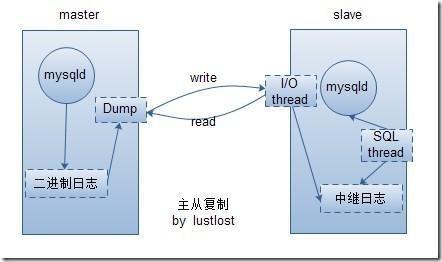
**master/slave**



**master/slave较为简单，master负责响应客户端的写请求，slave负责响应客户端的读请求**

**实现原理：**

**slave在启动两个线程，i/o线程和sql线程，master启动dump线程，每当master的数据发生改变时，master就会将对应的SQL语句存储在二进制日志文件中，slave的通过i/o线程连接master的dump线程并每隔段时间就会向master询问二进制文件是否发生改变，如果二进制文件发生改变slave的i/o线程就会去读取二进制文件中发生改变的内容存储在本地的中继日志中并通过sql线程对中继日志的内容进行回放，并记录本次读取到的二进制文件位置，完成复制**

**存在问题：**

**1、master压力过大**

**2、master宕机之后将无法完成用户写请求**

**3、节点数据可能不一致**

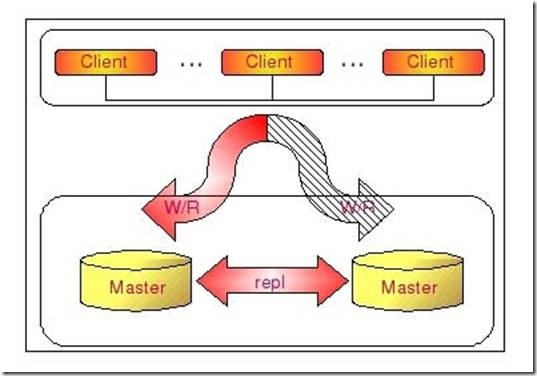
**4、默认使用异步复制方式**

**5、延迟**

**延迟的原因：**

1. **slave节点过多，master会为每个slave打开一个dump线程**
2. **并发写请求操作，由于mysql支持并发请求，如果多个写请求同时到达，将会将这些写操作都记录在二进制日志中，但在写入二进制文件时只能通过串行写入**
3. **slave通过i/o线程读取二进制文件并写入中继日志是也只能通过串行写入**

**master/master**



**master/master的出现正好解决了master/slave中如果master宕机之后用户的写请求无法完成的问题，但同时引入了比master/slave更多的问题，实现原理基本和master/slave相同，只需要在两台master中分别启动二进制日志和中继日志即可**

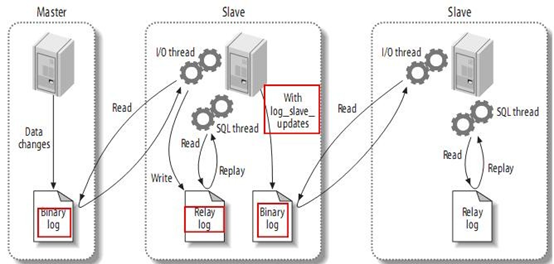
**注意问题：**

1. **避免主键重复**
2. **server-id全局唯一**

**存在问题：**

1. **经常出现两台master数据不一致(存在巨大风险)**
2. **主键冲突**
3. **延迟**

**级联复制**



**在有些应用场景中，可能读写压力差别比较大，读压力特别的大，一个Master可能需要上10台甚至更多的Slave才能够支撑注读的压力。这时候，Master就会比较吃力了，因为仅仅连上来的SlaveIO线程就比较多了，这样写的压力稍微大一点的时候，Master端因为复制就会消耗较多的资源，很容易造成复制的延时。这时候我们就可以利用MySQL可以在Slave端记录复制所产生变更的BinaryLog信息的功能，也就是打开log\_slave\_update选项。然后，通过二级（或者是更多级别）复制来减少Master端因为复制所带来的压力。**

**这种多层级联复制的架构，很容易就解决了Master端因为附属Slave太多而成为瓶颈的风险。**

**但下面阶级过多同一个变更传到最底层的Slave所需要经过的MySQL也会更多，同样可能造成延时较长的风险**

**存在问题：**

**1、可能导致延迟更为严重**

**2、借助于BlackHole存储引擎(黑洞引擎)**

**半同步复制**

**mysql复制方式：**

**异步：将二进制日志发送给slave不等待slave回应直接返回给客户端，数据储存成功**

**同步：将二进制日志发送给slave等待所有slave回应执行成功后返回给客户端，数据储存成功**

**半同步：将二进制日志发送给slave等待一个slave回应执行成功后返回给客户端，数据储存成功**