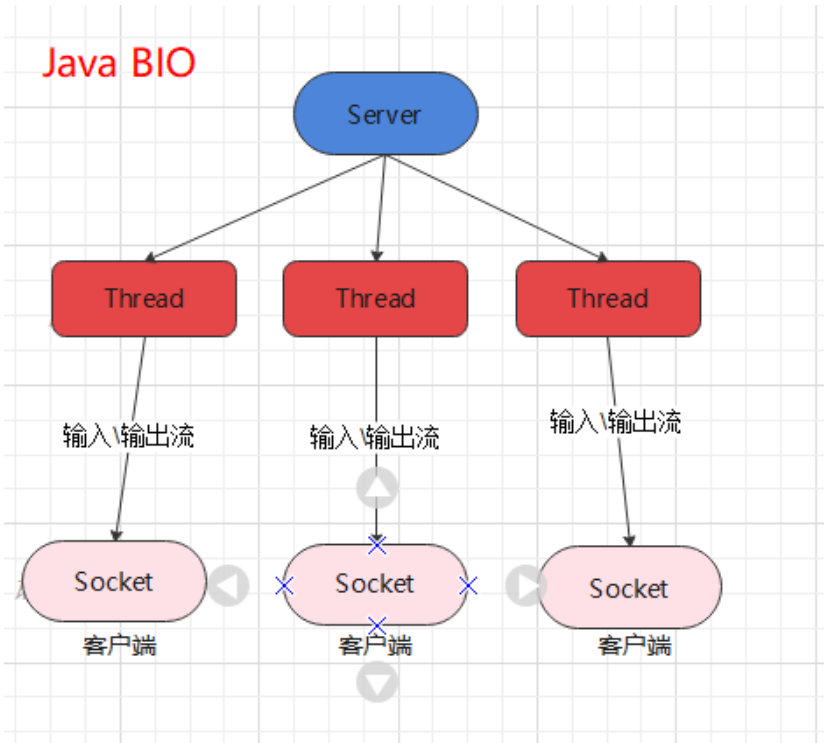
BIO

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)。

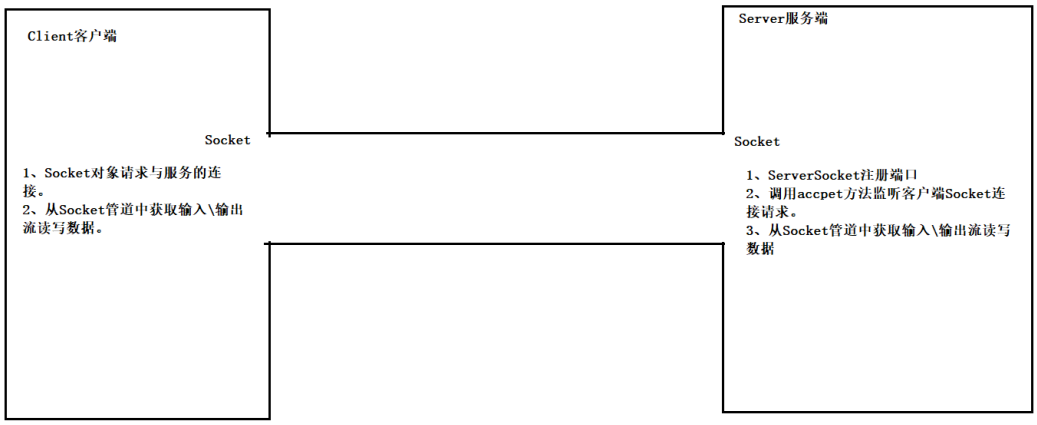


BIO 方式适用于连接数目比较小且固定的架构，这种方式对服务器资源要求比较高，并发局限于应用中，JDK1.4 以前的唯一选择，但程序简单易理解。基于 BIO 模式下的通信，客户端 - 服务端是完全同步，完全耦合的。

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 accpet 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：

```
//服务端
package com.IO.BIO.demo1;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 20:10
 * @Description: 服务端
 * 目标：客户端发送消息，服务端接受消息
 */
public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(9999);
            Socket socket = ss.accept();
            //从 socket 中得到字节输入流
            InputStream is = socket.getInputStream();
            //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
            BufferedReader br = new BufferedReader(new
InputStreamReader(is));
            String msg;
            while((msg = br.readLine()) != null){
                System.out.println("服务端接收到：" + msg);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

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

```
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 20:10
 * @Description: 客户端
 */
public class Client {
    public static void main(String[] args) throws Exception {
        Socket socket = new Socket("127.0.0.1",9999);
        OutputStream os = socket.getOutputStream();
        PrintStream ps = new PrintStream(os);
        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
        ps.flush();
    }
}
```

c. 客户端和服务端多发多收机制：

```
package com.IO.BIO.demo2;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 20:10
 * @Description: 服务端不断的接受消息
 *
 */
public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(9999);
            System.out.println("++服务端启动++");
            Socket socket = ss.accept();
            //从 socket 中得到字节输入流
            InputStream is = socket.getInputStream();
            //把字节输入流包装成缓冲字符输入流
            BufferedReader br = new BufferedReader(new InputStreamReader(is));
            String msg;
            while((msg = br.readLine()) != null){
                System.out.println("服务端接收到：" + msg);
            }

        } catch (Exception e) {
```

```
            e.printStackTrace();
        }
    }
}

```

```
package com.IO.BIO.demo2;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.util.Scanner;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 20:10
 * @Description: 客户端可以反复发送消息
 */
public class Client {
    public static void main(String[] args) throws Exception {
        Socket socket = new Socket("127.0.0.1",9999);
        OutputStream os = socket.getOutputStream();
        PrintStream ps = new PrintStream(os);
        Scanner sc = new Scanner(System.in);
        while(true){
            System.out.print("请输入：");
            String msg = sc.nextLine();
            ps.println(msg);
            ps.flush();
        }
    }
}

```

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。

```
package com.IO.BIO.demo3;
import java.io.BufferedReader;
import java.io.IOException;

```

```
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 20:10
 * @Description: 服务端实现同时接受多个客户端的 socket 通信需求
 *                   每接收到一个不同 socket 就创建一个新的线程处理
 *
 */
public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(9999);
            // 不断接受客户端的 socket 连接请求
            while(true){
                Socket socket = ss.accept();
                new ServerThreadReader(socket).start();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```
package com.IO.BIO.demo3;
import java.io.*;
import java.net.Socket;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 20:51
 * @Description: 每接受一个 socket，创建一个线程
 */
public class ServerThreadReader extends Thread {
    private Socket socket;
    public ServerThreadReader(Socket socket) {
        this.socket = socket;
    }
    @Override
    public void run() {
        try {
            InputStream is = socket.getInputStream();
            BufferedReader br = new BufferedReader(new
InputStreamReader(is));
            String msg;
            while((msg = br.readLine())!= null){
```

```
                System.out.println("服务器收到："+ msg);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
package com.IO.BIO.demo3;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.util.Scanner;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 20:10
 * @Description: 客户端可以反复发送消息
 */
public class Client {
    public static void main(String[] args) throws Exception {
        Socket socket = new Socket("127.0.0.1",9999);
        OutputStream os = socket.getOutputStream();
        PrintStream ps = new PrintStream(os);
        Scanner sc = new Scanner(System.in);
        while(true){
            System.out.print("请输入：");
            String msg = sc.nextLine();
            ps.println(msg);
            ps.flush();
        }
    }
}

```

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。

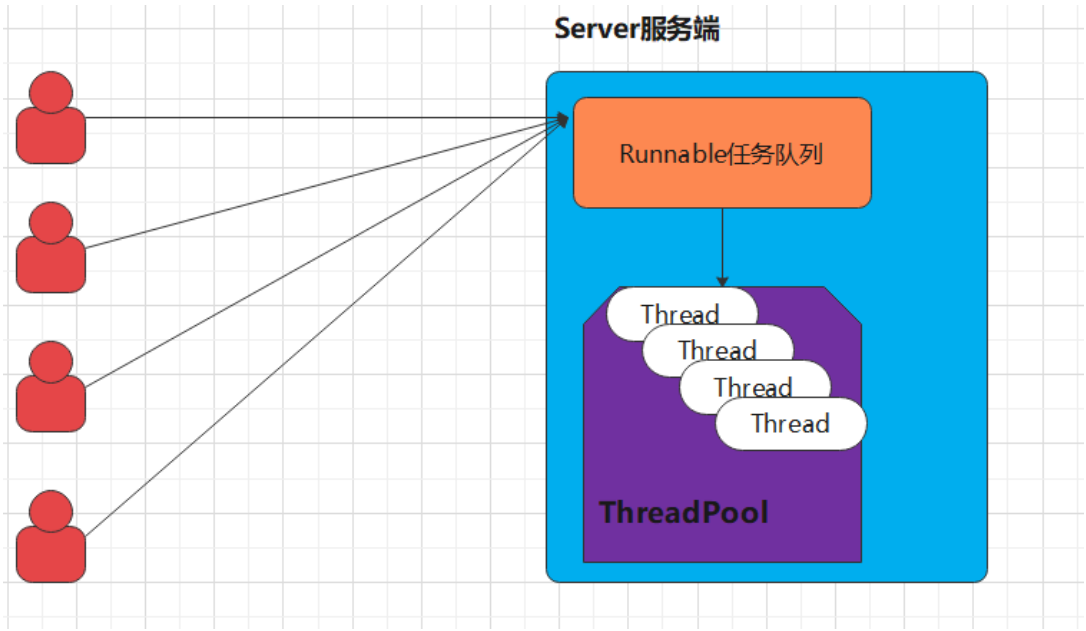
接下来我们采用一个伪异步 I/O 的通信框架，采用线程池和任务队列实现，当客户端接入时，将客户端的 Socket 封装成一个 Task(该任务实现 java.lang.Runnable 线程任务接口)交给后端的线程池中进行处理。JDK 的线程池维护一个消息队列和 N 个活跃的线程，对消息队列中 Socket 任务进行处理，由于线程池可以设置消息队列的大小和最大线程数，因此，它的资源占用是可控的，无论多少个客户端并发访问，都不会导致资源的耗尽和宕机。

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。




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

```
package com.IO.BIO.demo4;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 21:23
 * @Description: 创建线程池
 */
public class HandlerSocketServerPool {
    private ExecutorService executorService;
    //初始化线程池对象
    public HandlerSocketServerPool(int maxThreadNum,int queueSize) {
        executorService = new ThreadPoolExecutor(3,maxThreadNum,
                120, TimeUnit.SECONDS,new ArrayBlockingQueue<Runnable>
(queueSize));
    }
    // 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
    public void execute(Runnable target){
        executorService.execute(target);
    }
}
```

```
package com.IO.BIO.demo4;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 21:29
 * @Description: 变为线程的任务对象
 */
public class ServerRunnableTarget implements Runnable{
    private Socket socket;
    public ServerRunnableTarget(Socket socket) {
        this.socket = socket;
    }
    @Override
    public void run() {
        try {
            System.out.println("==服务端启动==");
            //从 socket 中得到字节输入流
            InputStream is = socket.getInputStream();
            //把字节输入流包装成缓冲字符输入流
```

```
            BufferedReader br = new BufferedReader(new
InputStreamReader(is));
            String msg;
            while((msg = br.readLine()) != null){
                System.out.println("服务端接收到：" + msg);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
package com.IO.BIO.demo4;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.util.Scanner;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 20:10
 * @Description: 客户端可以反复发送消息
 */
public class Client {
    public static void main(String[] args) throws Exception {
        Socket socket = new Socket("127.0.0.1",9999);
        OutputStream os = socket.getOutputStream();
        PrintStream ps = new PrintStream(os);
        Scanner sc = new Scanner(System.in);
        while(true){
            System.out.print("请输入：");
            String msg = sc.nextLine();
            ps.println(msg);
            ps.flush();
        }
    }
}

```

f. 基于 BIO 形式下的文件上传

```
package com.IO.BIO.demo5;
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.Socket;
/**

```

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

```
package com.IO.BIO.demo5;
import com.IO.BIO.demo4.ServerRunnableTarget;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 21:42
 * @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
 */
public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(9999);
            while(true){
                Socket socket = ss.accept();
                new ServerReadThread(socket).start();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
}
```

```
package com.IO.BIO.demo5;
import java.io.DataInput;
import java.io.DataInputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.util.UUID;
/**
 * @Author: 22936
 * @CreateTime: 2021-05-03 21:55
 * @Description:
 */
public class ServerReadThread extends Thread {
    private Socket socket;
    public ServerReadThread(Socket socket) {
        this.socket = socket;
    }
    @Override
    public void run() {
        try{
            DataInputStream dis = new
DataInputStream(socket.getInputStream());
            String suffix = dis.readUTF();
            System.out.println("服务端接受到了文件类型为："+suffix);
            //定义字节输出管道负责把客户端发来的文件数据写出去
            OutputStream os = new
FileOutputStream("E:\\Code\\JavaLearning\\IO\\src\\com\\IO\\BIO\\服务器路径
\\"
                    + UUID.randomUUID().toString()+suffix);
            byte[] buffer = new byte[1024];
            int len;
            while((len = dis.read(buffer)) >0){
                os.write(buffer,0,len);
            }
            os.close();
            System.out.println("服务端保存数据成功！");
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}

```

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。