数据持久化

1. 简介：

iOS中数据持久化方式，基本上有以下几种：1.属性列表 2.用户偏好 3.对象归档 4.SQLite3 5.Core Data，下面将分别对这几种的具体实现进行讲解。

1. 关于应用沙盒：
2. 简介：每个iOS应用都有自己的应用沙盒(即文件系统目录)，与其他文件系统隔离，应用必须待在自己的沙盒里，其他应用不能访问该沙盒。
3. 应用沙盒目录：
4. 应用程序包app：包含了所有资源文件和可执行文件，上架前经过数字签名，上架后不可修改
5. Documents：保存应用运行时生成的需要持久化的数据，如sqlite、coredata数据、游戏存档，不要保存从网络上加载的文件，否者无法上架！
6. Library目录：这目录下有Caches和Preferences

Preferences：包含应用程序便好文件。不应直接创建偏好设置文件，而应使用NSUserDefaults类来设置应用程序的偏好

Caches目录：用于存放体积大又不需要备份的数据，如图片、mp3文件、离线地图等，必须提供cache目录清理的解决方案

1. tmp目录：存放临时文件，不会被备份，而且这文件夹下数据可能随时被清除，保存后续不需要使用的文件，系统会自动处理，重启手机tmp目录会清空，系统磁盘空间不足时，系统也会自动处理
2. 常见沙盒目录获取方式
3. 沙盒根目录

NSSring \*home = NSHomeDirectory();

1. Document目录

方式一：利用沙盒根目录拼接目录名，不建议，由于新版本可能改目录名

方式二：利用NSSearchPathForDirectoriesInDomains

1. tmp临时目录

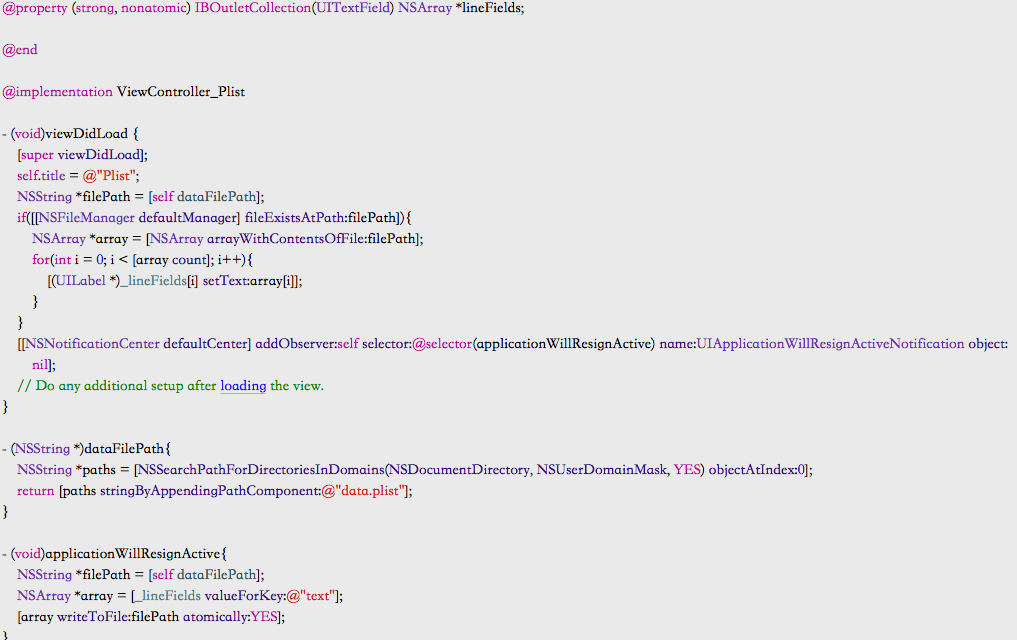
NSString \*tmp = NSTmporaryDirectory();

1. Library/Caches目录

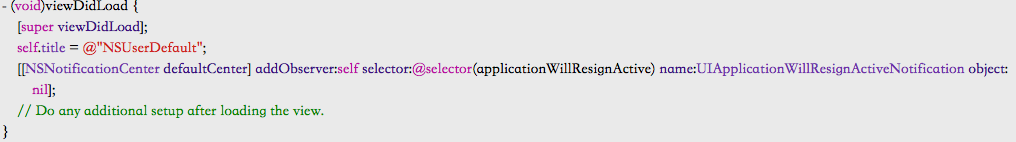
利用NSSearchPathForDirectoriesInDomains，第二个参数由NSDocumenDirecory改为NSCacheDirectory即可

1. 属性列表plist
2. 简介：可保存一些可序列化类，方便保存和加载数据。可序列化存储类:NSArray、NSDictionary、NSData、NSString、NSDate、NSNumber。
3. 缺点：无法将自定义对象序列化到plist；无法存储其他类，如UIColor、NSURL等；目前只能将一小部分对象存储在属性列表中

(3)示例：

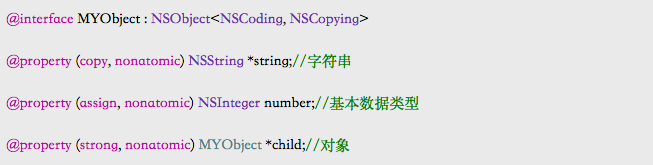


1. 用户偏好NSUserDefaults
2. 简介：本质也是一个plist文件，也是只可保存一些可序列化存储类，比较方便读写
3. 示例：

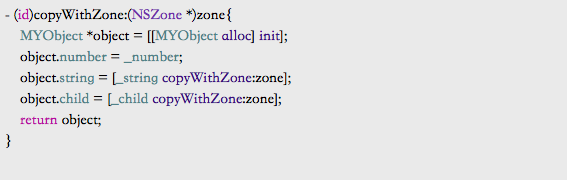




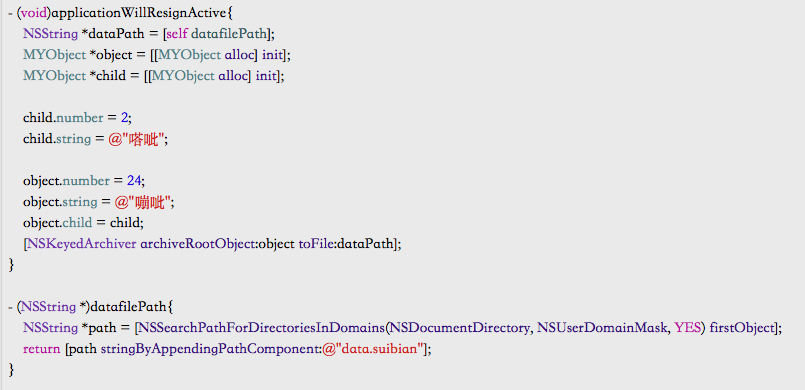
1. 模型对象归档NSKeyedArchiver
2. 简介：可以将对象存储到文件中(可任意拓展名保存)
3. 相比CoreData不能查询，不适合大型或复杂对象处理，存取速较慢。即使很多方面应用可能受益于CoreData，但就持久化而言，CoreData比较复杂。
4. 示例：



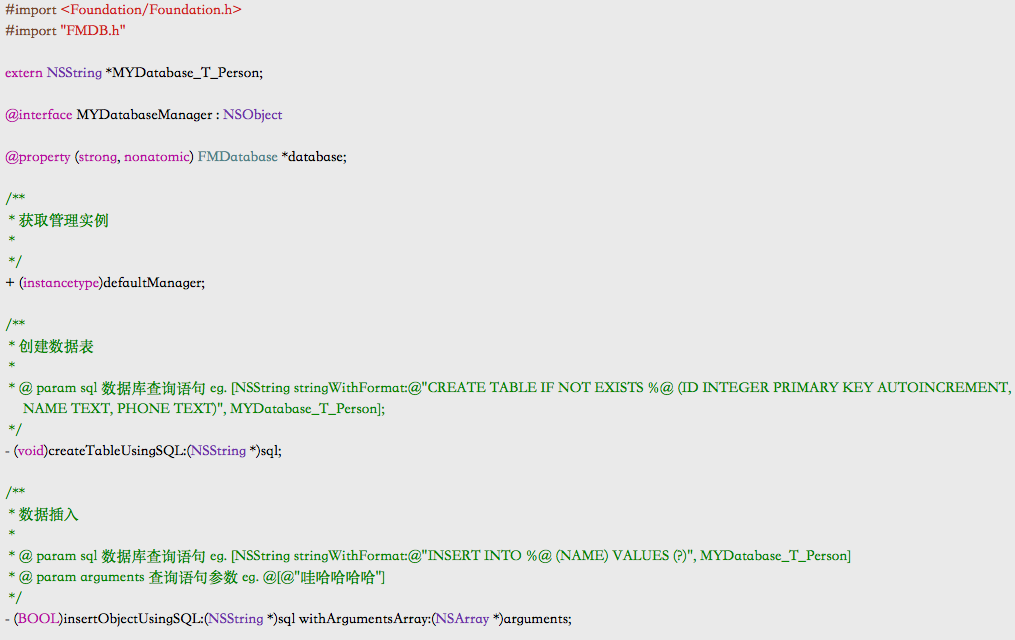




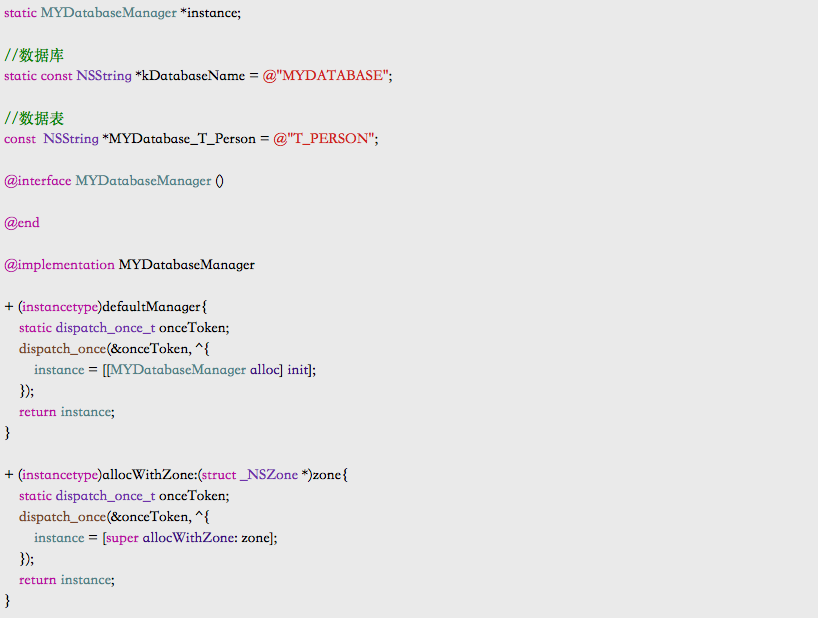




1. SQLite3之FMDB框架
2. 简介：sqlite是一个开源嵌入式关系数据库，可移植性好，易使用，很小，高效可靠。只是不建议直接操作sqlite库，而采用一些开源第三发库，如FMDB，对SQLite做了不错的封装
3. 示例：



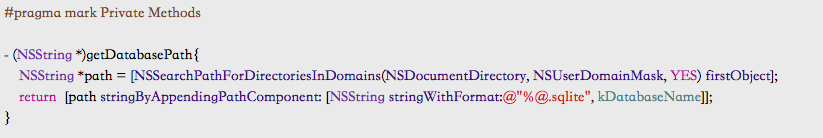


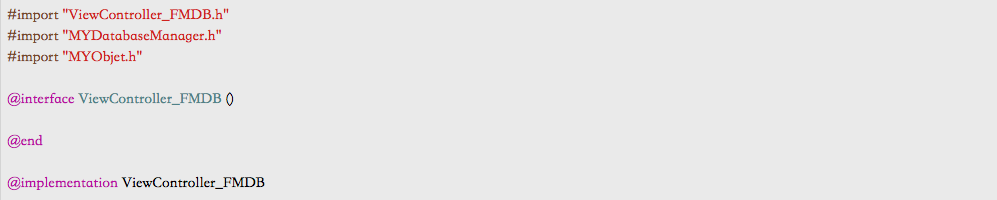














1. CoreData框架

在CoreData之前，创建数据模型的传统方式是创建NSObject的子类，并让其遵循NSCoding和NSCopying协议，以便进行归档。CoreData不需要创建类，而是先在数据模型编辑器中创建一些实体，然后在代码中为这些实体创建托管对象。实体与托管对象就相当于类与类的实例。

1. 实体由属性构成，属性分3种类型
2. 特性：如同实例变量的作用，用于保存数据
3. 关系：用于定义实体间的关系，一对一或一对多
4. 提取属性：关系的备选方法，如一个person对象可以有一个neighbor的提取属性，用于查找数据存储中与这个person的HomeAddress中拥有相同邮编的所有homeAddress对象，提取属性也是一种能跨越多个数据存储的关系。
5. 键-值编码

从托管对象中检索存储在name特性中的值，需调用：

let name = myManagedObject.valueForKey(“name”)

为托管对象属性设置新值，可执行：

myManagedObject.setValue(“fdafadsf”,forKey:”name”)

1. 在上下文中结合
2. 这些托管对象活动区域在持久存储中。默认情况下，CoreData应用将支持存储实现为存储在Documents的SQLite数据库。但CoreData框架中的类将完成加载和保存数据的所有操作。你只需要操作对象，内部工作由CoreData完成。虽然还支持其他方式如二进制文件、xml等，在几乎所有情况下，还是应采用默认设置，并使用SQLite作为持久存储
3. 除了创建持久化存储外(应用委托中实现)，我们通常不会直接操作持久存储，而使用所谓的托管对象上下文。上下文协调对持久存储的访问，同时保存自上次保存对象以来修改过的的属性信息。上下文还支持通过撤销管理器来注册所有更改，意味着可以撤销或回滚至上一次数据。可将多个上下文指向同一个持久存储，但一般应用只会用一个。
4. 许多核心数据调用都需要NSManagedObjectContext作为参数，或需在上下文中执行。除了一些更复杂、多线程的应用外，应用委托中都可只使用managedObjectContext属性，它是xcode项目模板自动为应用创建的上下文。除托管对象上下文和持久存储协调者外，所提供应用委托还包含一个NSManagedObjectModel实例，该类负责运行时加载和表示使用xcode中数据模型编辑器创建的数据模型。通常，不需直接与该类进行交互。该类由其他CoreData类在后台使用，因此可确定数据模型中定义了哪些实体和属性。只要使用所提供的文件创建数据模型，就完全不需要担心这个类。
5. 创建新的托管对象

创建托管对象新实例使用NSEntityDescription类中的insertNewObjectForEntityForName(\_,inManagedObjectContext:)工厂方法。NSEntityDescription的工作是跟踪在应用的数据模型中定义的实体，并让你创建这些实体的实例。此方法创建并返回一个实例，表示内存中的单个实体的正确属性设置的NSManagedObject实例；如果将实体配置为使用NSManagedObject的子类为实现，则返回该子类的实例。记住！实体类似于类。实体是对象的描述，用于定义特定的实体拥有哪些属性。

Let thing = NSEntityDescription.insertNewObjectForEntityForName(“” inManagedObjectContext:);

此方法除创建管理对象，还负责把对象插入上下文，返回这个对象。此时对象位于上下文中，但还不是持久存储中的一部分，下一次托管对象上下文的saveContext方法调用时，此对象将被添加到持久存储内。

1. 获取托管对象

要从持久存储中获取托管对象，可使用fetch请求，这是CoreData处理预定义查询的方式。例如：可要求返回所有eyesColor为blue的person对象

首先，创建获取请求后，为其指定一个NSEntityDescription，指定希望检索的一个或多个对象实体

let request = NSFetchRequest();

let entityDescription = NSEntityDescription.entityForName(“Thing”,inManagedObjectContext:);

request.entity = entityDescription;

也可以使用NSPredicate类为获取请求指定条件。谓词类似SQL的where语句

let pred = NSPredicate(format:”name=%@”, argumentArray:nameString);

request.predicate = pred;

表示：仅需获取哪些name属性为name string值的托管对象

可使用NSManagedObjectContext中的实例方法来执行获取请求

do{

let objects = try context.executeFetchRequest(request)

if object.count == 0

return;

for oneObject in objects{

//对象操作，结束后记得saveContext()  
}

}catch{

NSLog(“error”, [])  
}

1. CoreData增删改查
2. 添加操作

var person:NSManagedObject!=nil

person = NSEntityDescription.insertNewObjectForEntityForName(entityName,inManagedObjectContext:context) as NSManagedObject

person,.setValue(value, forKey:key);

context.saveContext();

1. 删除操作

let request = NSFetchReqeust(entityName:entityName)

do{

let objects = try context.executeFetchRequest(request)

if objects.count == 0{  
return

for oneObject in objects{

context.deleteObject(oneObject as! NSManagedObject)

}

context.saveContext()  
}catch{

NSLog(“error”,[])

}

1. 修改操作、查询操作

letpredic = NSPredicate(format:”name=%@”,”abc”);

let request = NSFetchRequest(entityName:entityName)

request.predicate = predict

do{

let objects = try context.executeFetchRequest(request)

if objects.count == 0{  
return

for oneObject in objects{

oneObject.setValue(“13123323395”,forKey:phoneKey)

}

context.saveContext()  
}catch{

NSLog(“error”,[])

}

1. CoreData管理类封装



