链接: https://www.cnblogs.com/wupeiqi/p/9078770.html

<https://mp.weixin.qq.com/s?__biz=MzA5ODUzOTA0OQ==&mid=2651689264&idx=1&sn=11fbc3076ad08dc22bf80eb975c8ee14&chksm=8b693276bc1ebb60cdf08d14a5856050edee89406b9f94679c0b78765921cd04b3702cae6381&mpshare=1&scene=1&srcid=0601iRcbOSz3LQTWArSeIWqS#rd>

## 列举常见的关系型数据库和非关系型都有那些？

关系型数据库与非关系型数据库的区别

非关系型数据库的优势：

性能NOSQL是基于键值对的，可以想象成表中的主键和值的对应关系，而且不需要经过SQL层的解析，所以性能非常高。

可扩展性同样也是因为基于键值对，数据之间没有耦合性，所以非常容易水平扩展。关系型数据库的优势：

复杂查询可以用SQL语句方便的在一个表以及多个表之间做非常复杂的数据查询。

事务支持使得对于安全性能很高的数据访问要求得以实现。

对于这两类数据库，对方的优势就是自己的弱势，反之亦然。

数据库(Database)是按照数据结构来组织、存储和管理数据的仓库，它产生于距今六十多年前，随着信息技术和市场的发展，特别是二十世纪九十年代以后，数据管理不再仅仅是存储和管理数据，而转变成用户所需要的各种数据管理的方式。数据库有很多种类型，从最简单的存储有各种数据的表格到能够进行海量数据存储的大型数据库系统都在各个方面得到了广泛的应用。

## MySQL常见数据库引擎及比较？

引擎，engine，生活中引擎很常见，比如飞机引擎，汽车引擎，游戏引擎，搜索引擎，中文搜索引擎，引擎也就是核心的意思，对于机动车来说引擎就是发动机，就是主要核心部件。在数据库中也同样如此，是可以理解为数据库核心部分，核心“部件”。

mysql常用的引擎包括MYISAM、Innodb和Memory，各自的特点如下：

1、MYISAM：应用时以读和插入操作为主，只有少量更新和删除，并且对事物的完整性和并发性要去不是很高时选择。MYISAM全表锁，拥有较高的执行速度，并发能力差，占用空间相对较小，mysql5.5 及以下仅MYISAM支持全文索引，不支持事务。

2、Innodb：行级锁（SQL都走索引查询），并发能力相对较强，占用空间是MYISAM的2.5倍，不支持全文索引（5.6开始支持），支持事务。

事务处理，以及并发条件下要求数据的一致性。除了插入和查询之外，包括很多的更新和删除。（Innodb有效地降低删除和更新导致的锁定）

  Memory：全表锁，存储在内存当中，速度快，但会占用和数量成正比的内存空间且数据在mysql重启时会丢失。数据保存在RAM，快速访问数据，要求表不能太大或者对mysql异常终止后不能恢复数据。

通过命令我们可以看到当前设置mysql默认引擎为Innodb，我们可以通过修改默认引擎来设置mysql存储引擎。（[http://www.cnblogs.com/yjwen/archive/2012/05/01/2478124.html](http://www.cnblogs.com/yjwen/archive/2012/05/01/2478124.html" \t "_blank)）

## 简述数据三大范式？

# [数据库的三大范式？](https://www.cnblogs.com/yi11/p/6680904.html)

第一范式:确保每列的原子性.  
 如果每列(或者每个属性)都是不可再分的最小数据单元(也称为最小的原子单元),则满足第一范式.  
 例如:顾客表(姓名、编号、地址、……)其中"地址"列还可以细分为国家、省、市、区等。  
第二范式:在第一范式的基础上更进一层,目标是确保表中的每列都和主键相关.  
 如果一个关系满足第一范式,并且除了主键以外的其它列,都依赖于该主键,则满足第二范式.  
 例如:订单表(订单编号、产品编号、定购日期、价格、……)，"订单编号"为主键，"产品编号"和  
主键列没有直接的关系，即"产品编号"列不依赖于主键列，应删除该列。  
第三范式:在第二范式的基础上更进一层,目标是确保每列都和主键列直接相关,而不是间接相关.  
 如果一个关系满足第二范式,并且除了主键以外的其它列都不依赖于主键列,则满足第三范式.  
 为了理解第三范式，需要根据Armstrong公里之一定义传递依赖。假设A、B和C是关系R的三个  
属性，如果A-〉B且B-〉C，则从这些函数依赖中，可以得出A-〉C，如上所述，依赖A-〉C是传递依赖。  
 例如:订单表(订单编号，定购日期，顾客编号，顾客姓名，……)，初看该表没有问题，满足第二范式  
，每列都和主键列"订单编号"相关，再细看你会发现"顾客姓名"和"顾客编号"相关，"顾客编号"和"订单编号"  
又相关，最后经过传递依赖，"顾客姓名"也和"订单编号"相关。为了满足第三范式，应去掉"顾客姓名"列，  
放入客户表中。

### 3.1 范式一

网址：<https://www.cnblogs.com/linjiqin/archive/2012/04/01/2428695.html>

**1．第一范式(确保每列保持原子性)**

第一范式是最基本的范式。如果数据库表中的所有字段值都是不可分解的原子值，就说明该数据库表满足了第一范式。

第一范式的合理遵循需要根据系统的实际需求来定。比如某些数据库系统中需要用到“地址”这个属性，本来直接将“地址”属性设计成一个数据库表的字段就行。但是如果系统经常会访问“地址”属性中的“城市”部分，那么就非要将“地址”这个属性重新拆分为省份、城市、详细地址等多个部分进行存储，这样在对地址中某一部分操作的时候将非常方便。这样设计才算满足了数据库的第一范式，如下表所示。



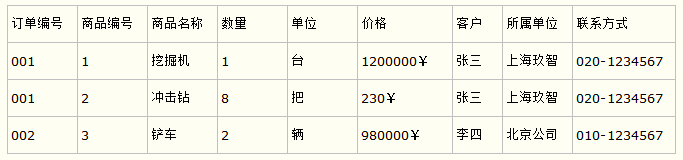
上表所示的用户信息遵循了第一范式的要求，这样在对用户使用城市进行分类的时候就非常方便，也提高了数据库的性能。

**2．第二范式(确保表中的每列都和主键相关)**

第二范式在第一范式的基础之上更进一层。第二范式需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。也就是说在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中。

比如要设计一个订单信息表，因为订单中可能会有多种商品，所以要将订单编号和商品编号作为数据库表的联合主键，如下表所示。

**订单信息表**



这样就产生一个问题：这个表中是以订单编号和商品编号作为联合主键。这样在该表中商品名称、单位、商品价格等信息不与该表的主键相关，而仅仅是与商品编号相关。所以在这里违反了第二范式的设计原则。

而如果把这个订单信息表进行拆分，把商品信息分离到另一个表中，把订单项目表也分离到另一个表中，就非常完美了。如下所示。



这样设计，在很大程度上减小了数据库的冗余。如果要获取订单的商品信息，使用商品编号到商品信息表中查询即可。

**3．第三范式(确保每列都和主键列直接相关,而不是间接相关)**

第三范式需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关。

比如在设计一个订单数据表的时候，可以将客户编号作为一个外键和订单表建立相应的关系。而不可以在订单表中添加关于客户其它信息（比如姓名、所属公司等）的字段。如下面这两个表所示的设计就是一个满足第三范式的数据库表。



这样在查询订单信息的时候，就可以使用客户编号来引用客户信息表中的记录，也不必在订单信息表中多次输入客户信息的内容，减小了数据冗余。

数据库的设计范式是数据库设计所需要满足的规范，满足这些规范的数据库是简洁的、结构明晰的，同时，不会发生插入（insert）、删除（delete）和更新（update）操作异常。反之则是乱七八糟，不仅给数据库的编程人员制造麻烦，而且面目可憎，可能存储了大量不需要的冗余信息。

范式说明

### 3.2 范式二

网址：https://blog.csdn.net/w\_\_yi/article/details/19934319

1.1 第一范式（1NF）无重复的列

所谓第一范式（1NF）是指数据库表的每一列都是不可分割的基本数据项，同一列中不能有多个值，即实体中的某个属性不能有多个值或者不能有重复的属性。如果出现重复的属性，就可能需要定义一个新的实体，新的实体由重复的属性构成，新实体与原实体之间为一对多关系。在第一范式（1NF）中表的每一行只包含一个实例的信息。简而言之，第一范式就是无重复的列。

说明：在任何一个关系数据库中，第一范式（1NF）是对关系模式的基本要求，不满足第一范式（1NF）的数据库就不是关系数据库。

例如，如下的数据库表是符合第一范式的：

而这样的数据库表是不符合第一范式的：

数据库表中的字段都是单一属性的，不可再分。这个单一属性由基本类型构成，包括整型、实数、字符型、逻辑型、日期型等。很显然，在当前的任何关系数据库管理系统（DBMS）中，傻瓜也不可能做出不符合第一范式的数据库，因为这些DBMS不允许你把数据库表的一列再分成二列或多列。因此，你想在现有的DBMS中设计出不符合第一范式的数据库都是不可能的。

1.2 第二范式（2NF）属性完全依赖于主键 [ 消除部分子函数依赖 ]

如果关系模式R为第一范式，并且R中每一个非主属性完全函数依赖于R的某个候选键， 则称为第二范式模式。

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或行必须可以被惟一地区分。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。这个惟一属性列被称为主关键字或主键、主码。

例如员工信息表中加上了员工编号（emp\_id）列，因为每个员工的员工编号是惟一的，因此每个员工可以被惟一区分。

简而言之，第二范式（2NF）就是非主属性完全依赖于主关键字。

所谓完全依赖是指不能存在仅依赖主关键字一部分的属性（设有函数依赖W→A，若存在XW，有X→A成立，那么称W→A是局部依赖，否则就称W→A是完全函数依赖）。如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。

假定选课关系表为SelectCourse(学号, 姓名, 年龄, 课程名称, 成绩, 学分)，关键字为组合关键字(学号, 课程名称)，因为存在如下决定关系：

(学号, 课程名称) → (姓名, 年龄, 成绩, 学分)

这个数据库表不满足第二范式，因为存在如下决定关系：

(课程名称) → (学分)

(学号) → (姓名, 年龄)

即存在组合关键字中的字段决定非关键字的情况。

由于不符合2NF，这个选课关系表会存在如下问题：

(1) 数据冗余：

同一门课程由n个学生选修，"学分"就重复n-1次；同一个学生选修了m门课程，姓名和年龄就重复了m-1次。

(2) 更新异常：

若调整了某门课程的学分，数据表中所有行的"学分"值都要更新，否则会出现同一门课程学分不同的情况。

(3) 插入异常：

假设要开设一门新的课程，暂时还没有人选修。这样，由于还没有"学号"关键字，课程名称和学分也无法记录入数据库。

(4) 删除异常：

假设一批学生已经完成课程的选修，这些选修记录就应该从数据库表中删除。但是，与此同时，课程名称和学分信息也被删除了。很显然，这也会导致插入异常。

把选课关系表SelectCourse改为如下三个表：

学生：Student(学号, 姓名, 年龄)；

课程：Course(课程名称, 学分)；

选课关系：SelectCourse(学号, 课程名称, 成绩)。

这样的数据库表是符合第二范式的， 消除了数据冗余、更新异常、插入异常和删除异常。

另外，所有单关键字的数据库表都符合第二范式，因为不可能存在组合关键字。

1.3 第三范式（3NF）属性不依赖于其它非主属性 [ 消除传递依赖 ]

如果关系模式R是第二范式，且每个非主属性都不传递依赖于R的候选键，则称R为第三范式模式。

满足第三范式（3NF）必须先满足第二范式（2NF）。第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。

例如，存在一个部门信息表，其中每个部门有部门编号（dept\_id）、部门名称、部门简介等信息。那么在的员工信息表中列出部门编号后就不能再将部门名称、部门简介等与部门有关的信息再加入员工信息表中。如果不存在部门信息表，则根据第三范式（3NF）也应该构建它，否则就会有大量的数据冗余。

第三范式（3NF）：在第二范式的基础上，数据表中如果不存在非关键字段对任一候选关键字段的传递函数依赖则符合第三范式。简而言之，第三范式就是属性不依赖于其它非主属性。

所谓传递函数依赖，指的是如果存在"A → B → C"的决定关系，则C传递函数依赖于A。

因此，满足第三范式的数据库表应该不存在如下依赖关系：

关键字段 → 非关键字段x → 非关键字段y

假定学生关系表为Student(学号, 姓名, 年龄, 所在学院, 学院地点, 学院电话)，关键字为单一关键字"学号"，因为存在如下决定关系：

(学号) → (姓名, 年龄, 所在学院, 学院地点, 学院电话)

这个数据库是符合2NF的，但是不符合3NF，因为存在如下决定关系：

(学号) → (所在学院) → (学院地点, 学院电话)

即存在非关键字段"学院地点"、"学院电话"对关键字段"学号"的传递函数依赖。

它也会存在数据冗余、更新异常、插入异常和删除异常的情况，读者可自行分析得知。

把学生关系表分为如下两个表：

学生：(学号, 姓名, 年龄, 所在学院)；

学院：(学院, 地点, 电话)。

这样的数据库表是符合第三范式的，消除了数据冗余、更新异常、插入异常和删除异常。

## 什么是事务？MySQL如何支持事务？

网址：https://blog.csdn.net/u013019431/article/details/78545749

### 4.1 什么是事务？

事务由一个或多个sql语句组成一个整体，如果所有的语句执行成功那么修改将会全部生效，如一条sql语句将销量+1，下一条再+1，倘若第二条失败，那么销量将撤销第一条sql语句的+1操作，只有在该事务中所有的语句都执行成功才会将修改加入到数据库中。

### 4.2 事务的特性

事务具体四大特性，也就是经常说的ACID

1. 原子性（Atomicity）

　　原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

2. 一致性（Consistency）

　　一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

　　拿转账来说，假设用户A和用户B两者的钱加起来一共是5000，那么不管A和B之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是5000，这就是事务的一致性。

3.隔离性（Isolation）

　　隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

　　即要达到这么一种效果：对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

4.持久性（Durability）

　　持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。Mysql中会保存有相应的操作日志，即使遭遇故障依然能够通过日志恢复最后一次更新。

　　例如我们在使用JDBC操作数据库时，在提交事务方法后，提示用户事务操作完成，当我们程序执行完成直到看到提示后，就可以认定事务以及正确提交，即使这时候数据库出现了问题，也必须要将我们的事务完全执行完成，否则就会造成我们看到提示事务处理完毕，但是数据库因为故障而没有执行事务的重大错误。

### 4.3 MySql中支持事务的引擎

在mysql中用的最多的存储引擎有：innodb，bdb，myisam ,memory 等。其中innodb和bdb支持事务而myisam等不支持事务。

二、事务的并发问题

　　1、脏读：事务A读取了事务B更新的数据，然后B回滚操作，那么A读取到的数据是脏数据

　　2、不可重复读：事务 A 多次读取同一数据，事务 B 在事务A多次读取的过程中，对数据作了更新并提交，导致事务A多次读取同一数据时，结果 不一致。

　　3、幻读：系统管理员A将数据库中所有学生的成绩从具体分数改为ABCDE等级，但是系统管理员B就在这个时候插入了一条具体分数的记录，当系统管理员A改结束后发现还有一条记录没有改过来，就好像发生了幻觉一样，这就叫幻读。

　　小结：不可重复读的和幻读很容易混淆，不可重复读侧重于修改，幻读侧重于新增或删除。解决不可重复读的问题只需锁住满足条件的行，解决幻读需要锁表

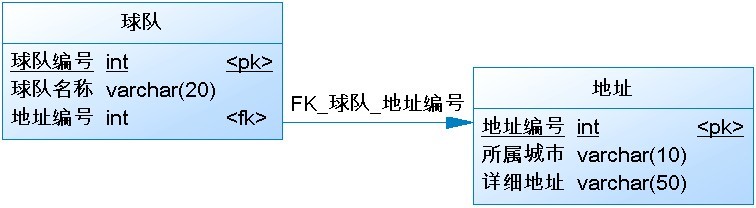
## 简述数据库设计中一对多和多对多的应用场景？

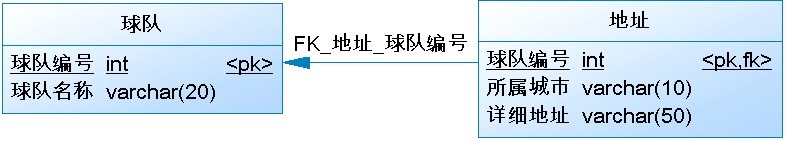
网址：https://www.cnblogs.com/lgxlsm/archive/2013/05/15/3080994.html

**关联映射：一对多/多对一**  
存在最普遍的映射关系，简单来讲就如球员与球队的关系；  
一对多：从球队角度来说一个球队拥有多个球员 即为一对多  
多对一：从球员角度来说多个球员属于一个球队 即为多对一数据表间一对多关系如下图：  

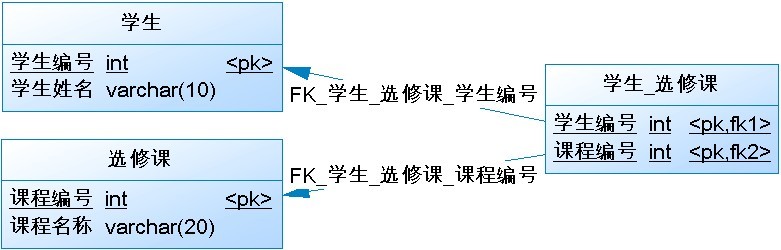

关联映射：一对一  
一对一关系就如球队与球队所在地址之间的关系，一支球队仅有一个地址，而一个地址区也仅有一支球队。  
数据表间一对一关系的表现有两种，一种是外键关联，一种是主键关联。

**一对一外键关联，图示如下：**

  
**一对一主键关联：要求两个表的主键必须完全一致，通过两个表的主键建立关联关系。图示如下：**



**关联映射：多对多**  
多对多关系也很常见，例如学生与选修课之间的关系，一个学生可以选择多门选修课，而每个选修课又可以被多名学生选择。  
数据库中的多对多关联关系一般需采用中间表的方式处理，将多对多转化为两个一对多。  
数据表间多对多关系如下图：



6、如何基于数据库实现商城商品计数器？

7、常见SQL（必备）  
https://www.cnblogs.com/wupeiqi/articles/5729934.html

## 8、简述触发器、函数、视图、存储过程？

9、MySQL索引种类

10、索引在什么情况下遵循最左前缀的规则？

11、主键和外键的区别？

12、MySQL常见的函数？

13、列举 创建索引但是无法命中索引的8种情况。

14、如何开启慢日志查询？

15、数据库导入导出命令（结构+数据）？

16、数据库优化方案？

17、char和varchar的区别？

18、简述MySQL的执行计划？

19、在对name做了唯一索引前提下，简述以下区别：

20、1000w条数据，使用limit offset 分页时，为什么越往后翻越慢？如何解决？

21、什么是索引合并？

22、什么是覆盖索引？

23、简述数据库读写分离？

24、简述数据库分库分表？（水平、垂直）

25、redis和memcached比较？

## 26、redis中数据库默认是多少个db 及作用？

27、python操作redis的模块？

28、如果redis中的某个列表中的数据量非常大，如果实现循环显示每一个值？

## 29、redis如何实现主从复制？以及数据同步机制？

http://www.cnblogs.com/zhoujinyi/p/5585723.html

https://www.cnblogs.com/zhoujinyi/p/5570024.html

## 30、redis中的sentinel的作用？

      Redis-Sentinel是Redis官方推荐的高可用性(HA)解决方案，当用Redis做Master-slave的高可用方案时，假如master宕机了，Redis本身(包括它的很多客户端)都没有实现自动进行主备切换，而Redis-sentinel本身也是一个独立运行的进程，它能监控多个master-slave集群，发现master宕机后能进行自动切换，更多的信息见[前一篇](http://www.cnblogs.com/zhoujinyi/p/5569462.html)说明。它的主要功能有以下几点：

1，不时地监控redis是否按照预期良好地运行;

2，如果发现某个redis节点运行出现状况，能够通知另外一个进程(例如它的客户端);

3，能够进行自动切换。当一个master节点不可用时，能够选举出master的多个slave(如果有超过一个slave的话)中的一个来作为新的master,其它的slave节点会将它所追随的master的地址改为被提升为master的slave的新地址。

31、如何实现redis集群？

https://www.zhihu.com/question/21419897

## 32、redis中默认有多少个哈希槽？

最近在学习redis,看到如果是集群的话，整个redis集群一共会拥有16384个slot, 我想知道如果16384个slot不够用的话应该怎么做，能自己设置slot数量吗，还是每个集群最多只能有16384个slot不够用的话就得搭建多个集群？

## 33、简述redis的有哪几种持久化策略及比较？

1).Master写内存快照，save命令调度rdbSave函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以Master最好不要写内存快照。

2).Master AOF持久化，如果不重写AOF文件，这个持久化方式对性能的影响是最小的，但是AOF文件会不断增大，AOF文件过大会影响Master重启的恢复速度。Master最好不要做任何持久化工作，包括内存快照和AOF日志文件，特别是不要启用内存快照做持久化,如果数据比较关键，某个Slave开启AOF备份数据，策略为每秒同步一次。

3).Master调用BGREWRITEAOF重写AOF文件，AOF在重写的时候会占大量的CPU和内存资源，导致服务load过高，出现短暂服务暂停现象。

4). Redis主从复制的性能问题，为了主从复制的速度和连接的稳定性，Slave和Master最好在同一个局域网内

34、列举redis支持的过期策略。

## 35、MySQL 里有 2000w 数据，redis 中只存 20w 的数据，如何保证 redis 中都是热点数据？

相关知识：redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。

redis 提供 6种数据淘汰策略：

voltile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰

volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰

volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰

allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰

allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰

no-enviction（驱逐）：禁止驱逐数据

36、写代码，基于redis的列表实现 先进先出、后进先出队列、优先级队列。

37、如何基于redis实现消息队列？

38、如何基于redis实现发布和订阅？以及发布订阅和消息队列的区别？

39、什么是codis及作用？

40、什么是twemproxy及作用？

41、写代码实现redis事务操作。

42、redis中的watch的命令的作用？

43、基于redis如何实现商城商品数量计数器？

44、简述redis分布式锁和redlock的实现机制。

45、什么是一致性哈希？Python中是否有相应模块？

46、如何高效的找到redis中所有以oldboy开头的key？

## 47、Redis cluster集群

<https://www.cnblogs.com/kerwinC/p/6611634.html>

redis使用中遇到的瓶颈

　　我们日常在对于redis的使用中，经常会遇到一些问题

　　1、高可用问题，如何保证redis的持续高可用性。

　　2、容量问题，单实例redis内存无法无限扩充，达到32G后就进入了64位世界，性能下降。

　　3、并发性能问题，redis号称单实例10万并发，但也是有尽头的。

redis-cluster的优势

　　1、官方推荐，毋庸置疑。

　　2、去中心化，集群最大可增加1000个节点，性能随节点增加而线性扩展。

　　3、管理方便，后续可自行增加或摘除节点，移动分槽等等。

　　4、简单，易上手。

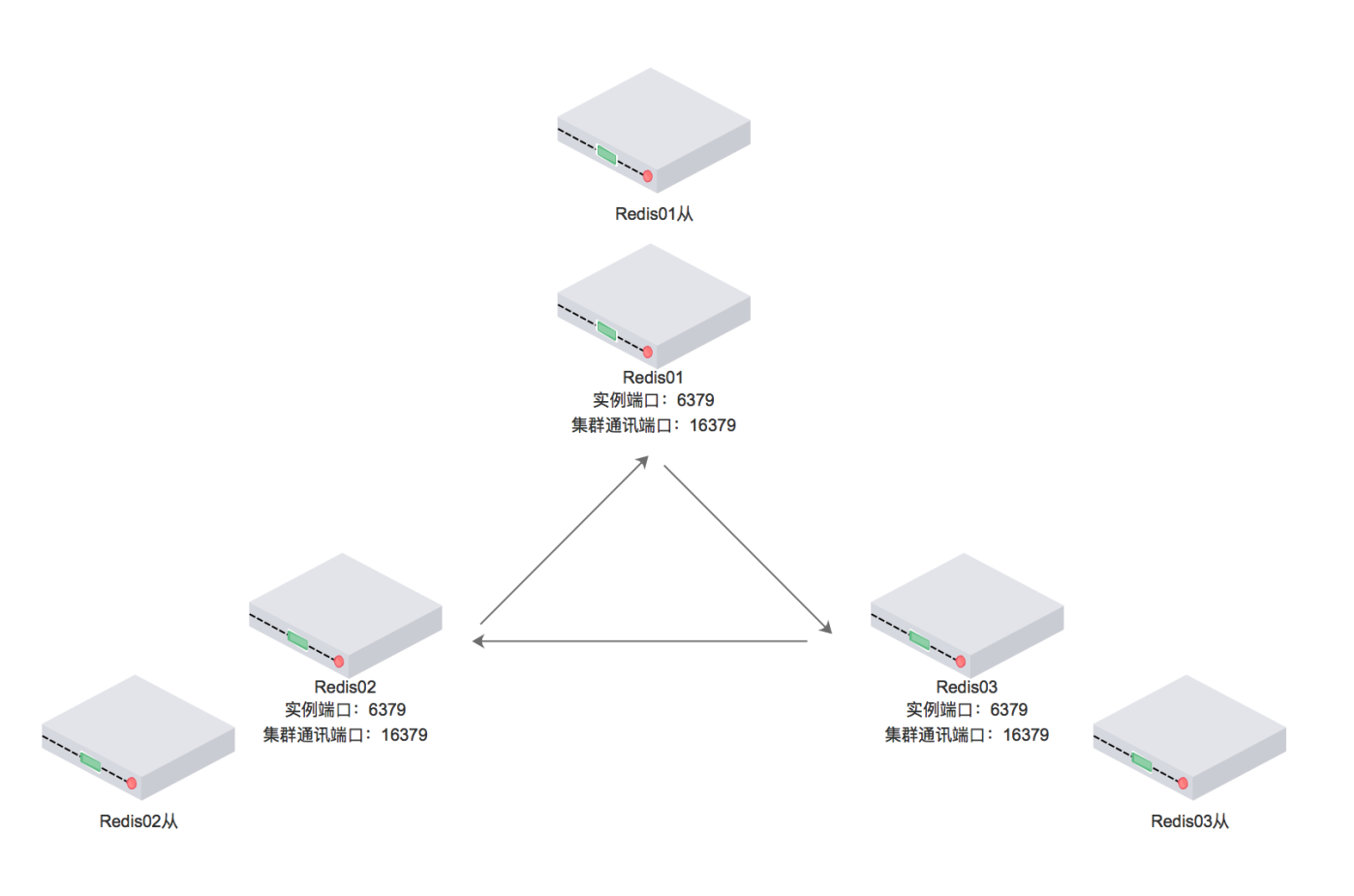
redis-cluster名词介绍

　　1、master　　主节点、

　　2、slave　　　从节点

　　3、slot　　　　槽，一共有16384数据分槽，分布在集群的所有主节点中。

redis-cluster简介



图中描述的是六个redis实例构成的集群

6379端口为客户端通讯端口

16379端口为集群总线端口

集群内部划分为16384个数据分槽，分布在三个主redis中。

从redis中没有分槽，不会参与集群投票，也不会帮忙加快读取数据，仅仅作为主机的备份。

三个主节点中平均分布着16384数据分槽的三分之一，每个节点中不会存有有重复数据，仅仅有自己的从机帮忙冗余。



2、什么是C/S和B/S架构？

29、haproxy是什么以及作用？

30、什么是负载均衡？

31、什么是rpc及应用场景？

32、简述 asynio模块的作用和应用场景。

33、简述 gevent模块的作用和应用场景。

34、twisted框架的使用和应用？

3、什么是magic string ？

4、如何创建响应式布局？

5、你曾经使用过哪些前端框架？

6、什么是ajax请求？并使用jQuery和XMLHttpRequest对象实现一个ajax请求。

7、如何在前端实现轮训？

8、如何在前端实现长轮训？

9、vuex的作用？

10、vue中的路由的拦截器的作用？

11、axios的作用？

## 48、[谈谈mongodb,mysql的区别和具体应用场景](https://www.cnblogs.com/printN/p/7199928.html)对数据库于

MySQL

关系型数据库。

在不同的引擎上有不同 的存储方式。

查询语句是使用传统的sql语句，拥有较为成熟的体系，成熟度很高。

开源数据库的份额在不断增加，mysql的份额页在持续增长。

缺点就是在海量数据处理的时候效率会显著变慢。

Mongodb

非关系型数据库(nosql ),属于文档型数据库。先解释一下文档的数据库，即可以存放xml、json、bson类型系那个的数据。这些数据具备自述性（self-describing），呈现分层的树状数据结构。数据结构由键值(key=>value)对组成。

存储方式：虚拟内存+持久化。

查询语句：是独特的Mongodb的查询方式。

适合场景：事件的记录，内容管理或者博客平台等等。

架构特点：可以通过副本集，以及分片来实现高可用。

数据处理：数据是存储在硬盘上的，只不过需要经常读取的数据会被加载到内存中，将数据存储在物理内存中，从而达到高速读写。

成熟度与广泛度：新兴数据库，成熟度较低，Nosql数据库中最为接近关系型数据库，比较完善的DB之一，适用人群不断在增长。

优势：

快速！在适量级的内存的Mongodb的性能是非常迅速的，它将热数据存储在物理内存中，使得热数据的读写变得十分快，

高扩展！

自身的Failover机制！

json的存储格式！

缺点：主要

分析一下Mysql和Mongodb应用场景

1.如果需要将mongodb作为后端db来代替mysql使用，即这里mysql与mongodb 属于平行级别，那么，这样的使用可能有以下几种情况的考量： (1)mongodb所负责部分以文档形式存储，能够有较好的代码亲和性，json格式的直接写入方便。(如日志之类) (2)从data models设计阶段就将原子性考虑于其中，无需事务之类的辅助。开发用如nodejs之类的语言来进行开发，对开发比较方便。 (3)mongodb本身的failover机制，无需使用如MHA之类的方式实现。

2.将mongodb作为类似redis ，memcache来做缓存db，为mysql提供服务，或是后端日志收集分析。 考虑到mongodb属于nosql型数据库，sql语句与数据结构不如mysql那么亲和 ，也会有很多时候将mongodb做为辅助mysql而使用的类redis memcache 之类的缓存db来使用。 亦或是仅作日志收集分析。