数据存储

sqlite中插入特殊字符的方法和接收到处理方法。

除'其他的都是在特殊字符前面加“/”，而 ' -> " 。方法：keyWord = keyWord.replace("/","//");

什么是NSManagedObject模型?

NSManagedObjcet是NSObject的子类，Core Date的重要组成部分。是一个通用类，实现了Core Date模型层所需的基本功能，用户可以通过NSManagedObjcet建立自己的数据模型。

你实现过多线程的Core Data么？NSPersistentStoreCoordinator，NSManagedObjectContext和NSManagedObject中的哪些需要在线程中创建或者传递？你是用什么样的策略来实现的?

```
1>  CoreData是对SQLite数据库的封装
2>  coreData中有三个对象是必须掌握的，
NSManagedObject :只要定义一个类继承于该类就会创建一张与之对应的表，也就是一个继承于该类的类就对应一张表. 每一个

NSManagedObjectContext: 用于操作数据库，只要有类它就能对数据库的表进行增删改查

NSPersistentStoreCoordinator: 决定数据存储的位置 (SQLite/XML/其它文件中)
3>  Core data本身并不是一个并发安全的架构所以在多线程中实现Core data会有问题.问题在于
>2.1  CoreData中的NSManagedObjectContext在多线程中不安全
>2.2如果想要多线程访问CoreData的话，最好的方法是一个线程一个NSManagedObjectContext
>2.3每个NSManagedObjectContext对象实例都可以使用同一个NSPersistentStoreCoordinator实例，这是因为NSManagedOb
```

简单描述下客户端的缓存机制?

- 缓存可以分为：内存数据缓存、数据库缓存、文件缓存
    - 每次想获取数据的时候
    - 先检测内存中有无缓存
    - 再检测本地有无缓存(数据库\文件)
    - 最终发送网络请求
    - 将服务器返回的网络数据进行缓存（内存、数据库、文件）， 以便下次读取

什么是序列化和反序列化，用来做什么

- 序列化把对象转化为字节序列的过程
- 反序列化把直接序列恢复成对象
- 把对象写到文件或者数据库中，并且读取出来

OC中实现复杂对象的存储

- 遵循NSCoding协议，实现复杂对象的存储，实现该协议后可以对其进行打包或者解包，转化为NSDate

iOS中常用的数据存储方式有哪些?

1. 数据存储有四种方案，NSUserDefault,KeyChain,File,DB.
2. 其中File有三种方式：plist,Archiver,Stream
3. DB包括core Data和FMDB

说一说你对SQLite的认识

- SQLite是目前主流的嵌入式关系型数据库，其最主要的特点就是轻量级、跨平台，当前很多嵌入式操作系统都将其作为数据库首选。

- 虽然SQLite是一款轻型数据库，但是其功能也绝不亚于很多大型关系数据库。

- 学习数据库就要学习其相关的定义、操作、查询语言，也就是大家日常说得SQL语句。和其他数据库相比，SQLite中的SQL语法并没有太大的差别，因此这里对于SQL语句的内容不会过多赘述，大家可以参考SQLite中其他SQL相关的内容，这里还是重点讲解iOS中如何使用SQLite构建应用程序。先看一下SQLite数据库的几个特点：

    - 基于C语言开发的轻型数据库
    - 在iOS中需要使用C语言语法进行数据库操作、访问（无法使用ObjC直接访问，因为libqlite3框架基于C语言编写）
    - SQLite中采用的是动态数据类型，即使创建时定义了一种类型，在实际操作时也可以存储其他类型，但是推荐建库时使用合适的类型（特别是应用需要考虑跨平台的情况时）
    - 建立连接后通常不需要关闭连接（尽管可以手动关闭）

- 在iOS中操作SQLite数据库可以分为以下几步（注意先在项目中导入libsqlite3框架）：

    - 打开数据库，利用sqlite3_open()打开数据库会指定一个数据库文件保存路径，如果文件存在则直接打开，否则创建并打开。打开数据库会得到一个sqlite3类型的对象，后面需要借助这个对象进行其他操作。
    - 执行SQL语句，执行SQL语句又包括有返回值的语句和无返回值语句。
    - 对于无返回值的语句（如增加、删除、修改等）直接通过sqlite3_exec()函数执行；

- 对于有返回值的语句则首先通过sqlite3_prepare_v2()进行sql语句评估（语法检测），然后通过sqlite3_step()依次取出查询结果的每一行数据，对于每行数据都可以通过对应的sqlite3_column_类型()方法获得对应列的数据，如此反复循环直到遍历完成。当然，最后需要释放句柄。

说一说你对FMDB的认识

- FMDB是一个处理数据存储的第三方框架，框架是对sqlite的封装，整个框架非常轻量级但又不失灵活性，而且更加面向对象。FMDB有如下几个特性：
  - FMDB既然是对于libsqlite3框架的封装，自然使用起来也是类似的，使用前也要打开一个数据库，这个数据库文件存在则直接打开否则会创建并打开。这里FMDB引入了一个MFDatabase对象来表示数据库，打开数据库和后面的数据库操作全部依赖此对象。
  - 对于数据库的操作跟前面KCDbManager的封装是类似的，在FMDB中FMDatabase类提供了两个方法executeUpdate:和executeQuery:分别用于执行无返回结果的查询和有返回结果的查询。当然这两个方法有很多的重载这里就不详细解释了。唯一需要指出的是，如果调用有格式化参数的sql语句时，格式化符号使用“?”而不是“%@”、等。
  - 我们知道直接使用libsqlite3进行数据库操作其实是线程不安全的，如果遇到多个线程同时操作一个表的时候可能会发生意想不到的结果。为了解决这个问题建议在多线程中使用FMDatabaseQueue对象，相比FMDatabase而言，它是线程安全的。
  - 将事务放到FMDB中去说并不是因为只有FMDB才支持事务，而是因为FMDB将其封装成了几个方法来调用，不用自己写对应的sql而已。其实在在使用libsqlite3操作数据库时也是原生支持事务的（因为这里的事务是基于数据库的，FMDB还是使用的SQLite数据库），只要在执行sql语句前加上“begin transaction;”执行完之后执行“commit transaction;”或者“rollback transaction;”进行提交或回滚即可。另外在Core Data中大家也可以发现，所有的增、删、改操作之后必须调用上下文的保存方法，其实本身就提供了事务的支持，只要不调用保存方法，之前所有的操作是不会提交的。在FMDB中FMDatabase有beginTransaction、commit、rollback三个方法进行开启事务、提交事务和回滚事务。

说一说你对Core Data的认识

Core Data使用起来相对直接使用SQLite3的API而言更加的面向对象，操作过程通常分为以下几个步骤：

- 创建管理上下文
  创建管理上下可以细分为：加载模型文件->指定数据存储路径->创建对应数据类型的存储->创建管理对象上下方并指定存储。

经过这几个步骤之后可以得到管理对象上下文NSManagedObjectContext，以后所有的数据操作都由此对象负责。同时如果是第一次创建上下文，Core Data会自动创建存储文件（例如这里使用SQLite3存储），并且根据模型对象创建对应的表结构。

- 查询数据
  对于有条件的查询，在Core Data中是通过谓词来实现的。首先创建一个请求，然后设置请求条件，最后调用上下文执行请求的方法。
- 插入数据
  插入数据需要调用实体描述对象NSEntityDescription返回一个实体对象，然后设置对象属性，最后保存当前上下文即可。这里需要注意，增、删、改操作完最后必须调用管理对象上下文的保存方法，否则操作不会执行。
- 删除数据
  删除数据可以直接调用管理对象上下文的deleteObject方法，删除完保存上下文即可。注意，删除数据前必须先查询到对应对象。
- 修改数据
  修改数据首先也是取出对应的实体对象，然后通过修改对象的属性，最后保存上下文。

OC中有哪些数据存储方式,各有什么区别?

- OC中有四种数据存储方式:
  - NSUserDefaults,用于存储配置信息
  - SQLite,用于存储查询需求较多的数据
  - CoreData,用于规划应用中的对象
- 使用基本对象类型定制的个性化缓存方案.
  - NSUserDefaults:对象中储存了系统中用户的配置信息,开发者可以通过这个实例对象对这些已有的信息进行修改,也 可以按照自己的需求创建新的配置项。
  - SQLite擅长处理的数据类型其实与NSUserDefaults差不多,也是基础类型的小数据,只是从组织形式上不同。开发者可以以关系型数据库的方式组织数据,使用SQL DML来管理数据。一般来说应用中的格式化的文本类数据可以存放在数据库 中,尤其是类似聊天记录、Timeline等这些具有条件查询和排序需求的数据。
  - CoreData是一个管理方案,它的持久化可以通过SQLite、XML或二进制文件储存。它可以把整个应用中的对象建模并进行自动化的管理。从归档文件还原模型时CoreData并不是一次性把整个模型中的所有数据都载入内存,而是根据运行时状态,把被调用到的对象实例载入内存。框架会自动控制这个过程,从而达到控制内存消耗,避免浪费。 无论从设计原理还是使用方法上

看,CoreData都比较复杂。因此,如果仅仅是考虑缓存数据这个需求,CoreData绝对不是一个优选方案。

- CoreData的使用场景在于:整个应用使用CoreData规划,把应用内的数据通过CoreData建模,完全基于CoreData架构应用。
- 使用基本对象类型定制的个性化缓存方案:从需求出发分析缓存数据有哪些要求:按Key查找,快速读取,写入不影响正常操作,不浪费内存,支持归档。这些都是基本需求,那么再进一步或许还需要固定缓存项数量,支持队列缓存,缓存过期等。

数据存储这一块, 面试常问, 你常用哪一种数据存储? 什么是序列化? sqlite是直接用它还是用封装了它的第三方库? 尤其是会问sqlite和core data的区别?

iOS平台怎么做数据的持久化?coredata和sqlite有无必然联系?coredata是一个关系型数据库吗?

- iOS中可以有四种持久化数据的方式: 属性列表、对象归档、SQLite3和Core Data
- coredata可以使你以图形界面的方式快速的定义app的数据模型,同时在你的代码中容易获取到它。
- coredata提供了基础结构去处理常用的功能,例如保存,恢复,撤销和重做,允许你在app中继续创建新的任务。
- 在使用coredata的时候,你不用安装额外的数据库系统,因为coredata使用内置的sqlite数据库。
- coredata将你app的模型层放入到一组定义在内存中的数据对象。
- coredata会 追踪这些对象的改变,同时可以根据需要做相应的改变,例如用户执行撤销命令。
- 当coredata在对你app数据的改变进行保存的时 候,core data会把这些数据归档,并永久性保存。
- mac os x中sqlite库,它是一个轻量级功能强大的关系数据引擎,也很容易嵌入到应用程序。可以在多个平台使用,sqlite是一个轻 量级的嵌入式sql数据库编程。
- 与coredata框架不同的是,sqlite是使用程序式的,sql的主要的API来直接操作数据表。
- Core Data不是一个关系型数据库,也不是关系型数据库管理系统(RDBMS)。
- 虽然Core Dta支持SQLite作为一种存储类型, 但它不能使用任意的SQLite数据库。

- Core Data在使用的过程种自己创建这个数据库。Core Data支持对一、对多的关系。

如果后期需要增加数据库中的字段怎么实现，如果不使用CoreData呢?

- 编写SQL语句来操作原来表中的字段
- 增加表字段：ALTER TABLE 表名 ADD COLUMN 字段名 字段类型;
- 删除表字段：ALTER TABLE 表名 DROP COLUMN 字段名;
- 修改表字段：ALTER TABLE 表名 RENAME COLUMN 旧字段名 TO 新字段名;

SQLite数据存储是怎么用?

- 添加SQLite动态库：导入主头文件：#import <sqlite3.h>
- 利用C语言函数创建\打开数据库，编写SQL语句

简单描述下客户端的缓存机制?

- 缓存可以分为：内存数据缓存、数据库缓存、文件缓存
- 每次想获取数据的时候
- 先检测内存中有无缓存
- 再检测本地有无缓存(数据库\文件)
- 最终发送网络请求
- 将服务器返回的网络数据进行缓存（内存、数据库、文件）以便下次读取

你实现过多线程的Core Data么？NSPersistentStoreCoordinator，NSManagedObjectContext和NSManagedObject中的哪些需要在线程中创建或者传递？你是用什么样的策略来实现的?

- CoreData是对SQLite数据库的封装
- CoreData中的NSManagedObjectContext在多线程中不安全
- 如果想要多线程访问CoreData的话，最好的方法是一个线程一个NSManagedObjectContext
- 每个NSManagedObjectContext对象实例都可以使用同一个NSPersistentStoreCoordinator实例，这是因为NSManagedObjectContext会在便用NSPersistentStoreCoordinator前上锁

Core Data数据迁移

博客地址: http://blog.csdn.net/jasonblog/article/details/17842535

FMDB的使用和对多张表的处理

博客地址: http://blog.csdn.net/wscqqlucy/article/details/8464398

说说数据库的左连接和右连接的区别

- 数据库左连接和右连接的区别：主表不一样通过左连接和右连接，最小条数为3（记录条数较小的记录数），最大条数为12（3×4）

- 技术博客的地址 : http://www.2cto.com/database/201407/317367.html

iOS 的沙盒目录结构是怎样的？ App Bundle 里面都有什么?

1. 沙盒结构

    - Application：存放程序源文件，上架前经过数字签名，上架后不可修改
    - Documents：常用目录，iCloud备份目录，存放数据,这里不能存缓存文件,否则上架不被通过
    - Library
        - Caches：存放体积大又不需要备份的数据,SDWebImage缓存路径就是这个
        - Preference：设置目录，iCloud会备份设置信息
    - tmp：存放临时文件，不会被备份，而且这个文件下的数据有可能随时被清除的可能

2. App Bundle 里面有什么

    - Info.plist:此文件包含了应用程序的配置信息.系统依赖此文件以获取应用程序的相关信息
    - 可执行文件:此文件包含应用程序的入口和通过静态连接到应用程序target的代码
    - 资源文件:图片,声音文件一类的
    - 其他:可以嵌入定制的数据资源

你会如何存储用户的一些敏感信息，如登录的 token

使用keychain来存储,也就是钥匙串,使用keychain需要导入Security框架

```
自定义一个keychain的类
#import <Security/Security.h>
@implementation YCKKeyChain

+(NSMutableDictionary *)getKeychainQuery:(NSString *)service {
return [NSMutableDictionary dictionaryWithObjectsAndKeys:
        (__bridge_transfer id)kSecClassGenericPassword,(__bridge_transfer   id)kSecClass,
        service, (__bridge_transfer id)kSecAttrService,
        service, (__bridge_transfer id)kSecAttrAccount,
        (__bridge_transfer id)kSecAttrAccessibleAfterFirstUnlock,   (__bridge_transfer id)kSecAttrAccess
```

```
        nil];
}

+(void)save:(NSString *)service data:(id)data {
// 获得搜索字典
NSMutableDictionary *keychainQuery = [self getKeychainQuery:service];
// 添加新的删除旧的
SecItemDelete((__bridge_retained CFDictionaryRef)keychainQuery);
// 添加新的对象到字符串
[keychainQuery setObject:[NSKeyedArchiver archivedDataWithRootObject:data]  forKey:(__bridge_transfer id
// 查询钥匙串
SecItemAdd((__bridge_retained CFDictionaryRef)keychainQuery, NULL);
}

+(id)load:(NSString *)service {
id ret = nil;
NSMutableDictionary *keychainQuery = [self getKeychainQuery:service];
// 配置搜索设置
[keychainQuery setObject:(id)kCFBooleanTrue forKey:(__bridge_transfer   id)kSecReturnData];
[keychainQuery setObject:(__bridge_transfer id)kSecMatchLimitOne forKey:    (__bridge_transfer id)kSecMa
CFDataRef keyData = NULL;
if (SecItemCopyMatching((__bridge_retained CFDictionaryRef)keychainQuery, (CFTypeRef *)&keyData) == noEr
    @try {
        ret = [NSKeyedUnarchiver unarchiveObjectWithData:(__bridge_transfer NSData *)keyData];
    } @catch (NSException *e) {
        NSLog(@"Unarchive of %@ failed: %@", service, e);
    } @finally {
    }
}
return ret;
}

+(void)delete:(NSString *)service {
NSMutableDictionary *keychainQuery = [self getKeychainQuery:service];
SecItemDelete((__bridge_retained CFDictionaryRef)keychainQuery);
}
```

在别的类实现存储,加载,删除敏感信息方法
// 用来标识这个钥匙串
static NSString * const KEY_IN_KEYCHAIN = @"com.yck.app.allinfo";
// 用来标识密码
static NSString * const KEY_PASSWORD = @"com.yck.app.password";

```
+(void)savePassWord:(NSString *)password
{
NSMutableDictionary *passwordDict = [NSMutableDictionary dictionary];
[passwordDict setObject:password forKey:KEY_PASSWORD];
[YCKKeyChain save:KEY_IN_KEYCHAIN data:passwordDict];
}

+(id)readPassWord
{
NSMutableDictionary *passwordDict = (NSMutableDictionary *)[YCKKeyChain     load:KEY_IN_KEYCHAIN];
return [passwordDict objectForKey:KEY_PASSWORD];
}

+(void)deletePassWord
```

```
{
[YCKKeyChain delete:KEY_IN_KEYCHAIN];
}
```

使用 NSUserDefaults 时，如何处理布尔的默认值？(比如返回 NO，不知道是真的 NO 还是没有设置过)

```
if([[NSUserDefaults standardUserDefaults] objectForKey:ID] == nil){
    NSLog(@"没有设置");
}
```

MD5和Base64的区别是什么，各自使用场景是什么？

做过加密相关的功能的，几乎都会使用到MD5和Base64，它们两者在实际开发中是最常用的。

- MD5：是一种不可逆的摘要算法，用于生成摘要，无法逆着破解得到原文。常用的是生成32位摘要，用于验证数据的有效性。
- Base64：属于加密算法，是可逆的，经过encode后，可以decode得到原文。在开发中，有的公司上传图片采用的是将图片转

plist文件是用来做什么的。一般用它来处理一些什么方面的问题。

- plist是iOS系统中特有的文件格式。我们常用的NSUserDefaults偏好设置实质上就是plist文件操作。plist文件是用来持久化存储数据的。
- 我们通常使用它来存储偏好设置，以及那些少量的、数组结构比较复杂的不适合存储数据库的数据。比如，我们要存储全国城市名称和id，那么我们要优先选择plist直接持久化存储，因为更简单。

当存储大块数据是怎么做?

- 你有很多选择，比如：
- 使用NSUerDefaults
- 使用XML, JSON, 或者 plist
- 使用NSCoding存档
- 使用类似SQLite的本地SQL数据库
- 使用 Core Data
- NSUserDefaults的问题是什么？虽然它很nice也很便捷，但是它只适用于小数据，比如一些简单的布尔型的设置选项，再大点你就要考虑其它方式了
- XML这种结构化档案呢？总体来说，你需要读取整个文件到内存里去解析，这样是很不经济的。使用SAX又是一个很麻烦的事情。
- NSCoding？不幸的是，它也需要读写文件，所以也有以上问题。

- 在这种应用场景下，使用SQLite 或者 Core Data比较好。使用这些技术你用特定的查询语句就能只加载你需要的对象。在性能层面来讲，SQLite和Core Data是很相似的。他们的不同在于具体使用方法。Core Data代表一个对象的graph model，但SQLite就是一个DBMS。Apple在一般情况下建议使用Core Data，但是如果你有理由不使用它，那么就去使用更加底层的SQLite吧。如果你使用SQLite，你可以用FMDB(https://GitHub.com/ccgus/fmdb)这个库来简化SQLite的操作，这样你就不用花很多经历了解SQLite的C API了

怎么解决sqlite锁定的问题

```
1>  设置数据库锁定的处理函数
int sqlite3_busy_handler(sqlite3*, int(*)(void*,int), void*);
函数可以定义一个回调函数，当出现数据库忙时，sqlite会调用该函数
当回调函数为ＮＵＬＬ时，清除busy handle，申请不到锁直接返回
回调函数的第二个函数会被传递为该由此次忙事件调用该函数的次数
回调函数返回非 0 ,数据库会重试当前操作，返回 0 则当前操作返回SQLITE_BUSY

2>  设定锁定时的等待时间
int sqlite3_busy_timeout(sqlite3*, 60);
定义一个毫秒数，当未到达该毫秒数时，sqlite会sleep并重试当前操作
如果超过ms毫秒，仍然申请不到需要的锁，当前操作返回sqlite_BUSY
当ms<=0时，清除busy handle，申请不到锁直接返回
```