**ZK**

1. 什么是Zookeeper ：分布式服务框架hadoop的子项目 主要用于分布式中经常用到的数据管理

本质是分布式的小文件（存储数据小）存储系统（目录树结构的），zk之所以能够进行分布式协调服务，与他自身的存储文件的目录树是息息相关的。Zk 可以对这些目录中的数据进行管理（增删改）和监控，通过数据的变化来达到对 分布式中数据的管理

如：集群管理、统一命名服务、分布式配置管理、分布式消息队列、分布式锁、分布式通知协调

使用场景 ： https://www.ibm.com/developerworks/cn/opensource/os-cn-zookeeper/

（1）、配置管理 ：对于分布式的项目 ， 公共的配置信息（每台服务器上的配置都配置） 假如这些信息要更换，如果手动一个个的更换的话，可能会出问题 那么可以使用zk 当一个server中的配置变化时 ，那么所有的server配置都会同时发生变化

（2）统一命名服务 ：分布式中的接口命名再项目初始化的时候在zk中没创建（保存在zk中是以目录树结构的形式保存的 (不重复) 并且该目录下的文件是一个ip地址, 当远程调用次接口时 ，实际上是找到了远程的ip地址 然后通过rpc 远程调用）

（3）集群管理 ： 集群中可能会有server挂掉，如果某个请求被分配到这个挂掉的server中就会出现问题 为了确保的服务的可用性 ， 可以在集群的nginx和server上安装zk ， server 和 nginx保持连接创建临时节点 当服务器正常运行时创建临时节点 但是服务挂掉时该节点这自动删除 （keepalived 也能实现）

（4） 分布式锁

（5） 队列管理 ：

A、当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达，这种是同步队列

创建目录，当有请求进入时就创建临时节点 ，然后判断目录中的节点数， 如果成员数未达到，则线程wait()

B、生产者、消费者 FIFO（先进先出）

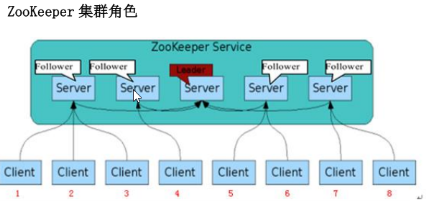
1. zk的特性
   1. 全局数据一致性：每台server上的数据都是一致的（文件夹下的数据都是一样的）
   2. 可靠性：如果消息（对目录中的数据进行增删改查）被其中一台服务器接受 ，那么将会被所有服务器接受然后一起操作
   3. 顺序性：包括全局有序 和 偏序。

假如再server1上先创建1.txt,再创建2.txt 。那么每台server创建文件夹的顺序都是一样的

* 1. 原子性：要么成功（成功的标准半数server节点成功，只要半数节点成功后面的节点会自动同步），要么失败
  2. 实时性：数据的实时

1. Zk中的集群角色（leader、follower、observer）： 分布式集群中：server承担着不同的角色， 角色承担不同的任务

如果访问量比较大的话，才会新增一个观察者的角色（提升项目的 非事物处理的能力）



Leader：处理事务性请求、调度follower 进行事物操作（对文件增删改）

（1）为了确保数据的一致性：请求分为事物性操作（对数据进行增删改） 和 非事务性操作

（2）当事物性操作发送请求给follower时，此时请求会被转发给leader，由leader进行全局的调度 。当发多个请求时并且是有顺序性的 ，leader还会对请求进行编号来确保数据的一致、顺序性

Follower：处理非事务性请求、参与集群中leader的选举，如果leader挂了，会再次选举一个

处理客户端的非事物性请求 (接受请求和选举)

Observer：处理非事务性请求、不会参与投票 （主要用于提升项目的 非事物处理的能力）

观察集群中节点的状态，并且实时同步数据。专门用于接收非事物请求，事物性请求转发给leader ，提高读的性能

1. 集群搭建 ： 由2n+1 奇数台集群（follower选举时，半数以上的投票才会产生leader） 偶数也行 至少有两台

服务器多 也只能提高读的性能，但是会降低事务性操作的性能（因为要同步数据） ，写只能由leader完成

增加Observer 能提高读的性能 ，写的性能影响也会稍微较低 但是没增加Follwoer大

1. Conf文件夹下zoo.cfg 文件 拷贝3 份
2. 配置： server.1 = ip地址:心跳端口:选举端口
3. 配置：myid文件的输出路径
4. 创建myid文件目录（创建名为myid无扩展名的文件，文件内容对应服务器的序号）
5. 再bin目录下拷贝3份zkServer.cmd 并配置zoo.cfg地址
6. Zookepper 数据模型

（1）Zk 存储数据的方式 跟操作系统存储文件的方式是一样的。 树形（节点）存储结构。 每一个存储节点 称为zNode

并且 每个节点 跟 常规的存储方式 有着不同之处

（2）Znode特点：

——每一个 节点都 兼具 文件 和 目录的 特点 （文件下还会有子目录）

——具有 原子 性操作

——大小有限制 kb 为单位

——文件的 项目引用 必须是 绝对路径 从根节点 / 开始

——序列化特性 ：如果再创建节点时指定此特性，zk会自动给此节点名字后追加一个序列号（序号是10位的数字从10000000000开始，如果有节点被创建则加1） 那么通过此特性就可以 判断 znode节点被创建的顺序

总结 每个Znode由3部分组成

Status 状态信息 ：描述该znode的版本和权限等信息

Data 数据 ：可以存储数据

Children 该节点如果关联目录的话，还可以关联子节点

1. ZNode的节点类型 ：　临时节点　　、　　永久节点

临时节点　：该临时节点 依赖他的会话（server与client的连接） ，一旦client断开与集群的连接 此节点就会被删除

注意：由于临时节点的特性 ，在该节点下不允许 创建子节点 （尤其是永久节点）

永久节点　：一旦该znode节点被创建 ，则会永久存在 除非手动命令删除

注意 ：永久节点不能修改 ， 只能删除 从新创建

1. 节点属性 （根据属性来判断 该Znode的相关信息 及 子节点相关信息 ）
   1. dataVersion数据版本号 ：每次节点进行 set操作 ，dataVersion值就会加1 。根据值来判断当前数据的新旧程度。 有效避免 数据更新时 出现先后顺序的问题
   2. cVersion 子节点版本号 ：当一个节点下的 孩子节点 发生变化 cversion的值就会加1 ，因此可以判断该节点下 的子节点 更新的 频繁程度
   3. ACLversion ：ACL 版本号 （了解） 安全认证 ，当其他客户端 访问zkserver ， 修改节点需要有认证

Zxid ：事物id ： zk 每进行一次 事务性的 操作 ，都会产生一个 全局的 唯一性的事务id（由leader 分配 id是有先后顺序的 ，因此 可以通过 事务id 来判断 事务性操作的 顺序 ） 这也是为什么事务性操作由 leader来完成

* 1. cZxid ：创建时的事务id
  2. mZxid ：修改时的事务id
  3. cTime ：节点创建的时间戳
  4. mTime ：节点修改的时间戳
  5. Ephemeralowner [ɪˈfɛmərəl] ： 如果该属性值 不为0 ， 则为临时节点 。 值为 临时节点的会话 sessionId

一旦客户端断开连接 ，该节点会自动删除

如果为 0 则为永久节点 。

1. 自带的zkShell 客户端 连接工具 操作 zk 集群
   1. 连接zk 集群 zkCli.sh -server ip地址 （linux） zkCli.cmd -server ip地址(windows)
      1. 如果不加 -server ip 客户端会到本机的默认2181端口 查找是否有zk集群 服务

如果加 则会到远程的服务去 连接 zk 集群服务

* + 1. 节点的增删该查
       1. Create [-s] [-e] path data acl 创建节点
          1. -s 可选 是否开启序列化节点 （默认不是序列化）
          2. -e 可选 是否开启临时节点 （默认永久节点）
          3. Path 创建节点的路径
          4. Data 数据
          5. Acl 权限控制
       2. 查询
          1. Ls path 节点下的所有子节点 （只能查看一级子节点）
          2. Get path 显示属性信息 和 保存的数据 和 子节点
          3. Ls2 path 只显示属性信息
       3. 更新 节点一旦被创建， 类型不能改变 但是数据是可以变动的 set path data version
          1. Date 修改的数据 、 version版本号 可以不指定
       4. 删除
          1. Delete path 如果删除的节点 存在子节点，不能进行删除操作 必须删除子节点
          2. Delete path rmr 强制递归删除

1. 常用命令 Quota : 配额、指标、额度
   1. quit 退出客户端
   2. setQuota 对节点增加限制 setquota -n | -b val path （注意：n | b 不能两个同时使用）
      1. 参数n：限制子节点的最大个数
      2. 参数b: 限制数据值的最大长度
      3. 参数vla：子节点最大个数 或 数据值的最大长度  
         注意 ： I、当对此节点设置最大子节点时 ， 节点本身也会算进去。

如： 对A 节点 的子节点设置为3 ， 此时A 节点自身算是一个节点 。 A节点下最多还能添加2 个子节点

II、如果子节点 个数超过 设置节点的 大小 ， 也不会报错 只会在日志中打一个警告

* 1. Listquota path 查询该节点是否进行 quota限制
  2. Delquota -n|-b path 删除节点的 quota限制 count的值为-1 表示大小无限制
  3. History 列出 历史操作命令
     1. 历史操作命令 会按照序号 显示出来
     2. 如果想重复执行 以上命令 redo 序号

1. Zk watcher 监听机制

Zk提供了分布式数据发布 / 订阅功能 （如我定了天气预报的业务 ，监听 监听到 那么当天气预报出来时 就会自动的吧天气预报的信息 推送给我） 分布式的服务 也有这种情况 ，为了解决这些类似的事情 zk 引入了wather机制。 zk允许 客户端向 服务端 注册一个监听 ， 监听感兴趣的事件（如：节点的创建、节点的变动、删除节点等） 一旦监听的事件发生了 ，那么设置的监听就会被触发 。 那么服务端 就会以 事件的形式 通知给客户端 来完成分布式数据发布 / 订阅功能

1. watcher 机制的实现过程
   * 1. 注册监听事件
     2. 一旦事件发生， 监听触发
     3. 服务器以事件的形式告诉客户端，客户端能够回调watcher 获取事件情况
2. watcher 机制的特点
   * 1. 一次性触发 ：同一种情况触发只实现一次 ， 如果数据再次发生变化则不会触发，监听会失效
     2. 事件的封装 ：zk 使用watcherEvent对象来封装服务端事件 ，并传递给客户端 。watcherEvent对象包含3个属性
        1. keeperStauts 通知的状态
        2. eventTpye 事件类型
        3. Path 事件路径 （当前事件发生在哪个路径下）

客户端那么客户端 可以 到watcherEvent中 根据 这三个属性 获取相应的

1. Event 异步发送 ： 事件 由 服务器 异步 发送到 客户端
2. 先注册 再触发 ：想要事件能够触发 必须先注册

1. Shell 客户端 实现 watcher机制

技巧 ： help 会列出所有 的命令， 其中带 watch 是跟 watcher机制有关的

* 1. 客户端1， 设置 节点变动 监听 : get path watch
  2. 当其他客户端操作使其节点数据 发生变化时
  3. 此时 客户端1 就会接收到 watcheEvent
  4. 如果再次 改变此节点数据 ， 则 不会发送通知

1. Java API 操作zk 集群

通过封装的 Zookeeper实例的对象可以连接zk服务器，并且操作znode节点

1. 、创建节点 create -e ：创建临时节点 -s 创建顺序节点
2. 获取节点 getData
   1. 参数1： path
   2. 参数2：是否对此节点设置监听 ture 监听此节点 ，false 不监听此节点
   3. 参数3：状态 可以为null
3. 修改 ：set Data
   1. 参数1 ：path
   2. 参数2：修改的数据
   3. 参数3：修改的版本 ， 对应当前版本 。 -1 表示版本维护交给服务器
4. Zk 的选举机制 ： 如何判断 哪台服务器做follower 哪台服务器做leader
   1. 采用fastLeaderElection采用投票数大于半数则胜
      1. 每个人都给自己投票
      2. 投票数过半 选举结束
5. zk数据发布 与 订阅模型 ： 全局的配置中心（全局管理的数据）既： 发布者将数据发布到 zk全局的配置中心 的节点上，供订阅者 动态的订阅数据 ，(多台客户端同时订阅)订阅者也可以设置监听 或更改这些数据 。实现配置信息的集中式管理 和 动态 更新

分布式的电商 ， 订单、购物车等模块不在同一个服务器 来实现 连接数据的数据的发布 和 订阅

1. 再服务启动的时候 连接zk 查询配置文件信息 zk.getData(“path”,true,null); 同时对此文件设置监听
2. 当配置文件改变时 ， 触发监听 通知客户端 ， 然后再次对文件 设置监听

这样就能完成 全局的消息发布和订阅功能

1. Zk 命名服务

在zk中，通过命名服务 客户端应用能根据指定的 名字 来获取资源 或 服务地址。 被命名的实体可以使 集群中的机器，可以是远程对象等等

dubbo通过使用zk 来作为其命名服务 。 因为zk中路径是唯一的，不存在两个一摸一样的路径， 当dubbo发布服务（ip路径或地址）的时候，此服务就会被注册到 zk中 ，形成 一对一的 关系 可以根据一方找到另一位方 从而完成服务的注册 。

1. Zk 分布式锁

类似多线程 ，多个线程操作同一资源时 可能会导致数据的不一致 。 分布式服务也会出现这种情况

锁服务分为两类 ： 保持独占 和 控制时序

持独占：只能有一个 客户端 能持有这把锁 ，当多个客户端去 获取同一个资源时 ，zk集群强制要求所有客户端先去zk上指定的目录创建节点（节点要求是一个非序列化的临时节点） 谁创建 成功 ，谁就可以获取访问数据的权限 当操作完成，客户端断开与zk的连接 节点就会被删除，其他客户端如果也需要操作这个文件的话，就监听上个文件创建的节点 ， 一旦断开连接 节点删除，其他客户端就会收到通知 然后再在这个文件下创建相同节点 ，然后就获得了 访问资源的权限

这样就 保证了 只有一个 应用能获取 一个资源的 能力

控制时序: 利用序列化的特性 。客户端创建 序列化的临时节点， 这样创建的节点 就会有一个顺序 。 那么可以根据这个序列号 进行优先级访问

分布式 ：面向机器，不同模块部署在不同的机器上

微服务 ：面向业务，对业务功能模块进行拆分

1. 分布式面临的问题 ：
   1. Session共享问题 解决：SpringSession 、redis 、tomcat之间的sesion 复制
   2. 分布式配置中的配置管理 解决：zk、disconf(百度开源）、diamond（阿里开源）
   3. 分布式事务

如下订单的问题 ： 1、rpc调：查 2、rpc调：保存订单 3、rpc调：减库存

而订单系统和库存管理系统 又是分布在不同的服务中 如何确保事务性操作

* 1. 分布式锁

当库存为 1 的时候 ， 有多个用户买这件商品时 ，如何确保不出现买错的现象

当下单后 需要将此商品锁起来

解决：1、可以直接在mysql中锁表 2、zk实现分布式锁 、redis实现分布式锁

* 1. 分布式定时任务 ： 多台机器定时跑定时任务 解决 ： zk 、XXL\_JOB、Elstic

1. CAP 理论：任何一个分布式系统都只能满足 一致性 、可用性 、分区容错性 这三特性中的两项

无论 redis 、zk 、还是自己做的系统 ，拿redis来说

* 1. 强一致性 ： 当 数据发生改变 所有的redis立即都会发生变化 （不能有延迟）
  2. 可用性 ： 当其中的一台服务挂掉 redis服务还可以用
  3. 分区容错性：即使其中有两台服务之间的网络断掉 仍然要对外保证 一致性和可用性

这一点的保证是需要基础设施的稳定来支撑的 （提高网络性能和稳定性）

目的就是 让一致性的效率更高

CA ： 严格意义上是不能满足的 如果两台服务离的较远，且网络较慢 这样不能满足可用性，或者降低

CP 可以 ：牺牲可用性 ，当修改数据时 数据且未同步期间 不允许使用此功能

AP 可以 ：牺牲一致性， 当修改数据时 为了让系统能够使用 但是就会有数据不一致的风险

1. BASE理论是对CAP的延伸 分别代表 ：BA 基本可用 、S 软状态 、E 最终一直性
2. ZK 可以对自己的数据进行管理（增删改）和监控，通过数据的变化来达到对 分布式中数据的管理（类似数据库） 目的就是为了 解决一致性
3. 节点类型
   1. 永久节点
   2. 临时节点
   3. 顺序节点
4. Zk的 客户端与服务断开 的session问题 ：当客户端与服务端建立连接时 会创建临时节点（session），然后这个临时节点就开始倒计时 ，另外每当连接的心跳刷新时 倒计时也会刷新
   1. 当 客户端 直接与 服务断开 临时（sesion）不会销毁 而是进入倒计时状态
   2. 当 使用命令 如 quit 断开时 ，临时节点（sessino）会被直接删除
5. Curator 框架 如何实现 zk的watcher机制

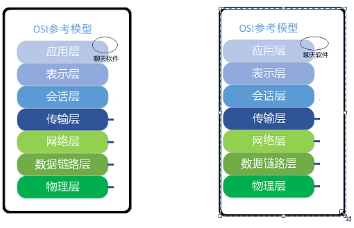


1. 现象 ： 当client 和server 时间由于网络不稳定 ，断开一段时间（超过了session的倒计时） 当网络恢复时 重新连上server时 ，临时节点（session） 就已经发生了变化，指点的的一些设置如 ：设置的监听节点操作、 临时节点的属性都会发生变化 Curator 也可以解决这一现象 ，即使网络断大于sesion倒计时 ，当重新连接时session也不会发生变化



**通信**

1. 网络通信 ： 利用编程语言 在 多台 计算机（服务器）上 实现信息 交互
2. 局域网或互联网网络通信的三个要素
   1. ip:Ip 和 网卡 对应 ，一台计算机 有多个 网卡就有 多个 ip（唯一标识）
      1. Ip分两个版本 ipV4 和 ipV6 表示方式两种：
         1. 32位4字节 例：010101
         2. 点分十进制 例：192.168.196.4
      2. Ip地址 分为 公网ip 和 局域网 ip
         1. 局域网ip 理论上能最多能组建 255台 。
         2. 192.168.1固定 后面 0-255
      3. Java如果想操作 本机的ip 可以通过 netAdderss类
   2. 端口：计算机上的每个软件对应一个端口号
      1. 每一个软件 对应 一个进程 ， 端口号就是 进程标识
      2. 一台计算机端口号 0-65535 ， 0-1024不能使用， 计算机系统会使用这些端口号
   3. 网络通信协议：传输协议通信双方 都遵守 传输数据的 格式
      1. 常见协议 ：UDP 用户数据报协议 、TCP 传输控制协议
      2. UDP：
         1. 不需要建立连接 （不可靠）
         2. 传输数据需要打包 （60K以内）
      3. TCP
         1. 需要建立（通过3次握手建立）连接,然后产生传输数据的通道
         2. 不打包 ， 不限大小
         3. 效率较低
3. 网络模型 ：研究计算机之间 是如何 实现 信息通信的 。 通常讲 通讯划分为 7 层



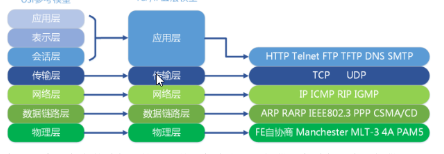
以两个两台计算机上的 两个聊天软件的通信为例

当A使用应用层发信息时 ， 信息不会直接到B 。 而是依次往下进行封装 直到到 物理层通过 光纤将信息 以

0101的方式传输到B。B的物理层接收到 进一步去解析拆分 0101 然后将信息编码转换传到应用层

也就是说 再网络数据传输过程中， 数据经历的 逐级封装 和 逐级拆分的过程。

七层详解：



1. Socket机制 网络编程又称 socket编程 soket里描述了 ip和端口。 本质 ： 数据再两个socket之间传输
   1. 两方或多方 都必须有socket
2. 数据再 在socket之间通过 1 0 传输
3. Socket封装了 ip 和 端口 ， 基于socket 再数据传输协议下就能实现 数据再网络间的 传输
4. Java 实现 socket 编程 ，基于UDP编程实现对象 ：DataGramSocket
5. 基于UDP编程实现

特点 ：　无需建立连接，需要打包

* 1. 创建DataGramSocket
  2. 打包数据
  3. 发送数据
  4. 释放资源

1. 基于TCP编程实现

特点需要建立连接　，　不需要打包 （客户端）

1. 创建 socket客户端 java Socket实现 ， 实现连接
2. 获取传输通道 输入流
3. 发送数据 输出流
4. 关闭通道

服务端

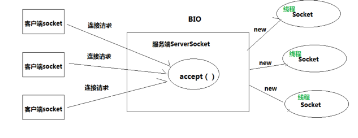
1. 创建Serversocket 服务端接收数据 监听本机的一个端口
2. 获取输入流
3. 读取数
4. 释放资源

IO（input 、 outPut） 通信模型 ： 网络通信的本质 是两太机器间 数据IO 之间的数据交互 。 数据交互时 可能就 会出现异常 因此就出现了 有阻塞 和 非阻塞的 问题

无论是 硬盘上 的IO 和 网络间 的IO 都是一样的 。 原因 IO 是基于操作系统 之上的 的 数据传输。 都需要操作系统发起 ， 假如两个 计算机建立了 连接 ， 其中一台服务端 会 一直 询问 操作系统， 是否有 客户端 访问他 ，如果有 系统来创建一个 socket对象 来接受数据 。

IO无论是 本地数据传输 还是 网络数据传输 都存在BIO 模式

BIO阻塞模式（jdk1.4之前） ： 如java 的 socekt 编程 ， 当serverSockrt服务端被创建 他的 accept（等待客户端） ，和 read （读取客户端信息） 就一直处于 阻塞状态 。 由于处于阻塞状态只能等待某一个客户端的访问 ， 如果想 同时可以处理多个任务 ，那么就需要多线程 。来一个客户端 就new socket对象 ，然就 使用不同的线程分别处理不同的事情



虽然可以利用多线程去处理不同的任务 ，但是 accpet 也是一个阻塞的点 ，所以多线程下的 阻塞模IO 只能优化部分效率

NIO阻塞模式（jdk1.4之后） 效率高

NIO 三大核心部分 ： channel 通道、buffer 缓冲区、seslector 选择器

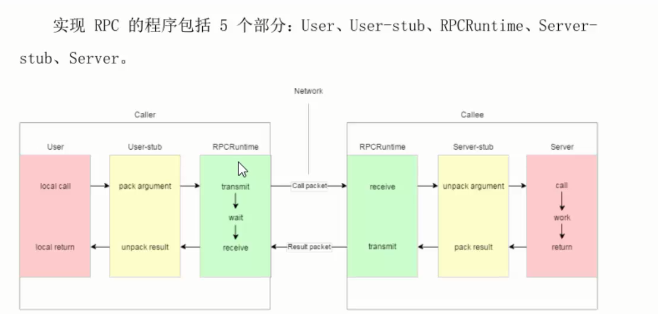
1. Buffer ：容器对象 用于存放需要操作的数据 。 在nio下 无论是写还是读 都要跟buffer交互
2. Channel ：通道类似 输入流、输出流 （单向） 但是 通道中的数据传输是双向的 可以同时进行读写、单 向读 单向写 。通道中交互的 是 一个个 buffer对象
3. Selector : 服务端 注册 channel的地方 ，如果有客户端 连接服务端进行数据交互 ， 交互前需到 selector去注册 ，然后就会不断的 轮询 服务端的 channel ， 一旦有channel就绪 ， 就直接打开通道 进 行数据交互

1. RPC 远程过程调用协议 ： 通过网络 连接向远程的计算机 去调用远程计算机上的 程序，函数等 。看上去就好像调用本地程序一样 而不需要了解底层协议的 技术。

特点 ：

1. 是一种 不需要了解 底层协议的，调用远程程序的 协议 （dubbo就是RPC的实现 ： 可以远程调用服务）
2. 网络协议 和 io 模型 都对其透明 ，rpc不关系底层的东西 。 基于rpc 自己的协议来实现 远程调用
3. 信息格式对rpc 透明 ， rpc实现程序调用 ， 参数 后 返回值 必须按照一定的格式书写
4. 跨语言能力 ： 调用远程程序，不需要关心 远程的程序是什么语言实现的

RPC 原理 ：



两个红色的 是调用方 和 被调用放 。 如果实现远程调用 中间必须 依赖 PCR 实例

如何 看起来向调用本地 服务一样 ：

当 user发起请求 先是 stub本地存根 接受到请求 并将参数打包 ，发送个 RPC 实例 。 rpc 进行网络传输 。 另一方的rpc就能接受到请求 。 解析数据 ，然后再由 server 服务端的 存根stu 调用程序。如果程序有返回值 ，还要对返回数据进行打包 返回 serverRPC ，再客户端 的 RPC ,然后到 客户端的 stub ，然后讲结果解析 返回

**ActiveMQ**

1. JMS(JAVA MESSAGE SERVICE) :java EE 消息服务规范接口， 用于两个程序之间 或 分布式 系统中 发布消息进行异步通信

JMS提供了应用之间的异步通信机制，当异步发送消息时，客户端不需要等待服务端处理消息结果

1. 适用场景

有A/B 两个应用程序 ， 他们之间也无调用关系 ， 更不知道各自 方法内部的实现 。

B可能对 A 程序的信息感兴趣 。更有可能N个B 对A都感兴趣 。

JMS 就是解决这类事情的, A 发送消息 ，B不需要在线， 当B上线时 jms消息服务能保证发送消息给A

1. 特点 ：

异步 :客户端不用主动发送请求 获取消息 ， 消息服务会自动推送

可靠 ：保证消息只传送一次， 避免消息重复 创建 。

Jms 常用类 ：

1. connectionFactory 连接工厂 接口，定义创建连接规范 ，连接provider服务
2. Connection 连接接口 封装连接信息
3. Session（类似hibernate的session）会话上下文，用于创建队列queue、

创建消息mesage、创建生产者producer、创建消费者comsumer等操作

1. Destination 消息发送和接收的 目的地(保存消息的地方)
2. Acknowledge 签收 ：当消费者接收到消息后 应基于中间件响应 ，当session提 交时会签收
3. Client 用于收 、 发消息的 java程序应用
4. No-jms client 使用jms provider 应用替换jms API实现收发消息的功能 （一般用 于扩展）
5. Administered object ：预定义JMS对象 ，通常在provider 规范中定义 提供给JMS 客户端来访问 ，如 connectionFactoey 、destination

Jms对象 ：（1） Destination：消息被寻址、发送以及接收的对象

p2p queue作为Destination

Pub/Sub topic主题作为Destinatio

1. Provider：实现jms的消息中间件服务（比如 ActiveMQ ）
2. Messsage： jms消息 ，jms消息由三部分组成
   1. 消息头 消息头属性都应有get/set方法
   2. 消息体
3. Producer：消息生产者，创建和发送jms消息的应用
4. Customer 消息的消费者 接受和处理jms消息的应用
   1. 同步消费 ：如果有消息就处理 ，没消息就阻塞 直到消息到达
   2. 异步消费 ：客户端注册监听器，当消息到达的时候 mq会回调监听
5. Domains: jms 规范定义的 消息传递域 ； p2p 和 pub/sub

****点对点（队列）消息传送模型 P2P****

程序由消息队列，发送者，接收者组成。每一个消息发送给一个特殊的消息队列， 该队列保存了所有发送给它的消息(除了被接收者消费掉的和过期的消息)。

特点 ： I、每个消息只能有一个消费者 （一个队列可以保存多个消息）

II、生产者和消费者面向的都是对列 .当消息发送者发送消息的时候，无论接收者程序在不在运行，都能往对列中存消息 ，无论生产者是否运行 只要对列中有消息消费者就可以取消息；

III、当接收者收到消息的时候，会发送确认收到通知,然后消息从队列中删除

III、消息发送者和消息接受者并没有时间依赖性

****发布/订阅消息传送模型**** Publish/Subscribe

发布者发布一个消息，该消息通过topic传递给客户端 。在这种模型中，发布者和订阅者彼此不知道对方，是匿名的且可以动态发布和订阅topic。topic主要用于保存和传递消息，且会一直保存消息直到消息被传递给客户端。

特点 ： I、一个消息可以传递给多个订阅者

II、发布者和订阅者有时间依赖性，只有当客户端创建订阅后才能接受消息，且 订阅者需一直保持活动状态以接收消息。 如果不订阅就不能接收到消息

III、为了缓和这样严格的时间相关性，JMS允许订阅者创建一个可持久化的订阅。 这样，即使订阅者没有被激活（运行），它也能接收到发布者的消息 。非持久化订阅 ，即使消费者订阅 如果离线 也接受不到消息

IIII、消息发送的方式， 是将消息复制 然后发送给不同的接收者

1. JMS消息的组成
   1. 消息头 ：包含识别信息 和 路由信息

消息头中的属性

* + - 1. JMS Destination 目的地 主要指queue 和 topic ，根据createQueue和createTopic 自动分配 开发者设置
      2. JmsDeliverymode 传送模式 开发者设置（可选） ：

1.持久模式 为了保证数据一定可以发送或接受 数据需要被持久 化 ，即使即使发生故障 也能接受到

2 .非持久模式

c.JMSExpiration 消息过期时间 ，发送 send方法 可以设置过期时间如果timeToLive 为0 表示永不过期 开发者设置（可选）

d.JMSpriority 消息发送优先级 ，发送 send方法 可以设置优先级

0 - 4 普通消息 ，5 -9 加急消息 值越大 ，优先级越高 默认值是 4 ， 自动分配 开发者设置（可选）

e.JMSMesageId ： 每个消息的 唯一标识 自动设置

f.JmsTimestamp : 当provider 调send方法时 的时间 和 接收者接收到时间 的差值，自动设置

g.JMSCorrelationID :用来连接到另一个消息，用于 回复消息时 连接到原消息

h.JMS replayTo :回复消息的地址

可以通过replayTo(Destination对象);getreplayTo()接收消息体 返回 Destination 还可以继续发送消息

I.JMSType :消息类型的识别符号 开发者设置

J.JMSDelivered 重新投递 ：如果需要消息由于其他原因未被签收，可以设置消息从新投递

JMSDelivered = true ，JMSDelivered = false

* 1. 消息体 ：就是针对发送的数对创建不同的消息体 。JMS API定义了5种消息格式，也称消息题 可以使用不同形式发送 接收数据

注意：再接收者接收消息时 接收到的消息应转换成对应的消息格式

* + - * 1. TextMEssage 普通文本信息
        2. MapMessage map 适合发送对象
        3. ByteMessage 数组
        4. StreamMessage 流式消息
        5. ObjectMessage 对象

* 1. 消息体属性
     + - 1. 发送消息时messageSetXXX(); 为待发送的消息 设置附加属性
         2. 接收消息时messageGetXXX(); 载接收消息时，获取相应的属性

1. JMS机制
   1. JMS机制可靠性：防止发送重复发送 同样的消息

消息接收确认 :消息确认之后才认为是被成功消费了 。如果不确认中间件 下一次还会发 同样的消息 。 消息何时被确认 取决于 创建会话时的应答 模式 。

消息成功消费包含3个阶段 ：客户接收消息、客户处理消息、消息被确认

* + - 1. session.AUTO\_ACKNOWWLEDGE在事务性会话中 当事物被提交时 session.commint()，确认默认发生。
      2. Session.CLIENT\_ACKNOWWLEDGE 通过消费方 对象调 acknowledge 方法手动确认消息

注意： 由于确认是在会话层进行的 ，一旦确认 会将之前所有已经消费的消息确认 ,一旦使用 这种方式 ，就将是否支持是否设置为false 和 不用session.commint() ，因为session.commint()是配合 session.AUTO\_ACKNOWWLEDGE使用

* + - 1. session.DUPS\_ACKNOWWLEDGE 迟钝的方式进行消息确认 ,性能稍好 。但是特殊情况 由于是延迟确认消息 ，就会存在 重复 收到消息的情况 ，比如 provider 服务突然挂了 ，重新开机 由于消息 还未确认 还会继续发送 那些未 确认过的消息 。

将JMS消息头 JMSRedelivered 设置为 true ，来对重复发送的消息进行检测

（种方式 ，就将是否支持是否设置为false 和 不用session.commint() ）

* 1. JMS消息提交模式
     + 1. PERSISTANT持久保存消息 ，provider服务可能会故障 消息会丢失

(也是可靠性的体现)

* + - 1. Non\_PERSISTANT 不持久化消息 ，要求消费者 先在线 ，生产者再发送
  1. Session.createTemporaryXX(); 创建临时目的地 他们存在的时间只限于创 建他们的连接所保持的时间 ， 只有创建临时目的地的消费者 才能从临 时目的中取消息
  2. 持久订阅 session.createDurableSubscriber(); provider会存储发布到 topic上的消息 ，当消费者 再次连接到 topic 的时候 ，发送者就会把订阅过后 消费者没有收到的消息再次发送给 消费端 。持久订阅在某个时刻 只能有一个激活的订阅者， 持久订阅被创建后会一直保留 ，直到调用 ununsubsciber方法

使用条件

A） 发送者 ：

producer.setDeliveryMode(DeliveryMode.PERSISTENT);第一个参数

ii、connection.start() 在持久化设置完再启动

1. 接收者

I、再连接之后需要创建消费者id : connection.setClientId(“”) ，

Ii、然后创建session.createDurableSubscriber()进行持久化订阅

Iii、最后再连接再 connection.start()

Iiii、注意 ：必须先运行 消费者 将设置的 connection Id，注册到服务器 ，然后 才能实现 持久化订阅

* 1. 本地事物 在创建session时可以设置事物是否开启

注意1、 ：一旦事物开启 ，业务需求又需要 有请求回复机制 rePlay(); 才能实现的功能 是比较耗资源的，因为 再回复之前 事物不会被提交 只会阻塞挂起直到回复完毕才会 提交事物

2、生产者和消费者 不能在同一个事物中，只有生产者 提交了事物 消息才会 发送到 目的地 ，然后消费端的事物开启接收消息

1. JMS消息传送模型
   1. ****点对点（队列）消息传送模型 P2P****

不相互独立 一个点 依赖一个点

****特点：****

* + 1. 如过消费端sesison关闭之前没有 签收 acknowledge（没有commint） ，那么消息对列中的消息就不被销毁 下次还会发送
    2. 如果用户在receive中设置了消息的选择条件 ，那么不符合条件的会留在队列中，不会被接收
    3. 队列可以长久的保存消息 ，即使消费端处于离线状态 ，重新上线 也会接收到消息 。除非消息过期
  1. Publis/subscribe

Topic 类似报纸 ， 消息的发布者类似出版社 ， 消息的接收者类似 读者

这使得 发送者 和 接收者 相互独立 ， 因此一个 主题 可以有多个程序订阅

****特点：****

* + 1. 消息订阅分为 持久订阅 和 非持久订阅
       1. 只要是非 持久订阅 只有客户端是 在线状态才能接收到消息，即使上线也收不到 ， 即使重新订阅也接收不到消息 ，除非 订阅后 客户端重新发送消息
       2. 持久订阅 消费端离线 ， 在上线时也可以接收到消息 。持久订阅时 消费端向jms中间件 注册了一个省份id， jms会根据id 为这个消费者保存消息， 当客户端再次连接上 jms服务时 消息就会发送过来
       3. 如果用户在receive中设置了消息的选择条件 ,那么不符合的消息 消费端不会接收
       4. 当消息 比较重要 必须接收 就是用持久订阅， 当消息可以适度丢失 可以使用非持久订阅

1. 接受消息方式
   1. 同步接收
   2. 异步接收

---------------------------------------------------------------------------------------------------------------------------------

1、是什么activityMQ

Apache下实现 消息推送的中间件 MOM (消息中间键).

什么是消息中间件 ： 当 A 方法调用本地项目 或者 远程项目中的方法 B ， 他们之间就形成了 强依赖的 关系。 这时可以添加消息中间件， A向 中间件发送消息 ， B 去取消息 ，当取到消息 就去处理 相应的功能 。 通过中间件 进行数据的传递来 解耦合 。

本质 ： 将信息 以消息的 形式 从一个应用程序 传送到 多个应用程序

2、activityMQ的作用

用于实现 JMS 的规范，完成相应功能 。用来实现 高可用、高性能、可伸缩 和 安全的 企业级 面向消息的 服务系统

1. activityMQ的特点

(1)、完全支持javaEE jms规 范

(2)、异步：即使B程序关闭，也可以在启动的时候接收到消息 ，也就是A无需等待B的处理 （类似 手机接收短信 ，作用：提高性能）

(3)、解耦 （作用：减少分布式的 多模块之间的依赖）

(4)、能够持久化消息

(5)、支持事物

(6)、支持多传输协议

(7)、和tomcat 易集成

1. 使用
2. bin 下启动 默认端口 61616
3. 进入 ActivityMQ 管理页面 http://localhost:8161/admin/

用户名/密码 ： admin

1. 创建消息队列的步骤

A/获取jms工厂

B/使用连接工厂创建连接

C/启动连接

D/从连接创建会话 session 三个参数。参数1：是否支持事物 a/性能搞 b/如果一次性 发送或接受多条消息 失败可以回滚

E/创建 Destination （p2p创建queue，Pub/Sub创建topic）

F/创建 producer

G/producer 发送消息到 destination

H/消息端 取消息

I/释放资源 关闭连接

1. 使用ActivieMQ 构建Broker(mq服务) 实例 ，java程序嵌入broker服务

两种方式

1. 、borker服务

BrokerService broker = new BrokerService();

broker.setUseJmx(true); //启动broker JMX监控功能

broker.setBrokerName("myBroker"); //设置名子

broker.addConnector("localhost:61616"); //连接地址

broker.start(); //启动

1. 、brock工厂启动broker （读取配置文件）

String url = "properties：borker.properties";

BrokerService broker = BrokerFactory.createBroker(url);

broker.addConnector("tcp://localhost:61616");

配置文件

useJmx = true

brokerName = myBroker

persistent = false

1. spring集成ActiveMQ
2. 、添加spring提供对jms的支持

<dependency>

<groupId>org.apache.activemq</groupId>

<artifactId>activemq-pool</artifactId>

<version>5.7.0</version>

</dependency>

1. 、ActiveMQ连接池 pool

<dependency>

<groupId>org.apache.activemq</groupId>

<artifactId>activemq-pool</artifactId>

<version>5.7.0</version>

</dependency>

配置：

<bean id="activeMQDataSource" class="org.apache.activemq.pool.PooledConnectionFactory">

<property name="maxConnections" value="100"/>

<property name="connectionFactory">

<bean class="org.apache.activemq.ActiveMQConnectionFactory">

<property name="userName" value="admin"></property>

<property name="password" value="admin"></property>

<property name="brokerURL" value="tcp://localhost:61616"/>

</bean>

</property>

</bean>

1. 、配置jmsTemplate （为使用mq开发时提供方便 类似jdbcTemplate）
2. 依赖 连接池
3. 依赖 destination
4. Destination

Queue配置：<bean id="destination" class="org.apache.activemq.command.ActiveMQQueue">

<constructor-arg index = "0" value="spring-queue"/>

</bean>

Topoic配置 <bean id="destinationTopic" class="org.apache.activemq.command.ActiveMQTopic">

<constructor-arg index = "0" value="spring-topic"/>

</bean>

配置：<bean class="org.springframework.jms.core.JmsTemplate">

<property name="connectionFactory" ref="activeMQDataSource"/>

<property name="defaultDestination" ref="destination"/>

<property name="messageConverter"> <!-- 消息转化 -->

<bean class="org.springframework.jms.support.converter.SimpleMessageConverter"/>

</property>

</bean>

8、spring内嵌broker

<bean id="myBroker" class="org.apache.activemq.broker.BrokerService" init-method="start" destroy-method="stop">

<property name="useJmx" value="true"/>

<property name="brokerName" value="broker"/>

<property name="persistent" value="false"></property>

<property name="transportConnectorURIs">

<list>

<value>tcp://localhost:61616</value>

</list>

</property>

</bean>

9、可以在spring中配合消费者监听 MessageListener ， 自动接收消息 ，无需再单独的写一个消费者进行接收消息，当发送者发送消息时 ，监听将自动接收消息

（1）<!-- 自定义消息接收监听 类，实现Messagelistener接口实现onMessage方法--> onMessage()：用于自动接收的方法

<bean id="messageListenerMQ" class="com.it.listener.MQMessageListener"/>

（2）<!-- 配置接收消费者监听类用户自动接收消息 -->

<bean id="MessageListenerContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer">

<property name="connectionFactory" ref="activeMQDataSource"/>

<property name="destination" ref="destinationTopic"/>

<property name="messageListener" ref="messageListenerMQ"/>

</bean>

1. 与spring整合开发的注意事项
   1. 、发送接收消息的核心 jmsTemplate对象， 隔离了打开连接 ，关闭sesion和producer 等频繁的操作 。让开发者只关注核心逻辑
   2. 、不建议使用jmsTemplate的receive方法来接收消息， 因为是同步的 阻塞性能降低 ，

建议使用自定义监听实现messageListener接口异步自动接收消息

(3)、建议配置DefaultMessageListenerContainer监听容器 ，它允许异步缓存session还可以 根据消息的数量合理的增加监听器和缩减监听器数量

1. ActiveMQ传输协议
2. 、connetor：activeMQ提供的用来实现通讯功能的一个组件。 两种情况
   * 1. Client-to-broker 客户端连接mq服务
     2. Broker-tobroker mq服务连接mq服务
3. 、Active允许多种协议 可以连接到 mq服务，可以在 conf下的 activemq.xml中配置

<transportConnectors>

<transportConnector name="openwire" uri="tcp://0.0.0.0:61616?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="amqp" uri="amqp://0.0.0.0:5672?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="stomp" uri="stomp://0.0.0.0:61613?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="mqtt" uri="mqtt://0.0.0.0:1883?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="ws" uri="ws://0.0.0.0:61614?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

</transportConnectors>

transportConnectors标签用于定义 ： 客户端如何连接到 broker上去

支持的协议包括 ： tcp、NIO、ssl、http（s）、udp 、 vm

1. 、tcp：是连接activemq的默认通讯协议 端口默认是61616 ，依赖openwire

用于流化数据 便于数据交互 。往后是地址加端口号 再往后是连接参数 最大连接数和数据传输大小

2）、Vm:如果客户端和broker在一个虚拟机下的话可以通过vm协议进行连接

如：内嵌到java虚拟机中 ，就是通过vm协议通讯

3）、扩展 ：如果有大量的cilent连接到 broker时 ，可以是使用NIO协议场景

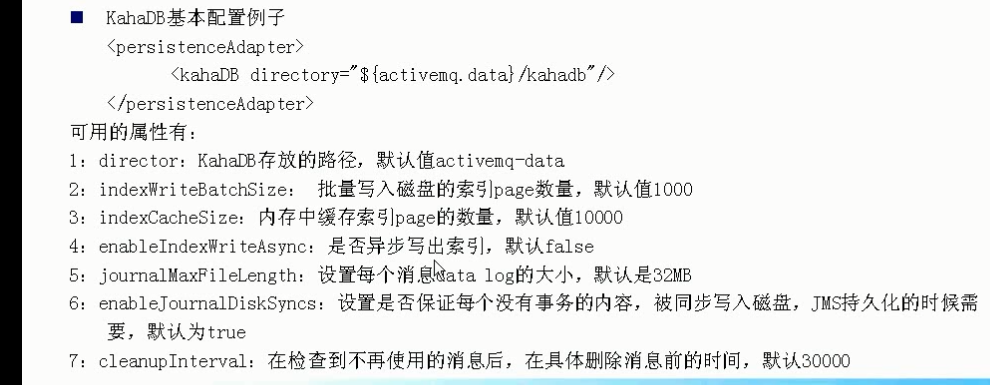
一般client 连接broker时 ，是被操作系统的线程数控制的 ，NIO能使用更少的线程数与broker建立连接

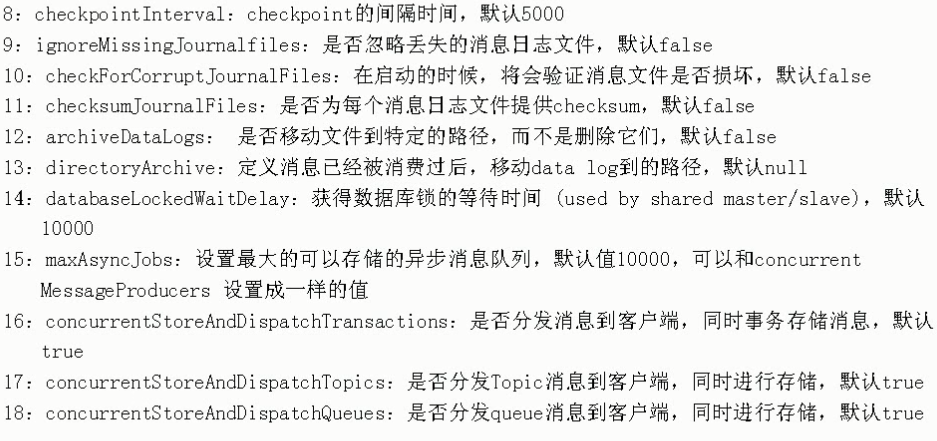
<transportConnector name=”nio” uri=”nio://localhost:端口”/>

1. ActiveMQ消息存储持久化

消息存储的几种方式 ：

1. 、AMQ消息存储， 基于文件的存储方式 是5.0版本以前默认的消息存储
2. 、KahaDB消息存储 是现在默认的存储方式（特点：提升了数据的容量和恢复能力）
   1. 可以用于任何场景
   2. 性能高
   3. 配置详情





1. 、JDBC消息存储 ，将消息持久化到表中
2. 、Merroy 基于内存的消息存储
3. 一台服务器集成多台 broker

