成缓冲字符输入流

```



```

24         BufferedReader br = new BufferedReader(new
InputStreamReader(is));
25         String msg;
26         while((msg = br.readLine()) != null){
27             System.out.println("服务端接收到：" + msg);
28         }
29     } catch (Exception e) {
30         e.printStackTrace();
31     }
32 }
33 }

```

```

1 package com.IO.BIO.demo4;
2 import java.io.OutputStream;
3 import java.io.PrintStream;
4 import java.net.Socket;
5 import java.util.Scanner;
6 /**
7  * @Author: 22936
8  * @CreateTime: 2021-05-03 20:10
9  * @Description: 客户端可以反复发送消息
10 */
11 public class Client {
12     public static void main(String[] args) throws Exception {
13         Socket socket = new Socket("127.0.0.1",9999);
14         OutputStream os = socket.getOutputStream();
15         PrintStream ps = new PrintStream(os);
16         Scanner sc = new Scanner(System.in);
17         while(true){
18             System.out.print("请输入：");
19             String msg = sc.nextLine();
20             ps.println(msg);
21             ps.flush();
22         }
23     }
24 }
25

```

## f. 基于 BIO 形式下的文件上传

```

1 package com.IO.BIO.demo5;
2 import java.io.DataOutputStream;
3 import java.io.FileInputStream;
4 import java.io.IOException;
5 import java.io.InputStream;
6 import java.net.Socket;
7 /**

```

```

8  * @Author: 22936
9  * @CreateTime: 2021-05-03 21:43
10 * @Description: 客户端上传任意类型的文件给服务端
11 */
12 public class Client {
13     public static void main(String[] args) {
14         try {
15             Socket socket = new Socket("127.0.0.1",9999);
16             DataOutputStream dos = new
DataOutputStream(socket.getOutputStream());
17             dos.writeUTF(".png"); // 发送后缀
18             InputStream is = new FileInputStream("F:\\GoogleDownload\\大厂
面试之 IO 模式详解资料\\文件\\java.png");
19             byte[] buffer = new byte[1024];
20             int len;
21             while((len = is.read(buffer)) > 0){
22                 dos.write(buffer,0,len);
23             }
24             dos.flush();
25             socket.shutdownOutput(); //防止服务端一直等待
26         } catch (Exception e) {
27             e.printStackTrace();
28         }
29     }
30 }

```

```

1  package com.IO.BIO.demo5;
2  import com.IO.BIO.demo4.ServerRunnableTarget;
3  import java.io.IOException;
4  import java.net.ServerSocket;
5  import java.net.Socket;
6  /**
7   * @Author: 22936
8   * @CreateTime: 2021-05-03 21:42
9   * @Description: 服务端实现接受客户端的任意类型文件，并保存到服务端磁盘
10  */
11 public class Server {
12     public static void main(String[] args) {
13         try {
14             ServerSocket ss = new ServerSocket(9999);
15             while(true){
16                 Socket socket = ss.accept();
17                 new ServerReadThread(socket).start();
18             }
19         } catch (Exception e) {
20             e.printStackTrace();
21         }
22     }

```

23 }

```
1 package com.IO.BIO.demo5;
2 import java.io.DataInput;
3 import java.io.DataInputStream;
4 import java.io.FileOutputStream;
5 import java.io.OutputStream;
6 import java.net.Socket;
7 import java.util.UUID;
8 /**
9  * @Author: 22936
10  * @CreateTime: 2021-05-03 21:55
11  * @Description:
12  */
13 public class ServerReadThread extends Thread {
14     private Socket socket;
15     public ServerReadThread(Socket socket) {
16         this.socket = socket;
17     }
18     @Override
19     public void run() {
20         try{
21             DataInputStream dis = new
22             DataInputStream(socket.getInputStream());
23             String suffix = dis.readUTF();
24             System.out.println("服务端接受到了文件类型为："+suffix);
25             //定义字节输出管道负责把客户端发来的文件数据写出去
26             OutputStream os = new
27             FileOutputStream("E:\\Code\\JavaLearning\\IO\\src\\com\\IO\\BIO\\服务器路径\\
28             + UUID.randomUUID().toString()+suffix);
29             byte[] buffer = new byte[1024];
30             int len;
31             while((len = dis.read(buffer)) >0){
32                 os.write(buffer,0,len);
33             }
34             os.close();
35             System.out.println("服务端保存数据成功！");
36         }catch (Exception e){
37             e.printStackTrace();
38         }
39     }
40 }
```

## g. BIO 模式下的端口转发思想

需求：需要实现一个客户端的消息可以发送给所有的客户端去接收。（群聊实现。基于 BIO 模式下的即时通信，我们需要解决客户端到客户端的通信，也就是需

要实现客户端与客户端的端口消息转发逻辑。

