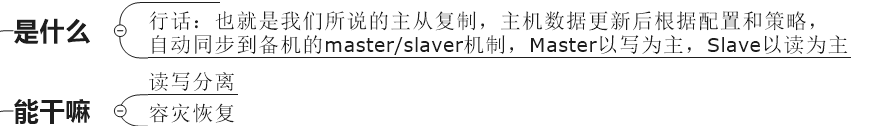
# 总结

## Redis

### 集群

为了分担些压力，将不同的数据放在不同的redis库中，每个Redis库搭建主从复制，在Redis3.0之前，没有将自动将数据按照一个算法取值，当值在某个区间时放在对应的库中，此时Java需要知道操作哪个库，便出现了代理中间件，比如tw…或者codis。在Redis3.0cluster出来后，即对key按某一算法取值，然后将数据放到对应的槽中。

### 主从复制



**主从复制的模式：**

1. **一主二仆模式**

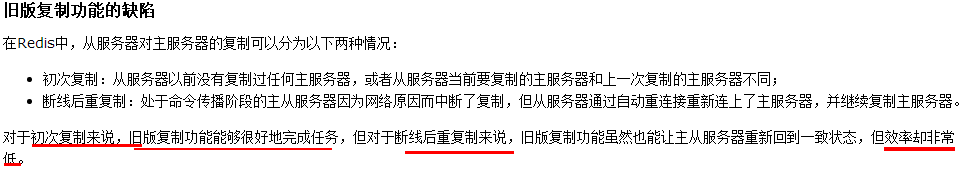
有一个主机作为master，负责写操作；两台主机作为slaver，只能负责读操作，不能有写操作。当一台从机加进来后，会向主机发送同步请求，此时主机进行备份（RDB），从机通过该备份文件进行数据同步，即会复制主机的所有数据（这个与Mysql主从复制不一样）。

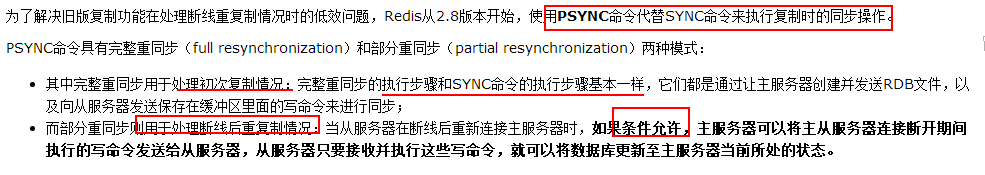
**细节：**

1. 主机宕机后，从机待命，一直等主机恢复，即从机不上位变为主机。
2. 主机恢复后，主机新增记录，从机可以复制到。
3. 从机宕机，不影响，当从机恢复后，不再是从机，需要手动再次声明为从机。
4. **从机从主机复制同步数据的原理**

从机联机后，向主机发送sync（同步）请求，主机收到请求后立即备份，发送RDB文件到从机，从机收到RDB文件后，执行文件完成复制操作。之后主机的写操作，都会立刻发送给从机，从机立即执行。

SYNC是一个如此消耗资源的命令





1. **薪火相传模式**

在复制原理中，我们发现主机不仅负责写操作，还负责将命令发送给它的每一个从机，比较消耗资源，所以，可以采用“链式操作”，即一台主机有一台从机（或者两台），然后该从机又是其它机器的主机。

这样可以分担主机的压力。对于既有主机角色又有从机角色的机器，只能负责读操作，没有写操作。

1. **哨兵模式**

一主二仆和薪火相传模式均为能解决当主机宕机，其它从机等待的问题。我们可以使用slave of no one 成为主机。但是为了自动化，引入一个哨兵，哨兵负责监控主机，会规律地发送ping命令测试主机的存活，若ping不通，此时当然一个哨兵不能下定论，因为有可能是ping的网络问题，所以可以在每个从机都安装哨兵，当有一个哨兵ping不通时，会通知其他哨兵执行ping操作，最后执行投票判决主机是否宕机。当主机宕机后，会投票选择从机上位成为主机（选取原则依次为：选取优先级靠前的；选取偏移量最大的，即数据量多的；选取runid最小的）。当从机重新成为主机后，哨兵发送slaveof命令给剩下的从机，并完成同步操作，当旧主机恢复后，哨兵发送slaveof命令，让其成为从机。

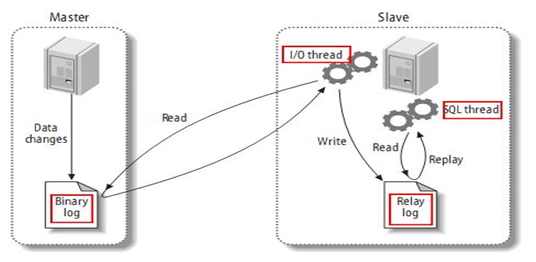
### 缓存穿透

## Mycat（mysql主从复制，分库分表）

### 主从复制

**Mysql主从复制：**

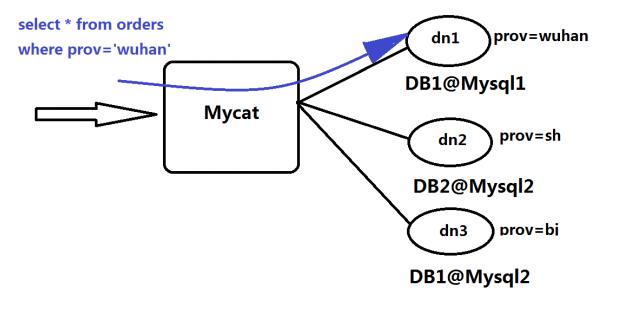
1. master将改变记录到二进制日志（binary log）。这些记录过程叫做二进制日志事件，binary log events；
2. slave将master的binary log events拷贝到它的中继日志（relay log）；
3. slave重做中继日志中的事件，将改变应用到自己的数据库中。 MySQL复制是异步的且串行化的



当实现了主从复制后，主机负责写操作，从机负责读操作，那么Java操作数据库时需要指定操作哪个库，这样很繁琐，因此引入中间件---Mycat。

Mycat屏蔽了客户端操作与数据库之间的直接关系，客户端操作时，只需要连接到Mycat即可。Mycat拦截到sql语句后，对sql做特定的分析，如分片，路由等等。然后操作真时的数据库，最后将结果返回给客户端。

配置主要是通过操作Mycat的schema.xml文件，见D:\JavaWeb\尚硅谷MySQL高级\SQL高级08&MyCat\Mycat.mmap



### 分库

同样是操作schema.xml文件。

分库原则：如果某个表访问量很大，规模很庞大，可以选择单独成库。但是当该表涉及到其他表的关联，可以使用冗余字段，冗余表等手段解决，具体分库策略还是得看业务。比如在订单表，订单详情表，订单类型表，用户表这四个表中，订单表有用户id和订单类型id，订单详情有订单id，即四表均关联，要想对这四个表分库，可以考虑将用户表为一库，与订单相关的表为一个库，因为一般查询订单时拿到用户id后才进行查询。

### 分表

类型：水平分表（按行分）；垂直分表（按列分）

考虑将大表拆分存储在不同的库中，具体拆分原则主要根据业务，拆分算法可以自定义（在rule.xml里定义）。

对于上述的订单表，用户表，订单详情表，订单类型表，现在用户表和订单相关表已经分库，考虑订单表太大，对订单表分表。

但是分表后当然要避免关联查询，所以可以针对用户id对订单表分表，这样可以保证同一个用户的所有订单在同一个表中。具体可以对用户id模二求余（则分两个表。那么模三求余就是三个表）。

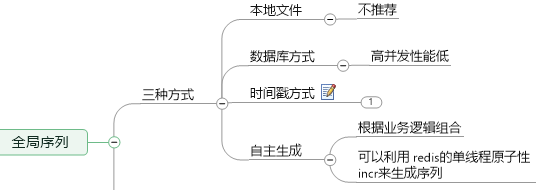
分好订单表后，订单详情表也是一个大表，因此也需要分表，但是订单详情表时关联订单表的，即订单详情表有订单id，因此不好使用某种算法针对某列来分表，Mycat提供了跨库关联，只需在schea.xml的订单表下添加：<childTable name="order\_detail" primaryKey="id" joinKey="order\_id" parentKey="id" />即可。那么对应订单表相关记录的订单详情表记录在插入时会进入该订单表的库。

E-R表的原理：

根据配置的关联id，找到主表所在的库，然后将从表插入到该库中。**疑问**：查找过程中的性能是否损耗较大？是否可以使用在订单详情表里添加冗余字段用户id，然后使用与订单表的相同策略进行分表？

最后还有一个订单类型表，这个表规模不大，与订单表有关联，所以把该表可以作为全局表。

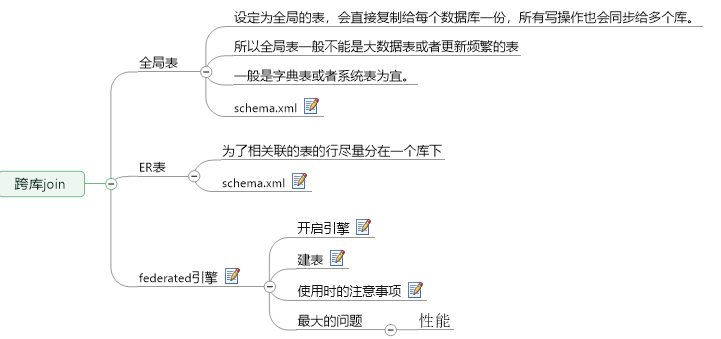
最后，为了避免不同库的相同表（分表）的主键相同，我们采用全局序列的时间戳方式来避免。



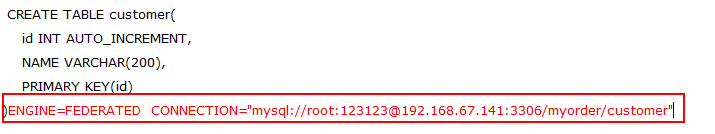
uuid的顺序不统一，即不是增加的或减少的，但时间戳是增长的，所以uuid会影响性能，不能使用。

时间戳操作语句：insert into `orders`(id,amount,customer\_id,order\_type) values(next value for MYCATSEQ\_GLOBAL,1000,103,10); （大小要是18位（BigInt类型））。

### 跨库关联表

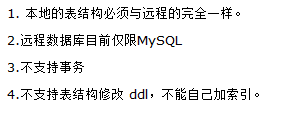


全局表和ER表（主从表）在上面已经说了，现在假如硬性要求不同库的两个表进行关联，直接将表复制多一份到另一个表有点伤，则可以通过**虚表**来解决问题。具体需要开启federated引擎，然后建表：



即表结构我有，但数据从远端来，但是表的结构必须一样，不能更改。

注意事项：



## Zookeeper

### 分布式环境的各种问题

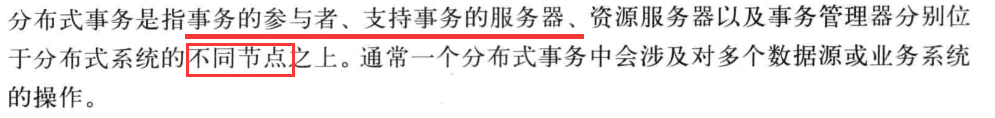
分布式的特点为：分布性，对等性（无主从之分），并发性，缺乏全局时钟，故障总是发生。

分布式的问题： 通信异常，网络分区（脑裂），三态（成功，超时，失败），节点故障。具体参考《从pasox到zookeeper》P21。

### 分布式事务

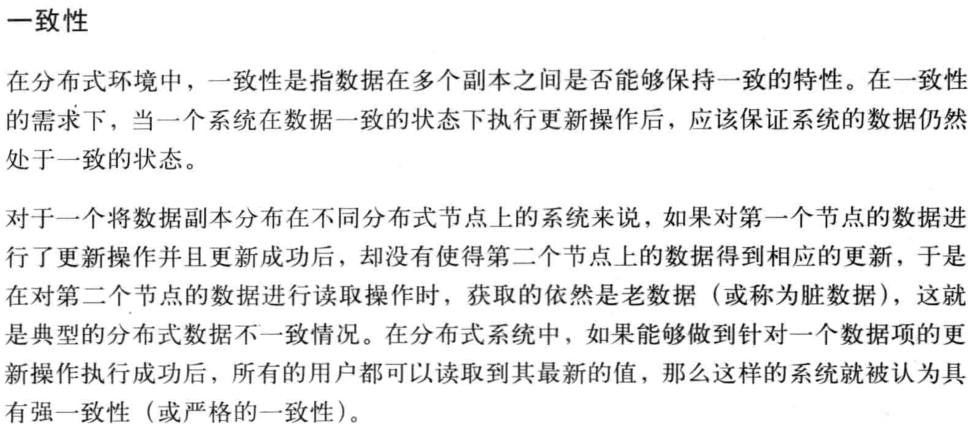
单机事务：ACID（原子性，一致性，隔离性，持久性）

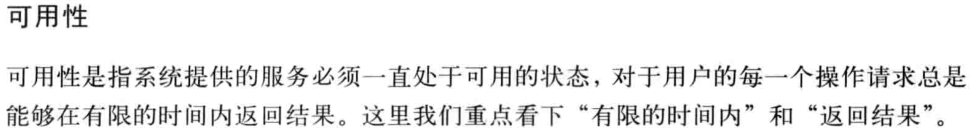
分布式事务（CAP理论）

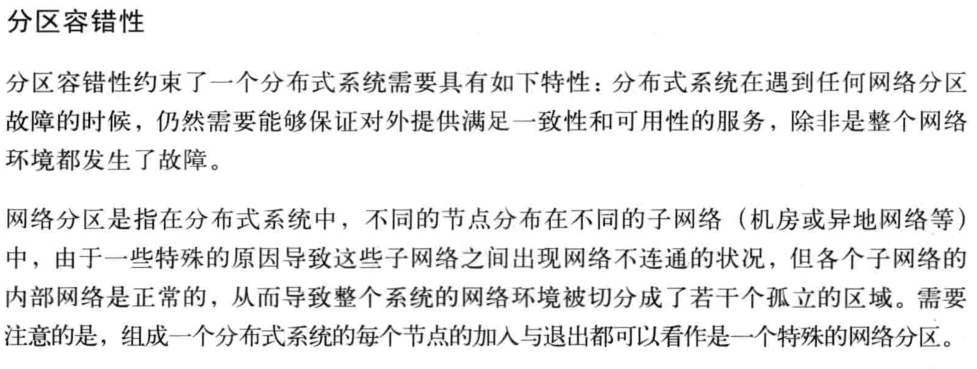


CAP理论：

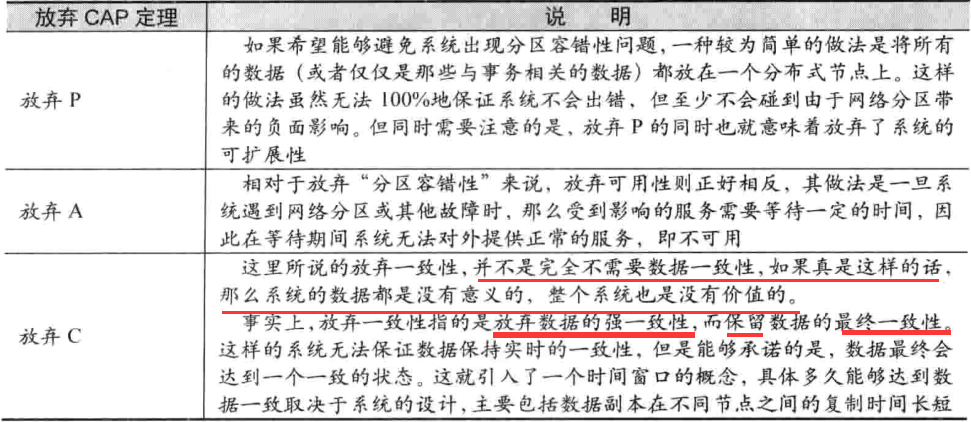
C：一致性，A：可用性，P：集群容错性；一个分布式系统，最多只能同时满足其中两项，不可能同时满足三项。







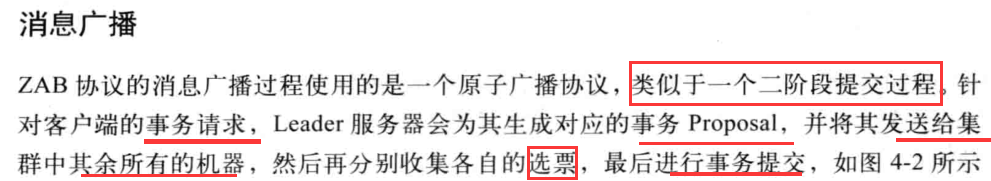
CAP理论的权衡选择:

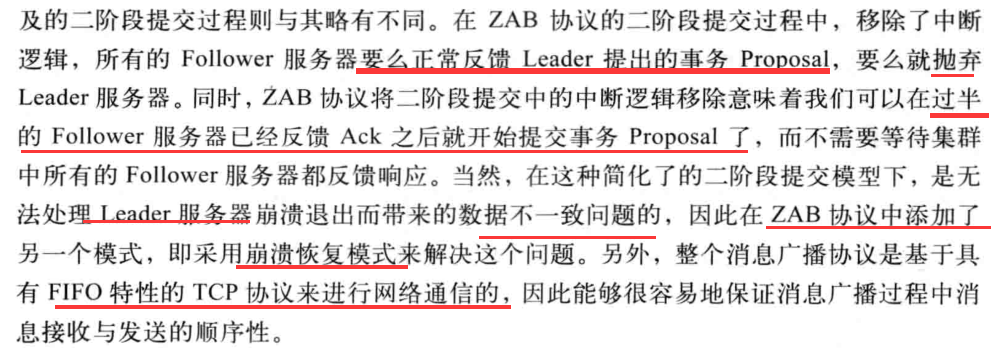


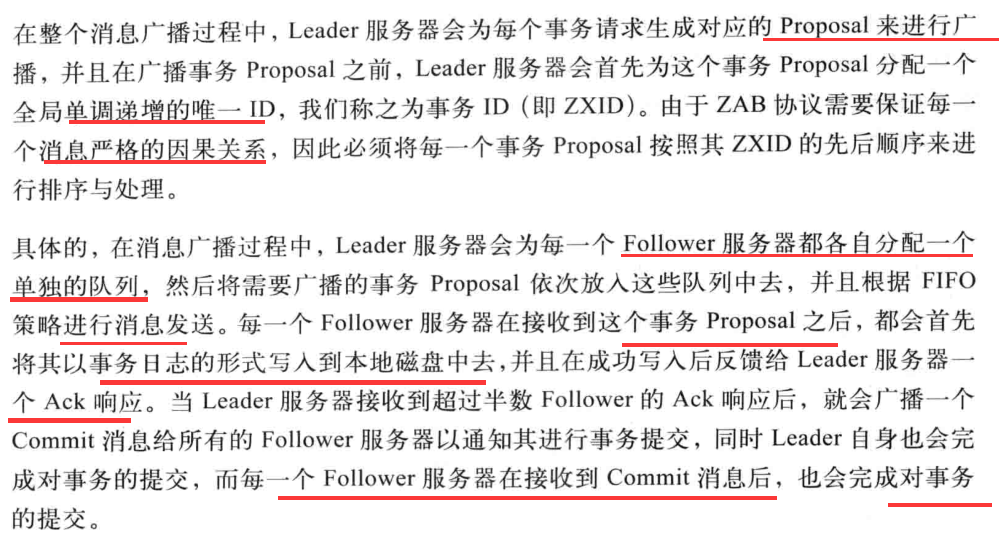
### 3、ZAB协议

ZAB协议分为崩溃恢复和消息广播模式，当集群中的leader宕机时，进入崩溃恢复模式，当选举出leader，并且集群中有过半的follower与leader完成同步，进入消息广播模式。

消息广播模式类似一个二阶段提交过程，但也有不同。

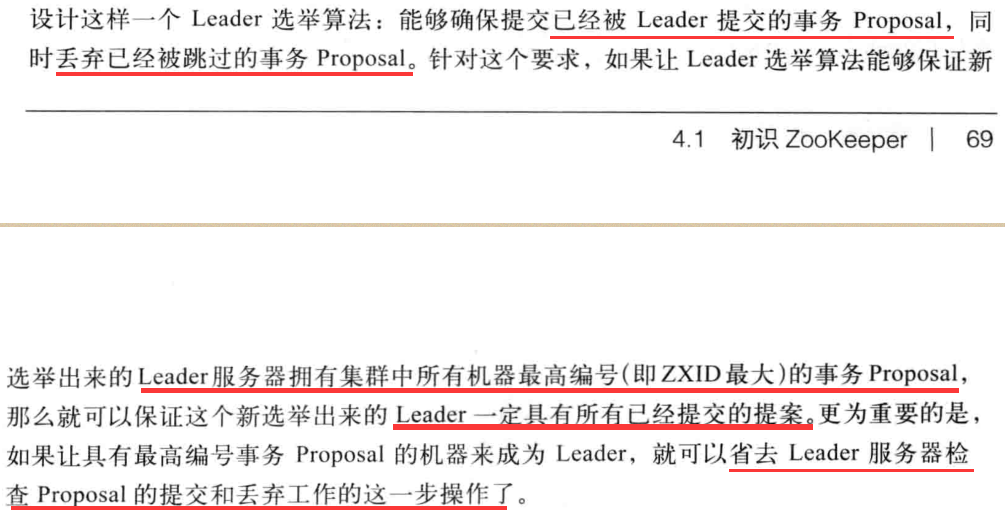


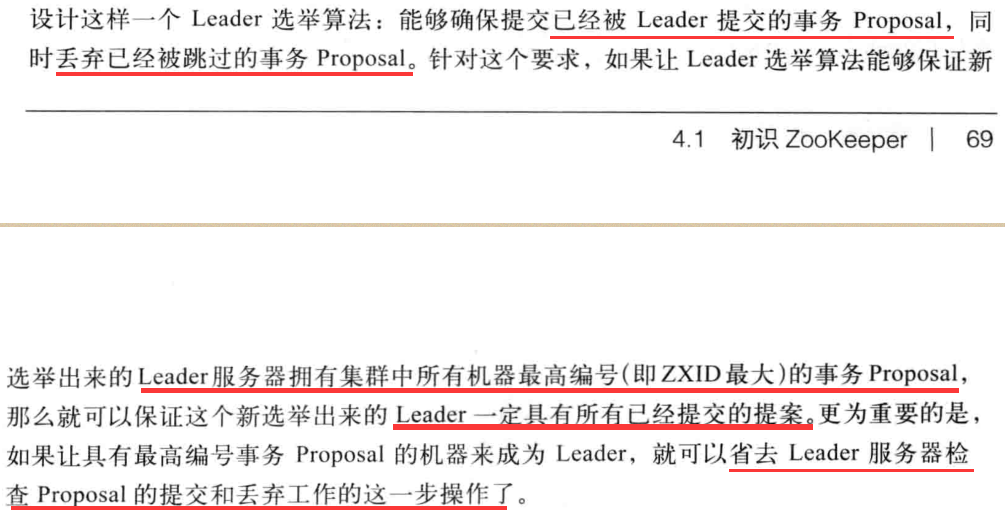




若leader失去与集群中半数以上follower的联系或者leader忽然崩溃，就会进入崩溃模式，选举新的leader。

需要完善的选举算法，保证数据一致性，zk遵循一个原则：若一个事务在一个服务器上提交了，那么必定要在其他服务器提交成功。



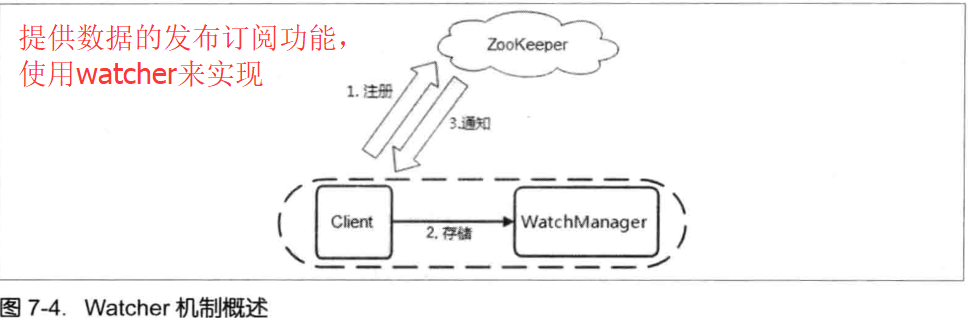


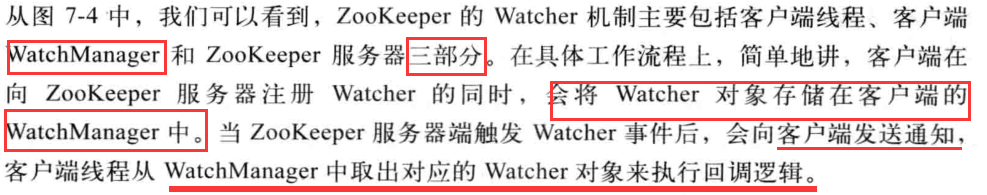
### 集群

### 4、分布式锁

### 5、分布式队列

### 6、Watcher机制





### 7、常见问题与实践

1、zookeeper集群中leader宕机后，集群选举新leader时，整个集群不可用，尝试过：leader宕机，follower的status为“not running”，过一段时间leader选举成功后就恢复正常。

## ActiveMQ

### 集群

## Dubbo

### 服务注册与发现

### 服务远程调用与本地调用

### 通信协议

默认是Dubbo协议，是基于传输层的协议，长连接，NIO方式。

其次有WebService协议（SOAP风格），也有rest风格的。

### 通信框架

默认使用Netty实现通信。

### 线程模型

### 常见问题

1. 服务提供者全部宕机，注册中心会删除对应的节点（因为创建的是 临时节点，与服务的生命周期关联），但此时消费者会**无限重连提供者**，知道提供者恢复。

<https://blog.csdn.net/luwei42768/article/details/54847427/>

## Springcloud

### 服务注册与发现

### 负载均衡

### 服务降级、熔断

### 路由

### 分布式配置

## 总体内容

### SOA架构与微服务架构

### 什么是微服务

### dubbo与Springcloud比较

### zookeeper与Eureka比较

https://blog.csdn.net/paincupid/article/details/80610441

### dubbo，niginx，Ribbon，Feign负载均衡比较

## 有机商城分析

### session共享问题



<https://blog.csdn.net/wojiaolinaaa/article/details/62424642>

<http://feitianbenyue.iteye.com/blog/2326408>

修改源码实现url重写，即把sessionId跟在url之后。

<https://www.cnblogs.com/duanxz/p/3482089.html>

redis操作类：RedisOperationsSessionRepository