# 分布式系统的特征

分布式系统：组件分布在联网的计算机上，组件之间通过消息来进行通信与协调：就是很多个体组成一个整体，协调工作；任何一个系统都是分布式系统，只不过组件之间的通讯方式不同；组件的并发性（独立）、缺乏全局时钟（独立）、组件故障的独立性（单点依赖）。

分布式系统发展的关键趋势：现代网络的泛在（无所不在）特性、移动与无处不在的计算、分布式多媒体系统。

分布式系统的挑战：组件的异构性、开放性、安全性、可伸缩性、故障处理、组件的并发性、透明性与提供服务的质量的问题。

## 简介

书主要是解释影响系统设计与实现的联网的计算机的特征，给出可帮助完成设计与实现的主要概念与技术（解决方法：概念与技术）

分布式系统的特征：并发（共享资源）、缺乏全局时钟（消息协作的时序问题）、故障的独立性（隔离感知）。构建与使用分布式系统的主要动力是对共享资源的期望 （变化的与不变的）。

## 分布式系统的例子

所有基于网络的服务都是分布式系统。

* 金融与商业：电商与支付；
* 信息社会：互联网，web搜索与wiki等知识网站；
* 创意产业与娱乐：游戏、视频；
* 医疗与保健：医疗信息化与远程医疗；
* 教育：电子教育；
* 交通与物流：地图等；
* 科学；
* 环境管理：结合传感器技术。

### Web搜索

对万维网的内容建立索引，google使用分布式系统完成这个任务。

### 大型多人在线游戏

快速响应时间与事件通知机制，对共享世界的一致的视图；c/s架构。

### 金融交易

## 分布式系统的趋势

### 泛在联网和现代互联网

现代互联网是各种网络的集合，也是一个大型的分布式系统，设计好的互联网通信协议连接了各个网络，这些网络在生活中无处不在

### 1.3.2无处不在的计算

设备小型化，每个设备都是一个分布式的资源，连接到分布式系统；移动计算与物联网（无处不在的计算）。

### 分布式多媒体系统

### 把分布式计算作为一个公共设施

PaaS/IaaS/云计算等；

## 关注共享资源

共享资源包括硬件、服务、数据等；共享的方式包含2种：一种是各个使用方不相关，一种是使用方相关的（协作）。服务用于管理共享资源与提供访问；服务定义了资源上的操作；服务是通过管理资源的程序提供的，这个程序也叫做服务器；服务器接受客户端请求，这个方案是客户-服务器计算；当客户发送请求时，就称为调用服务气上的操作，一次完整的请求与响应的过程称为远程调用。

## 挑战

* 异构性：网络、计算机硬件、操作系统、编程语言；中间件是指一个软件层，提供了编程抽象，屏蔽了底层网络、硬件、操作系统与编程语言的异构性；中间件为服务器与分布式应用提供了一致的计算模型；移动代码：applet或者javascript；
* 开放性：系统可以被扩展，也可以被重新实现；开放的分布式系统允许增加硬件或者软件比如（分布式协议与规约等）而进行扩展；
* 安全性：机密性（防止泄漏给未授权的个人）、完整性（防止被改变或破坏）、可用性（访问资源的手段进行干扰），拒绝服务攻击（大量无意义的请求）；
* 可伸缩性：资源数量能满足不断增长的用户数量就是可伸缩的，需要控制物理资源的开销，控制性能损失，防止软件资源用尽，避免性能瓶颈；
* 故障处理：分布式系统的故障是独立的，涉及的难点有检测故障、掩盖故障（重传、使用副本）、容错、故障恢复、冗余（实现容错比如副本机制，多种选择）；
* 并发性：分布式系统的共享资源服务要并法的处理请求，需要进行并法控制；
* 透明性：透明性指对用户或者应用程序员屏蔽分布式系统的组件的分离性，使系统被认为是一个整体；而不是独立组件的集合，透明性包含访问透明性、位置透明性、并发透明性、复制透明性、故障透明性、移动透明性、性能透明性、伸缩透明性；
* 服务质量：可靠性、安全性与性能、适应性；

## 实例研究：万维网

Web是扩展与开放的，基于下面3个主要的技术组件：

* HTML 超文本描述语言，指定显示形式；
* 统一资源定位器URL；
* 客户-服务器系统体系结构；

HTML制定内容的布局方式与组织方式；URL定位资源，具有统一的格式；HTTP协议定义了浏览器与服务器的交互方式。

## 练习

* 1. 列出能被共享的五种类型的硬件资源和五种类型的数据与软件资源，并举出他们在实际的分布式系统中发生共享的例子？

答：打印机、磁盘、传感器、文件、web搜索、支付、数据库、服务;共享的例子忽略。

* 1. 不参考外部时间源的情况下，通过本地网络连接的2台计算机的时钟如何同步？什么因素限制了你描述过程的准确性，有互联网大量连接的计算机的时钟是如何同步的？讨论该过程的准确性？

答：通过消息同步的，不确定消息的格式；外部基准时间源同步。

* 1. 考虑1.2.2节讨论的大型多人在线游戏的实现策略，采用单一服务器方法表示多个游戏玩家的游戏状态的好处是什么？这存在什么问题以及如何解决？

答：能够获得一致性的地图；单机负载过高，必须使用集群结构做分解。这种结构需要设计集群之间的通信机制。

* 1. 一个用户随身携带可以无线联网的PDA，来到一个从没有到过的火车站，请给出建议，在用户不输入火车站的名称与属性的前提下，如何得到关于本地服务和火车站环境的情况？要解决哪些技术问题？

答：

# 系统模型

分布式系统3种模型：

* 物理模型：硬件设备与连通方式；
* 体系结构模型：基于角色来思考分布式系统（客户-服务器或者对等模型）；
* 基础模型：抽象的观点描述分布式系统的问题。

## 2.1 简介

## 2.2 物理模型

物理模型是从计算机和所用网络技术的特定细节中抽象出来的分布式系统地层硬件元素的表示。

基线物理模型：计算机设别结点

* + 早期的分布式系统：局域网；
  + 互联网规模的分布式系统：网络的网络，异构性增加，中间件出现；
  + 当代的分布式网络：移动互联网与物联网与云计算。

系统的分布式系统也就是系统的系统。

## 2.3 体系结构模型

系统的体系结构指的是系统的组件以及组件之间的关系；目标是确保结构能满足将来的需求，主要关心的是系统的可靠性、可管理性、适应性、与性价比。

### 2.3.1 体系结构的元素

* 通信实体：系统角度是进程，也可以是计算物理结点，也可以是线程；从编程角度看是对象（分布式对象是面向对象的思维）、组件（面向问题的实体）、web服务（基于互联网协议的行为封装）；
* 通信泛型：进程间通信，就是简单的2台机器的基于互联网协议的进程间通信；远程调用，双向交换的技术，基于的是请求-应答的通信模式，是客户-服务器计算的结构的主要通信模式，分为远程过程调用与远程方法调用，远程过程调用（RPC），是从组件的角度来说的，从对象的角度来书就是远程方法调用（RMI），分布式对象之间的方法调用。上述的通信模型都是同步的，双方必须同时存在，通过第三方实体，允许发送者与接受者解耦合，达到空间解耦合（不知道发给谁）与时间解耦合（不需要同时存在），这个也叫做间接通信，间接通信的关键技术是：组通信，一对多通信，通过组标识符发送给组，组管理组的成员。发布-订阅系统（分布式基于事件的系统）一对多。消息队列，点对点服务。元组空间，结构化数据放入持久空间，消费者处理特定模式的元组。分布式共享内存（DSM）。
* 角色与责任：基于角色，分布式系统体系结构风格有客户-服务器风格与对等风格；客户-服务器是最重要也是最简单有效的体系结构；对等结构是所有进程都是相同的，扮演同样的角色，不区分客户或者服务器；相比于客户-服务器（访问的资源与服务受限于服务器的能力），对等体系解决了这个问题，每个拥有资源与服务的机器都参与计算；对等应用的形式有分解（每个进程有一部分数据）与冗余（增加副本，分散负载，容灾配置）2种；服务与数据如何放置到真实的物理结点涉及到分布式系统的性能、可靠性与安全性；放置策略：1）将服务映射到多个服务器，可以集中式（一台服务器），也可以是分布式的（服务分解），集群（副本）2）缓存，用于存储最近使用的对象；3)移动代码；4）移动代理。

### 2.3.2 体系结构模式

1.分层体系结构

分层与抽象紧密相关，上层依赖下层，每层都是一个软件抽象，更高的层对细节透明。分布式系统的分层相当于把服务垂直组织成服务层；分层体系结构的常规视图：



中间件是为了屏蔽异构性，给应用程序员提供方便的编程模型，中间件也是计算机上的进程或者对象，可以是软件的基础构造块，用来抽象底层操作，比如远程方法调用、通信、事件通知等。

1. 层次化的体系结构

层次化与分层其实差不多，不过这个层次化指的是对某个分层进行层次化，就是再次分层，比如说应用层的MVC模型。

1. 瘦客户

将复杂性从最终用户设备移向互联网服务；

1. 代理

主要用于支持远程过程调用与远程方法调用透明性。

1. 业务代理（注册中心）
2. 反射。

### 2.3.3 相关的中间件解决方案

中间间用来提供一个抽象层，主要是为了屏蔽底层基础设施的异构性；提升互操作性与移植性；中间件的类别有：

|  |  |  |
| --- | --- | --- |
| 主类 | 子类 | 系统例子 |
| 分布式对象 | 标准 | RM-ODP |
| 平台 | CPRBA,RMI |
| 分布式组件 | 轻量级组件 | Fractal、OpenCOM |
| 应用服务器 | SUN EJB、JBoss |
| 发布-订阅系统 |  | JMS、Scribe,CORBA事件服务 |
| 消息队列 |  | WebsphereMQ、JMS |
| Web服务 | Web服务 | Apache Axis |
| 网格服务 | Globus Toolkit |
| 对等 | 路有覆盖网 | Pastry、Tapestry |
| 应用年特定的 | Squirrel、Ivy、Gnutella |

中间件在分布式系统中完成通信与数据共享，但是服务的可靠性还是需要应用层面作处理；实际就是中间件与上层服务的功能划分，比较困难，因为服务的正确性是依赖于上层服务的应用逻辑。

## 2.4 基础模型（这里没有理解）

分布式系统的设计的要求：进程与网络的高性能、高可靠性以及系统中资源的安全。

### 2.4.1 交互模型

分布式系统的进程要交互，相比于单个进程，分布式进程多了消息传递，不能很好的控制每个机器上的操作的执行时序，单机可以控制自己的进程的执行时序；影响进程交互的2个因素：

* 通信性能是一个限制特性；
* 不可能维护一个全局时间概念；

通信会有延迟（数据发送返回、网络堵塞、进程处理）、带宽（可以使用的共享带宽）、抖动（消息传递时间不稳定）。

通信涉及到的本地时钟因为时钟漂移的原因，可能影响时序。

同步分布式系统：进程执行时间有限制、消息传递时间有限制、本地时钟的偏移在给定的范围内；

异步分布式系统：对上述的延迟时间没有限制，时钟偏移也没有限制；典型的就是互联网。

分布式系统的分布式事件排序，比如2个分布式进程的事件发生顺序。

### 2.4.2 故障模型

* + 遗漏故障： 进程与通信信道故障；
  + 随机故障：随时可能出现的错误；
  + 时序故障：里面的所有的处理时间等故障；
  + 故障屏蔽：

### 2.4.3 安全模型

保证进程、通信信道、进程里面的对象数据的安全就是分布式系统的安全；

# 网络与网际互连

## 3.1 简介

分布式系统的网络需要：

* 传输介质；
* 硬件设备；
* 软件组件。

未分布式系统提供通信功能的所有组件叫做通信子系统，使用通信子系统通信的叫做主机，主机+交换机=结点。

分布式系统的联网问题：

* + 性能；
  + 可伸缩性；
  + 可靠性；
  + 安全性；
  + 移动性；
  + 服务质量；
  + 组播。

## 3.2 网络类型

* 个域网：
* 局域网；
* 广域网；
* 城域网；
* 无线局域网；
* 无线城域网；
* 无线广域网；
* 互联网络；
* 网络错误。

## 3.3 网络原理

包交换技术，不同于电路交换技术。

1. 数据包的传输：信息分割为小的数据包，是为了满足网络中计算机的缓冲空间要求，避免数据太大需要等待通信信道的时间过长；
2. 数据流协议，因为数据流的特点，对带宽与延迟有较高的要求，需要预先分配足够的资源，RSVP协议与RTP协议；
3. 交换模式：广播（以太网，蜂窝网络）、电路交换（传统的电话交换网络）、包交换（存储转发网络）、帧中继（针对包交换需要存储然后转发造成延迟高的特点，包更小（帧），不存储帧，只根据头转发，形成了一种流）式的结构；
4. 协议：进程间通信需要用到的规则与格式，协议包含2个部分：消息的顺序与消息的格式，协议层与协议栈，因为网络是分层处理，每层有很多的协议，OSI 7层协议栈：

* 应用层：满足特定应用的通信需求定义的，比如HTTP，  
  FTP，STMP；
* 表示层：数据转换，比如加密（TLS）；
* 会话层：可靠性与适应性，故障检测与自动恢复（SIP）；
* 传输层：处理消息的最低一层与端口相关联，比如TCP、UDP；
* 网络层：在网络路径中传输数据包比如IP、ATM虚电路；
* 数据链路层：有直接物理连接的2个结点间传输数据包，比如：MAC、PPP、ATM;
* 物理层：驱动网络的硬件与电路或者光纤或者无线网络电磁信号。

1. 数据包组装，因为网络中的存储要对包的大小的要求，这个就是MTU(Maximum Transfer Unit),端口是计算机中进程的目的点，0-1023 是公共端口，其他是个人端口；
2. 数据包传递：2种方法传递数据包：

* 数据报报包传递：发送后不管；
* 虚电路包传递：建立虚电路路由路径，包中保存虚电路号，在中间结点通过虚电路号转发。

1. 路由：自适应路由，路由信息会根据网络的情况定期的变更；路由算法包含2个部分：决定路由路径与根据网络情况或者故障动态变更信息；
2. 距离向量算法或者叫做Bellman-Ford算法，Router Infomation Protocol 路由信息协议，相邻的路由器互相交换路由信息，新增或者更新路由表信息，故障路由也会被检测到，最终替换故障路由的路径；
3. 网络的能力受通信链路性能与交换结点性能的限制，当链路与结点的负载接近极限时，发送的主机就会建立队列，当队里持续增长，超过缓冲区，就会发生数据包丢失，拥塞控制，通知路由上的结点，缓存数据包或者端到端的流量控制，通过额外的拥塞信息，降低发送的速率；
4. 路由器、网桥、集线器、交换机、隧道；

## 3.4 互联网协议

* 1. IP寻址：互联网中的每个主机都有一个IP地址，分为2个部分，子网编号与主机编号；分为A，B，C，D（组播），E（保留） 几种类别，具体可以参考网上的一些说明，3类可以使用的地址分别用于不同的用途，主机标示部分全为0代表本机，全为1代表广播消息，广播给子网内的全部主机；地址不够的3种方案：1.IPv6，2.无等级域间路由，3.NAT；
  2. IP协议：IP数据报包含头部与数据2部分，并且提供了头部的校验和，IP报文没有传输的保证，可能丢失，重复，错乱等，组装好后，放入数据链路层，数据链路层需要根据IP得到下一跳路由器的MAC地址（底层网络使用的地址），地址解析协议（ARP），IP伪冒，就是填入虚假的发送方地址；
  3. IP路由：主干（自治系统的概念），路由协议，链路-状态算法，解决路由器路由表过大的2种方案：1.IP地址人为设定地域，2.默认路由；网关路由器有个互联网地址，每个地址都对应一个网络；
  4. CIDR，分配连续的C类地址或者分割B类地址，增加掩码域，用来任意的分割IP地址中的子网地址与主机地址，相比分类地址更为灵活；
  5. NAT，不需要注册，直接访问网络，NAT算法的寻址过程：1.路由器接到内部计算机发送的TCP或者UDP，根据IP与端口号生成一个翻译表项，2.路由器使用一个虚拟端口号，指向翻译表项，并用它与路由器的IP地址替换包中的源端口与IP地址，3.当接到返回的包时，根据目的端口号查找翻译表项，然后替换包中的IP地址与端口号，然后转发到内部网；
  6. IPv6：IP地址使用128比特表示。
  7. 移动IP：移动计算机访问互联网，如果只是客户端，不需要保留固定的IP地址，移动时接入不同的子网就行，子网使用DHCP(动态主机配置协议)分配IP地址，但是如果是打电话这种，必须保持，手机的IP地址是固定的，才能接收到别人打过来的电话，这时需要使用移动IP的解决方案，在所有的子网中，有固定的IP地址，剩下的没看懂；
  8. TCP与UDP：端口号，16位整数，UDP不可靠传输，但是开销最小，TCP需要额外的信息保证是可靠传输，可靠传输的机制有：1.排序，TCP报文分割后有序号，2.流控制，比如确认机制、窗口控制、重传机制、缓冲与校验和。
  9. 域名：域名解析。什么是域？什么是域名。
  10. 防火墙：一组进程，用于控制外网与内部网的通信，用于服务控制、行为控制、用户控制。

## 3.5 实例研究：以太网、WiFi、蓝牙

### 3.5.1 以太网

802.3标准，类似一个有分支的总线连接，通过集线器或者中继器连接为一个以太网，通过交换机或者网桥连接多个以太网，以太网使用“具有冲突检测的载波监听多路访问”的机制传输数据，也是一种竞争总线的策略。

### 3.5.2 IEEE 802.11 无线LAN

也叫作WIFI，扩展了802.3的载波监听与多路复用协议；通过基站接入有线网络叫做基于基础设置的无线网络，还有没有基站的叫做自组织网络，802.11标准使用无线电传输信号，有多个版本，每个版本工作在不同的频率上，可以互相独立组成无线网，并且也是用MAC（媒体访问控制协议），由于无线点信号传输的特殊性，可能会造成子网内：1.工作站隐蔽（2个主机离的太远，检测不到对方），2.信号衰减，3.冲突屏蔽（因为本地的信号用于比远程传过来的好，这样一对比，主机就看不到冲突了）；CSMA/CD加入时隙保留机制变成CSMA/CA协议；主要的机制就是加锁-完成-通知的一种逻辑，这整个的过程中，时隙代表约定的加锁的时间；就是发送方要发送信息时，广播一个RTS帧，代表请求时隙，告诉别的主机，信道我要用，你们都别用，然后别的主机回复一个CTS帧，代表，你加锁成功，然后传输完成，基站通知所有的主机信道空闲了。

### 3.5.3 IEEE 802.15.1 蓝牙无线PAN

无线个域网，名字取自与一个海盗的名字，蓝牙比无线网带宽少，能耗小，用于低开销传输设备。

# 进程间通信

## 4.1 简介



UDP应用程序接口提供了消息传递抽象；TCP应用程序接口提供了进程对之间的双向流抽象。

## 4.2 互联网协议的API

### 4.2.1 进程间通信的特征

每个消息目的地都维持了一个队列，发送者就是生产者，接收者就是消费者，send/receive是2个通信原语，通信也可以是异步或者同步的：

* 同步：发送进程与接收进程在每个消息上同步（必须关联发生，在这期间不能干别的，必须都完成了，都才能向下走，这就是同步），send/receive都是阻塞操作，发送者在send后，一直阻塞直到接收端receive并正确处理，receive的消息处理完后，服务端也会一直阻塞等待新的消息；
* 异步：send操作是非阻塞的，消息被复制到缓冲区，发送者去干别的，生产与发送是同时进行的，异步的意思就是（不管你干没干完，我这边不用管，我继续干我的），receive操作可以是阻塞的，与同步的形式一样，也可以是非阻塞的，那就是接收进程receive后复制到缓冲区，继续receive，不用等处理完，接收这需要轮训或者定期的检查事件去读取消息。
* 每个消息都有消息目的地，里面含有端口；
* 可靠性；
* 排序。

### 4.2.2 套接字

TCP/UDP都使用套接字抽象，套接字是进程通信的端点的抽象，起源于UNIX，每个套接字绑定到<IP,PORT>对；java语言中，InetAddress抽象表示互联网地址。

### 4.2.3 UDP数据报通信

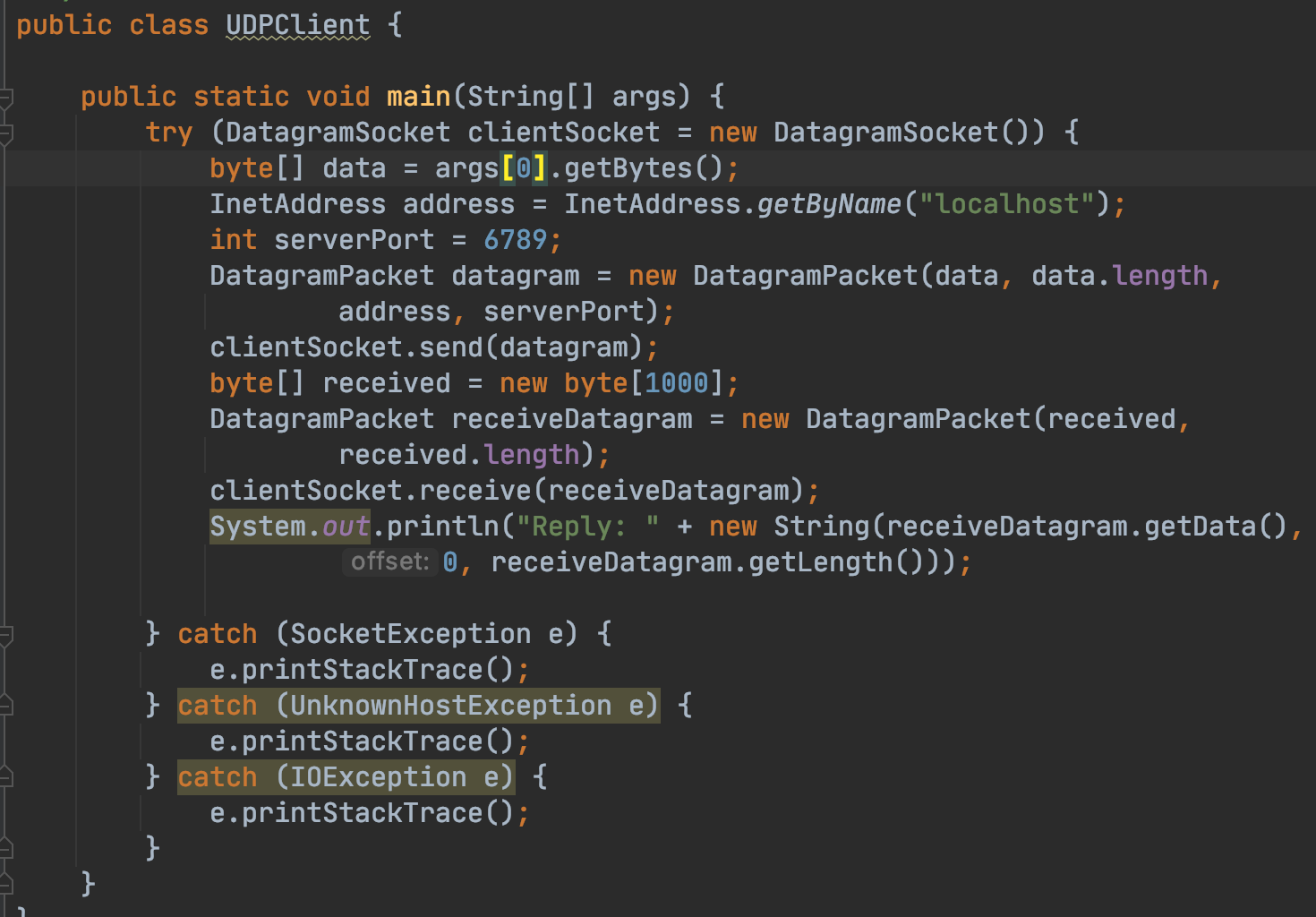
这种通信模式只是简单的发送IP数据报，但是通信也有一些注意点：

* + 消息大小：接收进程对接收消息的大小有限制，超出则会截断；
  + 阻塞：send是非阻塞，复制到缓冲区，不管发没发送，receive一般是阻塞，等到消息，到来后，送入单独的线程中处理；
  + 超时：设定receive的等待时间；
  + 任意接收：可以限制接收的来源根据IP与端口号。

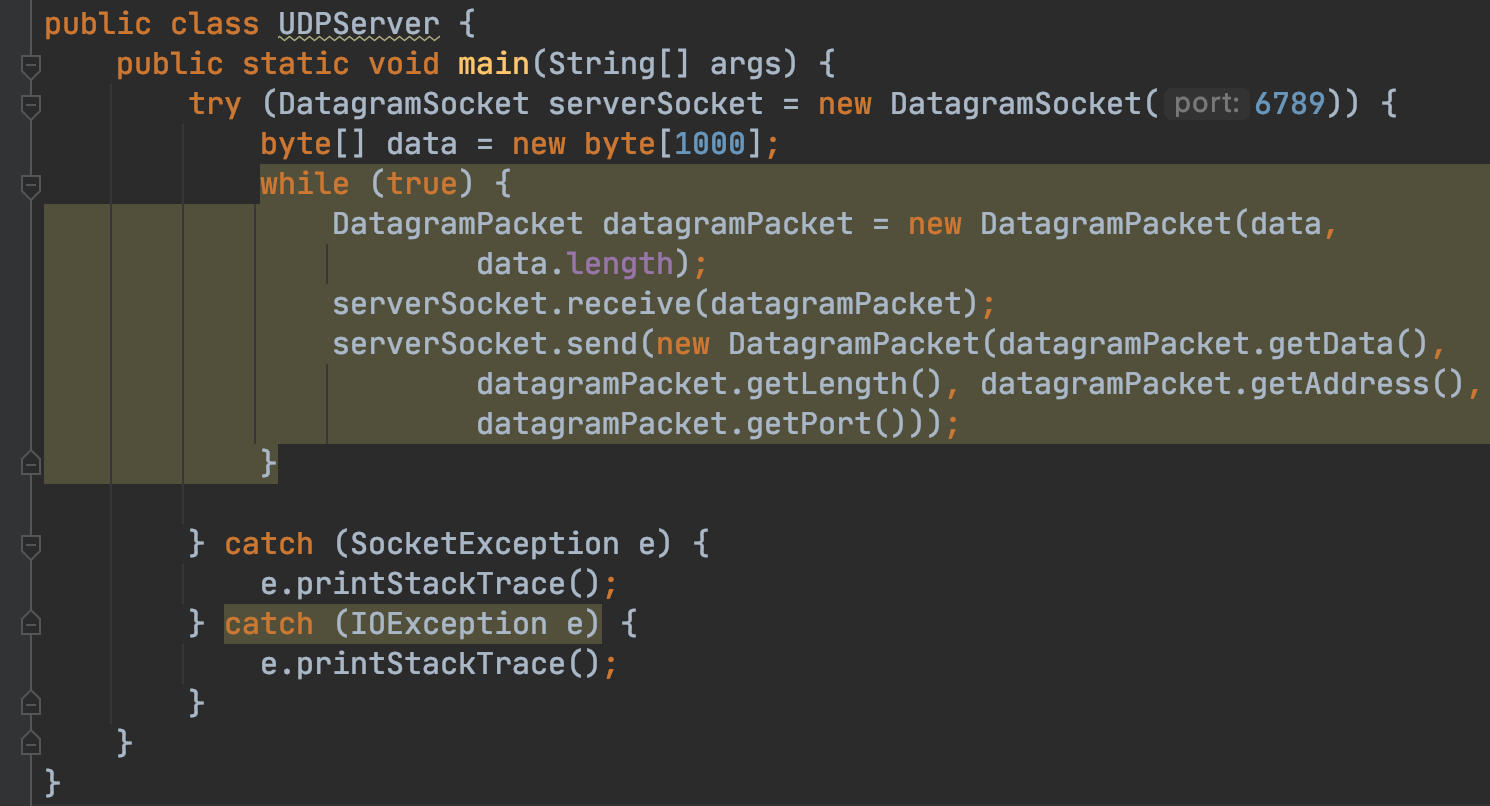
UDP数据报故障模型：

* 遗漏故障：也就是消息丢失；
* 乱序；

java中对UDP的抽象类是DatagramSocket与DatagramPacket。



服务端的设置如下：



### 4.2.4 TCP流通信

TCP提供了一种网络通信中的流抽象，流的特点是可靠双向传输，底层的保证机制有：

* + 消息大小可以通过分片的机制变成任意的，可以随时发送；
  + 消息重传机制，或者滑动窗口机制保证消息不会丢失；
  + 消息传送的速度控制；
  + 消息重复丢弃与消息编号保证顺序；
  + 只需要建立一次连接开销；

套接字有点消息队列的意思，发送队列与接收队列，好好想象；每个套接字都有一个输入流一个输出流，分别用于发送与接收；close套接字的时候，关闭输出流（记住没有关闭输入流），并在最后发送的数据末尾添加EOF，指明流已经结束。

流通信的模型需要注意的是，双方需要确保应用层协议的一致性，否则通信无意义，读写消息队列都容易造成阻塞，流控也会出发阻塞，使用单独的线程处理连接后的通信。

TCP使用校验纠正或者丢弃破坏的数据包，使用序号丢弃重复的包并按顺序组合，使用超时与重传处理丢失的数据包，

java中TCP的抽象类是ServerSocket与Socket.



TCP server的代码：



关闭套接字后，流不可用，但是远程端还可以用，最后读到EOF，如果还继续读，则会有EOFException异常，如果向关闭的流写数据则是IOException。

## 4.3 外部数据表示与编码

进程通信中的消息都是字节码，发送端与接收端对于字节码与数据对象的对应关系是不同的，比如大端/小端表示法，解决办法是：

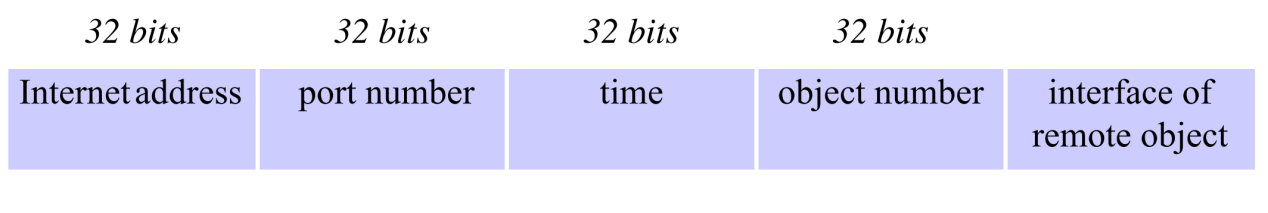
* 使用一种一致的外部格式，发送端负责转为外部格式，接收端负责把外部格式转化为本地格式，就是定义一种双方都能识别交流的语言进行通信，这种格式叫做外部数据表示；
* 按发送端格式传送，并在里面设置格式信息，接收端根据格式信息转为本地格式。

编码：将数据组装成适合消息传递的格式；

解码：达到后分解消息反过来的过程。

3种数据表示与编码法：

1. CORBA：定义了数据类型，没搞明白；
2. Java对象序列化：实现Serializable接口表明类对象可以被序列化，序列化对象是将对象转化成为适合存储与传输的二进制信息，解序列化，是从二进制恢复为对象，因为解序列化时需要目标类型信息，所以类型信息是包含在序列化后的数据中的；内部包含的对象也会跟随序列化，引用会被序列化为一个句柄，作为一个指向其他序列化对象的引用，序列化使用Object(Input|Output)Stream中的方法，反射，查询类的属性；因为反射的功能，可以编写通用的编解码的功能；
3. XML（Extensible Markup Language）标记语言：web上的结构化文档，XML的元素与属性，XML文档关键字符转义处理、XML命名空间、XML模式；
4. 远程对象引用：用于远程方法调用中识别对象的一种标识，在整个的分布式系统中唯一，远程对象的引用在整个分布式系统中是唯一的，可以通过引用得知是否引用相同的对象，远程对象引用必须保证在时间与空间上的唯一性，确保持有陈旧的引用不会调用成功，格式：IP+port+time+object no



这种远程对象引用实际就是对象的地址了，但是如果应用可以被定位到多个计算机中的多个进程的一个对象的话，就不能使用这种地址的表示，P2P系统中的远程对象引用就不是使用位置表示的。

## 4.4 多播

广播的场景不适合请求/应答模式下的p-to-p通信，多播通信适合做这个，组播是发送者的一个消息依次发送到一个组中的所有的成员的操作，但是组对发送者来说是透明的，感觉就像是发送给一个单一的进程，使用多播的分布式系统特点如下：

* 用于容错的多副本服务：
* 自发网络中的服务发现，用于注册服务与发现服务接口；
* 用于提升性能的多副本数据，就是一写多复制的情况，就会使用广播消息通知副本完成数据变更；
* 事件通知：特别是发布-订阅系统，使用组播发送通知给订阅者；

### 4.4.1 IP多播

IP多播是一个发送者发送一个IP数据包到一组计算机，发送者不知道多播组有多少计算机，一个多播组是用一个D类IP（1110开头）地址描述的；多播组中的计算机可以在任何时刻加入或者离开多播组，也可以同时加入多个多播组；IP多播只能传送UDP报文，在IP数据报中写入D类多播地址与端口号就可以多播了，加入多播组只需要进程套接字关联的IP地址是多播地址就可以了，当广播消息到达一个计算机是，消息的副本会被转发给本地多播组中所有的套接字；

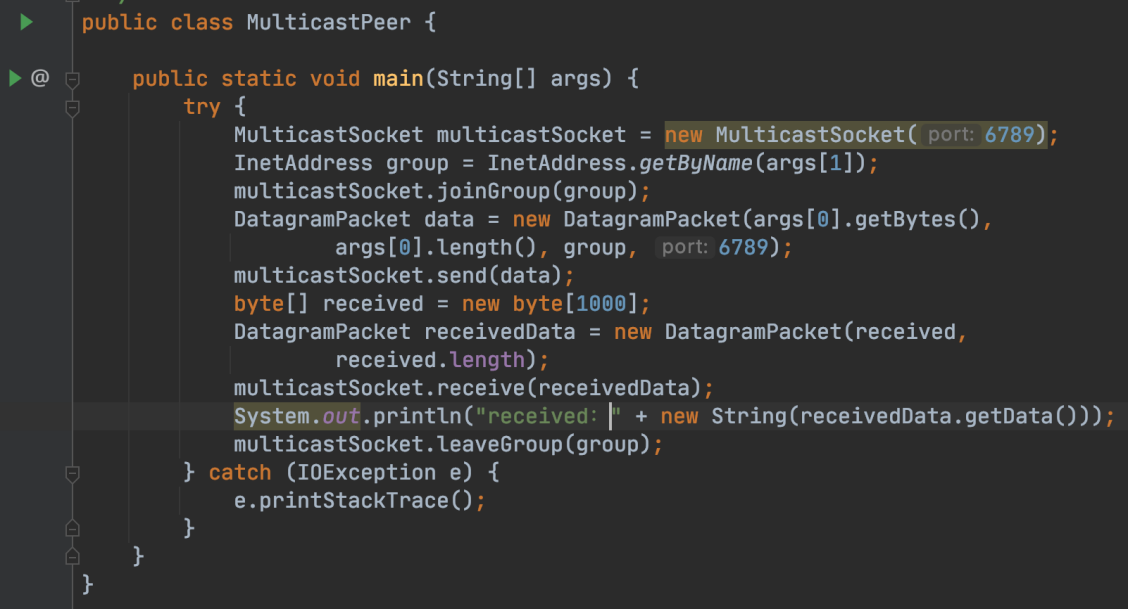
IPv4地址下多播的过程：

* + 多播路由器，本地网络使用本地网络的多播，互联网多播需要用到多播路由器，用于转发数据报到其他网络的路由器，为了限制多播数据报传播的距离，sender可以指定数据报经过的路由器上限个数，这个叫做time to live也就是TTL；
  + 多播地址分配，D类地址范围224.0.0.0～239.255.255.255，这写地址也被分成了多个用途部分：224.0.0.0～224.0.0.225用于本地的多播，224.0.1.0 ~224.0.1.225用于互联网的多播，224.0.2.0～224.0.255.0 用于自组织网络的多播，239.0.0.0～239.255.255.255 用于多播通信的管理。

组播地址可以是永久的，也可以是暂时的。

组播的的故障与UDP一样的，因为都是不可靠传输，信息可能会丢失。

Java实现：MulticastSocket。



## 4.8 网络虚拟化：覆盖网络

互联网通信协议提供了很多的工具api，同时是有很多的网络科技构造而成，这造成了，互联网通信协议可能不是对所有的互联网应用都适配，所以提出了一个网络虚拟化的概念。

网络虚拟化就是在互联网上建一个虚拟网络，虚拟网络对应这一个分布式的应用。

### 4.5.1 覆盖网络

一个由节点与虚拟链接组成的虚拟网络，位于一个底层之上，特点是：

* 满足某一类应用的需求或者比较高层的服务；
* 需要引入网络层协议，这些协议可以完成覆盖网络的功能。

## 4.6 实例研究：MPI

MPI是一个跨语言的通讯协议，用于编写并行计算机。支持点对点和广播。MPI是一个信息传递应用程序接口，包括协议和和语义说明，他们指明其如何在各种实现中发挥其特性。MPI的目标是高性能，大规模性，和可移植性。MPI在今天仍为高性能计算的主要模型。

并发程序模型就是消息传递模型，其实就是进程之间通过传递消息协同工作。

# 远程调用

本章是进程通信的小改进，分布式系统2个最主要的远程调用技术：

* 远程过程调用(RPC)：一个调用过程可以像调用本地节点上的过程去调用远程节点上的过程；
* 远程方法调用(RMI)：与RPC类似，但是使用了面向对象的编程概念，把对象引用扩展到分布式环境中，在远程调用中可以把对象引用作为参数。

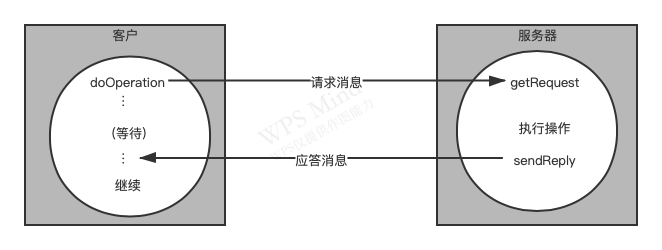
## 5.1 简介

远程调用是基于请求-应答协议的，请求-应答协议是消息传递的一种机制，提供了通信的底层支持，也就是TCP/UDP/IP协议等，一开始是远程过程调用，后来面向对象的编程模型出来后，远程过程调用演变为远程方法调用。

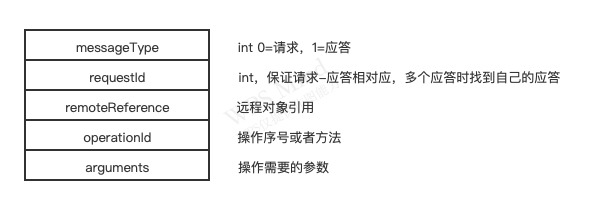
RPC（远程过程调用）和RMI（远程方法调用）是两种可以让用户从一台电脑调用不同电脑上面的方法的的机制（也可以称作规范、协议）。两者的主要不同是他们的使用方式或者称作范式，RMI使用面向对象的范式，也就是用户需要知道他调用的对象和对象中的方法；RPC不是面向对象也不能处理对象，而是调用具体的子程序。

## 5.2 请求-应答协议

请求-应答协议基本就是客户/服务器结构，请求应答协议基于3个通信元语：doOperation、getRequest、sendReply。



消息的结构如下：



任何消息都有唯一的消息标识符，通常由<IP,PORT>+requestId构成。

请求应答协议由UDP实现的话，存在UDP的故障类型，即遗漏故障语乱序故障。当遗漏发生时，doOperation会等待超时，这是可以抛出异常；重复发送的消息根据标识符会被丢弃，对于服务端来说，处理重复的请求要求结果是幂等的。

交互协议的3种类型：

* + 请求协议（R）
  + 请求-应答协议（RR）
  + 请求-应答-确认应答（RRA）

为了简化请求应答协议，使用TCP流是比较好的选择，帮助处理通信故障，而且可以传输任意大小的数据。

请求-应答协议的例子是HTTP协议。

## 5.3 远程过程调用

远程过程调用就是为了屏蔽分布式，使像单机一样操作，传统的本地的过程调用转为分布式调用，RPC框架屏蔽了底层的细节，像本地调用一样。

### 5.3.1 RPC的设计问题

1. 接口编程

软件规模变大，整个程序分为各个模块，做好模块的交互就是要定义好交互的接口，也就是依赖倒置，依赖接口而不是实现，这样也解决了分布式系统异构性的问题。分布式系统的接口就是服务接口（服务提供的操作，也就是过程以及相关的参数）。接口定义语言（IDL）定义了一种通用的接口表示，方便转化为各种编程语言的实现，在不同语言的进程互相调用时，遵循了同样的标准。

2）RPC调用模式

RPC传输保证方式：1.重发请求消息，2.过滤重复请求，3.重传结果；

RPC的调用模式：

获取调用：就是可能执行了或者没执行，反正不知道；

至少一次调用：发生异常重发消息，避免了遗漏故障，但是重发的消息可能多次执行，造成数据错误（幂等操作除外）；

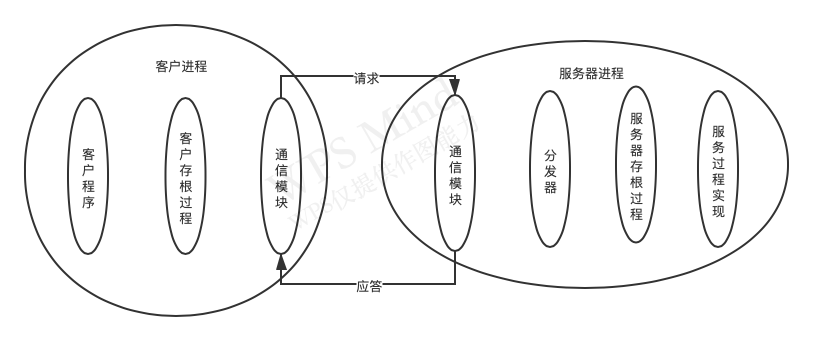
至多一次调用语义：是至少一次调用的增强版，就是缓存了应答，可以重传结果。

3）透明性

远程调用是透明的，RPC提供过程的位置与访问透明性。

### 5.3.2 RPC的实现

stub(存根，也可以理解为接口副本)，



分发器根据过程标识符选择一个存根过程，存根过程对参数解码，调用服务过程，然后对结果进行编码。分发器与存根过程都可以由接口定义语言编译器生成。

### 5.3.3 实例研究：Sun RPC

## 5.4 远程方法调用

远程方法调用也就是RMI，其实就是RPC，但是在分布式对象领域（面向对象）称为RMI，都支持接口编程（接口的形式不一样），都是使用的请求-应答协议，都提供远程操作的透明性。RMI可以使用一切面向对象的编程方式，并且传递对象引用。

### 5.4.1 RMI的设计问题

1. 对象模型设计

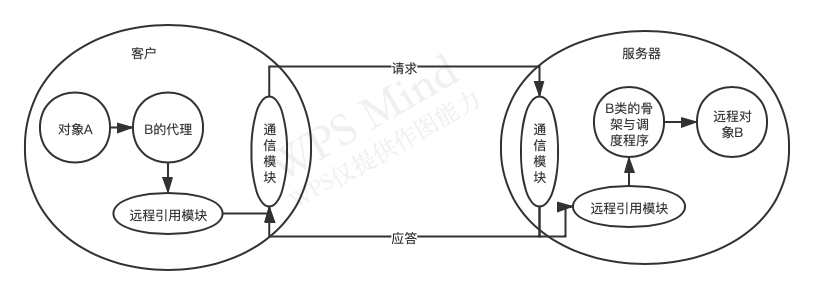
传统对象都有封装、继承、多态与抽象的特点，分布式对象也有，但是分布式对象的数据只能通过方法访问；对象引用、接口、动作、异常与内存回收；

分布式对象就是对象分布在不同的进程中，使用客户-服务器体系结构传统的方法调用编程了消息的传递。

不同进程的对象之间的调用是远程方法调用，同一进程中的对象间的调用称为本地方法调用。

远程方法调用必须持有远程对象的引用，并且持有远程对象的接口定义也就是操作，远程对象应用是整个分布式系统中对象的标识符，用于指定对象位置，可以作为远程方法调用的参数与结果传递。

### 5.4.2 RMI的实现



* 通信模块：执行请求-应答协议，传递请求与应答消息；
* 远程引用模块：负责在本地对象与远程对象之间翻译，维护一个远程对象表记录远程对象引用与本地对象的映射关系
* 远程对象B就是远程对象实例；
* RMI软件：由位于应用层对象与通信模块远程引用模块之间的中间件层组成，主要的中间件有代理（隐藏远程方法调用的细节，实现了消息的编解码与通信工作，实现了远程对象的接口保证接口一致性）、分发器（每个分布式对象实例都有，用来传递消息，根据methodId在骨架中找到方法）、骨架（消息编解码，调用实际对象执行）。
* 创建代理类、分发器类与骨架类：由接口编译器自动创建，如果编译期还不能确定远程对象的接口，那么可以使用通用的动态调用的方式或者直接下载类的方法；
* 服务器与客户程序：服务期程序包含分发器类与骨架类，客户程序包含所有远程对象的代理类；
* 绑定程序：实际就是注册中心，名字->远程对象地址的映射；
* 服务器线程：就是每个远程调用一个线程处理。

剩余的TMD我也读不懂。

### 5.4.3 分布式无用单元收集

就是对象，没有被引用了，那么对象就会被收集，简单的收集算法如下：

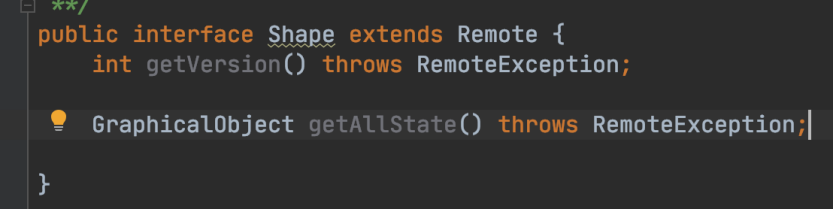
1. 服务器端为每个远程对象维护一个object-> reference list的映射；
2. 当客户需要远程对象引用时，生成代理，并addRef，向远程的refrence list中加入自己的标识；
3. 当客户不在使用到远程对象引用时，调用removeRef，向远程的reference list中移除自己；
4. 若reference list为空，则本地垃圾收集器回收远程对象。

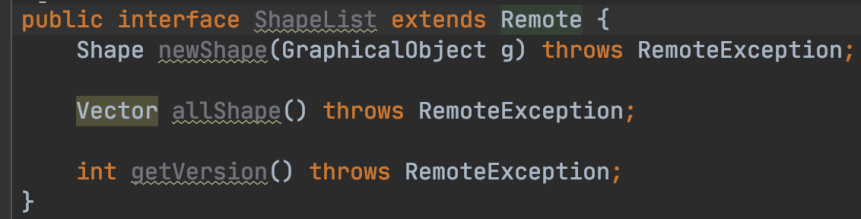
## 5.5 实例研究：java RMI

Java RMI扩展了Java的对象模型为分布式对象，但是对象之间是能知道调用是远程的，不是本地的，因为调用者要处理RemoteException，远程对象要实现Remote接口。

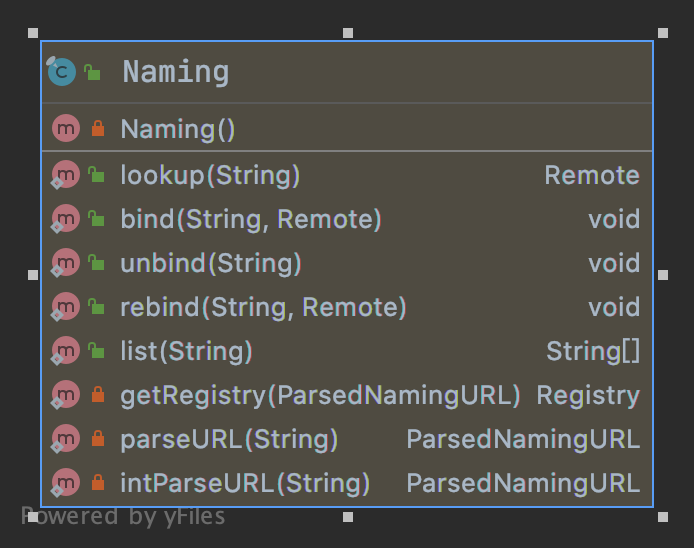
例子是共享画板。

* 1. Java RMI中的远程接口，远程接口必须扩展java.rmi.Remote，其中的方法必须必须抛出RemoteException异常，下面是Shape与ShapeList远程接口的定义，注意远程对象与普通对象都可以作为远程接口的参数与结果，如果是远程对象传递远程对象引用，普通对象传递序列化的对象数据是值传递。





* 1. 传递参数与结果，JavaRMI中的参数与结果都必须实现Serializable接口， 数据类型可以从RMI系统中下载，传递远程对象就是传递的远程对象引用（代理类，里面包含远程对象引用），非远程对象传递传递的都是序列化后的值（里面包含类信息（URL），可以从RMI下载）。
  2. 类的下载，client或者server会自动你同步本身没有的类的类信息。
  3. RMIregistry，Java RMI的绑定程序，运行在server端，维护一张表，key是远程对象引用地址 格式是//computername（IP）:port/objectName，value是内存中的远程对象地址，这个表可以通过Naming静态方法读写，Naming接口如下：



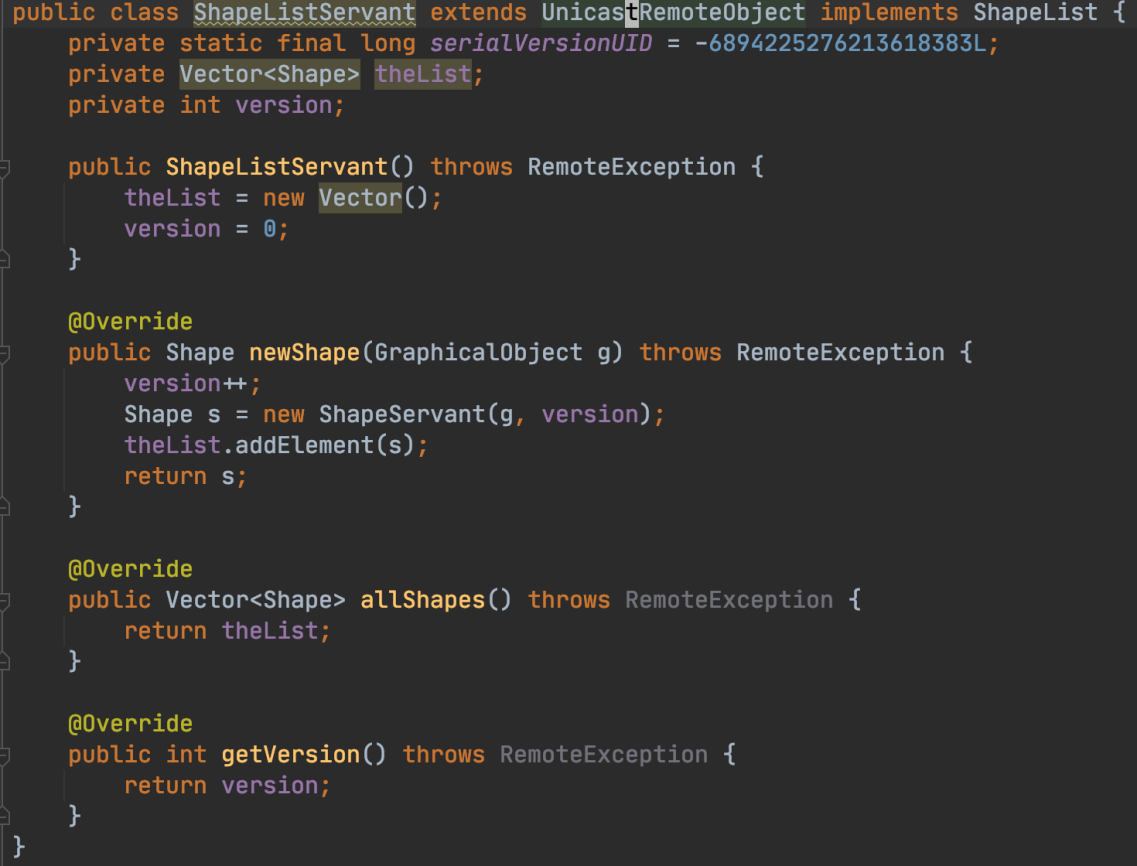
Naming静态方法底层使用的是LocateRegistry类，LocateRegistry是实际的RMIregistry的目标类，LocateRegistry.getRegistry可以在本地构建一个绑定到网络中的RMIregistry的Registry，Naming底层就是通过Registry对象完成注册表读写的。

### 5.5.1 创建客户和服务器程序

首先创建一个rmiregistry命令，需要注意classpath的设置或者在本地项目的当前目录下，否则注册中心无法找到server中的类，默认是1099端口，可以指定：



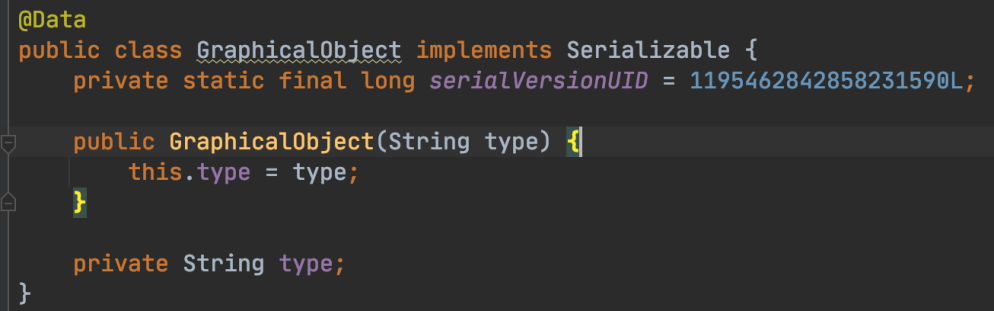
实现远程对象接口ShapeList：



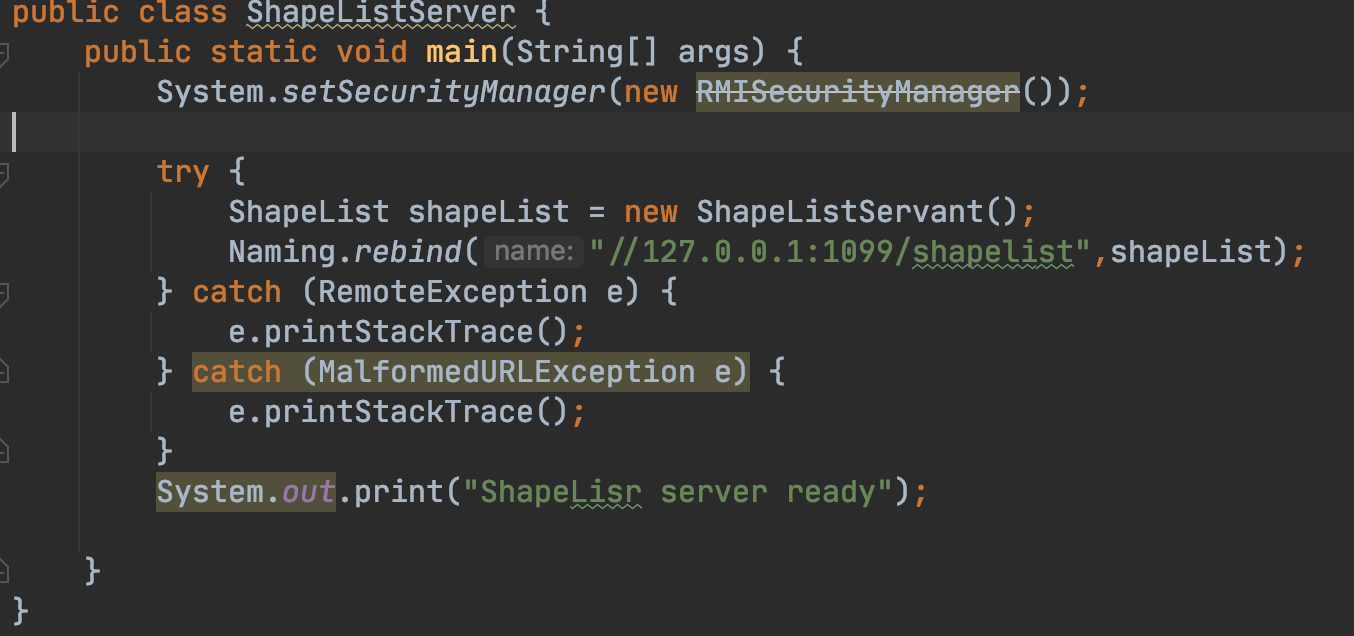
注意这里继承了UnicastRemoteObject类，逻辑是每添加一个GraphicObject类，版本号加1，同时生成一个Shape类对象，实现逻辑是：



GraphicObject类的实现如下：

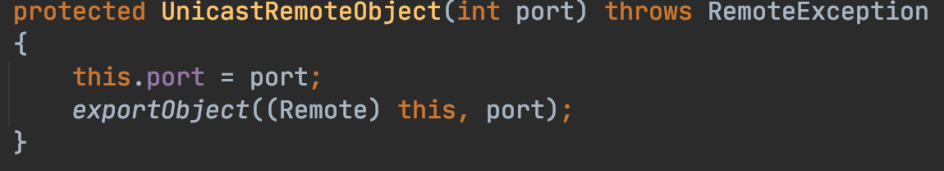


以上是所有的服务端业务逻辑类的实现，下面是Server的实现：



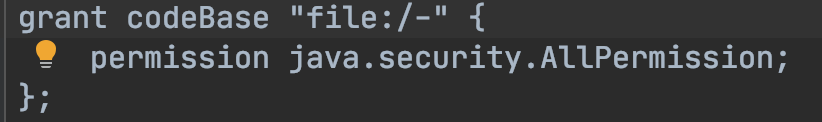
首先设置一个安全性管理者，这个与默认的不同，以后需要好好看这里。

接着创建对象，这里的对象在创建时，调用了UnicastRemoteObject的构造方法，里面调用了exportObject方法，这个方法使得该对象可以被远程调用。



Naming类然后完成对象与名字的绑定。

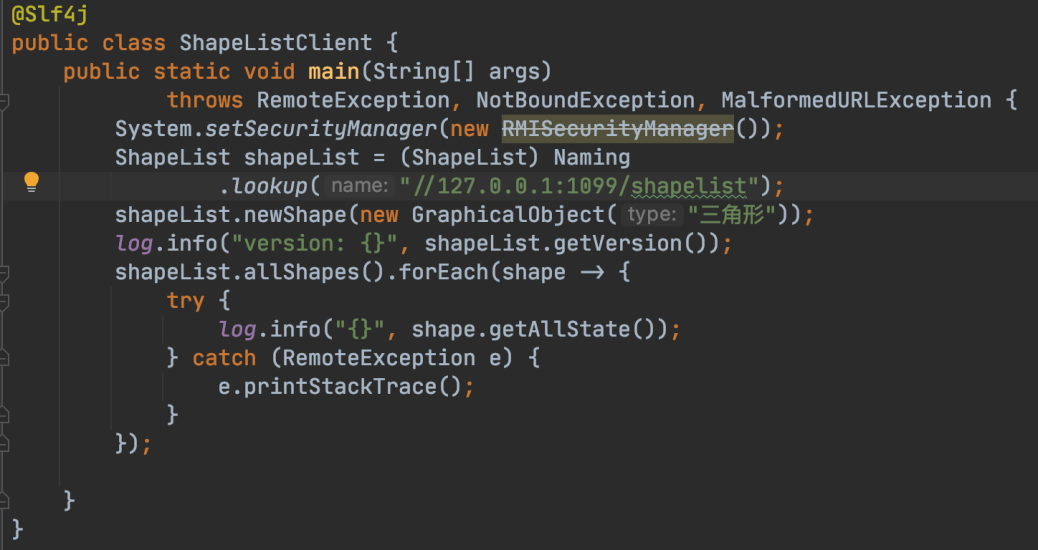
需要注意的是，因为使用了网络，需要定义安全策略文件，否则会报access denied的错误。



同时通过虚拟机参数或者环境变量传递进来（客户端一样）

-Djava.security.policy=/Users/zyx/Documents/java-concept/java-introduction/src/main/resources/security.policy。

客户端的代码如下：



回调：客户不用为找出某个事件的发生而轮询服务器，而是当事件发生时，有服务器通知客户，实际就是传递方法参数，从而实现底层执行上层的逻辑，展示的现象就是底层调用上层（常规都是上层调用下层），RMI中的实现方式：

1. 客户端创建一个远程对象的实现，实现CallBack接口；
2. 向服务端通知该Callback对象；
3. 服务端持有客户端的Callback对象远程引用，当事件发生时，调用Callback对象。

### 5.5.2 RMI的设计与实现

反射：主要是使用到了Method对象，RMI的请求中，序列化了Method对象与请求参数，还有远程对象地址，到了服务端的分发器，调用Method对象的invoke()方法就可以，这是一个通用的分发器。

# 间接通信

## 6.1 简介

间接是计算机科学中的基本概念，著名的断言是：

* 计算机科学中所有的问题都可以通过某个层次上的间接方式解决；
* 没有通过消除某种层次上的间接方式解决不了的性能问题.

分布式系统中的间接通信就是通过中介者通信，没有发送者与接受者的耦合。这种间接通信有2个主要特点：

* + 空间解耦，就是发送者与接受者互相不知道，自己替换、更新、迁移不影响别的系统；
  + 时间解耦，发送者与接收者有独立的生命周期，不需要2方同时存在；

间接通信用于预期会发生改变的系统或者用作事件分发。间接通信的缺点是增加间接层带来的性能开销并且难于管理。

|  |  |  |
| --- | --- | --- |
|  | 时间耦合 | 时间解偶 |
| 空间耦合 | 直接同步通信，必须同时存在 | IP异步通信，微信消息等 |
| 空间解耦 | IP组播多播或者发布订阅系统 | 间接通信 |

时间解偶的通信信道都是存储消息的，这样才能支持时间解偶，任何时刻接收消息。

异步通信与时间解偶是不同的概念，异步通信是指不阻塞，而时间解偶是指接受者可能在发送时不存在。

## 6.2 组通信

定义：消息先被发送到组中，然后传送到组中的所有成员；组通信的应用领域：

* 面向大量客户分发信息；
* 协作应用，事件被分发到多个用户，保留一个共同的用户视图；
* 容错策略，数据复制等高可用性的实现；
* 系统监控与管理

### 6.2.1 编程模型

核心概念是组与组成员，就是消息被传到组就是传到组内所有的策划姑娘元，组通信实现了组播通信；与系统中的所有进程通信，被称为广播(broadcast)，与单个进程通信被称为单播(unicast)； 组播一个消息只向组发送一次，不会多次发送，节省带宽，这是通过路由器的多播功能实现的，

组分2种对象组与进程组：

* + 进程组，通信的实体是进程，消息是二进制；
  + 对象组，通信的实体是对象，会有一个前端代理，前端代理负责组通信，通信的内容是有具体的编码格式的，有点类似与负载均衡；

封闭组与开放组：组外的通信实体是否可以向组内的通信实体发送消息；

重叠组与非重叠组：通信实体是否可以加入多个组；

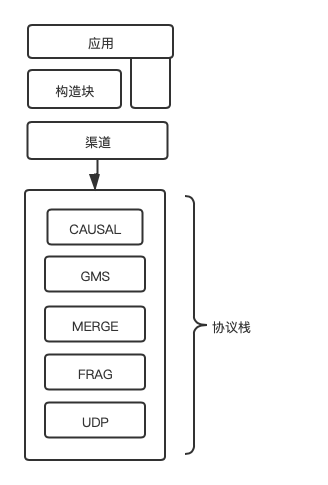
### 6.2.2 实现问题

组通信的可靠性与排序保证：

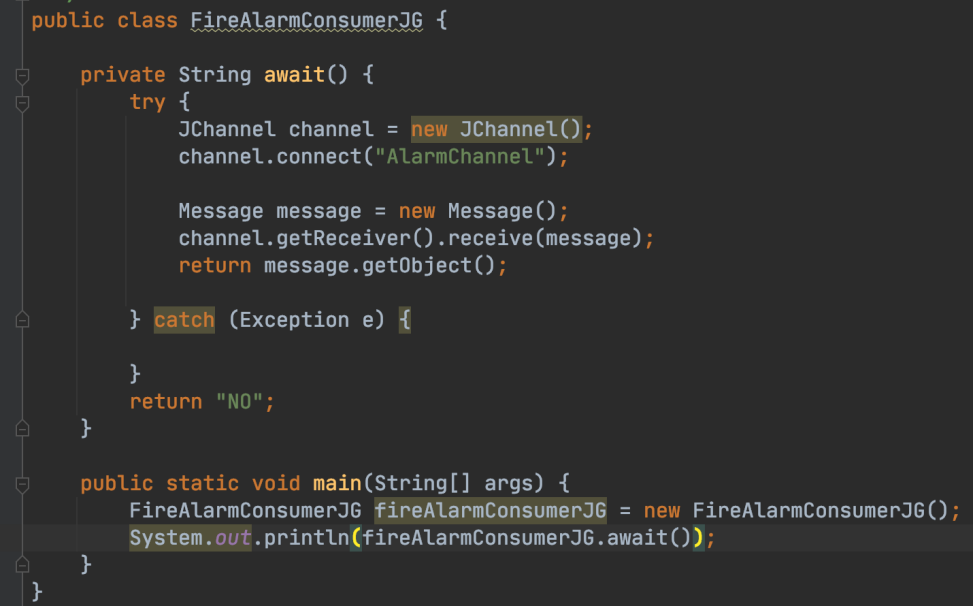
* 可靠性：完整性、有效性、协定；
* 排序：FIFO、因果序、全序；
* 组成员管理：提供组成员改变的接口（加入或者离开）、故障检测、成员变更通知、组地址扩展；

### 6.2.3 实例研究：JGroups工具箱

java语言编写的可靠组通信工具箱，提供完整进程组的组通信服务；包括组成员管理，与通信相关的保证；架构图如下：



* + 渠道，代表组通信与成员管理的接口，一个进程通过渠道与组交互，渠道代表的就是组，创建渠道就代表加入了一个确定的组，并可以执行一些组上的接口操作；代码如下：



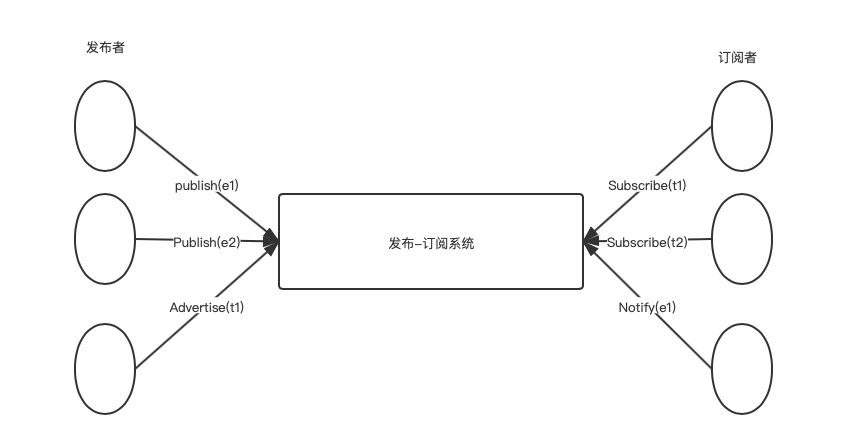
* + 构造块，是在渠道上的更高的抽象，提供对常见的通信模式的支持，常见的有MessageDispatcher、RpcDispatcher、NotificationBus等；
  + 协议栈，通过协议层构成的协议栈，有5层构成
  + UDP层，IP组播层，点对点通信。

## 6.3 发布-订阅系统

发布-订阅系统是间接通信中应用最广泛的。这种系统的特征：

* 异构性；
* 异步性；

### 6.3.1 编程模型



发布-订阅系统的操作集合：

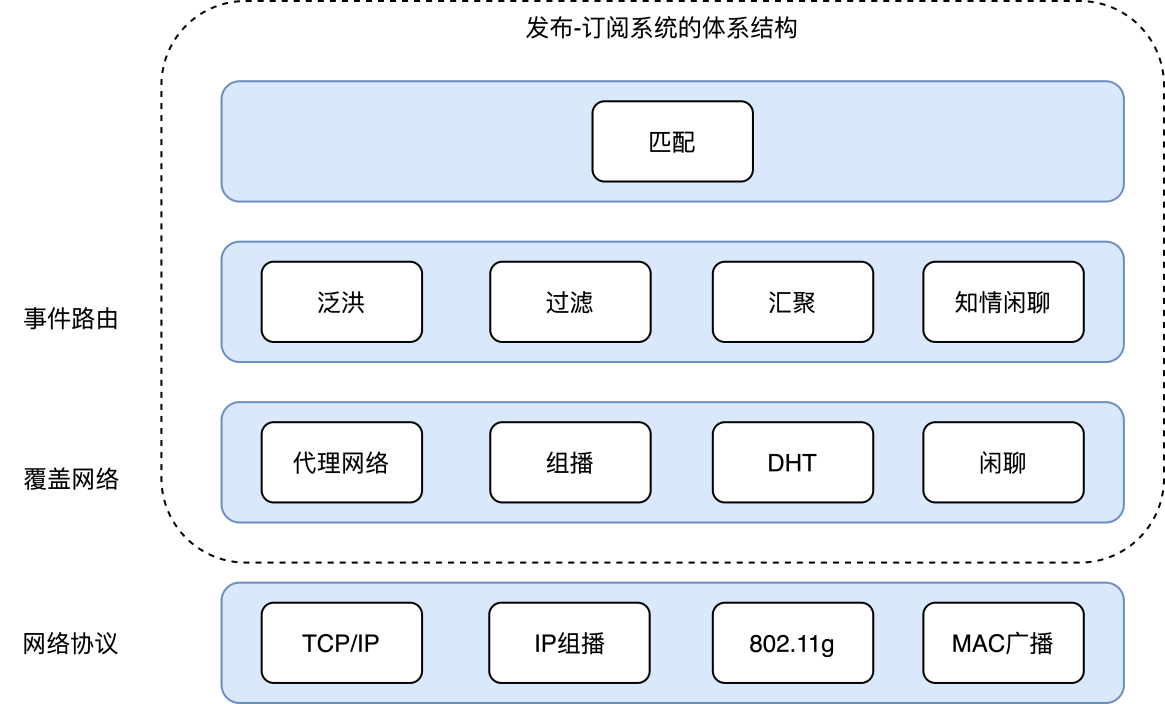
* + [un]Publish(event)：发布事件；
  + [un]Subscribe(filter): 订阅符合filter规则的事件，filter是事件模式；
  + [un]Notify(event)：通知事件；
  + [un]Advertise(type)：将要发布的事件的定义或者类型；

目前已经支持的事件模式有以下几种：

* 基于渠道：发布者发布事件到命名渠道，订阅者订阅渠道并接收事件；
* 基于主题：事件中显式或者隐式的含有主题的定义，订阅者根据主题来确定是否接收事件；
* 基于内容：根据事件的内容订阅；
* 基于类型：根据事件的类型或者对象的类型来订阅；

### 6.3.2 实现问题

发布-订阅系统一般都是分布式实现的，这是为了提升性能， 发布-订阅系统的体系结构：



* + 底层是进程间通信服务；
  + 事件路由层执行路由到指定的订阅者服务；
  + 顶层实现匹配逻辑。

路由层的实现：

* 泛洪：事件发送到所有订阅者，订阅者自己匹配，或者订阅发送到发布者，发布者发布时直接广播到指定的订阅者；
* 过滤：就是系统维护一个到订阅者的路由路径，上面匹配到了，就查路径直接转发，有点类似于IP路由协议；
* 广告：不知道讲了啥；
* 汇聚：没看，翻译的狗屎一样；

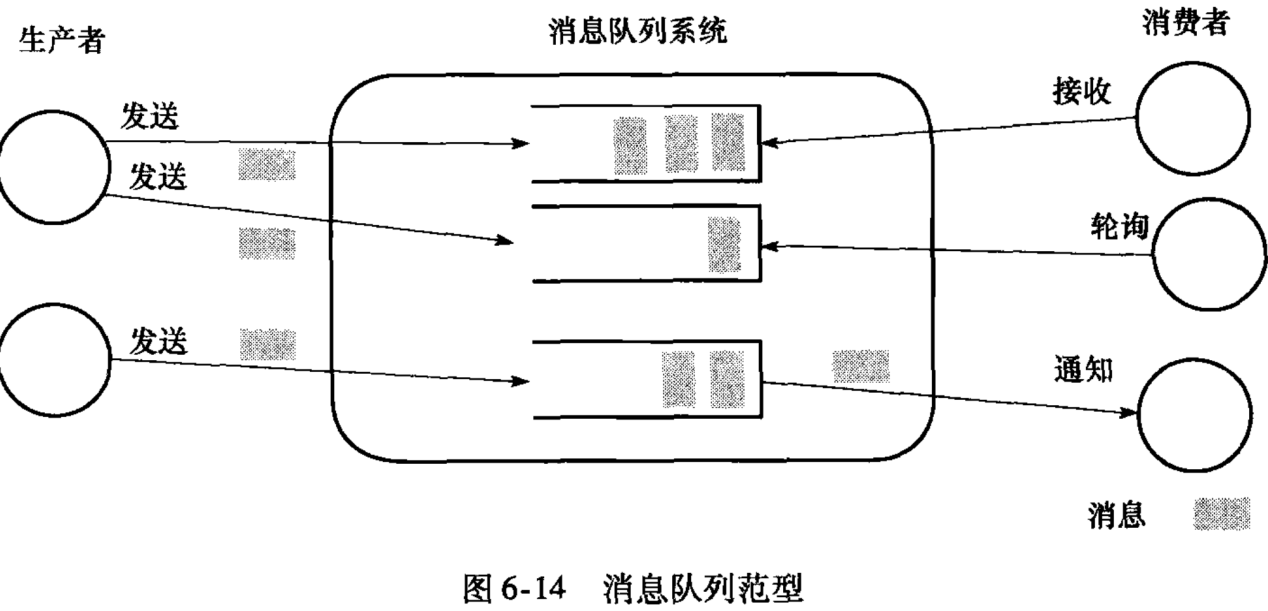
## 6.4 消息队列

消息队列也就是分布式消息队列，提供点对点通信，由于队列的特性提供时间空间解偶。

### 6.4.1 编程模型

编程模型就是队列，有生产者与消费者；接收目前有3种方式：

* + 阻塞接收：保持阻塞直到有消息可用；
  + 非阻塞接收(轮询)：检查队列的状态，可用就返回，或者返回不可用；
  + 通知操作：队列可用时，发出事件通知。



队列种排队的策略是FIFO，也支持优先级的概念，消息由目的地、消息元数据、消息体3个部分组成，消息是持久存储的，所以是可靠传递，此外支持爹特性包括：

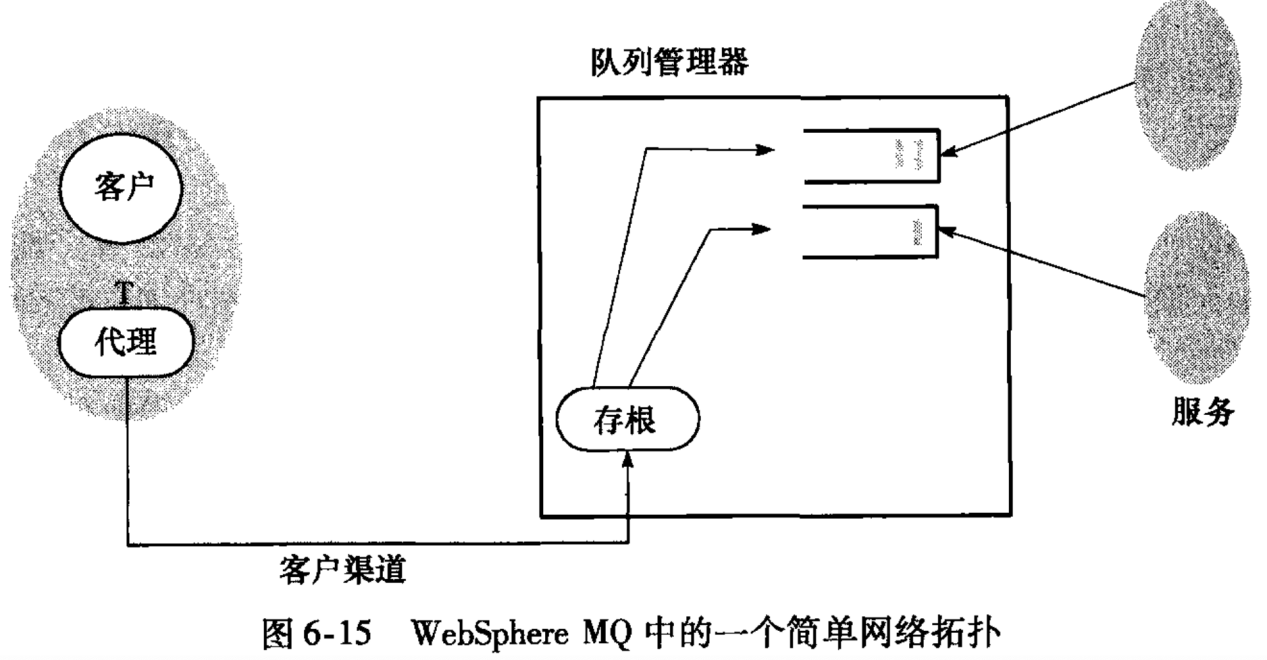
* 消息的发送与接收都是在一个事务中，原子性；
* 支持消息转换，类似于适配器，将到达的消息改成另外一种消息；
* 安全性支持；

### 6.4.2 实现问题

集中式消息队列容易出现单点瓶颈或者单点故障，分布式实现较好。例子是WebSphere MQ。

WebSphere MQ通过队列管理器管理队列，应用通过MQI(Message Queue Interface)访问队列，MQI包括连接/断连与发送/接收等操作。

客户应用与WebSphere 通信需要通过本地机器上的客户端通道（client channel），也就是代理；代理通过RPC与WebSphere 通信。



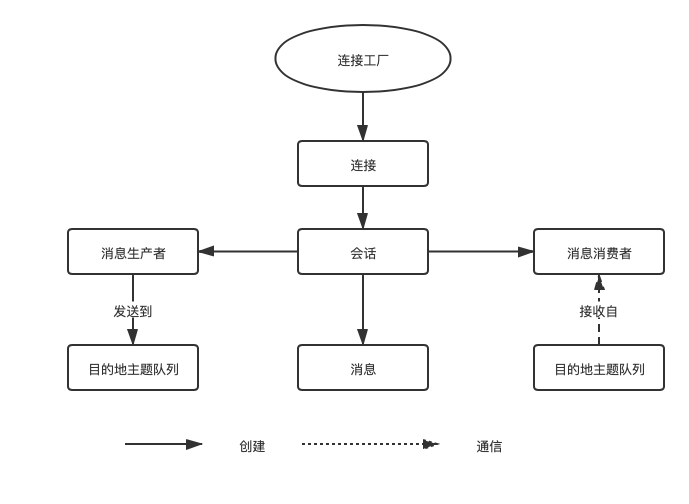
队列管理器如果分布在不同的机器上，通过消息渠道（Message Channel）通信，用于转发消息。

### 6.4.3 实例研究：Java消息服务

JMS是分布式Java程序间接通信的标准化规约，支持主题与消息队列，是一种发布-订阅系统与消息队列的混合体，有很多对规约的实现。JMS的组件：

* + JMS客户：生产者或者消费者；
  + JMS服务提供者；
  + JMS消息；
  + JMS目的地，通常是类似于主题的消息队列。

JMS编程模型如下：



会话对象是JMS操作的核心，可以创建消息、生产者于消费者。

* 消息由3部分组成，头部、特性集于消息体；头部主要包含系统或者路由信息，比如目的地（主题）、消息的优先级、过期日期、消息ID、时间戳等；特性集由用户定义的消息元信息，消息体不透明，可以是任何数据；
* 生产者；
* 消费者，订阅主题，含有关联的消息选择器（过滤消息）于接收操作，接收操作可以是阻塞的或者非阻塞的。

## 6.5 共享内存的方式

分布式共享内存与元组空间。

### 6.5.1 分布式共享内存

就是内存是远程机器上的，多个客户端机器共享，并且就像操作本地内存一样操作远程内存，类似与多个进程操作一块大内存，更新可以及时看到。这种技术是为了解决并行计算的，并行计算的研究主题就是如何通过多核处理器最大速度与吞吐量的访问内存，因为这些多核处理器都是通过总线访问内存的，这涉及到总线竞争，实际的极限是10核。

消息传递与DSM的比较：

* + 提供的服务： DSM不需要对消息传递与编码，但是容易因为共享内存而相互影响，同时，异构计算机共享内存也是问题；
  + 效率：差不多。

### 6.5.2 元组空间通信

相比于共享内存操作的单位是字节，元组操作的元组结构化数据；比如git系统就是元组空间通信的。元组存在于元组空间，元组空间存储的一种类型的元组，可以write、read、take，都是阻塞操作；元组空间因为是共享内存的方式所以是空间与时间解耦的。

因为元组空间是分布式实现的，这涉及到一些问题：

* 复制：全量复制/分片复制；使用组播的方式。

实例研究：JavaSpaces，太老了，我也不会用了。

# 操作系统支持

## 7.1 简介

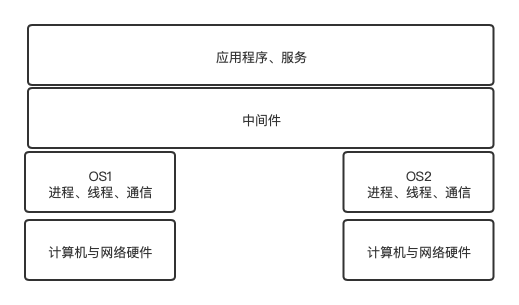
操作系统的作用就是提供物理层资源抽象；

* 网络操作系统：内置联网功能，对网络资源的访问跟在本地一样，比如ssh登陆机器；
* 分布式操作系统：就是网络上的资源是一体的，统一管理，看起来就是一个系统，而不是网络操作系统自己管理自己。

中间件与网络操作系统的结合为自治性需求与网络透明的资源访问之间提供了平衡。

## 7.2 操作系统层

中间件运行在不同的异构的计算机，也可能是不同的操作系统上，使用操作系统提供的资源抽象来提供分布式的通信功能，图示如下：



中间件因为要提供分布式共享资源的能力，对操作系统内核与机器上运行的进程有一系列的要求：

* + 封装：提供访问资源的服务接口，屏蔽底层细节；
  + 保护：资源的安全保护；
  + 并发处理：

访问已封装资源的手段叫做调用； 服务端内核、库等通过通信的方式接收调用的参数，然后在内核上调度执行，最终返回资源； 所以要求核心操作系统的功能有：进程与线程管理、内存管理以及进程间通信。

核心操作系统组件与他们的职责是：

* 进程管理器：负责进程的创建与操作，进程是资源管理单元；
* 线程管理器：负责线程创建、同步与调度；
* 通信管理器：负责进程中的线程通信；
* 内存管理器：管理无力内存与虚拟内存；
* 管理器：负责处理中断、异常调用，同时也控制处理器、寄存器、硬件缓存与内存单元。

## 7.3 保护

错误的执行了资源不提供的操作，资源保护的几种方式：

* + 密码；
  + 类型安全的编程方式；
  + 内核在管理模式下运行，其他进程在用户模式下运行，内核有自己的地址空间， 运行在内核态，进入到内核地址空间，运行在用户态就是用户地址空间。

## 7.4 进程与线程

进程是由一个执行环境与多个线程组成，线程是任务的操作系统抽象，执行环境是资源管理的基本单位，是一个进程拥有的所有的资源，主要包括：

* 地址空间：就是一块内存；
* 线程同步与通信资源：操作系统层面上的资源，比如套接字等资源；
* 高级资源：外设硬件资源等，比如文件与窗口等。

线程共享执行环境，但是在进程之间不共享，在不同的操作系统中，线程的叫法不一致。

### 7.4.1 地址空间

就是进程拥有的虚拟内存空间，很大，虚拟内存空间分为很多区域，每个区域有范围、访问权限与扩展机制等；包含3个主要的区域：

* + 程序代码所在的正文区域；
  + 堆；
  + 栈；

共享内存区域可以作为多个进程内的一部分，可以存储共享库、内核代码、数据共享与通信。

### 7.4.2 新进程的生成

新进程的创建是操作系统提供的原子操作，比如UNIX的fork与exec；分布式系统因为是多机的，所以创建进程需要首先确定目标机器，然后创建。

目标机器的选择是一个策略的问题：

* 转移策略：目标机器的选择根据本机的负载的轻重；
* 定位策略：取决于负载、体系结构或者机器的角色。

分布式系统的负载机制可能是集中式（有一个负载管理器）、层次化（多级负载管理器）、分散化（对等系统），这些负载管理器用来决定到哪个目标机器上创建进程，还分为发送方与接收方负载算法。

执行环境的创建分为初始化与复制2种，第一种就是静态初始化一个地址空间，第二中类似于fork，使用的是写时拷贝的策略。

### 7.4.3 线程

为了增加吞吐量，提出多线程服务器，有一下几种体系结构：

* + Worker pool architecture，工作池体系结构，就是线程放到池，任务放到队列中；
  + 一请求一线程体系结构(thread-per-request architecture): 每个请求生成线程处理，吞吐量最大，但是创建与销毁线程的开销比较大；
  + 一连接一线程体系结构(thread-per-connection architecture)：为每个连接分配一个线程；
  + 一对象一线程体系结构(thread-per-object architecture): 每个远程对象分别与一个线程相连。

客户端最好也用多线程机制。

多线程运行与多进程运行的优势：

* 线程的创建与管理开销比较小；
* 线程更容易共享资源，而不必要进行上下文切换。

进程与线程含有的包含状态的部分：

|  |  |
| --- | --- |
| 进程 | 线程 |
| 地址空间表 | 被保存的处理器寄存器 |
| 通信接口 | 优先级与执行状态 |
| 信号量与其他同步对象 | 软件中断处理信息 |
| 线程标识符列表 | 执行环境标识符 |

进程与线程的比较总结：

* + 创建线程比创建进程开销小；
  + 线程切换比进程切换开销小；
  + 线程方便共享资源；
  + 线程的安全性比较弱。

线程编程：

概念：race condition竞争条件、critical section临界区、monitor监视器、condition variable条件变量、semaphore信号量。

java语言提供了线程支持，Java提供了线程的创建、销毁、与同步的方法，

|  |  |
| --- | --- |
| 方法 | 用途 |
| Thread(ThreadGroup group, Runnable target, String name) | 创建线程 |
| setPriority() getPriority() | 设置/返回优先级 |
| run() | 执行任务 |
| start() | 将线程设置成runable状态 |
| sleep() | 将线程设置成SUSPENDED状态 |
| yield() | 进入READY状态并调用调度程序 |
| destroy() | 销毁 |
| Join() | 调用进程阻塞指定的时间直到进程终止 |
| Interrupt() | 中断thread，使它从阻塞得方法中返回 |
| Wait() | 阻塞调用线程到指定的时间或者通过notify方法唤醒或者被中断 |
| Notify() | 唤醒一个或者多个在object上调用wait()的线程 |

线程执行完run方法或者调用destroy，生命周期结束，线程可以按组管理，根源是安全性，别的组的线程不能操作当前组的线程。

线程同步的难点在于共享对象与线程协调合作。

线程调度分为抢占式调度与非抢占式调度：

* 抢占式调度：任何时候都可能发生调度；
* 非抢占式调度：占有了就是不让，领导说让也不管，非要自己决定（发生一次线程调用或者系统调用）让时才让；

非抢占式调度因为独占式运行不能使用多处理器，同时因为长期运行的一段自有代码可能使别的线程没有办法得到执行，常常需要加yield()方法。Java不支持实时处理，因为实时对线程的调度有要求。

很多操作系统都提供了多线程进程的抽象与系统调用，并且，线程实现是在用户态。

## 7.5 通信与调用

调用的作用就是在不同的地址空间上执行对资源的操作。操作系统提供了底层的通信原语来支持TCP/UDP协议，高层的通信机制是由中间件来实现的，因为在用户态实现这些机制比较容易，操作系统提供对标准协议的支持。

### 7.5.1 调用性能