1.AutoRelease的嵌套是怎么处理的

通过一个自动释放池的栈（详细解释参考以下链接：黑幕背后的Autorelease）

<http://www.cocoachina.com/ios/20141031/10107.html>

AutoRelease让系统自行的选择合适的时机release对象，防止由于开发者操作已经release的对象产生野指针错误。

2.ARC的实现原理

什么是ARC

Automatic Reference Counting，自动引用计数，即ARC，可以说是WWDC2011和iOS5所引入的最大的变革和最激动人心的变化。

 ARC 会自动帮助你添加 retain 和release/autorelease语句。

ARC编译器分为（1）前端编译器 （2）优化器  
（1）前端编译器 －－－－给“拥有的”每个对象插入相应的release语句。－－－－某个方法里面创建了一个对象，前端编译器会在方法末尾自动添加release语句销毁它。－－－－类拥有的对象（实例变量／属性）在dealloc方法内释放

（2）ARC优化器－－－当代码中出现多个对 retain 和release的重复调用，ARC优化器负责移出多余的 retain 和release语句。

如果需要持有一个对象，那么对其发送retain 如果之后不再使用该对象，那么需要对其发送release（或者autorealse） 每一次对retain,alloc，copy，mutable copy或者new的调用(自动引用计数＋1)，一次release或autorealse调用 (自动引用计数-1).

3.关于线程安全的问题

参考文章 线程安全类的设计

<http://www.cocoachina.com/industry/20131213/7535.html>

使用FMDB时候的线程安全注意点：

在多个线程中同时使用一个FMDatabase实例是不明智的。不要让多个线程分享同一个FMDatabase实例，它无法在多个线程中同时使用。 如果在多个线程中同时使用一个FMDatabase实例，会造成数据混乱等问题。所以，请使用 FMDatabaseQueue，它是线程安全的。以下是使用方法：

* 创建队列。

FMDatabaseQueue \*queue = [FMDatabaseQueue databaseQueueWithPath:aPath];

* 使用队列

[queue inDatabase:^(FMDatabase \*database) {

          [database executeUpdate:@"INSERT INTO t\_person(name, age) VALUES (?, ?)", @"Bourne\_1", [NSNumber numberWithInt:1]];

          [database executeUpdate:@"INSERT INTO t\_person(name, age) VALUES (?, ?)", @"Bourne\_2", [NSNumber numberWithInt:2]];

          [database executeUpdate:@"INSERT INTO t\_person(name, age) VALUES (?, ?)", @"Bourne\_3", [NSNumber numberWithInt:3]];

          FMResultSet \*result = [database executeQuery:@"select \* from t\_person"];

         while([result next]) {

         }

}];

而且可以轻松地把简单任务包装到事务里：

[queue inTransaction:^(FMDatabase \*database, BOOL \*rollback) {

          [database executeUpdate:@"INSERT INTO t\_person(name, age) VALUES (?, ?)", @"Bourne\_1", [NSNumber numberWithInt:1]];

          [database executeUpdate:@"INSERT INTO t\_person(name, age) VALUES (?, ?)", @"Bourne\_2", [NSNumber numberWithInt:2]];

          [database executeUpdate:@"INSERT INTO t\_person(name, age) VALUES (?, ?)", @"Bourne\_3", [NSNumber numberWithInt:3]];

          FMResultSet \*result = [database executeQuery:@"select \* from t\_person"];

             while([result next]) {

             }

           //回滚

           \*rollback = YES;

    }];

FMDatabaseQueue 后台会建立系列化的G-C-D队列，并执行你传给G-C-D队列的块。这意味着 你从多线程同时调用调用方法，GDC也会按它接收的块的顺序来执行。

4. cancel当前正在运行的NSOperation

参考文章 知其然亦知其所以然--NSOperation并发编程

http://www.cocoachina.com/game/20151201/14517.html

所谓的队列的取消操作方法cancelAllOperation只能取消还没有执行的任务，不能取消正在执行的任务。

比如100个操作已经执行到20个，后面还有80个未执行，此时如果调用cancelAllOperation取消之后，那么后面的80个任务以后也不会继续执行了。

但是第20个还会继续完成，不会中断

5.GCD的原理

GCD的工作原理是：让程序平行排队的特定任务，根据可用的处理资源，安排他们在任何可用的处理器核心上执行任务。

一个任务可以是一个函数(function)或者是一个block。 GCD的底层依然是用线程实现，不过这样可以让程序员不用关注实现的细节。

GCD中的FIFO队列称为dispatch queue，它可以保证先进来的任务先得到执行。

dispatch queue分为下面三种：

**Serial**

又称为private dispatch queues，同时只执行一个任务。Serial queue通常用于同步访问特定的资源或数据。当你创建多个Serial queue时，虽然它们各自是同步执行的，但Serial queue与Serial queue之间是并发执行的。

**Concurrent**

又称为global dispatch queue，可以并发地执行多个任务，但是执行完成的顺序是随机的。

**Main dispatch queue**

它是全局可用的serial queue，它是在应用程序主线程上执行任务的。

6.http状态码 自写http框架，缓存，异步，并发高性能解决方案

常见：

* 成功的状态码：
* 200 – 服务器成功返回网页
* 304 – 未修改   （自从上次请求后，请求的网页未被修改过。服务器返回此响应时，不会返回网页内容）
* 失败的状态码：
* 404 – 请求的网页不存在
* 503 – 服务器暂时不可用
* 500 – 服务器内部错误

参考文章 HTTP状态码详解

<http://blog.csdn.net/qiaqia609/article/details/45741149>

7.ios动画性能优化

参考文章

iOS-Core-Animation-Advanced-Techniques(六)

http://www.cocoachina.com/ios/20150106/10839.html

8.ios本地数据库如何高效率查询存储

参考文章 提高SQL查询效率

<http://www.cnblogs.com/hlxs/archive/2012/05/07/2487082.html>

http://www.2cto.com/database/201503/381295.html

9. 数据库索引的作用以及优缺点分析

<http://blog.csdn.net/mistygod/article/details/5706292>

10.block底层实现原理

参考文章 Block技巧与底层解析

<http://www.jianshu.com/p/51d04b7639f1>

11.ios 性能优化

http://blog.csdn.net/yangxt/article/details/8173412