**第一节知识点**

**双十一的成败在不同人的眼中  
 电商老板: 广告投入、带来了客户**

**运营人员: 活动策划和产品设计的好**

**技术人员: 稳定的数据库和服务器环境的稳定**

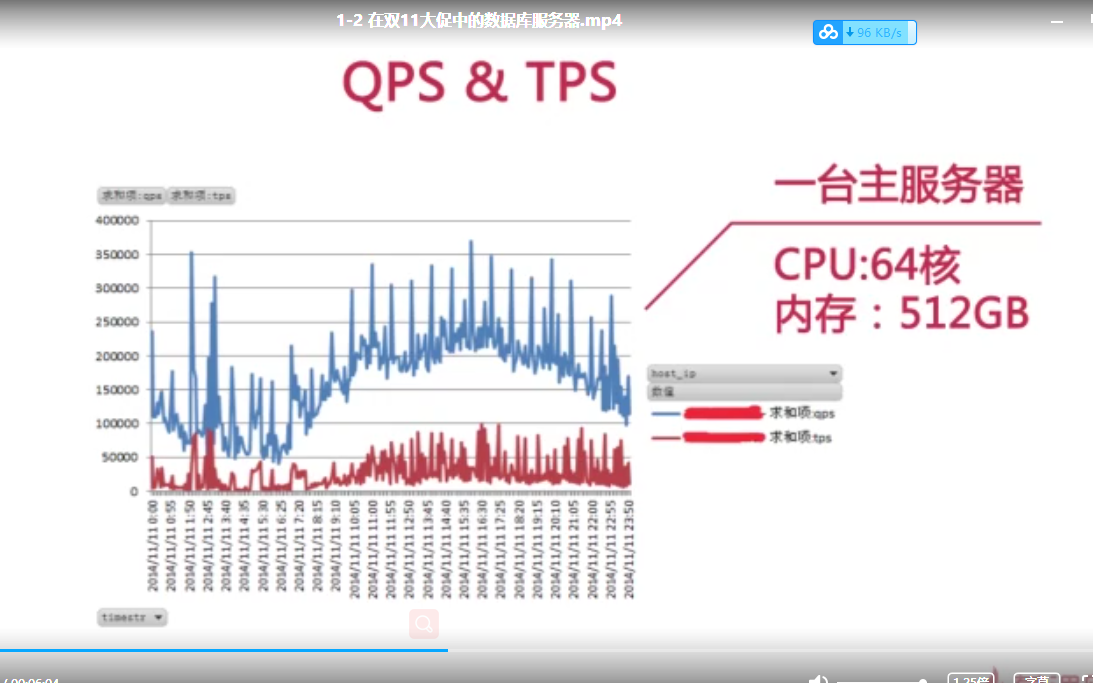
**Web服务器 : 可以横向扩展(增加服务器、部署web服务)**

**数据库服务器 :**

**特性: 完整性&一致性**

**第二节知识点:**

1. **数据库架构:**
2. **主从复制组件**
3. **监控信息**

 **（1）QPS & TPS （吞度量）**

**QPS: 每秒中处理的查询量**

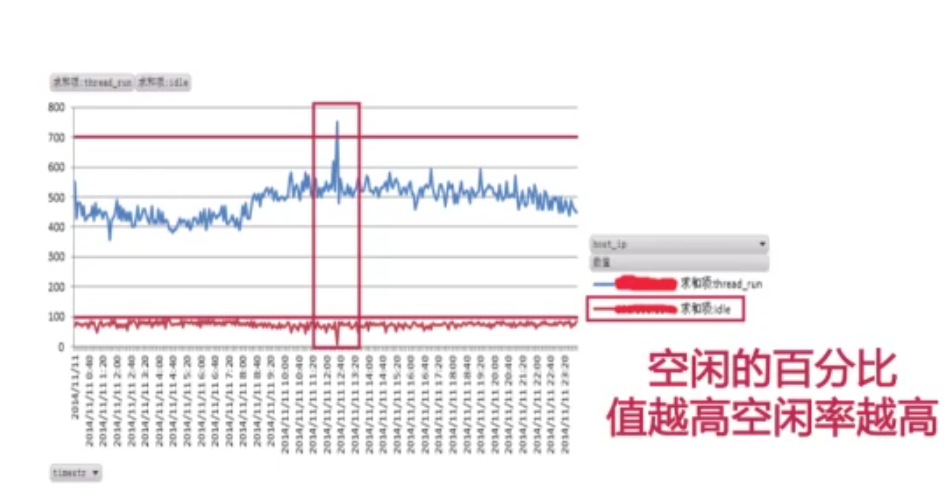
**QPS计算**



**TPS:**

**超高的QPS和Tps(吞吐量.访问量)的时候 :**

**效率低下的sql 会造成风险**



**（2）并发量 & cpu 使用量  
 并发量: 同一时间处理的请求的数量；**

**同时连接数: 同一时间内链接的数量**

**并发量和同时连接数的不同:**

**并发量是在同时链接的数量中拿出一批处理，**

**剩下的会处于sleep状态**

**Cpu使用率: 上面的图中代表的是空间的百分比**

**大量的并发和超高的cpu使用率:**

**风险1 : 大量的并发**

**会造成数据库连接数被占满。(max\_connections默认100)**

**当数据库连接数超过了mysql连接值 则前台服务器就会返回404 、500 影响用户体验**

**风险2 : cpu的高使用率**

**会使cpu资源被耗尽而宕机**



**（3）磁盘IO（数据库的瓶颈）**

**IO高 会造成服务器性能下降，导致阻塞**

**风险1 : 磁盘IO性能突然下降(**

**解决 : 更换更快的磁盘设备)、**

**风险2 : 其他大量消耗磁盘性能的计划任务**

**解决: /调整计划任务(如将备份在从数据库上进行)**

**/做好磁盘维护（现在磁盘出现问题会报警）**

**第三节知识点:**

**影响数据库的因素:**

1. **SQL 查询的速度**
2. **服务器硬件**
3. **网卡流量**

**风险1：网卡IO被沾满（1000mb/8=100mB）**

1. **减少从服务器的数量**

**(从服务器要去主服务器复制log日志,所以从服务器越多主服务器压力越大)**

1. **进行分级缓存**

**(要避免缓存失效对服务器冲击)**

1. **避免使用”select \* ” 进行查询**

**（查询不必要的会浪费流量）**

1. **分离业务网络和服务器网络**
2. **磁盘IO**

**第四节知识点**

**大表的概念:**

1. **记录行数巨大，单表超千万**
2. **表数据文件巨大，超10G**

**当然也是根据业务和磁盘I0来的比如只是用来select和insert**

**大表带来的问题:**

1. **建立索引需要很长时间**

**风险1:**

**mysql<5.5**

**默认的innoDB建立索引会锁表**

**Mysql>=5.5 inodbplass**

**不会锁表但是mysql的机制是先在主服务器中进行建索引然后在同步到从服务器中这时候会有延迟造成业务出错**

1. **修改表结构要很长时间**

**风险1:**

**会造成长时间的主从延迟**

**Mysql的机制对所有的ddl操作都是在主服务器上进行操作，然后通过日志传送到从库上然后在从库上进行相同的操作，从而实现表的复制；**

**风险2**

**影响正常的数据操作**

**单线程，多线程（5.6）**

1. **慢查询的产生:**

**查询大表的时候很难再一定的时间内进行过滤**

**如何解决:**

1. **分库分表**
   1. **分表主键的选择**
   2. **分表后跨分区数据的查询和统计**
2. **大表的历史数据归档**
   1. **减少对前后端业务的影响**
   2. **归档的时间点选择**
   3. **如何进行归档操作**

**第四节知识点**

**事务:**

**1.是数据库系统区别于其他一切文件系统的重要特性之一**

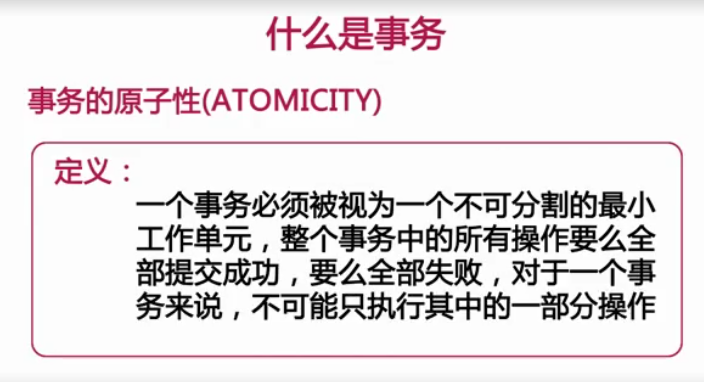
**例子:**

如果要保证两个文件的一致性；当我们修改一个文件后，系统崩溃了，然而系统恢复后文件就很难保持一致了

而数据库在系统恢复后根据事务的特点可以恢复数据库的数据

**2.事务是一组具有原子性的SQL语句是sql语句的集合，或是一个独立的工作单元**

**3. 事务的原子性:**



**例子:**

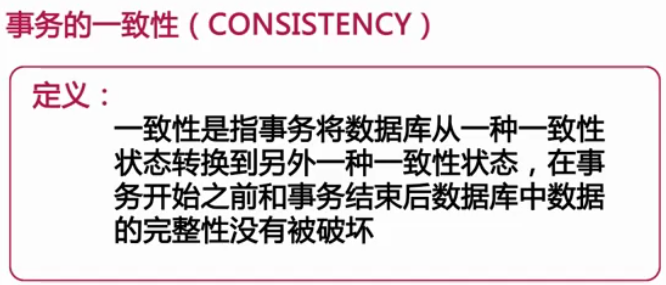
**银行转账,当我们从理财中拿出200元转到银行卡中我们需要这么几个操作:**

1. **判断理财中是否大于200元**
2. **从理财中减去200元**
3. **在银行卡中添加200元**

**当我们在第二步完成的时候系统崩溃了，那么当系统恢复的时候我们第三步还没有执行相当对我们理财中减去200，银行卡中没有加200，这样就亏大了；但是加上事务后我们系统中就会在日志中判断事务是否提交了，没有提交就回滚到第一步之前的状态；**

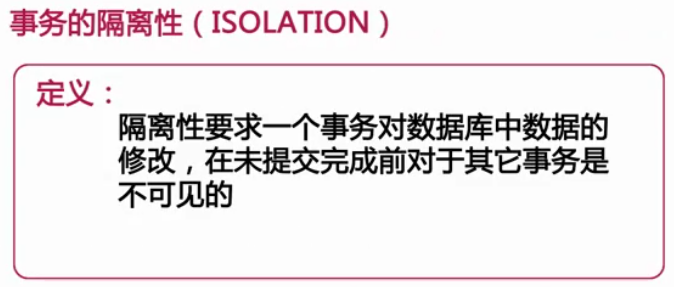
**也就是 要么全部完成，或者全部失败**

**4. 事务的一致性**



**接着上面的例子来说:就是个人的总金额在事务前和事务后必须是一致的;**

**5. 事务的隔离性**



1. **未提交读 （READ ）**
   1. **在未提交读模式中**

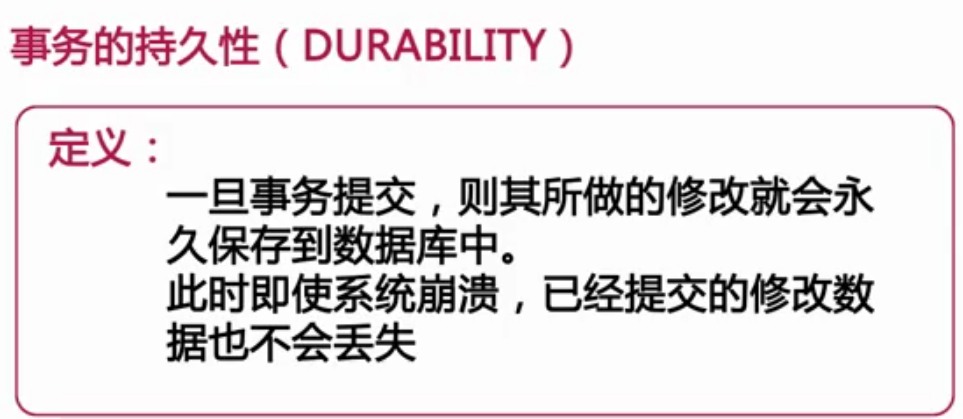
**事务改即使没有被提交对其他事务也是可见的，**

**事务可以读取未提交的数据，被称为脏读**

**未提交的数据我们称之为脏数据**

1. **已提交读**
2. **可重复读**
3. **可串行化**

**6. 事务的持久性**



**什么是大事务?**

**定义:运行时间比较长,操作的数据比较多的事务  
 风险:**

1. **锁定太多的数据，造成大量的阻塞和锁超时**
2. **回滚时所需要的比较长**
3. **执行时间长，主从延迟**

**如何处理:**

1. **避免一次处理太多的数据**
2. **移出不必要在事务中的select 操作**

**例子:**

**比如余额宝,我们每天需要计算用户的收益；如果一个事务对所有的用户进行事务处理，那么就会发生，事务时间长，事务锁造成阻塞，连接数满了,等**