服务器Server

- 搭建服务器（psvm、无参构造）
- 开启服务器，设置端口号
- 监听端口，将连接信息封装成socket对象
- 从socket中得到读取流，用于读取客户端给服务器的数据
- 从socket中得到写入流，用于发送数据到客户端
- 接收客户端发送的数据，并以字符串的方式打印出来
- 向客户端发送数据，并刷新
- 关闭流

客户端Client

- 创建客户端（psvm、无参构造）
- 连接指定ip和端口号的服务器
- 监听端口，将连接信息封装成socket对象
- 从socket中得到读取流，用于读取服务器给客户端的数据
- 从socket中得到写入流，用于发送数据到服务器
- 向服务器发送数据，并刷新
- 接收服务器发送的数据，并以字符串的方式打印出来
- 关闭流

编码时常见注意事项

数据库的添加操作，利用注解实现

数据库的删除和修改操作，利用反射实现

数据库的查询操作，

数据库

```
USE mydb;

DROP TABLE IF EXISTS t_user;
CREATE TABLE t_user(
    pk_userId INT PRIMARY KEY AUTO_INCREMENT,
    u_name VARCHAR(20),
    u_pwd VARCHAR(20),
    u_birthday DATE,
    u_sex ENUM('man','woman')
);
DROP TABLE t_user;

TRUNCATE TABLE t_user;
INSERT INTO t_user(u_name,u_pwd,u_birthday,u_sex)
```

```
            VALUES('loth','123','1998-03-06','man'),
                  ('sotho','456','1999-05-07','woman'),
                  ('wadf','789','1997-03-06','man'),
                  ('dfdf','147','1996-03-06','woman');

SELECT * FROM t_user;

DELETE FROM t_user WHERE pk_userId = 1;
UPDATE t_user SET u_pwd = "778" WHERE pk_userId = 4;
SELECT*FROM t_user WHERE pk_userId = 1;
```

步骤

1、MyServer

服务器MyServer类

```
//搭建服务器
//运行时先开服务器，在开客户端
public class MyServer {
    public MyServer(){
        try {
            //开启服务器，开放8088端口，端口号自己定：8088
            ServerSocket server = new ServerSocket(8088);
            //解决服务器开了后执行一次就关闭的情况
            while(true) {
                // 如果有客户端连接到服务器，就将连接信息封装成socket对象
                //监听端口，有服务的时候进行数据交互，
                //有客户端连接到服务器时才会执行后面的语句，
                //线程阻塞：监听一直存在，但一直没有客户端连接
                Socket socket = server.accept();
                //从socket中得到读取流，用于读取客户端给服务器的数据
                InputStream in = socket.getInputStream();
                //从socket中得到写入流，用于发送数据到客户端
                OutputStream out = socket.getOutputStream();

                //接收客户端发送的数据，并以字符串的方式打印出来
                byte[] by = new byte[1024];
                in.read(by);
                //trim()用于去掉空格
                String info = new String(by).trim();
                System.out.println("客户端发送：  " + info);

                //发送数据到客户端
                out.write("你好，欢迎光临".getBytes());
                out.flush();

                //关闭流
                out.close();
                in.close();
                socket.close();
            }
```

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

    }

    public static void main(String[] args) {
        new MyServer();
    }
}
```

2、测试

运行MyServer类，在浏览器中输入http://localhost:8088/，测试客户端是否响应

此时的浏览器就是我们的客户端，参考Client类

此时服务器端接收到的客户端发送的数据，打印出来为：

```
GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: zh-Hans-CN,zh-Hans;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like
Gecko
Accept-Encoding: gzip, deflate
Host: localhost:8088
Connection: Keep-Alive
Cookie: Idea-e4e944d1=6d3f90e8-b2c3-459e-af67-ffb5dce7978f
```

```
请求方式    请求服务的路径url（统一资源定位符）
Accept：客户端能接收什么数据
Accept-Language:能接收什么语种
User-Agent:用户的浏览器类型
Accept-Encoding:压缩类型
Host:端口
Connection:连接类型
```

该信息包括服务器的端口号8088；客户端访问的服务器中的位置；
以get方式请求文件、以get方式请求业务组件、以post方式请求业务组件

3、SocketThread

新建一个线程类SocketThread

继承Runnable接口，重写run方法

定义套接字socket为私有属性

在类SocketThread中定义以套接字socket为形参的构造方法，该构造方法用于完成多线程的开启

将MyServer类中监听端口之后的处理操作移动到SocketThread类重写的run方法中

```
//线程类
```

```
public class SocketThread implements Runnable{
    //socket对象
    private Socket socket;
    public SocketThread(Socket socket) {
        this.socket = socket;
        //启动线程，只要new这个类的时候，就会启动线程，完成多线程的开启
        new Thread(this).start();
    }

    @Override
    public void run() {
        try {
            //从socket中得到读取流，用于读取客户端给服务器的数据
                InputStream in = socket.getInputStream();
                //从socket中得到写入流，用于发送数据到客户端
                OutputStream out = socket.getOutputStream();

                //接收客户端发送的数据，并以字符串的方式打印出来
                byte[] by = new byte[1024];
                in.read(by);
                //trim()用于去掉空格
                String info = new String(by).trim();
                System.out.println("客户端发送：  " + info);

                //发送数据到客户端
                out.write("你好，欢迎光临".getBytes());
                out.flush();

                //关闭流
                out.close();
                in.close();
                socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

4、建立联系

如何将MyServer类和SocketThread类联系起来

在监听端口语句后创建一个SocketThread对象

```
public class MyServer {
    public MyServer(){
        try {
            //开启服务器，开放8088端口，端口号自己定：8088
            ServerSocket server = new ServerSocket(7808);
            //解决服务器开了后执行一次就关闭的情况
            while(true) {
                // 如果有客户端连接到服务器，就将连接信息封装成socket对象
                //监听端口，有服务的时候进行数据交互，
                //有客户端连接到服务器时才会执行后面的语句，
                //线程阻塞：监听一直存在，但一直没有客户端连接
                Socket socket = server.accept();
```

```
                //新建一个SocketThread类来解决多用户不能同时访问的问题
                //将原本写在监听语句后的代码放到SocketThread的run方法中
            //循环监听端口，每循环一次新建一个线程对象对象，为每个用户新建一个线程专门为其服务
                new SocketThread(socket);
            }

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

    public static void main(String[] args) {
        new MyServer();
    }
}
```

5、测试

测试一下，运行MyServer，打开浏览器，输入http://localhost:7808/.

刷新之后发现程序运行正常，有结果。

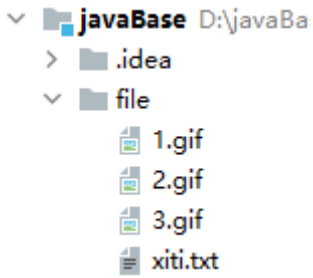
刷新就是向服务器重新发送一次请求，重新发送请求意味着服务器需要重新处理一次，频繁的进行刷新不会更快，还会增加服务器负担。

现在解决了不同多用户同时访问的问题，虽然看不出效果，因为只有我们自己进行访问，一个用户

6、Response

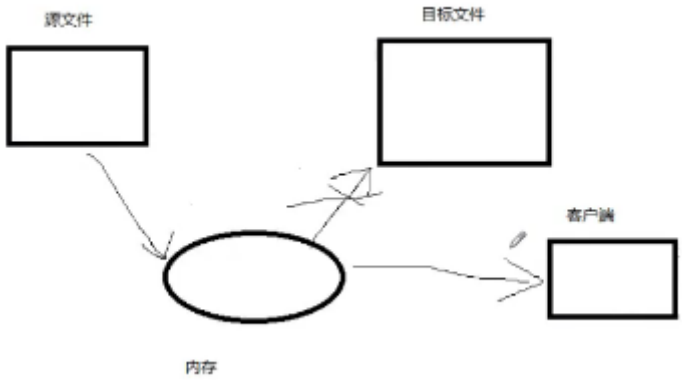
将本地的工程文件信息发送给客户端，不要客户每次访问都只是看见“你好，欢迎光临”

在工程下新建一个文件夹file，随便加入点文件，图片、文本都可以，文本中中文容易出现乱码



新建一个响应类Response，用于将本地文件读取出来后发送给客户端。

sendFile方法用于发送文件数据，从源文件读取数据到内存，在发送给客户端




```
//处理写入流的响应类
//先由客户端发送请求，由这个类来响应
public class Response {

    //socket写入流
    //这个写入流的关闭在类SocketThread中
    private OutputStream out;

    //此构造方法完成写入流的初始化
    public Response(OutputStream out) {
        this.out = out;
    }

    //向客户端发送文本数据
    public void sendMessage(String info){
        try {
            this.out.write(info.getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //向客户端发送文件数据
    public void sendFile(String filePath){
        File f = new File(filePath);
        //判断文件是否存在，如果不存在，则跳出方法
        if(f.exists() == false){
            return;
        }

        //给本地文件创建一个读取流
        InputStream in = null;
        try {
            in = new FileInputStream(filePath);

            //进行文件拷贝，建立byte数组，一个缓冲的地方，一次读取1024字节，
            //在建一个变量len用于统计当前读了多少了字节
            //每次循环读取1024个字节到byte数组并让变量len接收，当前不等于-1就循环（数据还未
读完）
            byte[] by = new byte[1024];
            int len = 0;

            while ((len = in.read(by))!= -1){
                //out写入流来自于SocketThread类，来自于客户端
                out.write(by,0,len);
```

```
//                this.out = out;
            }

        } catch (Exception e) {
            e.printStackTrace();
        }finally {
            try {
                in.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

7、建立联系

让Response类和SocketThread类联系起来

SocketThread类的run方法：
新建一个Response响应对象，去掉“你好，欢迎光临”。发送一张图片到客户端

```
@Override
    public void run() {
        try {
            //从socket中得到读取流，用于读取客户端给服务器的数据
                InputStream in = socket.getInputStream();
                //从socket中得到写入流，用于发送数据到客户端
                OutputStream out = socket.getOutputStream();

                //将写入流封装成响应对象，以便更好的发送数据
            Response response = new Response(out);

                //接收客户端发送的数据，并以字符串的方式打印出来
                byte[] by = new byte[1024];
                in.read(by);
                //trim()用于去掉空格
                String info = new String(by).trim();
                System.out.println("客户端发送：  " + info);


          // out.write("你好，欢迎光临".getBytes());
          //硬编码
                response.sendFile("file/2.gif");

                out.flush();

                //关闭流
                out.close();
                in.close();
                socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

8、硬变软

问题：这里是硬编码，因为我们将数据写死了，没有灵活性
如何解决：让客户想访问什么就访问什么。

那么我们需要分析流的格式。
输入地址http://localhost:8088/file.3gif，比较服务器端接收到的客户端发送的数据，返回的信息可以看到，我们在端口号之后输入的信息会放在GET之后 file.3gif ，这个被称为rul地址

目的：截取file.3gif ，让这个信息动态的放入response.sendFile() 中。通过截取字符串完成此操作

```
客户端发送： GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: zh-Hans-CN,zh-Hans;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like
Gecko
Accept-Encoding: gzip, deflate
Host: localhost:8088
Connection: Keep-Alive
Cookie: Idea-e4e944d1=6d3f90e8-b2c3-459e-af67-ffb5dce7978f


客户端发送： GET /file.3gif HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: zh-Hans-CN,zh-Hans;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like
Gecko
Accept-Encoding: gzip, deflate
Host: localhost:8088
Connection: Keep-Alive
Cookie: Idea-e4e944d1=6d3f90e8-b2c3-459e-af67-ffb5dce7978f
```

9、Request

处理读取流的请求操作，完成字符串的截取。
此类专门处理请求信息。
request和respose连个类将请求信息和响应信息分别处理。让我们更好地进行数据地读写。

读取流作为形参放在Request地构造方法中。
SocketThread类的run方法中接收客户端数据部分转移到此类中

```
//处理读取流的请求类
public class Request {
    public Request(InputStream in) {
        try {
            //接收客户端发送的数据，并以字符串的方式打印出来
            byte[] by = new byte[1024];
            in.read(by);
            //trim()用于去掉空格
            String str = new String(by).trim();
            System.out.println("客户端发送： " + info);

            String[] array =  str.split("\\s+");
```

```
//            this.url = array[1].substring(1)

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public String getUrl() {
        return url;
    }
}
```

10、得到url

此处字符串的截取要根据协议来，http协议，它规定数据怎么放

有三种情况：
以get方式请求文件、以get方式请求业务组件、以post方式请求业务组件

```
GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: zh-Hans-CN,zh-Hans;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like
Gecko
Accept-Encoding: gzip, deflate
Host: localhost:8088
Connection: Keep-Alive
Cookie: Idea-e4e944d1=6d3f90e8-b2c3-459e-af67-ffb5dce7978f
```

```
请求方式    请求服务的路径url（统一资源定位符）
Accept：客户端能接收什么数据
Accept-Language:能接收什么语种
User-Agent:用户的浏览器类型
Accept-Encoding:压缩类型
Host:端口
Connection:连接类型
```

拆分字符串
新建属性url路径
为路径赋值
用get方法使url能够被访问

```
//处理读取流的请求类
public class Request {
    //客户端访问的url路径
    private String url;

    public Request(InputStream in) {
        try {
            //接收客户端发送的数据，并以字符串的方式打印出来
            byte[] by = new byte[1024];
            in.read(by);
            //trim()用于去掉空格
            String str = new String(by).trim();
```

```
            System.out.println("客户端发送：  " + info);

            //按空格拆分
            String[] array =  str.split("\\s+");
            //第二个元素即为我们想要的元素，第二个元素开头有个斜杠，将它截掉
           this.url = array[1].substring(1)

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public String getUrl() {
        return url;
    }
}
```

11、建立联系

建立Request和SocketThread之间的联系

SocketThread类的run方法：
新建读取流对象，
getUrl(),
response.sendFile()方法中的参数由固定的变为了可变的，url

```
@Override
    public void run() {
        try {
            //从socket中得到读取流，用于读取客户端给服务器的数据
                InputStream in = socket.getInputStream();
                //从socket中得到写入流，用于发送数据到客户端
                OutputStream out = socket.getOutputStream();

                //将写入流封装成响应对象，以便更好的发送数据
            Response response = new Response(out);

               //将读取流封装成请求对象，以便更好地读取数据
            Request request = new Request(in);
            //得到Url
            String url = request.getUrl();

                response.sendFile(url);

                out.flush();

                //关闭流
                out.close();
                in.close();
                socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

12、测试

在浏览器这个客户端访问file目录下的文件，查看代码能否实现功能:
http://localhost:8088/file/3.gif
http://localhost:8088/file/1.gif

这些路径就是我们访问网页时的链接和广告，这些路径对应着服务器端相应的文件

13、html文件

为我们的图片添加一些描述信息

SocketThread类的run方法:
//当想在图片后加一行描述信息时，如果把sendMessage这句话放在sendFile前，则图片显示乱码，将图片当成了文字
// 若把sendMessage这句话放在sendFile后，则描述信息显示不出来，描述信息被当成了图片
//新建一个html文件来解决

```
@Override
    public void run() {
        try {
            //从socket中得到读取流，用于读取客户端给服务器的数据
                InputStream in = socket.getInputStream();
                //从socket中得到写入流，用于发送数据到客户端
                OutputStream out = socket.getOutputStream();

                //将写入流封装成响应对象，以便更好的发送数据
            Response response = new Response(out);

               //将读取流封装成请求对象，以便更好地读取数据
            Request request = new Request(in);
            //得到Url
            String url = request.getUrl();

                response.sendFile(url);

             //当想在图片后加一行描述信息时，如果把sendMessage这句话放在sendFile前，则图片显示乱码，将图片当成了文字
            // 若把sendMessage这句话放在sendFile后，则描述信息显示不出来，描述信息被当成了图片
            //新建一个html文件来解决
//              response.sendMessage("神奇动物");

                out.flush();

                //关闭流
                out.close();
                in.close();
                socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

新建一个html文件来解决

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
</head>
<body>
神奇动物<img src="file/1.gif" width="300" height="300"> <!--br用来换行 -->
<img src="file/2.gif" width="300" height="300"><br>

</body>
</html>
```

在浏览器进行访问的时候，常常因为路径问题导致文档或图片不显示，可通过 右键→属性，查看文件路径，做到及时调整

14、测试

客户端访问此地址，login.html是我们html文件的文件名，它是直接放在工程下面的。
一定一定要注意路径问题

http://localhost:7808/login.html

html:称为超文本标记语言。以标记和子标记描述数据的一种语言。
超文本表示传输的数据不仅仅限于文本数据，而可以通过一系列的标记，关联各式各样的资源（图片、音频、视频、css . js等)。
客户端要想正确的访问html，除了要下载文本数据以外，还需要将这些标记链接的资源一一下载。

14、登录

怎么样将客户端填的数据给服务器

login.html文件

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
神奇动物<img src="file/1.gif" width="300" height="300"> <!--br用来换行 -->
<img src="file/2.gif" width="300" height="300"><br>

    <form action = "land" method="post">
        用户名：<input type = "text" name = "userName"><br>
        密码：<input type = "password" name = "pwd"><br>
        <input type="submit" value="提交">
    </form>

</body>
</html>
```

写好html文件后，重新写一下Request类

登录操作是一个业务，我们在Request中新建Map属性

http协议的三种情况：以get方式请求文件、以get方式请求业务组件、以post方式请求业务组件

15、http的三种情况

```
以GET方式请求文件       ------------封装url
GET /login.html HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: zh-Hans-CN,zh-Hans;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like
Gecko
Accept-Encoding: gzip, deflate
Host: localhost:8088
Connection: Keep-Alive

以GET方式请求业务组件  ---------- land---->url
 表单数据------>  paramMap     (userName=tom       pwd=123)
GET /land?userName=tom&pwd=123 HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Referer: http://localhost:8088/login.html
Accept-Language: zh-Hans-CN,zh-Hans;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like
Gecko
Accept-Encoding: gzip, deflate
Host: localhost:8088
Connection: Keep-Alive

以POST方式请求业务组件  ---------- land---->url
 表单数据------>  paramMap     (userName=tom       pwd=123)
POST /land HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Referer: http://localhost:8088/login.html
Accept-Language: zh-Hans-CN,zh-Hans;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like
Gecko
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Host: localhost:8088
Content-Length: 20
Connection: Keep-Alive
Cache-Control: no-cache

userName=tom&pwd=123
```

16、Request

对应于客户端请求的三种不同的流的格式，根据不同的格式，对request中的数据进行不同的封装

```
//处理读取流的请求类
public class Request {
```

```
    //客户端访问的url路径
    private String url;

    //封装表单数据的Map集合
    private Map<String, String> paramMap = new HashMap<>();

    public Request(InputStream in) {
        try {
            //接收客户端发送的数据，并以字符串的方式打印出来
            byte[] by = new byte[1024];
            in.read(by);
            //trim()用于去掉空格
            String str = new String(by).trim();

            //分别处理以get方式请求文件、以get方式请求业务组件、以post方式请求业务组件这三种
情况
            String[] array =  str.split("\\s+");

            if(str.startsWith("GET")){
                pressGet(array[1]);
            }
            else if(str.startsWith("POST")){
                pressPost(array);
            }
//            System.out.println(paramMap);
//            this.url = array[1].substring(1);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public String getUrl() {
        return url;
    }

    @Override
    public String toString() {
        return "server.Request{" +
                "url='" + url + '\'' +
                ", paramMap=" + paramMap +
                '}';
    }

    /**
     * 处理Post请求
     * @param array
     */
    public void pressPost(String[] array){
            this.url = array[1].substring(1);

            this.fullParam(array[array.length-1]);
    }

    /**
     * 处理Get请求
     * @param url
     */
    private void pressGet(String url) {
```

```
        if (url.indexOf("?") == -1) {
            this.url = url.substring(1);
        } else {
            //这里“[?]”是将“？”和正则表达式中的问号区分开来
            //?：等价于{0,1} 前一个规则可以不出现，最多出现一次
            String[] urlArray = url.split("[?]");
            this.url = urlArray[0].substring(1);
            this.fullParam(urlArray[1]);
        }

    }

    /**
     * 将账户密码放入map集合
     * @param s
     */
    private void fullParam(String s){
        String[] paramArray =s.split("&");
        for (String str : paramArray) {
            String[] parray = str.split("=");
            //考虑到用户输入的非法性，进行判断
            if(parray.length == 2) {
                paramMap.put(parray[0], parray[1]);
            }
            else{
                this.paramMap.put(parray[0],"");
            }
        }
    }

    /**
     * 根据表单名得到表单值
     * @param key 表单名
     * @return 表单值
     */
    public String getParameter(String key){
        return this.paramMap.get(key);
    }
}
```

单例模式实现业务

工厂类中利用单例模式实现各业务

```
//单例模式实现,立即加载类型
private static Map<String, Object> servletMap = new HashMap<>();
```

```
static{
        try {
            pro.load(new FileReader("web.txt"));

            //得到properties键的集合
           Set<String> kset =  pro.stringPropertyNames();
           for(String s : kset){
               //根据键得到值，值为servlet类路径
               String classPath = pro.getProperty(s);
               Class c = Class.forName(classPath);
               servletMap.put(s,c.getDeclaredConstructor().newInstance());
           }

        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println(servletMap);
    }
```

```
   public static Object getServlet(String url){
//        String classPath = pro.getProperty(url);
//        System.out.println(classPath + "======================");
//        if(classPath == null){
//            return null;
//        }
//
//        try {
//            //加载类，得到类模板
//            Class c = Class.forName(classPath);
//            return c.getDeclaredConstructor().newInstance();
//        } catch (Exception e) {
//            e.printStackTrace();
//        }
//        return null;


        return servletMap.get(url);
    }
```

利用注解实现业务

工厂模式中利用注解实现各业务