## 网络编程简介

### 概念理解

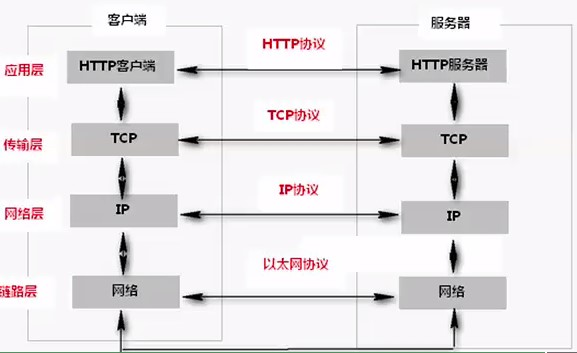
网络编程称为Socket编程或TCP/IP编程。主要的工作就是在发送端把信息通过规定好的协议进行组装包，在接收端按照规定好的协议把包进行解析，从而提取出对应的信息，达到网络通信的目的即通过网络将各个孤立的设备进行连接，通过信息交换实现人与人，人与计算机，计算机与计算机之间的通信。

任何客户端与服务端的信息交互都要用到网络编程，如navicat连接数据库则会用到数据库连接api如jdbc、odbc，驱动获取连接底层则会用到socket编程连接远程数据库。

### **[网络协议](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1%E5%8D%8F%E8%AE%AE/4438611" \t "https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1/_blank)**

网络通信中最重要的就是[网络通信协议](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1%E5%8D%8F%E8%AE%AE/4438611" \t "https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1/_blank)，是网络之间沟通、交流的桥梁，只有相同网络协议的[计算机](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA" \t "https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E9%80%9A%E4%BF%A1/_blank)才能进行信息的沟通与交流。TCP/IP协议族是协议的总称。

TCP/IP协议族四层模型图



## ip协议

### 相关概念

**ip**

IP（Internet Protocol）是一种网络层协议，通过IP数据包和[IP地址](https://baike.baidu.com/item/IP%E5%9C%B0%E5%9D%80/150859" \t "https://baike.baidu.com/item/IP%E5%8D%8F%E8%AE%AE/_blank)屏蔽掉了不同的物理网络（如[以太](https://baike.baidu.com/item/%E4%BB%A5%E5%A4%AA/518267" \t "https://baike.baidu.com/item/IP%E5%8D%8F%E8%AE%AE/_blank)网、[令牌环网](https://baike.baidu.com/item/%E4%BB%A4%E7%89%8C%E7%8E%AF%E7%BD%91/2076446" \t "https://baike.baidu.com/item/IP%E5%8D%8F%E8%AE%AE/_blank)等）的帧格式、地址格式等各种底层物理网络细节，使得各种物理网络的差异性对上层协议不复存在，从而使网络互联成为可能。IP协议的主要功能有：无连接数据报传送、数据报路由选择和差错控制。

**IP地址**

IP地址是IP协议提供的一种统一的地址格式，它为互联网上的每一个网络和每一台主机分配一个逻辑地址，以此来屏蔽物理地址的差异，现在使用的是IPv4，既由4个0-255之间的数字组成，在计算机内部存储时只需要4个字节即可。在计算机中，IP地址是分配给网卡的，每个网卡有一个唯一的IP地址，如果一个计算机有多个网卡，则该台计算机则拥有多个不同的IP地址，在同一个网络内部，IP地址不能相同。IP地址的概念类似于电话号码、身份证这样的概念。

**域名**

由于IP地址不方便记忆，所以有专门创造了域名(Domain Name)的概念，其实就是给IP取一个字符的名字，例如163.com、sina.com等。IP和域名之间存在一定的对应关系。其实在网络中只能使用IP地址进行数据传输，所以在传输以前，需要把域名转换为IP，这个由称作DNS（Domain Name System:域名系统）的服务器解析。在网络编程中，使用IP或域名来标识网络上的一台设备。

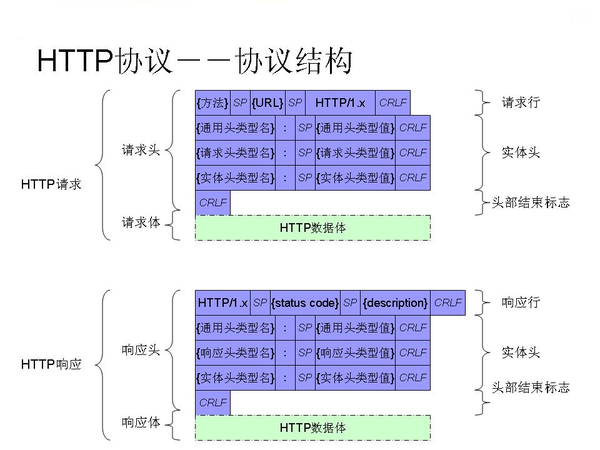
**端口**

为了在一台设备上可以运行多个程序，人为的设计了端口(Port)的概念，即ip找到具体主机再根据端口找到这个主机上的具体应用程序。规定一个设备有216个，也就是65536个端口，每个端口对应一个唯一的程序。每个网络程序，无论是客户端还是服务器端，都对应一个或多个特定的端口号。由于0-1024之间多被[操作系统](http://lib.csdn.net/base/operatingsystem" \t "_blank" \o "操作系统知识库)占用，所以实际编程时一般采用1024以后的端口号。使用端口号，可以找到一台设备上唯一的一个程序。所以如果需要和某台计算机建立连接的话，只需要知道IP地址或域名即可，但是如果想和该台计算机上的某个程序交换数据的话，还必须知道该程序使用的端口号。

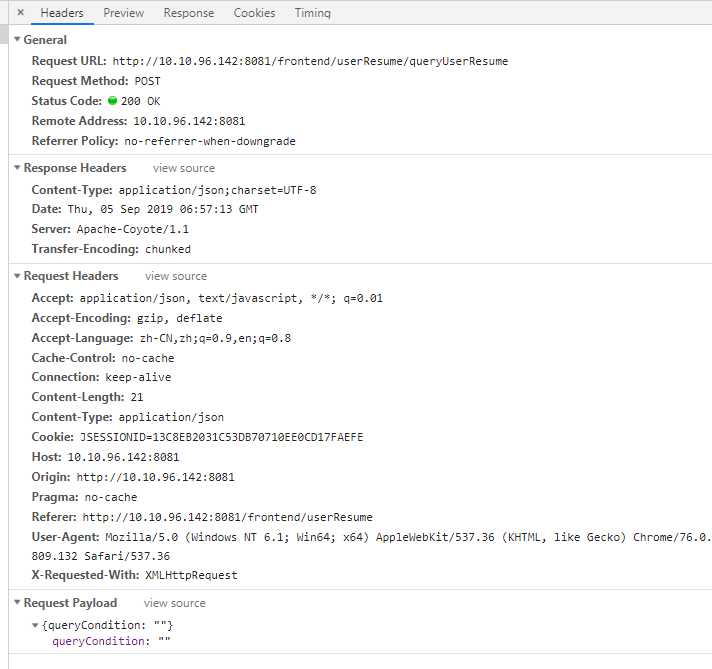
## http协议

### http简介

HTTP协议是一个简单的请求-响应协议，它通常运行在TCP之上。浏览器客户端和服务器端之间数据传输的格式规范。



这种报文结构是固定的以便解析，如tomact将请求报文解析封装成HttpRequest，将HttpResponse对象解析成响应报文返回给客户端,浏览器一般都集成抓包工具，按F12即可，浏览器已经将请求报文和响应报文解析解析成如图所示，可以更方便的查看请求和响应的各项信息。



### http请求

#### 协议版本

http1.0：当前浏览器客户端与服务器端建立连接之后，只能发送一次请求，一次请求之后连接关闭。

http1.1：当前浏览器客户端与服务器端建立连接之后，可以在一次连接中发送多次请求。（基本都使用1.1）

#### 请求资源

URL: 统一资源定位符。http://localhost:8080/day09/testImg.html。只能定位互联网资源。是URI的子集。

URI：统一资源标记符。/day09/hello。用于标记任何资源。可以是本地文件系统，局域网的资源（//192.168.14.10/myweb/index.html），可以是互联网。

#### 请求方式

根据HTTP标准，HTTP请求可以使用多种请求方法。HTTP1.0定义了三种请求方法： GET, POST 和 HEAD方法。HTTP1.1新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

|  |  |
| --- | --- |
| GET | 请求指定的页面信息，并返回实体主体 |
| POST | 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改 |
| HEAD | 类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头 |
| PUT | 从客户端向服务器传送的数据取代指定的文档的内容 |
| DELETE | 请求服务器删除指定的页面。 |
| CONNECT | HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。 |
| OPTIONS | 允许客户端查看服务器的性能。 |
| TRACE | 回显服务器收到的请求，主要用于测试或诊断 |

常用的请求方式：GET 和 POST

**GET方式提交特点**

* 地址栏（URI）会跟上参数数据。以？开头，多个参数之间以&分割。
* GET提交参数数据有限制，不超过1KB。
* GET方式不适合提交敏感密码。
* 浏览器直接访问的请求，默认提交方式是GET方式

|  |
| --- |
| GET /day09/testMethod.html?name=eric&password=123456 HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Referer: http://localhost:8080/day09/testMethod.html  Connection: keep-alive |

**POST方式提交特点**

* 参数不会跟着URI后面，参数而是跟在请求的实体内容中，多个参数之间以&分割。
* POST提交的参数数据没有限制。
* POST方式适合提交敏感数据。

|  |
| --- |
| POST /day09/testMethod.html HTTP/1.1  Host: localhost:8080  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,en-us;q=0.8,zh;q=0.5,en;q=0.3  Accept-Encoding: gzip, deflate  Referer: http://localhost:8080/day09/testMethod.html  Connection: keep-alive  name=eric&password=123456 |

**请求体**

只有POST提交的参数会放到实体内容中

**发起请求方式**

1. 客户端：浏览器、Postmen(谷歌插件)、RestClient、form表单
2. 服务端：阿帕奇的httpclient、jdk自带的HttpURLConnection
3. 前端：ajax（Asynchronous javascript and xml 异步的 JavaScript 和 XML）

注意：普通的B/S模式就是同步，而ajax请求默认是异步的，打断点测试时可能不是按顺序从上往下走返回成功或失败的方法。浏览器请求会自动处理响应，ajax需要自定义失败和成功的处理，如spring security登录拦截，发送的请求未登录返回登录页面，如果是浏览器发起则会自动处理响应展示返回的页面，如果是ajax则要程序员自己手动解析处理。

#### 重复提交

场景一：网络延时

在平时开发中，如果网速比较慢的情况下，用户提交表单后，发现服务器半天都没有响应，那么用户可能会以为是自己没有提交表单，就会再点击提交按钮重复提交表单，我们在开发中必须防止表单重复提交。

场景二：重新刷新

表单提交后用户点击【刷新】按钮导致表单重复提交

场景三：点击浏览器的【后退】按钮回退到表单页面后进行再次提交，用户提交表单后，点击浏览器的【后退】按钮回退到表单页面后进行再次提交

解决方案

前段javascript 解决

|  |
| --- |
| 1.表单是否已经提交标识，默认为false  var isFlag = false;  function submitFlag() {  if (!isFlag) {  isFlag = true;  return true;  } else {  return false;  }  }  2.将提交按钮设置为不可用，让用户没有机会点击第二次提交按钮  function dosubmit(){  //获取表单提交按钮  var btnSubmit = document.getElementById("submit");  //将表单提交按钮设置为不可用，这样就可以避免用户再次点击提交按钮  btnSubmit.disabled= "disabled";  //返回true让表单可以正常提交  return true;  } |

后端服务解决，对于【场景二】和【场景三】导致表单重复提交的问题，既然客户端无法解决，那么就在服务器端解决，在服务器端解决就需要用到session了。

具体的做法：  
1、获取用户填写用户名和密码的页面时向后台发送一次请求，这时后台会生成唯一的随机标识号，专业术语称为Token(令牌)。

2、将Token发送到客户端的Form表单中，在Form表单中使用隐藏域来存储这个Token，表单提交的时候连同这个Token一起提交到服务器端。

3、服务器端判断客户端提交上来的Token与服务器端生成的Token是否一致，如果不一致，那就是重复提交了，此时服务器端就可以不处理重复提交的表单。如果相同则处理表单提交，处理完后清除当前用户的Session域中存储的标识号。

#### 幂等性

**概念**

HTTP/1.1中对幂等性的定义是：一次和多次请求某一个资源对于资源本身应该具有同样的结果（网络超时等问题除外）。也就是说，其任意多次执行对资源本身所产生的影响均与一次执行的影响相同。

**举例说明**

以SQL为例，有下面三种场景，只有第三种场景需要开发人员使用其他策略保证幂等性

SELECT col1 FROM tab1 WHER col2=2，无论执行多少次都不会改变状态，是天然的幂等。

UPDATE tab1 SET col1=1 WHERE col2=2，无论执行成功多少次状态都是一致的，因此也是幂等操作。

UPDATE tab1 SET col1=col1+1 WHERE col2=2，每次执行的结果都会发生变化，这种不是幂等的。

**项目应用场景**

一、MQ出现非幂等性的情况

1.生成者重复发送消息给MQ:生成者把消息发送给MQ之后，MQ收到消息在给生产者返回ack的时候，网络中断了。这时MQ明明已经接收到了消息，但是生产者没接收到确定消息，就会认为MQ没有接收到消息。因此，在网络重新连接后，生产者会把已经发送的消息再次发送到MQ，如果MQ没有去重措施的话，那么就接收到了重复的消息。

2.MQ重复发送消息给消费者:消费者从MQ中拉取消息进行消费，当消费者已经消费了消息但还没向MQ返回ack的时候，消费者宕机或者网络断开了。所以消费者成功消费了消息的情况，MQ并不知道。当消费者重启或网络重连后，消费者再次去请求MQ拉取消息的时候，MQ会把已经消费的消息再次发送给消费者，如果消费者没有去重就直接消费，那么就会造成重复消费的情况。便会造成数据的不一致

二、同一个支付订单非幂等的情况

重复提交

**实现幂等解决方案**

1、token令牌

这种方式分成两个阶段：申请token阶段和支付阶段。 第一阶段，在进入到提交订单页面之前，需要订单系统根据用户信息向支付系统发起一次申请token的请求，支付系统将token保存到Redis缓存中，为第二阶段支付使用。 第二阶段，订单系统拿着申请到的token发起支付请求，支付系统会检查Redis中是否存在该token，如果存在，表示第一次发起支付请求，删除缓存中token后开始支付逻辑处理；如果缓存中不存在，表示非法请求，分布式系统接口调用一般就用token保证请求安全。 实际上这里的token是一个信物，支付系统根据token确认。不足是需要系统间交互两次，流程较上述方法复杂。

2、分布式锁

这里使用的防重表可以使用分布式锁代替，比如Redis。订单发起支付请求，支付系统会去Redis缓存中查询是否存在该订单号的Key，如果不存在，则向Redis增加Key为订单号。查询订单支付已经支付，如果没有则进行支付，支付完成后删除该订单号的Key。通过Redis做到了分布式锁，只有这次订单订单支付请求完成，下次请求才能进来。相比去重表，将放并发做到了缓存中，较为高效。思路相同，同一时间只能完成一次支付请求。

3、防重表

使用订单号orderNo做为去重表的唯一索引，每次请求都根据订单号向去重表中插入一条数据。第一次请求查询订单支付状态，当然订单没有支付，进行支付操作，无论成功与否，执行完后更新订单状态为成功或失败，删除去重表中的数据。后续的订单因为表中唯一索引而插入失败，则返回操作失败，直到第一次的请求完成（成功或失败）。可以看出防重表作用是加锁的功能。

4、乐观锁

如果只是更新已有的数据，没有必要对业务进行加锁，设计表结构时使用乐观锁，一般通过version来做乐观锁，这样既能保证执行效率，又能保证幂等。例如： UPDATE tab1 SET col1=1,version=version+1 WHERE version=#version# 不过，乐观锁存在失效的情况，就是常说的ABA问题，不过如果version版本一直是自增的就不会出现ABA的情况 。

### http响应

**响应报文示例**

|  |
| --- |
| HTTP/1.1 200 OK --响应行  Server: Apache-Coyote/1.1 --响应头（key-vaule）  Content-Length: 24  Date: Fri, 30 Jan 2015 01:54:57 GMT  --一个空行  ＜html＞  ＜head＞  ＜title＞Wrox Homepage＜/title＞  ＜/head＞  ＜body＞ --响应体：实体内容  ＜!-- body goes here --＞  ＜/body＞  ＜/html＞ |

#### 状态码

|  |  |
| --- | --- |
| 1\*\* | 信息，服务器收到请求，需要请求者继续执行操作 |
| 2\*\* | 成功，操作被成功接收并处理 |
| 3\*\* | 重定向，需要进一步的操作以完成请求 |
| 4\*\* | 客户端错误，请求包含语法错误或无法完成请求 |
| 5\*\* | 服务器错误，服务器在处理请求的过程中发生了错误 |

#### 响应体

有的是页面有的是数据，主要取决于后台设置。

### http头信息（重点）

**实体头主要属性含义**

|  |  |
| --- | --- |
| 属性 | 含义 |
| Accept | 浏览器接受的数据类型 |
| Accept-Charset | 浏览器接受的编码格式 |
| Accept-Encoding | 浏览器接受的数据压缩格式 |
| Host | 必须的，当前请求访问的目标地址 |
| Referer | 当前请求来自于哪里，可以知道在哪个页面发起的请求，从而防止非法链接(referer) |
| Content-Type | 定义网络文件的类型和网页的编码，决定浏览器将以什么形式、什么编码读取这个文件 |
| Cookie | 浏览器保存的cookie信息，服务端的叫session，在web应用中，使用http协议是无状态的，但是我们需要的web是有状态的，因此加入了cookie、session等机制实现有状态的的web |
| Connection | 浏览器跟服务器连接状态close: 连接关闭 keep-alive：保存连接，一般为长连接省去较多的TCP建立和关闭的操作，减少浪费，节约时间 |
| Set-Cookie | 表示服务器发送给浏览器的cookie信息 |
| Location | 表示重定向的地址，该头和302的状态码一起使用，详情查看下面的重定向 |
| ... | ... |

#### Cookie、Set-Cookie协议之会话技术

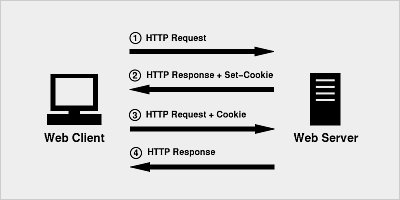
会话:从打开浏览器到关闭浏览器的过程

Cookie和Session是为了在无状态的HTTP协议之上维护会话状态，使得服务器可以知道当前是和哪个客户在打交道，因为HTTP协议是无状态的，即每次用户请求到达服务器时，HTTP服务器并不知道这个用户是谁、是否登录过等。现在的服务器之所以知道我们是否已经登录，是因为服务器在登录时设置了浏览器的Cookie

Cookie的局限：Cookie只能存字符串类型。不能保存对象只能存非中文。1个Cookie的容量不超过4KB。如果要保存非字符串，超过4kb内容，只能使用session技术保存在服务器端。

**实现机制：**

浏览器向某个URL发起HTTP请求，对应的服务器收到该HTTP请求如果是第一次访问创建session对象，给session对象分配一个唯一的ID叫JSESSIONID,把JSESSIONID作为Cookie的值发送给浏览器保存Cookie cookie = new Cookie("JSESSIONID", sessionID),response.addCookie(cookie),返回给浏览器的HTTP响应,在响应头加入Set-Cookie字段，它的值是要设置的Cookie，浏览器收到来自服务器的HTTP响应。浏览器在响应头中发现Set-Cookie字段，就会将该字段的值保存在内存或者硬盘中，浏览器下次给该服务器发送HTTP请求时，将会在请求头中添加Cookie**:** JSESSIONID=3CB452DB3C0AD5D9CB161A8DBDCE81C0。浏览器可以存储多个域名下的Cookie，但只发送当前请求的域名曾经指定的Cookie， 这个域名也可以在Set-Cookie字段中指定。服务器收到这个HTTP请求，发现请求头中有Cookie字段， 便知道之前就和这个用户打过交道了，根据JSESSIONID获取已经存在的session，这样就保证了同一次会话中不同requset中的session相同。如果没有则当时第一次请求，继续创建session。服务端保存的HttpSession对象默认是30分钟有效时间，而当客户端关闭后客户端的cookie被清除，再打开浏览器发送请求那么客户端找不到对应的sessionId，所以会创建新的HttpSession对象即一个新的会话对象，而之前的session也不会立马清除，会到期后再清除。



#### Location协议之重定向

重定向：当客户端向服务器访问A资源时，服务器返回一个302代码和一个URL给客户端如response.setStatus(302)，response.setHeader("Location",”www.ba.com”),客户端见到302，立马就向服务器给的URL进行访问,是2次请求，地址url发生变化。

服务端的一个概念-转发

转发：当客户端向服务器请求访问A资源，服务器将其转发到B这，返回B资源，首先转发是服务器的操作，通过请求分派器RequestDispatcher实现，如

 req.getRequestDispatcher("servlet别名").forward(req.resp)，而重定向是客户端的操作，转发只有一次请求，tomcat等容器内部跳转url不变，而重定向有两次请求，转发只能在站内转发，重定向可以随意定向。

#### Referer协议之同源策略

**同源的定义**

如果两个页面的协议，端口（如果有指定）和主机都相同，则两个页面具有相同的源。我们也可以把它称为“协议/主机/端口 tuple”，或简单地叫做“tuple". ("tuple" ，“元”，是指一些事物组合在一起形成一个整体，比如（1，2）叫二元，（1，2，3）叫三元).

**同源策略**

浏览器内发起的ajax（XMLHttpRequest）请求要符合两个URI同源当且仅当它们的协议://host:port相同，一种是限制发起AJAX请求(XMLHttpRequest，Fetch)，另一种是可以发送请求但拦截其他跨站请求的返回结果，这取决于请求是否为简单请求。同源策略是http的一种安全机制，但没有刺不穿的盾，只是攻击的成本和攻击成功后获得的利益成不成正比，就和加密破解一样，再严格的加密也能破解，但是成本太高破解就没有意义了。

**作用**

* 防止csrf攻击（Cross-site request forgery跨站请求伪造）。

Csrf攻击步骤：登录受信任网站A，并在本地生成Cookie，在不登出A的情况下，访问危险网站B，B网站内部发起对A网站的访问，浏览器会带上之前生成的Cookie，这样相当于B网站已经登录了A网站。

* 限制的Dom查询

由于没有同源策略的限制，钓鱼网站可以直接拿到别的网站的Dom const iframe = window.frames['yinhang'] const node = iframe.document.getElementById('你输入账号密码的Input')

**跨域问题**

在同源策略下的确能防御非法的攻击，但有的正常需求如前后端分离的项目不同源要能正常访问，这就产生了跨域问题。

**解决方法**

CORS（Cross-origin resource sharing，跨域资源共享）标准的跨域解决方案

CORS背后的基本思想是使用自定义的HTTP头部允许浏览器和服务器相互了解对方，从而决定请求或响应成功与否.

Access-Control-Allow-Origin:指定授权访问的域  
Access-Control-Allow-Methods：授权请求的方法（GET, POST, PUT, DELETE，OPTIONS等)

|  |
| --- |
| public class CORSFilter implements Filter {  @Override  public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {  HttpServletResponse response = (HttpServletResponse) servletResponse;  HttpServletRequest request = (HttpServletRequest)servletRequest;  response.setHeader("Access-Control-Allow-Origin", \*);  response.setHeader("Access-Control-Allow-Methods", "POST, GET, OPTIONS, DELETE");  response.setHeader("Access-Control-Max-Age", "3600");  response.setHeader("Access-Control-Allow-Headers", "x-requested-with,Authorization");  response.setHeader("Access-Control-Allow-Credentials", "true");  response.setHeader("Access-Control-Allow-Headers","Accept,Origin,,sessiomAllowModify,token");//当在头部加上自定属性值，需要将该属性放在里面。  String method = request.getMethod();  if(method.equalsIgnoreCase("OPTIONS")){//预请求  response.setStatus(HttpServletResponse.SC\_OK);  servletResponse.getOutputStream().write("OPTIONS returns OK".getBytes("utf-8"));  }else{  filterChain.doFilter(servletRequest, servletResponse);  }  } } |

还可以设置response的header中的Access-Control-Allow-Origin的值。使用接口网关使用nginx转发。使用JSONP

### https基础

Hypertext Transfer Protocol Secure，超文本传输安全协议 ，SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，http 和 https 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443。

#### 工作原理

我们都知道 HTTPS 能够加密信息，以免敏感信息被第三方获取，所以很多银行网站或电子邮箱等等安全级别较高的服务都会采用 HTTPS 协议。

客户端在使用 HTTPS 方式与 Web 服务器通信时有以下几个步骤，如图所示。



　　（1）客户使用 https 的 URL 访问 Web 服务器，要求与 Web 服务器建立 SSL 连接。

　　（2）Web 服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。

　　（3）客户端的浏览器与 Web 服务器开始协商 SSL 连接的安全等级，也就是信息加密的等级。

　　（4）客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。

　　（5）Web 服务器利用自己的私钥解密出会话密钥。

　　（6）Web 服务器利用会话密钥加密与客户端之间的通信。

#### 优点

http 是超文本传输协议，信息是明文传输，https 则是具有安全性的 ssl(Secure Sockets Layer [安全套接层](https://baike.baidu.com/item/%E5%AE%89%E5%85%A8%E5%A5%97%E6%8E%A5%E5%B1%82" \t "https://baike.baidu.com/item/ssl/_blank))加密传输协议，比 http 协议安全。

#### 缺点

HTTPS 协议握手阶段比较费时，会使页面的加载时间延长近 50%，增加 10% 到 20% 的耗电；

HTTPS 连接缓存不如 HTTP 高效，会增加数据开销和功耗，甚至已有的安全措施也会因此而受到影响；

http升级https要申请证书,SSL 证书需要钱，功能越强大的证书费用越高，个人网站、小网站没有必要一般不会用。

SSL 证书通常需要绑定 IP，不能在同一 IP 上绑定多个域名，IPv4 资源不可能支撑这个消耗。

HTTPS 协议的加密范围也比较有限，在黑客攻击、拒绝服务攻击、服务器劫持等方面几乎起不到什么作用。最关键的，SSL 证书的信用链体系并不安全，特别是在某些国家可以控制 CA 根证书的情况下，中间人攻击一样可行。

### XSS攻击

（此不属于http协议内容，暂时放在此文档里）

概念

[跨站脚本攻击](https://baike.baidu.com/item/%E8%B7%A8%E7%AB%99%E8%84%9A%E6%9C%AC%E6%94%BB%E5%87%BB/8186208" \t "https://baike.baidu.com/item/xss/_blank)(Cross Site Scripting)，为了不和[层叠样式表](https://baike.baidu.com/item/%E5%B1%82%E5%8F%A0%E6%A0%B7%E5%BC%8F%E8%A1%A8" \t "https://baike.baidu.com/item/xss/_blank)(Cascading Style Sheets, [CSS](https://baike.baidu.com/item/CSS/5457" \t "https://baike.baidu.com/item/xss/_blank))的缩写混淆，故将跨站脚本攻击缩写为XSS。恶意攻击者往Web页面里插入恶意的Script代码，当用户浏览该页之时，嵌入其中Web里面的Script代码会被执行，从而达到恶意攻击用户的目的。

示例

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <title></title>  </head>  <body>  <input type="text" id='liuyan'></input>  <button id="btn\_submit">提交</button>  <div>  <li id="show">  </li>  </div>  <script src="js/jquery-1.9.1.min.js"></script>  <script>  $("#btn\_submit").click(function(){  $("#show").html( $("#liuyan").val())  })  </script>  </body>  </html> |

当在输入框中输入<script>alert("你是sb")</script>后提交会有弹框出来,如果这里面是获取cookie并向指定的服务器发送，那么就能利用cookie用于模仿合法用户。

解决方法

前段方式

|  |
| --- |
| <script>  $("#btn\_submit").click(function() {  var value = $("#liuyan").val();  var full = value + '';  var dd = full.replace(/<\/?.+?>/g, "");  var dds = dd.replace(/ /g, "");  $("#show").html(dds)  })  </script> |

后端方式：定义过滤器，实际项目中都引入第三方包的过滤器。

|  |
| --- |
| /\*\*  \* 拦截防止sql注入、xss注入  \*/  public class XssFilter implements Filter {  public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)throws IOException,ServletException {  XssHttpServletRequestWrapper xssRequest = new XssHttpServletRequestWrapper(  (HttpServletRequest) request);  filterChain.doFilter(xssRequest, response);  }  }  /\*\*  \* xss过滤  \*/  public class XssHttpServletRequestWrapper extends HttpServletRequestWrapper {  HttpServletRequest orgRequest = null;  public XssHttpServletRequestWrapper(HttpServletRequest request) {  super(request);  orgRequest = request;  }  /\*\*  \* 覆盖getParameter方法，将参数名和参数值都做xss过滤。  \* 如果需要获得原始的值，则通过super.getParameterValues(name)来获取  \* getParameterNames,getParameterValues和getParameterMap也可能需要覆盖  \*/  @Override  public String getParameter(String name) {  String value = super.getParameter(xssEncode(name));  if (value != null) {  value = xssEncode(value);  }  return value;  }  /\*\*  \* 覆盖getHeader方法，将参数名和参数值都做xss过滤。  \* 如果需要获得原始的值，则通过super.getHeaders(name)来获取  \* getHeaderNames 也可能需要覆盖  \*/  @Override  public String getHeader(String name) {  String value = super.getHeader(xssEncode(name));  if (value != null) {  value = xssEncode(value);  }  return value;  }  /\*\*  \* 将容易引起xss漏洞的半角字符直接替换成全角字符  \* @param s  \* @return  \*/  private static String xssEncode(String s) {  if (s == null || "".equals(s)) {  return s;  }  StringBuilder sb = new StringBuilder(s.length() + 16);  for (int i = 0; i < s.length(); i++) {  char c = s.charAt(i);  switch (c) {  case '>':  sb.append('＞');//全角大于号  break;  case '<':  sb.append('＜');//全角小于号  break;  case '\'':  sb.append('‘');//全角单引号  break;  case '\"':  sb.append('“');//全角双引号  break;  case '&':  sb.append('＆');//全角  break;  case '\\':  sb.append('＼');//全角斜线  break;  case '#':  sb.append('＃');//全角井号  break;  default:  sb.append(c);  break;  }  }  return sb.toString();  }  /\*\*  \* 获取最原始的request  \*  \* @return  \*/  public HttpServletRequest getOrgRequest() {  return orgRequest;  }  /\*\*  \* 获取最原始的request的静态方法  \* @return  \*/  public static HttpServletRequest getOrgRequest(HttpServletRequest req) {  if (req instanceof XssHttpServletRequestWrapper) {  return ((XssHttpServletRequestWrapper) req).getOrgRequest();  }  return req;  }  public static void main(String[] args) {  System.out.println(XssHttpServletRequestWrapper.xssEncode("<script>alert(1)</script>"));  } |

## tcp/udp协议

### 简介

TCP（Transmission Control Protocol [传输控制协议](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E6%8E%A7%E5%88%B6%E5%8D%8F%E8%AE%AE/9727741" \t "https://baike.baidu.com/item/TCP/_blank)）是一种面向连接的、可靠的、基于字节流的[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82/4329536" \t "https://baike.baidu.com/item/TCP/_blank)通信协议

UDP (User Datagram Protocol用户数据报协议)是一种无连接的[传输层](https://baike.baidu.com/item/%E4%BC%A0%E8%BE%93%E5%B1%82" \t "https://baike.baidu.com/item/UDP/_blank)通信协议

Tcp和udp的区别

udp:

是面向无连接, 将数据及源的封装成数据包中,不需要建立建立连接

每个数据报的大小在限制64k内

因无连接,是不可靠协议

不需要建立连接,速度快，如传长视频用此协议

tcp：

建议连接，形成传输数据的通道.

在连接中进行大数据量传输，以字节流方式

通过三次握手完成连接,是可靠协议

必须建立连接m效率会稍低

### Tcp连接和终止

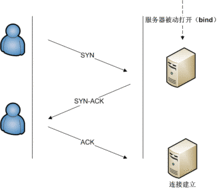
**连接建立**

TCP是因特网中的传输层协议，使用[三次握手协议](https://baike.baidu.com/item/%E4%B8%89%E6%AC%A1%E6%8F%A1%E6%89%8B%E5%8D%8F%E8%AE%AE" \t "https://baike.baidu.com/item/TCP/_blank)建立连接。当主动方发出SYN连接请求后，等待对方回答SYN+ACK [1] ，并最终对对方的 SYN 执行 ACK 确认。这种建立连接的方法可以防止产生错误的连接，TCP使用的流量控制协议是可变大小的滑动窗口协议。

SYN - 创建一个连接，FIN -  终结一个连接，ACK - 确认接收到的数据

TCP三次握手的过程如下：

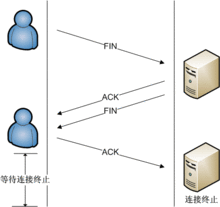
1. 客户端发送SYN（SEQ=x）报文给服务器端，进入SYN\_SEND状态。
2. 服务器端收到SYN报文，回应一个SYN （SEQ=y）ACK(ACK=x+1）报文，进入[SYN\_RECV](https://baike.baidu.com/item/SYN_RECV" \t "https://baike.baidu.com/item/TCP/_blank)状态。
3. 客户端收到服务器端的SYN报文，回应一个ACK(ACK=y+1）报文，进入Established状态。

[](https://baike.baidu.com/pic/TCP/33012/0/a1ad16fa330e9cae59ee90ca?fr=lemma%26ct=single)

三次握手完成，TCP客户端和服务器端成功地建立连接，可以开始传输数据了。

**连接终止**

建立一个连接需要三次握手，而终止一个连接要经过四次握手，这是由TCP的半关闭（half-close）造成的。具体过程如下图所示。

[](https://baike.baidu.com/pic/TCP/33012/0/bf4875638e06d1590c33faf9?fr=lemma%26ct=single)

1. 首先进行关闭的一方即发送第一个FIN将执行主动关闭，而另一方收到这个FIN执行被动关闭。
2. 当服务器收到这个FIN，它发回一个ACK，确认序号为收到的序号加1。和SYN一样，一个FIN将占用一个序号。
3. 同时TCP服务器还向应用程序（即丢弃服务器）传送一个文件结束符。接着这个服务器程序就关闭它的连接，导致它的TCP端发送一个FIN
4. 客户必须发回一个确认，它发回一个ACK

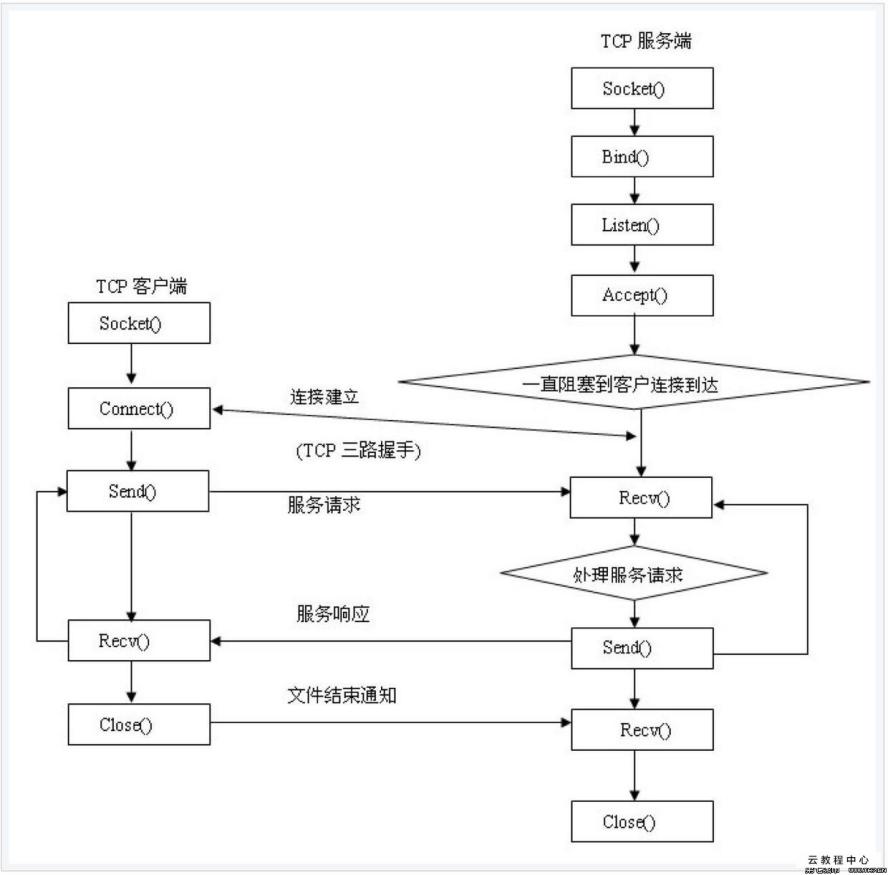
## socket

### 简介

套接字（socket）是一个抽象层，应用程序可以通过它发送或接收数据，可对其进行像对文件一样的打开、读写和关闭等操作。套接字允许应用程序将I/O插入到网络中，并与网络中的其他应用程序进行通信，通讯的两端都有Sokcet，网络通讯其实就是Sokcet间的通讯，数据在两个Sokcet间通过IO传输。网络套接字是IP地址与端口的组合。网络编程即使用套接字来达到进程间通信，称为Socket编程或TCP/IP编程。

任何2台计算通信都要用到网络编程

### 调用流程



### java.net

java.net包就是java网络编程相关的api。socket 是一种通讯标准简称，socket编程语言有c、java等，java 本身不带socket通讯底层实现的，而是通过调用系统底层的winsock API 进行的二次封装，如对ip、socket等等信息封装成java对象，用JAVA发开出来的socket程序可以在任何支持JAVA虚拟机上运行，不用修改任何代码。

**java网络编程简单示例**



先启动服务端再启动客户端后显示如下

