**[C:\Users\ADMINI~1\AppData\Local\Temp\enhtmlclip\YX0014902e.png](file:///C:\Users\Administrator\AppData\Local\Temp\enhtmlclip\20170722笔记.docx)[C:\Users\ADMINI~1\AppData\Local\Temp\enhtmlclip\YX0014903d.png](file:///C:\Users\Administrator\AppData\Local\Temp\enhtmlclip\20170723-课堂笔记.docx)[C:\Users\ADMINI~1\AppData\Local\Temp\enhtmlclip\YX0014904d.png](file:///C:\Users\Administrator\AppData\Local\Temp\enhtmlclip\7月19号上课笔记.docx)[C:\Users\ADMINI~1\AppData\Local\Temp\enhtmlclip\YX0014930b.png](file:///C:\Users\Administrator\AppData\Local\Temp\enhtmlclip\20170726分布式通信协议-http笔记.docx)**

**[C:\Users\ADMINI~1\AppData\Local\Temp\enhtmlclip\YX0014931b.png](file:///C:\Users\Administrator\AppData\Local\Temp\enhtmlclip\0729分布式通信框架-RMI.docx)[C:\Users\ADMINI~1\AppData\Local\Temp\enhtmlclip\YX0014932a.png](file:///C:\Users\Administrator\AppData\Local\Temp\enhtmlclip\分布式通信框架-webservice分析笔记.docx)**

**作为一名架构师，我们要专业，要能看懂代码，即使光着臂膀去机房，也能独挡一面！即使同事搞不定问题，**

**或者撂挑子，你也能给老大一个坚定的眼神：不怕，有我在！还能在会议室上滔滔不绝，如若无人，**

**让不懂技术的妹子看你时眼神迷离，就好想落霞与孤鹜齐飞！**

**高并发、海量数据**

**什么是分布式？**

1. **任务分解**
2. **节点通信**

**分布式和集群的关系？**

**电商平台：  用户、 商品、订单、 交易**

**分布式： 一个业务拆分成多个子系统，部署在不同的服务器上**

**集群：  同一个业务，部署在多个服务器上**

**计算机的发展历史**

1. **1946  情人节**
2. **1964  IBM  SYSTEM/360**

**超强的计算能力、 高可靠性**

1. **X86 CPU**
2. **RISC CPU  小型机**

**软件架构往集中式发展， 成为当时软件架构的主流**

**分布式架构的发展**

1. **时机成熟了**

**PC机的性能不断提升**

1. **企业必须要做**

**去IOE**

**IBM小型机 、 Oracle Rac、 EMC存储设备**

**PC            mysql    mariadb**

**2013年5月17号，最后一台IMB小型机下线**

**单机计算机的架构->分布式计算机架构**

**架构的发展演变过程**

**lamp**

**BAT**

**什么是大型网站**

1. **访问量（tps、qps）**
2. **数据量（存储数据量）**

**电商平台的发展**

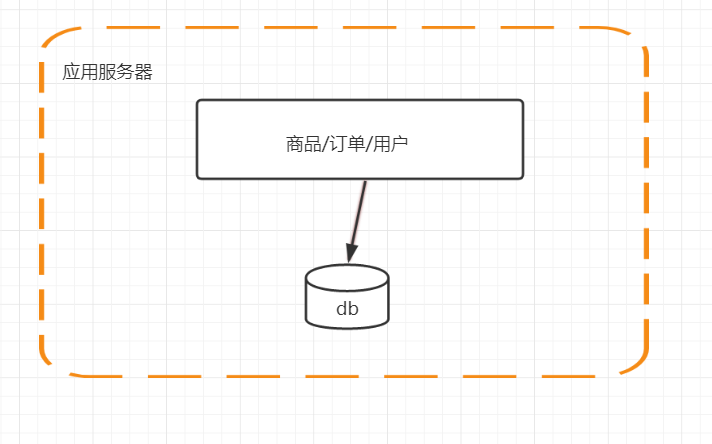
**用户、 商品、订单**

**容器： tomcat   ；jsp/servlet**

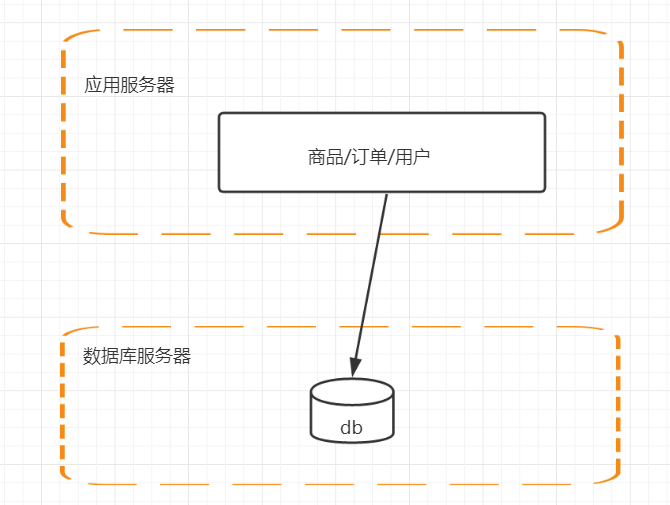
**数据库存储： mysql**

**网络层面的知识：tcp/udp**

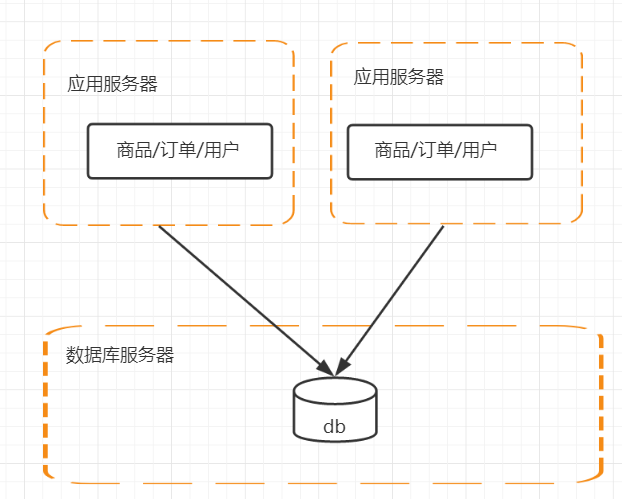
**第一版应用**



**第二版 单击负载越来越高，数据库服务器和应用服务器分离**



**第三版  应用服务器做集群**



**1 .session**

**cookie 存储的是JSESSIONID**

**ConcurrentMap   key JSESSIONID  values session**

**解决session跨域共享问题**

1. **session sticky**
2. **session replication**
3. **session 集中存储**

**存储在db、 存储在缓存服务器 （redis）**

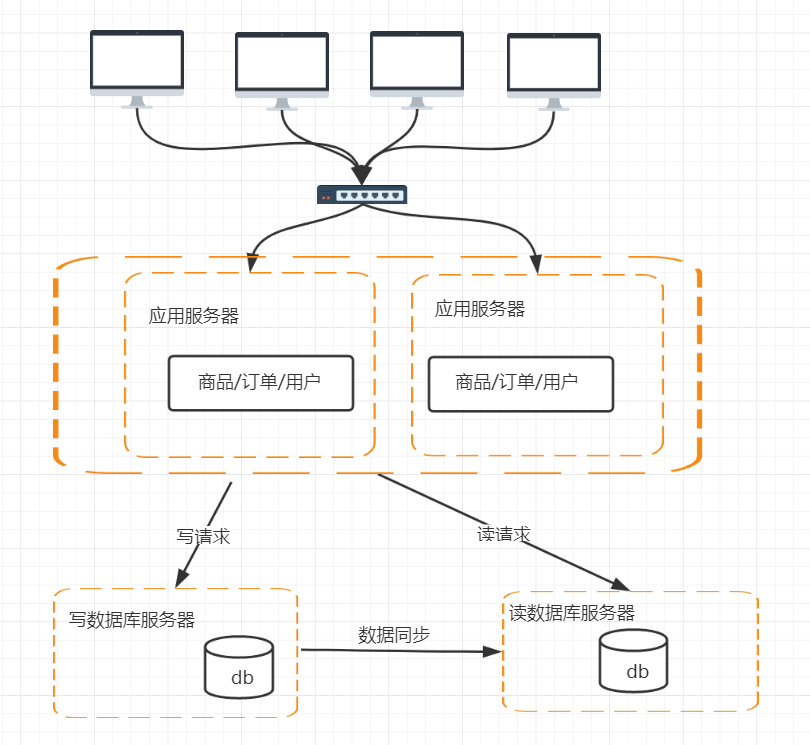
1. **cookie (主流)**

**access\_token(userid/token/timestamp)**

**soa架构和微服务架构**

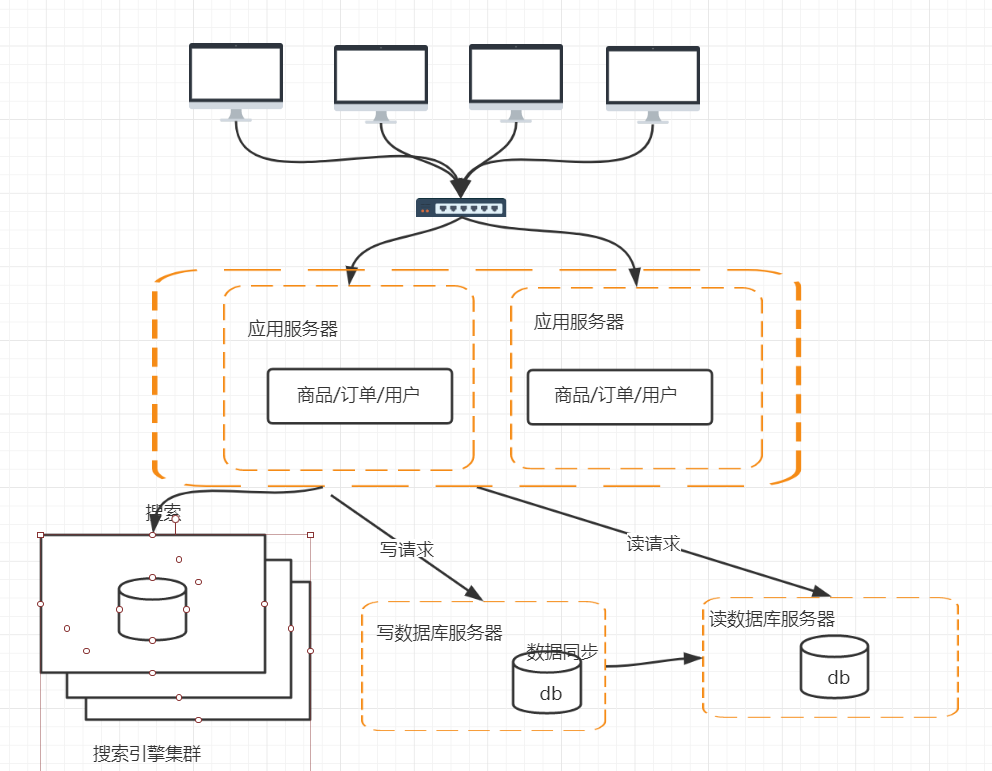
1. **如何做请求转发**

**第四版  数据库的高性能操作**



1. **数据库读写分离怎么操作**
2. **数据库的数据同步**
3. **数据库路由 mycat**

**电商平台最多的操作： 搜索商品**



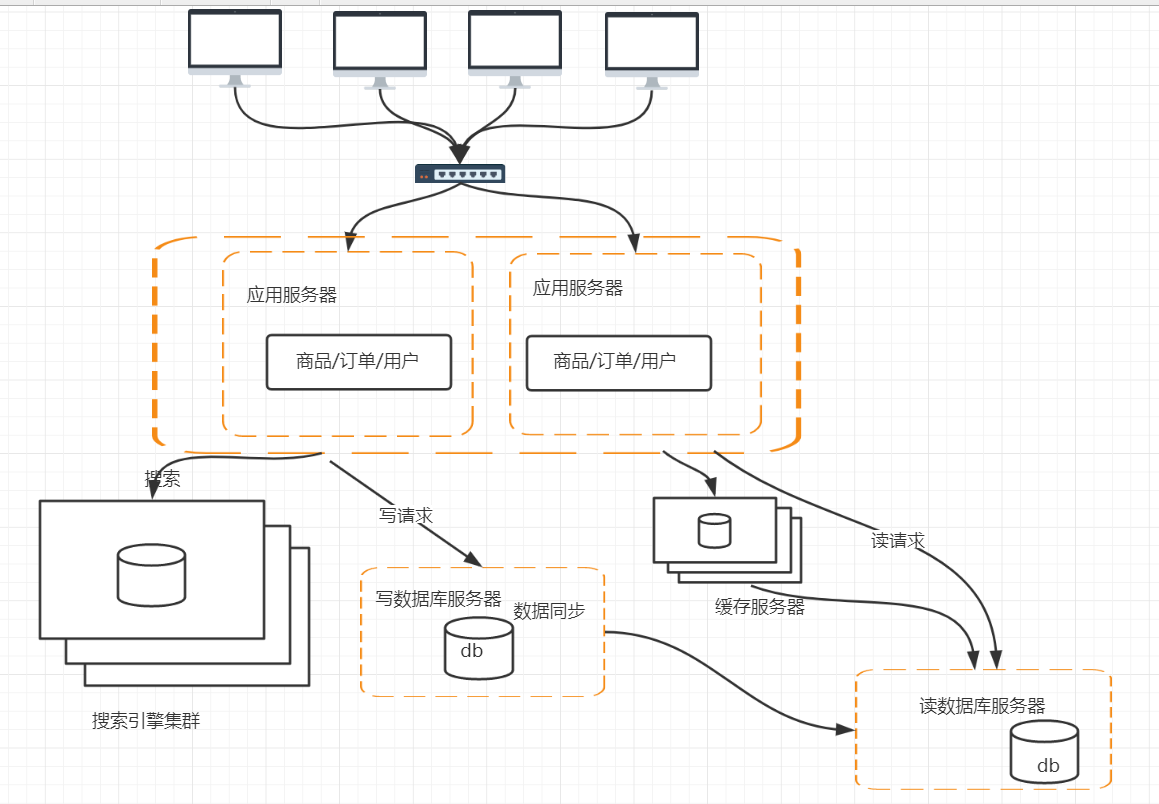
**问题：**

1. **搜索引擎的索引数据怎么去做同步，实时增量同步？ 还是定时全量同步？**

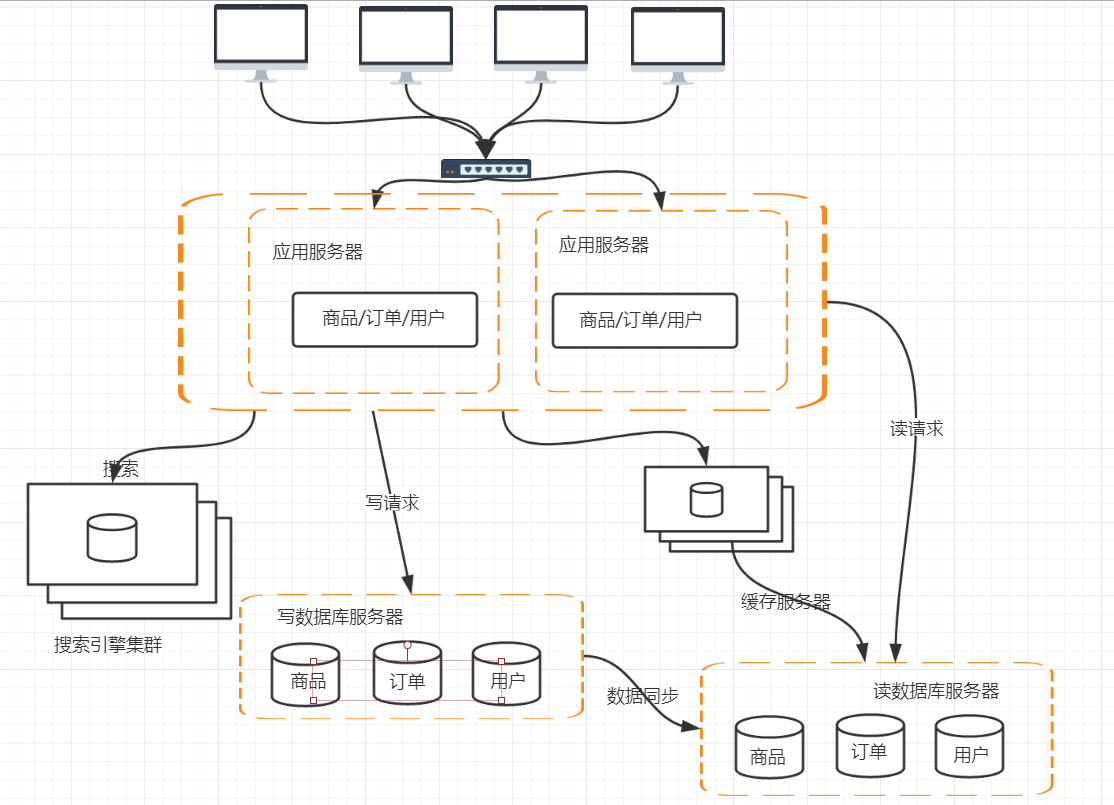
**第六版 解决访问量持续增高，引入缓存机制**

**用户量是没有上限的**

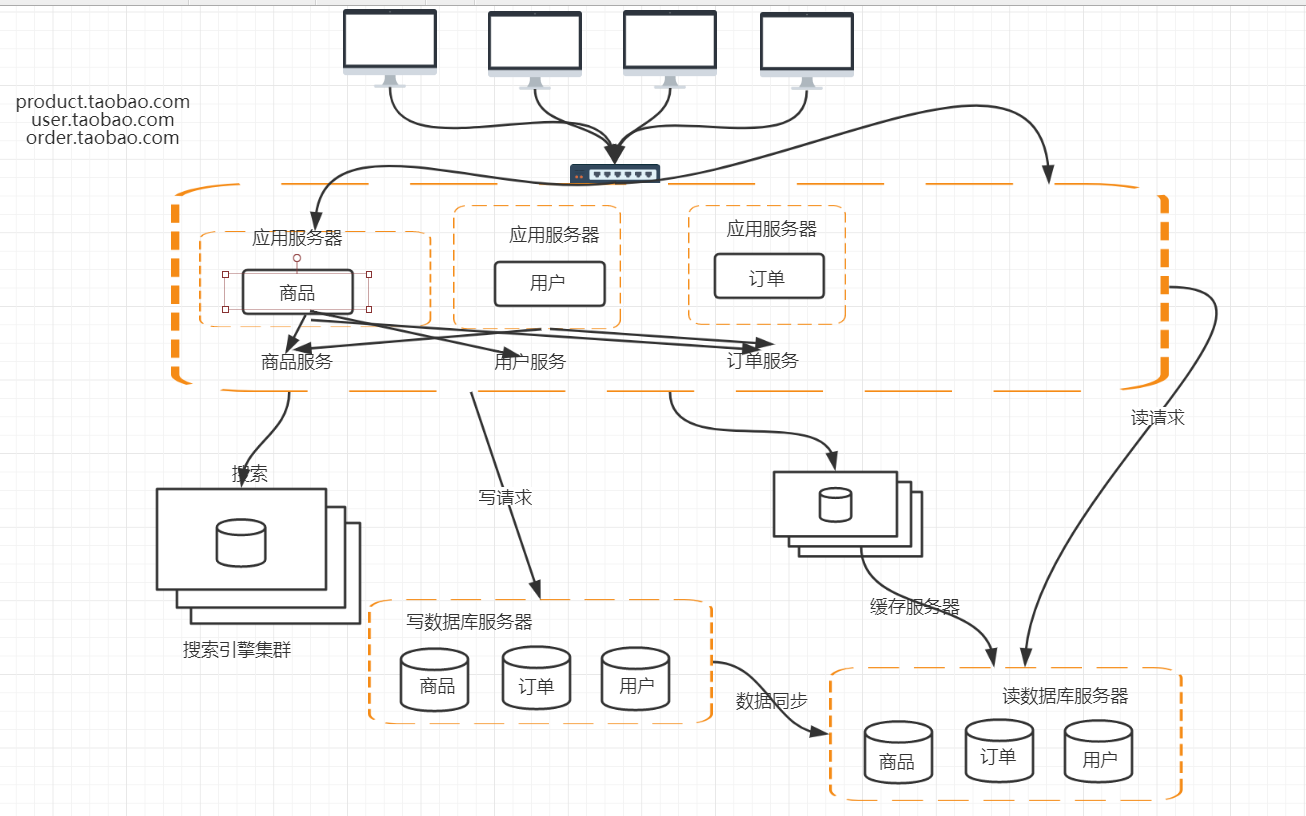
**缓存、 限流、 降级**



**第8️版本 数据库的水平/垂直拆分**



**第9版本**



# 第二节课

**网络协议： TCP/IP 和UDP/IP**

**TCP/IP**

TCP/IP（Transmission Control Protocol/Internet Protocol）是一种可靠的网络数据传输控制协议。定义了主机如何连入因特网以及数据如何在他们之间传输的标准。

TCP/IP协议参考模型把所有TCP/IP系列协议归类到四个抽象层中；

每一个抽象层建立在低一层提供的服务上，并且为高一层提供服务

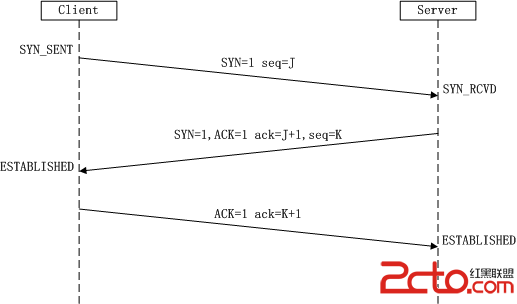
|  |  |
| --- | --- |
| C:\Users\ADMINI~1\AppData\Local\Temp\enhtmlclip\Image.jpg | ICMP:控制报文协议  IGMP:internet组管理协议  ARP:地址解析协议  RARP:反向地址转化协议 |

OSI模型（开放式系统互联通信参考模型），它是由国际标准化组织提出的，试图使各种计算机在世界范围内互联为网络的标准框架

OSI模型多了表达层、会话层

**3次握手协议**

所谓三次握手（Three-Way Handshake）即建立TCP连接，就是指建立一个TCP连接时，需要客户端和服务端总共发送3个包以确认连接的建立



（1）第一次握手：Client将标志位SYN置为1，随机产生一个值seq=J，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。

（2）第二次握手：Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。

（3）第三次握手：Client收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。

*SYN攻击：*

*在三次握手过程中，Server发送SYN-ACK之后，收到Client的ACK之前的TCP连接称为半连接（half-open connect），此时Server处于SYN\_RCVD状态，当收到ACK后，Server转入ESTABLISHED状态。SYN攻击就是Client在短时间内伪造大量不存在的IP地址，并向Server不断地发送SYN包，Server回复确认包，并等待Client的确认，由于源地址是不存在的，因此，Server需要不断重发直至超时，这些伪造的SYN包将产时间占用未连接队列，导致正常的SYN请求因为队列满而被丢弃，从而引起网络堵塞甚至系统瘫痪。SYN攻击时一种典型的DDOS攻击，检测SYN攻击的方式非常简单，即当Server上有大量半连接状态且源IP地址是随机的，则可以断定遭到SYN攻击了，使用如下命令可以让之现行：*

*#netstat -nap | grep SYN\_RECV*

**4次挥手协议**

三次握手耳熟能详，四次挥手估计就听得比较少了，所谓四次挥手（Four-Way Wavehand）即终止TCP连接，就是指断开一个TCP连接时，需要客户端和服务端总共发送4个包以确认连接的断开

|  |  |
| --- | --- |
| C:\Users\ADMINI~1\AppData\Local\Temp\enhtmlclip\Image(9).png | **单工**：数据传输只支持数据在一个方向上传输  **半双工**：数据传输允许数据在两个方向上传输，但是在某一时刻，只允许在一个方向上传输，实际上有点像切换方向的单工通信  **全双工**：数据通信允许数据同时在两个方向上传输，因此全双工是两个单工通信方式的结合，它要求发送设备和接收设备都有独立的接收和发送能力 |

由于TCP连接时全双工的，因此，每个方向都必须要单独进行关闭，这一原则是当一方完成数据发送任务后，发送一个FIN来终止这一方向的连接，收到一个FIN只是意味着这一方向上没有数据流动了，即不会再收到数据了，但是在这个TCP连接上仍然能够发送数据，直到这一方向也发送了FIN。首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭，上图描述的即是如此。

（1）第一次挥手：Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN\_WAIT\_1状态。

（2）第二次挥手：Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE\_WAIT状态。

（3）第三次挥手：Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST\_ACK状态。

（4）第四次挥手：Client收到FIN后，Client进入TIME\_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入CLOSED状态，完成四次挥手。

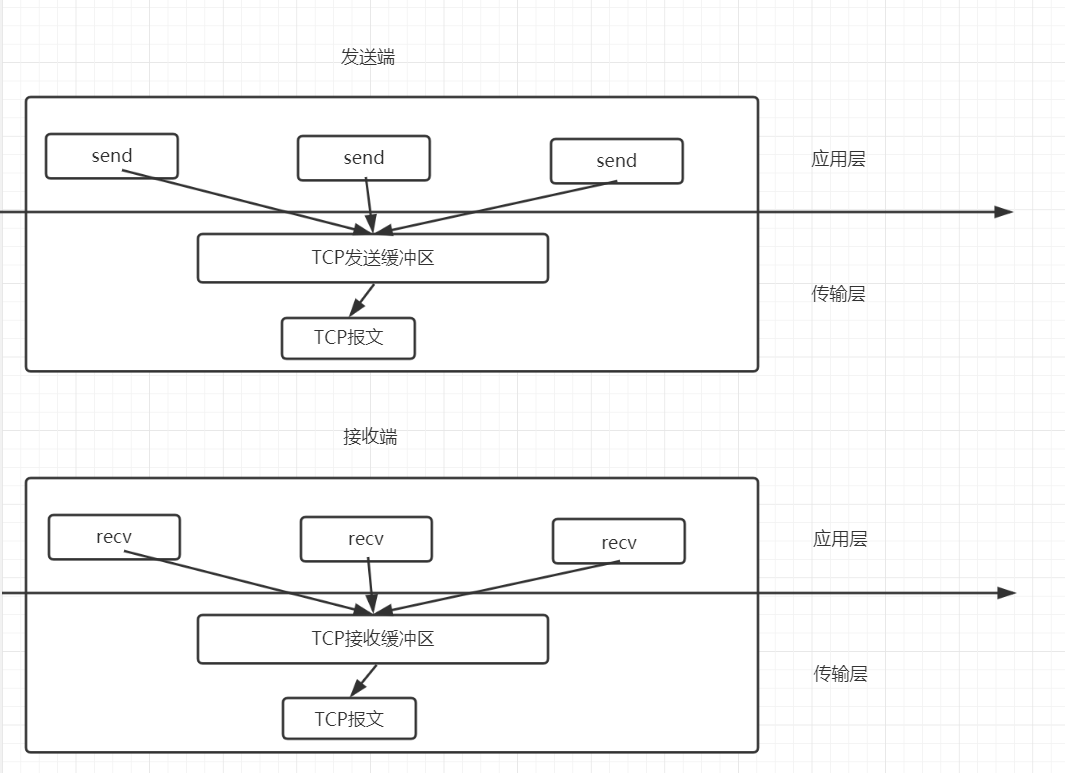
**TCP通信原理**

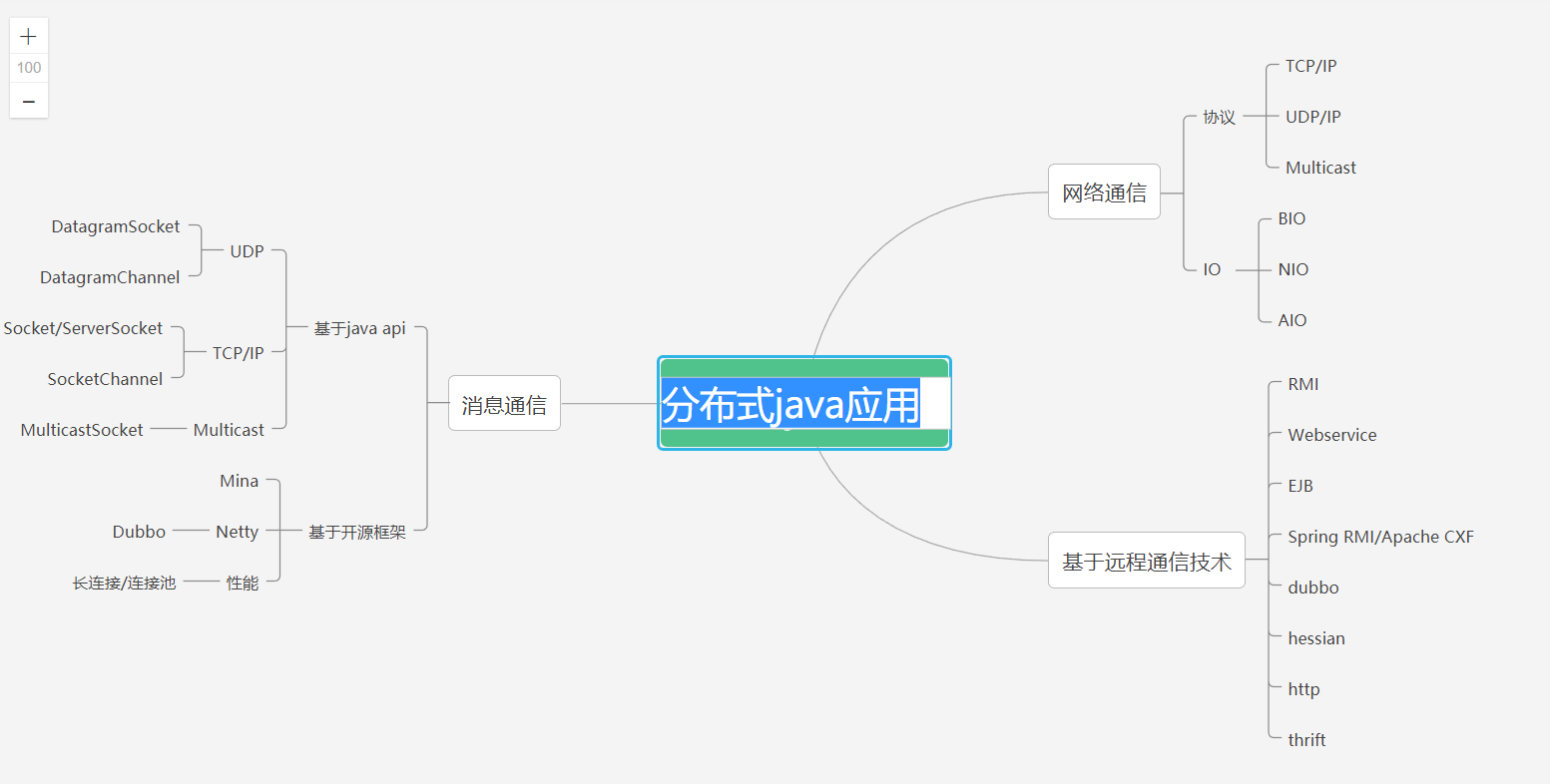
首先，对于TCP通信来说，每个TCP Socket的内核中都有一个发送缓冲区和一个接收缓冲区，TCP的全双工的工作模式及TCP的滑动窗口就是依赖于这两个独立的Buffer和该Buffer的填充状态。

接收缓冲区把数据缓存到内核，若应用进程一直没有调用Socket的read方法进行读取，那么该数据会一直被缓存在接收缓冲区内。不管进程是否读取Socket，对端发来的数据都会经过内核接收并缓存到Socket的内核接收缓冲区。

read索要做的工作，就是把内核接收缓冲区中的数据复制到应用层用户的Buffer里。

进程调用Socket的send发送数据的时候，一般情况下是讲数据从应用层用户的Buffer里复制到Socket的内核发送缓冲区，然后send就会在上层返回。换句话说，send返回时，数据不一定会被发送到对端。

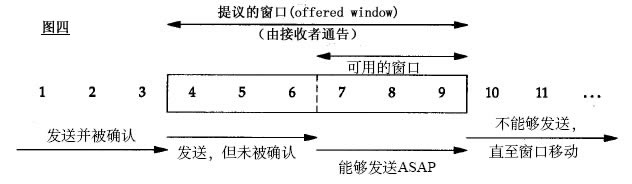




**什么是滑动窗口协议**

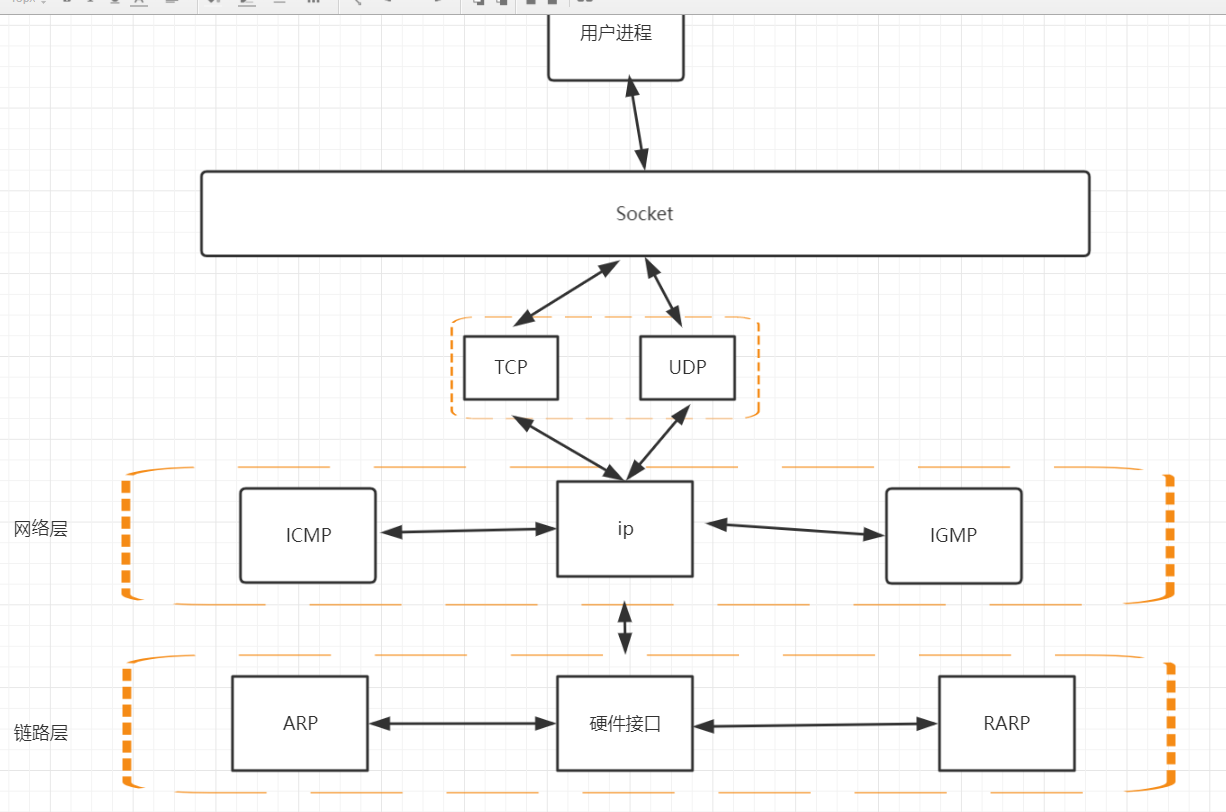
发送方和接收方都会维护一个数据帧的序列，这个序列被称作窗口。发送方的窗口大小由接收方确认，目的是控制发送速度，以免接收方的缓存不够大导致溢出，同时控制流量也可以避免网络拥塞。

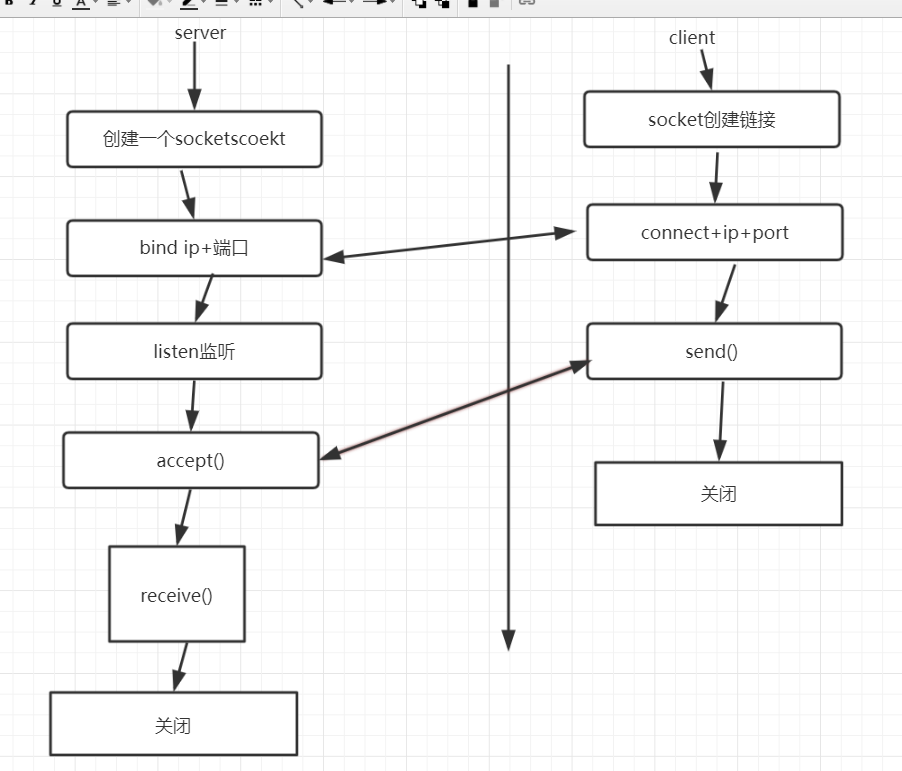
下面图中的4,5,6号数据帧已经被发送出去，但是未收到关联的ACK，7,8,9帧则是等待发送。可以看出发送端的窗口大小为6，这是由接受端告知的（事实上必须考虑拥塞窗口cwnd，这里暂且考虑cwnd>rwnd）。此时如果发送端收到4号ACK，则窗口的左边缘向右收缩，窗口的右边缘则向右扩展，此时窗口就向前“滑动了”，即数据帧10也可以被发送



明白了Socket读写数据的底层原理，我们就很容易理解“阻塞模式”：对于读取Socket数据的过程而言，如果接收缓冲区为空，则调用Socket的read方法的线程会阻塞，知道有数据进入接收缓冲区；而对于写数据到Socket中的线程来说，如果待发送的数据长度大于发送缓冲区空余长度，则会阻塞在write方法上，等待发送缓冲区的报文被发送到网络上，然后继续发送下一段数据，循环上述过程直到数据都被写入到发送缓冲区为止

从前面分析的过程来看，传统的Socket阻塞模式直接导致每个Socket都必须绑定一个线程来操作数据，参与通信的任意一方如果处理数据的速度较慢，会直接拖累到另一方，导致另一方的线程不得不浪费大量的时间在I/O等待上，所以这就是Socket阻塞模式的“缺陷”。但是这种模式在少量的TCP连接通信的情况下，双方都可以快速的传输数据，这个时候的性能是最高的。





# 第三节课

**java序列化机制Serialize接口**

**java本身的序列化机制存在的问题**

1. 序列化数据结果比较大、传输效率比较低
2. 不能跨语言对接

以至于在后来的很长一段时间，基于XML格式编码的对象序列化机制成为了主流，一方面解决了多语言兼容问题，另一方面比二进制的序列化方式更容易理解。以至于基于XML的SOAP协议及对应的WebService框架在很长一段时间内成为各个主流开发语言的必备的技术。

再到后来，基于JSON的简单文本格式编码的HTTP REST接口又基本上取代了复杂的Web Service接口，成为分布式架构中远程通信的首要选择。但是JSON序列化存储占用的空间大、性能低等问题，同时移动客户端应用需要更高效的传输数据来提升用户体验。在这种情况下与语言无关并且搞笑的二进制编码协议就成为了大家追求的热点技术之一。首先诞生的一个开源的二进制序列化框架-MessagePack。它比google的Protocol Buffers出现得还要早

恰当的序列化协议不仅可以提高系统的通用性、强壮型、安全性、优化性能。同时还能让系统更加易于调试和扩展

**序列化和反序列化的概念**

把对象转化为字节序列的过程称之为对象的序列化

反之，称之为反序列化

**怎么去实现一个序列化操作**

1. 实现Serializable接口
2. ObjectInputStream  : 表示读取指定的字节数据转换成对象
3. ObjectOutputStream ：

**科普小知识**

**serialVersionUID的作用**

文件流中的class和classpath中的class，也就是修改过后的class，不兼容了，处于安全机制考虑，程序抛出了错误，并且拒绝载入。从错误结果来看，如果没有为指定的class配置serialVersionUID，那么java编译器会自动给这个class进行一个摘要算法，类似于指纹算法，只要这个文件有任何改动，得到的UID就会截然不同的，可以保证在这么多类中，这个编号是唯一的。所以，由于没有显指定 serialVersionUID，编译器又为我们生成了一个UID，当然和前面保存在文件中的那个不会一样了，于是就出现了2个序列化版本号不一致的错误。因此，只要我们自己指定了serialVersionUID，就可以在序列化后，去添加一个字段，或者方法，而不会影响到后期的还原，还原后的对象照样可以使用，而且还多了方法或者属性可以用。

**静态变量的序列化**

序列化并不保存静态变量的状态

**Transient关键字**

transient关键字表示指定属性不参与序列化

**父子类问题**

如果父类没有实现序列化，而子类实现列序列化。那么父类中的成员没办法做序列化操作

**序列化的存储规则**

对同一个对象进行多次写入，打印出的第一次存储结果和第二次存储结果，只多了5个字节的引用关系。

并不会导致文件累加

**序列化实现深度克隆**

**浅拷贝（浅复制、浅克隆）**：被复制对象的所有变量都含有与原来的对象相同的值，而所有的对其他对象的引用仍然指向原来的对象。

　　换言之，浅拷贝仅仅复制所拷贝的对象，而不复制它所引用的对象。

**深拷贝（深复制、深克隆）**：被复制对象的所有变量都含有与原来的对象相同的值，除去那些引用其他对象的变量。

　　那些引用其他对象的变量将指向被复制过的新对象，而不再是原有的那些被引用的对象。

　　换言之，深拷贝把要复制的对象所引用的对象都复制了一遍

**总结**

1. 在java中，只要一个类实现了java.io.Serializable接口，那么它就可以被序列化
2. 通过ObjectOutputStream和ObjectInputStream对对象进行序列化合反序列化操作
3. 对象是否允许被反序列化，不仅仅是取决于对象的代码是否一致，同时还有一个重要的因素（UID）
4. 序列化不保存静态变量
5. 要想父类对象也参与序列化操作，那么必须要让父类也实现Serializable接口
6. Transient关键字，主要是控制变量是否能够被序列化。如果没有被序列化的成员变量反序列化后，会被设置成初始值，比如String -> null
7. 通过序列化操作实现深度克隆

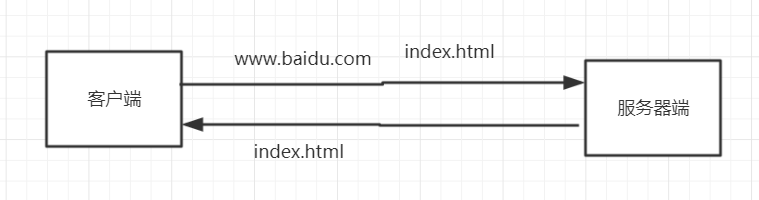
**主流的序列化技术有哪些**

JSON/Hessian(2) /xml/protobuf/kryo/MsgPack/FST/thrift/protostuff/Avro

# 第四节课

**HTTP协议的概述**

1. 客户端和服务器端



1. 资源

html/文本、word、avi电影、其他资源

1. 媒体类型

MIME类型。  text/html、 image/jpeg

1. URI和URL

URI:web服务器资源的名字。  index.html

  http://[www.gupaoedu.com](http://www.gupaoedu.com/):80/java/index.html[?query-string] #location

  schema: http/https/ftp.

  host: web服务器的ip地址或者域名

  port: 服务端端口， http默认访问的端口是80

  path: 资源访问路径

  query-string: 查询参数

1. 方法

GET/PUT/DELETE/POST/HEAD

**报文**

request参数、 response响应参数

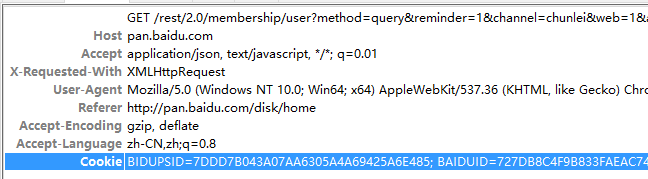
request消息结构包含三部分： （起始行、首部字段、主体）

METHOD /path / http/version-number

Header-Name:value

空行

主体 optional request body

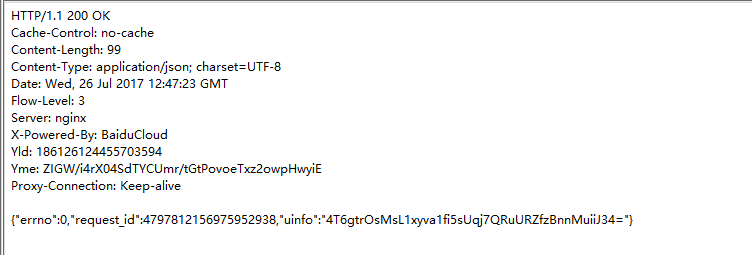


response

http/version-number   status code message

header-name:value

body



**状态码**

http/1.1版本的协议里面定义了五种类型的状态码

1XX    提示信息

2XX    成功

3XX    重定向

4XX    客户端错误

5XX    服务器端的错误

**缓存**

HTTP协议的特点

1. 无状态

cookie+session

1. 多次请求
2. 基于TCP协议

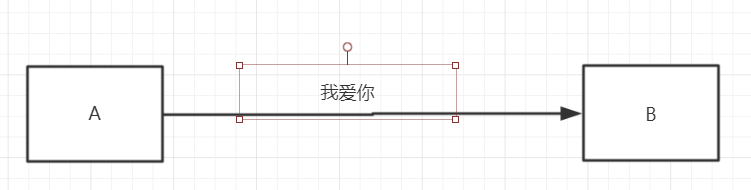
**HTTPS**

SSL/TLS

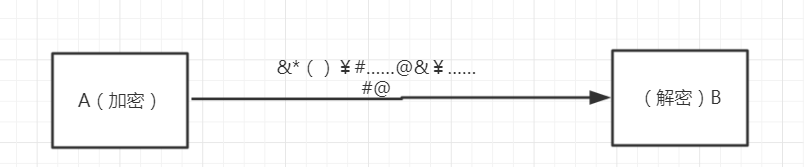
SSL3.0

ISOC  在SSL的基础上发布了升级版本 TLS1.2

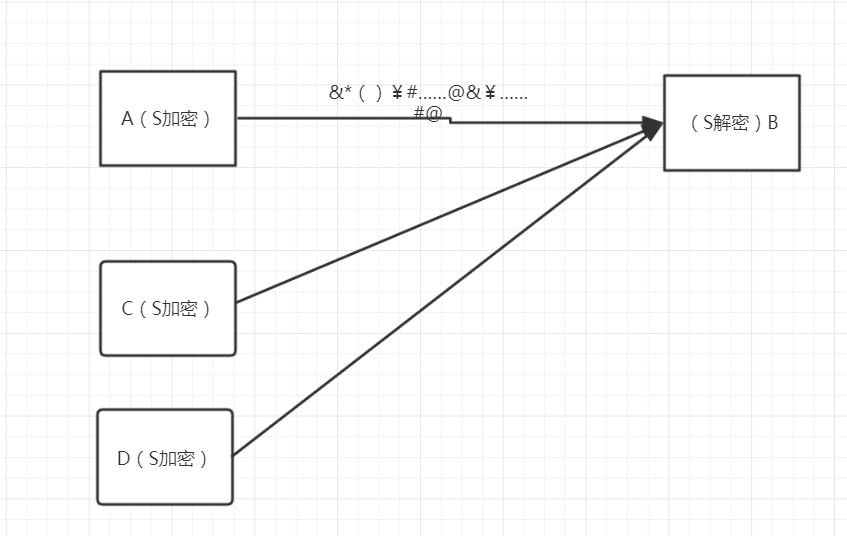
HTTPS的工作原理



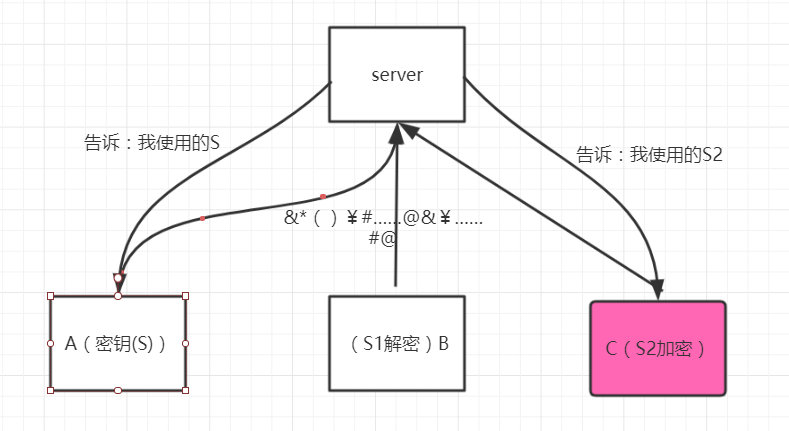
第一步， 使用对称加解密



第二步，密钥是公开的，所有的客户端都可以拿到



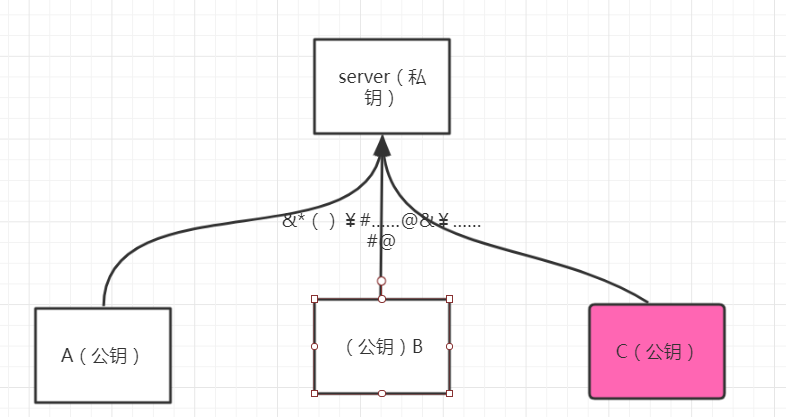
第三步 针对不同的客户端使用不同的密钥



问题：协商过程是没有加密的，所以还会出现被截断的问题

第四步：使用非对称加密

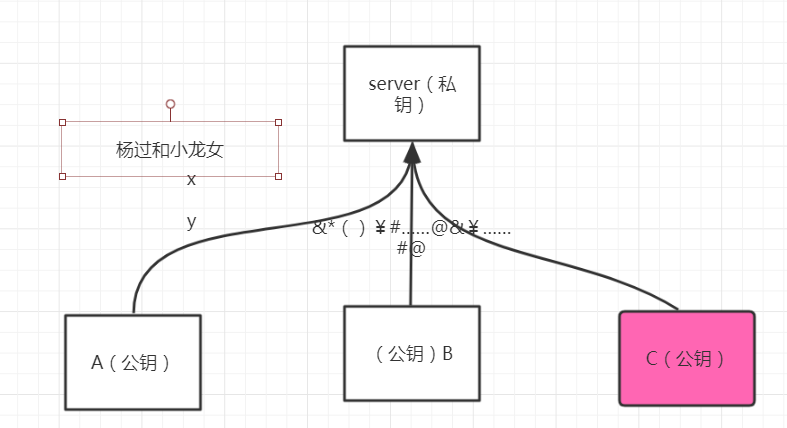
非对称：公钥和私钥的概念



问题： 客户端如何拿到公钥

1. 服务器端把公钥发送给每一个客户端
2. 服务器端把公钥放到远程服务器，客户端可以请求到
3. 让浏览器保存所有的公钥（不现实）

第五步 **公钥被调包的问题按照上面的方案，永远存在**。



第六步：使用第三方机构来解决

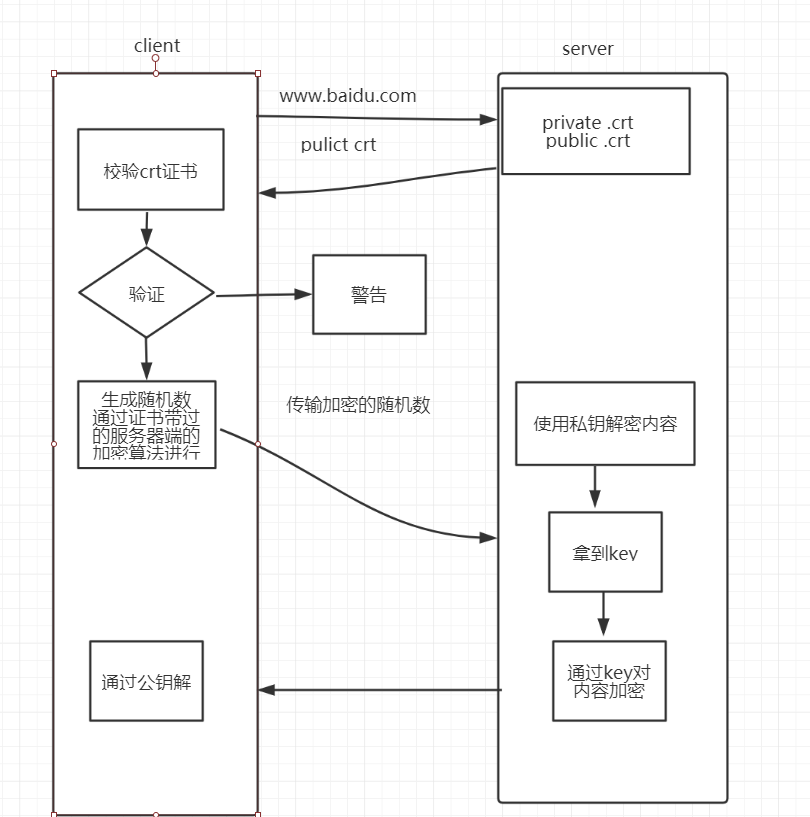
通过第三方机构，使用第三方机构的私钥对我们【需要传输的公钥】进行加密

第七部分

 数字证里面包含的内容：

 公司信息、网站信息、数字证书的算法、公钥

连接过程



**RESTful**

REST  表述性状态转移

使用WEB标准来做一些准则和约束。

RESTful的基本概念

1. 在REST中，一切的内容都被认为是一种**资源**
2. 每个资源都由URI唯一标识
3. 使用统一的接口处理资源请求（POST/GET/PUT/DELETE/HEAD）
4. 无状态

**资源和URI**

1. [/]表示资源的层级关系
2. ？过滤资源
3. 使用\_或者-让URI的可读性更好

统一接口

 GET  获取某个资源。 幂等

 POST 创建一个新的资源

 PUT 替换某个已有的资源（更新操作） ， 幂等

 DELETE 删除某个资源

 PATCH/HEAD

资源表述

MIME 类型（）

accept: text/xml   html文件

Content-Type告诉客户端资源的表述形式

资源链接

 超媒体即应用状态引擎

状态转移

 服务器端不应该保存客户端状态。

应用状态-  >服务器端不保存应用状态

访问订单   根据接口去查询

访问商品   查询

RESTful的最佳设计

1. 域名

[http://api.gupaoedu.com](http://api.gupaoedu.com/)

<http://api/gupaoedu.com/api>

1. 版本

<http://api.gupaoedu.com/v1/user/1>

header里面维护版本

1. 路径

<http://api.gupaoedu.com/v1/users>\_list  //获取用户列表

<http://api.gupaoedu.com/v1/goods-list>  //商品列表

<http://api.gupaoedu.com/v1/users/>{id}

1. 过滤信息

<https://api.github.com/user/repos?page=2&per_page=100>

[https://developer.github.com/v3/#rate-limiting](https://developer.github.com/v3/%20/l%20rate-limiting)

1. 状态码

业务状态码

http状态码

**第五节课**

**课程回顾**

1. 分布式架构的定义以及分布式架构的演进。
2. 分布式架构和集群的区别
3. TCP/UDP、全双工、半双工、3次握手协议、4次挥手协议
4. FIN标识的报文给到server端
5. server端接收到FIN报文以后，表示Client端没有数据要发给Server端了
6. server端发送ACK给到Client端，表示Server端的数据已经发完了。准备关闭链接
7. client端收到ACK报文以后，知道可以关闭连接了，发送ACK请求到Server端，自己进入TIME-WAIT
8. Server端接收到ACK以后，表示可以断开连接了
9. Client端等待一定时间后，没有收到回复，表示Client可以关闭连接
10. TCP的非阻塞IO
11. 序列化
12. SerialVersionUID
13. 静态变量序列化问题、Transient关键字、父子类的序列化问题
14. kryo、FST、JSON、XML、protobuf、Hessian、Avro、Thrift
15. http和https协议、RESTful规范
16. 客户端发起一个https请求
    1. 客户端支持的加密方式
    2. 客户端生成的随机数（第一个随机数）
17. 服务端收到请求后，拿到随机数，返回
    1. 证书（颁发机构（CA）、证书内容本身的数字签名（使用第三方机构的私钥加密）、证书持有者的公钥、证书签名用到的hash算法）
    2. 生成一个随机数，返回给客户端（第二个随机数）
18. 客户端拿到证书以后做验证
    1. 根据颁发机构找到本地的跟证书
    2. 根据CA得到根证书的公钥，通过公钥对数字签名解密，得到证书的内容摘要 A
    3. 用证书提供的算法对证书内容进行摘要，得到摘要 B
    4. 通过A和B的对比，也就是验证数字签名
19. 验证通过以后，生成一个随机数（第三个随机数），通过证书内的公钥对这个随机数加密，发送给服务器端
20. （随机数1+2+3）通过对称加密得到一个密钥。（会话密钥）
21. 通过会话密钥对内容进行对称加密传输

**分布式通信框架-RMI讲解**

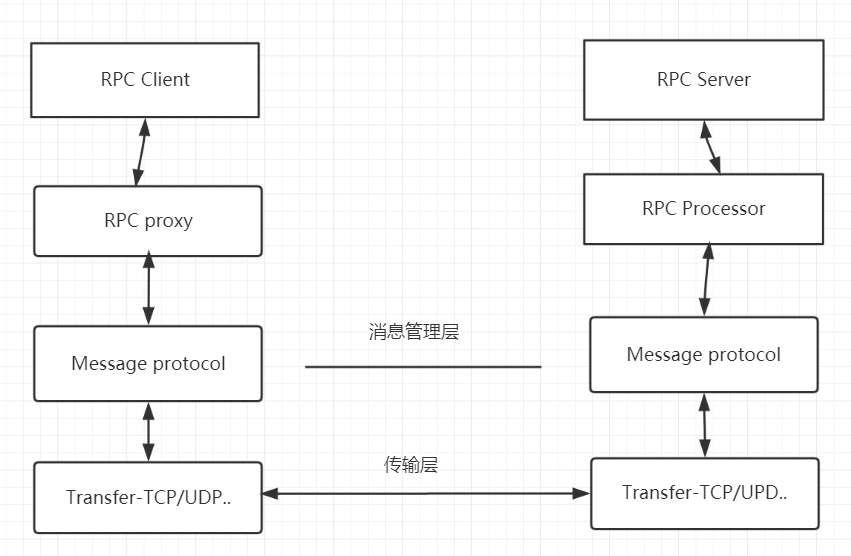
**什么是RPC**

Remote procedure call protocal

RPC协议其实是一个规范。Dubbo、Thrif、RMI、Webservice、Hessain

网络协议和网络IO对于调用端和服务端来说是透明；

**一个RPC框架包含的要素**



**RMI的概述**

RMI(remote method invocation)  , 可以认为是RPC的java版本

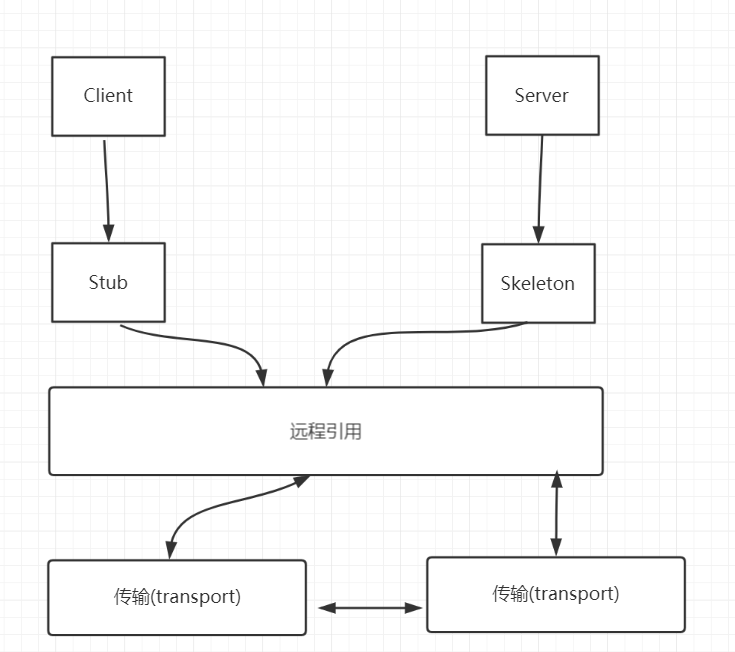
RMI使用的是JRMP（Java Remote Messageing Protocol）, JRMP是专门为java定制的通信协议，所以踏实纯java的分布式解决方案

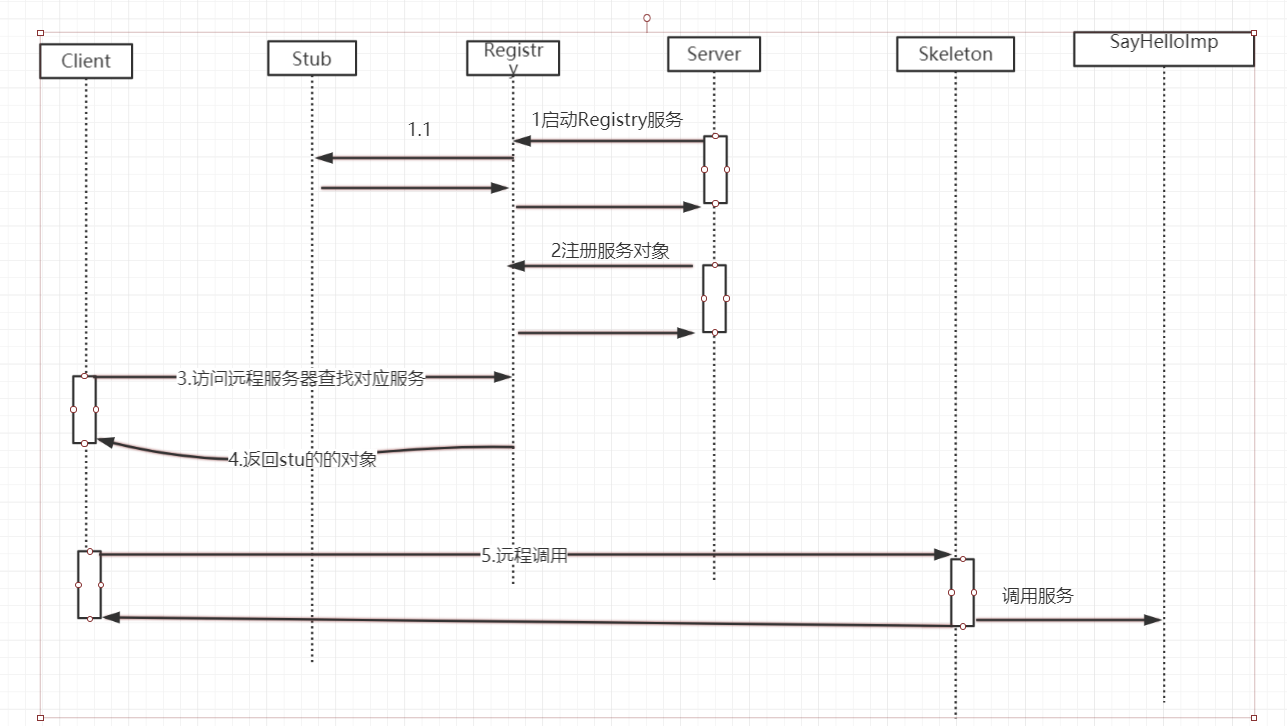
**如何实现一个RMI程序**

1. 创建远程接口， 并且继承java.rmi.Remote接口
2. 实现远程接口，并且继承：UnicastRemoteObject
3. 创建服务器程序： createRegistry方法注册远程对象
4. 创建客户端程序

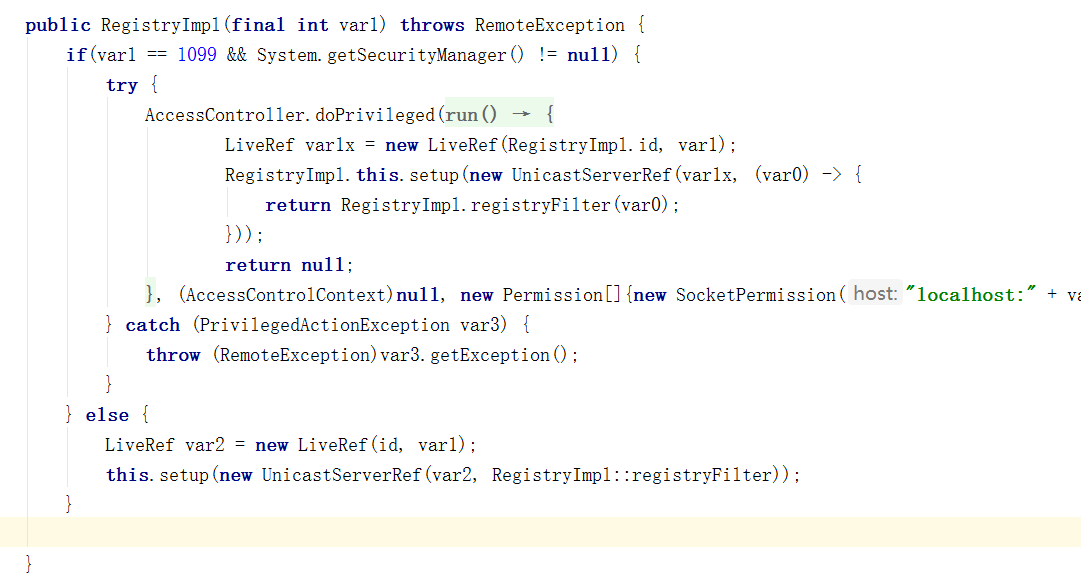
**如果自己要去实现一个RMI**

1. 编写服务器程序，暴露一个监听， 可以使用socket
2. 编写客户端程序，通过ip和端口连接到指定的服务器，并且将数据做封装（序列化）
3. 服务器端收到请求，先反序列化。再进行业务逻辑处理。把返回结果序列化返回





**源码分析**



BIO 、 NIO

**第六节课**

记得录屏

RPC 包含的要素

webservice

        协议层：tcp/ip

应用层： http协议

SOAP： http+xml

**分布式通信框架-webservice分析**

**什么是webservice**

webservice也可以叫xml web service webservice, 轻量级的独立的通讯技术

1. 基于web的服务：服务端提供的服务接口让客户端访问
2. 跨平台、跨语言的整合方案

**为什么要使用webservice**

跨语言调用的解决方案

**什么时候要去使用webservice**

电商平台，订单的物流状态。

 .net实现的webservice服务接口

**webservice中的一些概念**

**WSDL(web service definition language  webservice 定义语言)**

webservice服务需要通过wsdl文件来说明自己有什么服务可以对外调用。并且有哪些方法、方法里面有哪些参数

wsdl基于XML（可扩展标记语言）去定义的

1. 对应一个.wsdl的文件类型
2. 定义了webservice的服务器端和客户端应用进行交互的传递数据和响应数据格式和方式
3. 一个webservice对应唯一一个wsdl文档

**SOAP（simple object access protocal简单对象访问协议）**

http+xml

webservice通过http协议发送和接收请求时， 发送的内容（请求报文）和接收的内容（响应报文）都是采用xml格式进行封装

这些特定的HTTP消息头和XML内容格式就是SOAP协议

1. 一种简单、基于HTTP和XML的协议
2. soap消息：请求和响应消息
3. http+xml报文

**SEI（webservice endpoint interface webservice的终端接口）**

webservice服务端用来处理请求的接口，也就是发布出去的接口。

**开发一个webservice的实例**

》》》》》》》》详见代码》》》》》》》

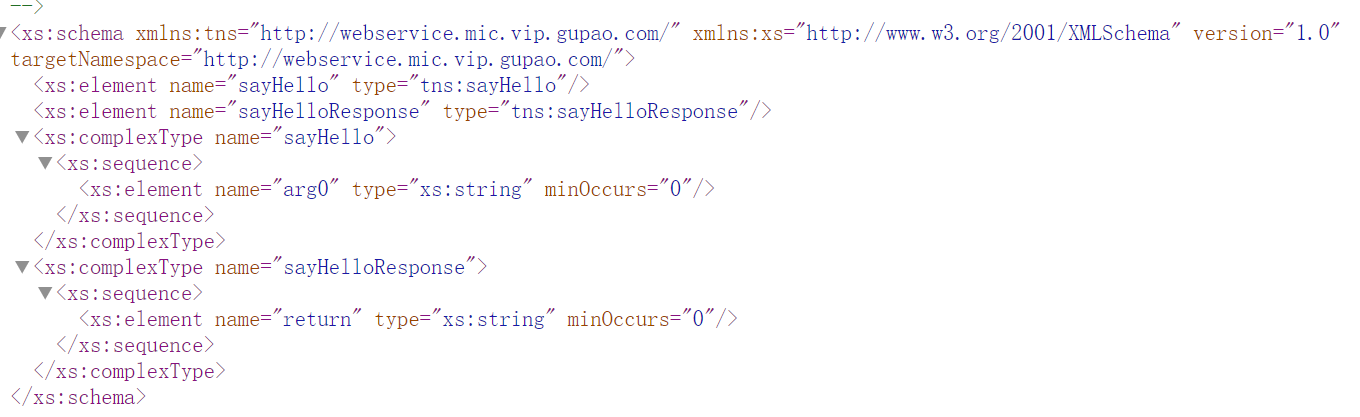
**分析WSDL文档**



**Types标签**

定义整服务端的数据报文

**Schema标签**



<sayHello>

   <arg0>String</arg0>

</sayHello>

<soapenv:Envelope xmlns:soapenv="<http://schemas.xmlsoap.org/soap/envelope/>" xmlns:web="<http://webservice.mic.vip.gupao.com/>">

   <soapenv:Header/>

   <soapenv:Body>

      <web:sayHello>

         <!--Optional:-->

         <arg0>?</arg0>

      </web:sayHello>

   </soapenv:Body>

</soapenv:Envelope>

<sayHelloResponse>

   <return>string</return>

</sayHelloResponse>

<S:Envelope xmlns:S="<http://schemas.xmlsoap.org/soap/envelope/>">

   <S:Body>

      <ns2:sayHelloResponse xmlns:ns2="<http://webservice.mic.vip.gupao.com/>">

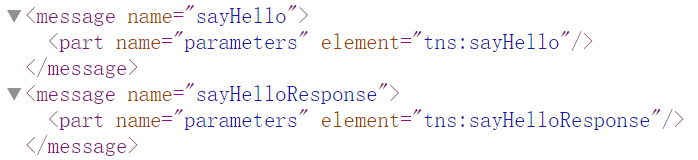
         <return>Hello ,Mic,I'am 菲菲</return>

      </ns2:sayHelloResponse>

   </S:Body>

</S:Envelope>

**Message**



定义了在通信中使用的消息的数据结构

**PortType**



定义服务器端的SEI

input/output表示输入/输出数据

**Binding标签**

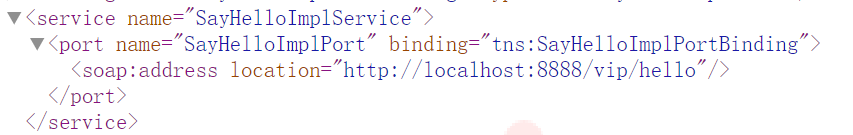


1. type属性： 引用porttype

<soap:binding style=”document”>

1. operation : 指定实现方法
2. input/output 表示输入和输出的数据类型

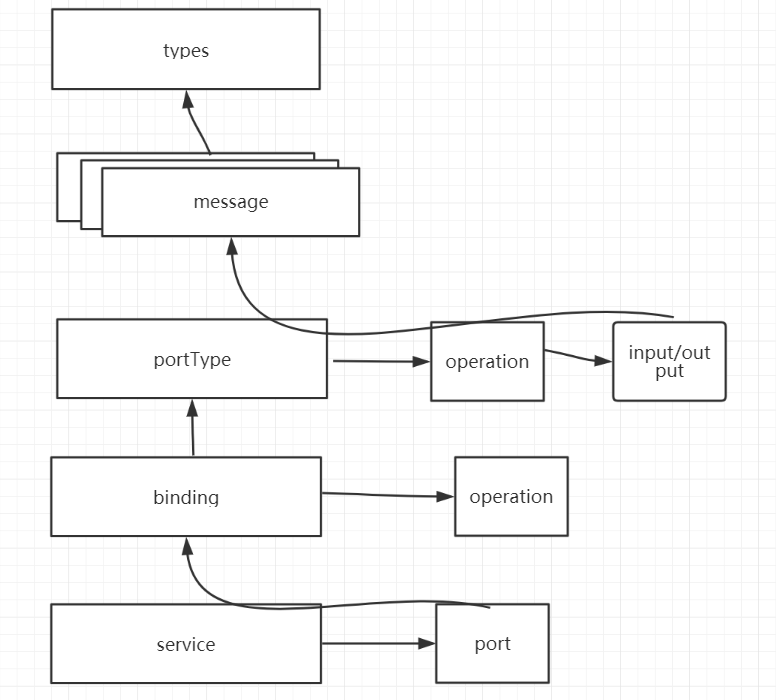
**Service标签**

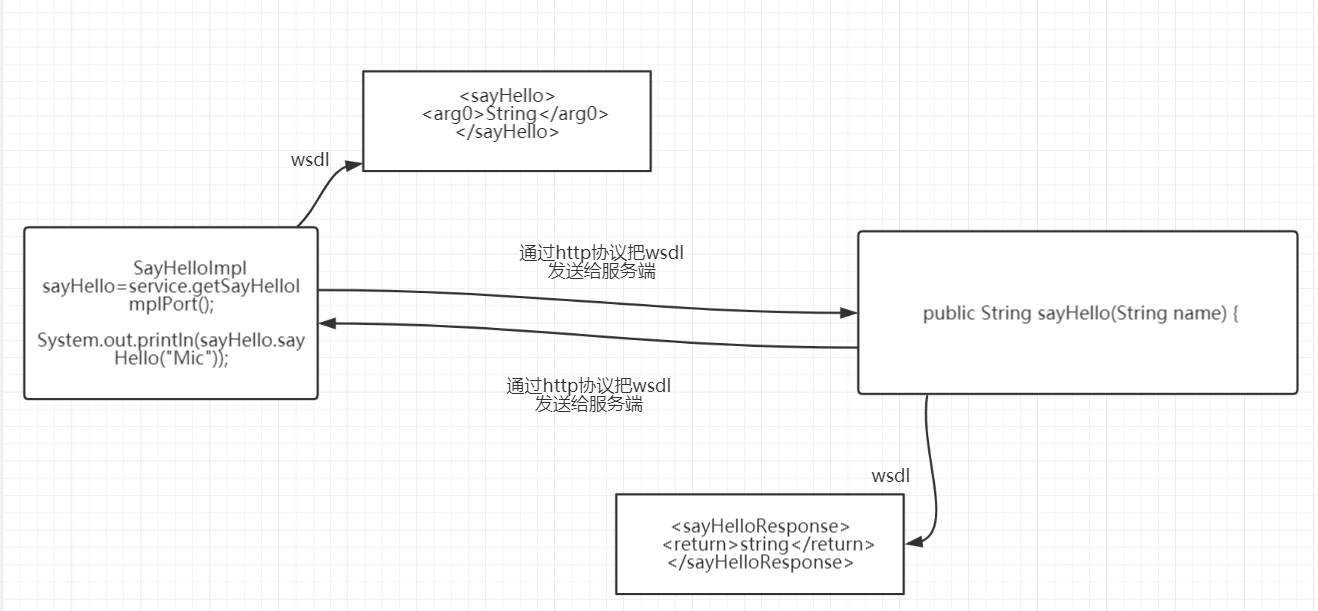


service： 服务器端的一个webservice的容器

name属性： 指定客户端的容器类

address： 当前webservice的请求地址





**Axis/Axis2**

apache开源的webservice工具

**CXF**

Celtix+Xfire 。 用的很广泛，因为集成到了spring

**Xfire**

高性能的Webservice

HTTP+JSON (新的webservice)

HTTP+XML

**spring cxf+REST实现一个webservice服务**

springmvc+REST实现的新webservice

linux： centos7;  vm可以copy 设置一个备份点

jdk、tomcat

RMI、 http协议/https、webservice、 TCP协议、UDP协议、

socket编程、bio /nio模型、分布式架构、集群、架构演进过程