## sql索引理解和优化（算法理解二叉树，B树）

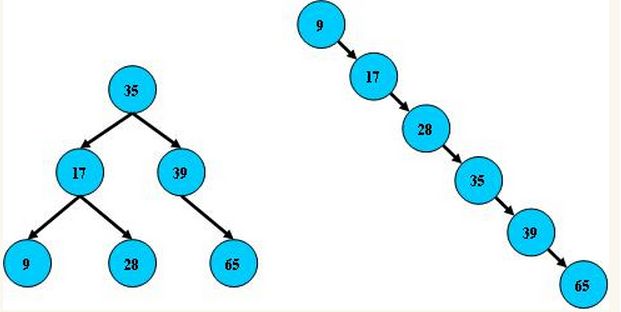
### 生成索引

Create index name\_index on person(name)

### 效率问题

提高SQL语句执行速度，可以根据SQL语句的查询条件在表中创建合适的索引，以此来提升SQL语句的执行速度；当然创建这个索引也需要有一定的代价，需要存储空间来存放，  总的来说，为SQL语句（select，update，delete）创建必要的索引是必须的，这样虽然有一定的性能和空间消耗，但是是值得，尤其是在大并发的请求下，大量的数据被扫描造成系统IO和CPU资源消耗完，进而导致整个数据库不可服务。

### B树（二叉树）



平衡二叉树特点：

（1）非叶子节点最多拥有两个子节点；

（2）非叶子节值大于左边子节点、小于右边子节点；

（3）树的左右两边的层级数相差不会大于1;

（4）没有值相等重复的节点;

## .net有哪些非托管资源

### 概念

**托管资源**指的是.NET可以自动进行回收的资源，主要是指托管堆上分配的内存资源。托管资源的回收工作是不需要人工干预的，有.NET运行库在合适调用垃圾回收器进行回收。

**非托管资源**指的是.NET不知道如何回收的资源，最常见的一类非托管资源是包装操作系统资源的对象，例如文件，窗口，网络连接，数据库连接，画刷，图标等。这类资源，垃圾回收器在清理的时候会调用Object.Finalize()方法。默认情况下，方法是空的，对于非托管对象，需要在此方法中编写回收非托管资源的代码，以便垃圾回收器正确回收资源。

### 使用

  Microsoft为非托管资源的回收专门定义了一个接口：IDisposable，接口中只包含一个Dispose()方法。任何包含非托管资源的类，都应该继承此接口。

      Despose()方法只能由类的使用者调用。

在C#中，凡是继承了IDisposable接口的类，都可以使用using语句，从而在超出作用域后，让系统自动调用Dispose()方法。  
         一个资源安全的类，都实现了IDisposable接口和析构函数。提供手动释放资源和系统自动释放资源的双保险。

## redis

### redis里有哪些类型，怎么应用

**1.String（字符串）**

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象 。

string类型是Redis最基本的数据类型，一个redis中字符串value最多可以是512M

**2.Hash（哈希）**

Redis hash 是一个键值对集合。  
Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。

类似Java里面的Map<String,Object>

**3.List（列表）**

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素导列表的头部（左边）或者尾部（右边）。  
它的底层实际是个链表

**4.Set（集合）**

Redis的Set是string类型的无序集合。它是通过HashTable实现实现的，

**5.zset(sorted set：有序集合)**

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。  
不同的是每个元素都会关联一个double类型的分数。  
redis正是通过分数来为集合中的成员进行从小到大的排序。zset的成员是唯一的,但分数(score)却可以重复。

### Redis的应用场景

1. **缓存——热数据 （我们现在用到的）**
2. **计数器**
3. **队列**
4. **位操作（大数据处理）**
5. **分布式锁与单线程机制**
6. **最新列表**
7. **排行榜**

参考 <https://www.cnblogs.com/NiceCui/p/7794659.html>

### 和memcached的区别

1、Redis和Memcache都是将数据存放在内存中，都是内存数据库。不过memcache还可用于缓存其他东西，例如图片、视频等等；

2、Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，hash等数据结构的存储；

3、虚拟内存--Redis当物理内存用完时，可以将一些很久没用到的value 交换到磁盘；

4、过期策略--memcache在set时就指定，例如set key1 0 0 8,即永不过期。Redis可以通过例如expire 设定，例如expire name 10；

5、分布式--设定memcache集群，利用magent做一主多从;redis可以做一主多从。都可以一主一从；

6、存储数据安全--memcache挂掉后，数据没了；redis可以定期保存到磁盘（持久化）；

7、灾难恢复--memcache挂掉后，数据不可恢复; redis数据丢失后可以通过aof恢复；

8、Redis支持数据的备份，即master-slave模式的数据备份；

### 与mongodb比较

MongoDB和Redis都是NoSQL，采用结构型数据存储。二者在使用场景中，存在一定的区别，这也主要由于二者在内存映射的处理过程，持久化的处理方法不同。

 MongoDB建议集群部署，更多的考虑到集群方案，Redis更偏重于进程顺序写入，虽然支持集群，也仅限于主-从模式。

|  |  |  |  |
| --- | --- | --- | --- |
| 比较指标 | MongoDB(v2.4.9) | Redis(v2.4.17) | 比较说明 |
| 实现语言 | c++ | c/c++ | - |
| 协议 | BSON,自定义二进制 | 类telnet | - |
| 性能 | 依赖内存,TPS较高 | 依赖内存,TPS非常高 | Redis优于MongoDB |
| 可操作性 | 丰富的数据表达,索引;最类似于关系型数据库,支持丰富的查询语句 | 数据丰富,较少的IO | MongoDB优于Redis |
| 内存及存储 | 适合大数据量存储,依赖系统虚拟内存,采用镜像文件存储;内存占用率比较高,官方建议独立部署在64位系统 | Redis2.0后支持虚拟内存特性(VM) 突破物理内存限制;数据可以设置时效性,类似于memcache | 不同的应用场景,各有千秋 |
| 可用性 | 支持master-slave,replicatset(内部采用paxos选举算法,自动故障恢复),auto sharding机制,对客户端屏蔽了故障转移和切片机制 | 依赖客户端来实现分布式读写;主从复制时,每次从节点重新连接主节点都要依赖整个快照,无增量复制;不支持auto sharding,需要依赖程序设定一致性hash机制 | MongoDB优于Redis；单点问题上,MongoDB应用简单,相对用户透明,Redis比较复杂,需要客户端主动解决.(MongoDB一般使用replicasets和sharding相结合,replicasets侧重高可用性以及高可靠,sharding侧重性能,水平扩展) |
| 可靠性 | 从1.8版本后,采用binlog方式(类似Mysql) 支持持久化 | 依赖快照进行持久化;AOF增强可靠性;增强性的同时,影响访问性能 |  |
| 一致性 | 不支持事务,靠客户端保证 | 支持事务,比较脆,仅能保证事务中的操作按顺序执行 | Redis优于MongoDB |
| 数据分析 | 内置数据分析功能(mapreduce) | 不支持 | MongoDB优于Redis |
| 应用场景 | 海量数据的访问效率提升 | 较小数据量的性能和运算 | MongoDB优于Redis |

.net有哪些集合类型，如何选取

hash算法的理解

反射的好处和坏处，如何优化反射

观察者模式实现的好的方法

单例模式的理解

异常处理（对于几个代码层级），哪些处理，怎么存数据

http协议的理解（重要）三次握手是指什么？

tcp/ip协议的理解

## varchar nvarchar char等sql数据库字段区别

CHAR:

1.长度固定，当char（15）的元素中只有“abc”时,其余十二位用空格填补；

VARCAHR:

1.长度不固定，该列中元素有多少位，就是多少位；

2.当数据为空时，该字段为空字符串；

3.VARCHAR只对汉字或全角字符占两个字节处理，数字和英文等都是一个字节；

VARCHAR2:

1.当数据为空时，该数据显示为NULL;

2.VARCHAR2（一般情况下）把所有字符都按两个字节处理，VARCHAR2字符要用几个字节存储，要看数据库使用的字符集，  
比如GBK，汉字就会占两个字节，英文1个  
如果是UTF-8，汉字一般占3个字节，英文还是1个。   
但是一般情况下，我们都认为是两个字节处理，因为oracle安装时候默认我们都选择GBK的编码格式，但是我们在页面做输入字符串长度的校验的时候，还是以数据库设计字段最大长度除3来作为最大长度-----防止数据库移植时设置不同编码格式；

## GC回收原理，什么时候决定一个类已经死亡，循环引用时呢

　1、只管理内存，非托管资源，如文件句柄，GDI资源，数据库连接等还需要用户去管理。

　　2、循环引用，网状结构等的实现会变得简单。GC的标志-压缩算法能有效的检测这些关系，并将不再被引用的网状结构整体删除。

　　3、GC通过从程序的根对象开始遍历来检测一个对象是否可被其他对象访问，而不是用类似于COM中的引用计数方法。

　　4、GC在一个独立的线程中运行来删除不再被引用的内存。

　　5、GC每次运行时会压缩托管堆。

　　6、你必须对非托管资源的释放负责。可以通过在类型中定义Finalizer来保证资源得到释放。

　　7、对象的Finalizer被执行的时间是在对象不再被引用后的某个不确定的时间。注意并非和C++中一样在对象超出声明周期时立即执行析构函数

　　8、Finalizer的使用有性能上的代价。需要Finalization的对象不会立即被清除，而需要先执行Finalizer.Finalizer，不是在GC执行的线程被调用。GC把每一个需要执行Finalizer的对象放到一个队列中去，然后启动另一个线程来执行所有这些Finalizer，而GC线程继续去删除其他待回收的对象。在下一个GC周期，这些执行完Finalizer的对象的内存才会被回收。

　　9、.NET GC使用"代"(generations)的概念来优化性能。代帮助GC更迅速的识别那些最可能成为垃圾的对象。在上次执行完垃圾回收后新创建的对象为第0代对象。经历了一次GC周期的对象为第1代对象。经历了两次或更多的GC周期的对象为第2代对象。代的作用是为了区分局部变量和需要在应用程序生存周期中一直存活的对象。大部分第0代对象是局部变量。成员变量和全局变量很快变成第1代对象并最终成为第2代对象。

　　10、GC对不同代的对象执行不同的检查策略以优化性能。每个GC周期都会检查第0代对象。大约1/10的GC周期检查第0代和第1代对象。大约1/100的GC周期检查所有的对象。重新思考Finalization的代价：需要Finalization的对象可能比不需要Finalization在内存中停留额外9个GC周期。如果此时它还没有被Finalize，就变成第2代对象，从而在内存中停留更长时间。

hashcode的生成，如何去避免生成重复的hashcode

多线程编程有哪些要注意的，有哪些当时可以保证线程安全

lock语法糖实际等同于哪种写法

angular（或react、vue)与jquery的区别

分布式开发的CAP原则

Mysql在EF上的开发和sqlserver上的区别

多个字段的快速查询，如何设计？

业务问题：

如何设计无状态的用户验证

前后端分离的情况下如何设计登录超时自动返回登录页面

如何拆预付款（多个合同一个预付款）来保证两方利益

如何在设计层面上设计报表模块