会写代码的猪

没谱青年，资深80后，大爱加菲猫，不炫技会死！

对比iOS中的四种数据存储

作者：gaosboy | 时间：April 2, 2013 | 分类：猪在写代码

本文首发于InfoQ中文站

你是用什么方法来持久保存数据的？这是在几乎每一次关于iOS技术的交流或讨论都会被提到的问题，而且大家对这个问题的热情持续高涨。本文主要从概念上把“数据存储”这个问题进行剖析，并且结合各自特点和适用场景给大家提供一个选择的思路，并不详细介绍某一种方式的技术细节。

谈到数据储存，首先要明确区分两个概念，数据结构和储存方式。所谓数据结构就是数据存在的形式。除了基本的NSDictionary、NSArray和NSSet这些对象，还有更复杂的如：关系模型、对象图和属性列表多种结构。而存储方式则简单的分为两种：内存与闪存。内存存储是临时的，运行时有效的，但效率高，而闪存则是一种持久化存储，但产生I/O消耗，效率相对低。把内存数据转移到闪存中进行持久化的操作称成为归档。

二者结合起来才是完整的数据存储方案，我们最常谈起的那些：SQLite、CoreData、NSUserDefaults等都是数据存储方案。当然在这些框架提供的方案之外，我们自己也可以按照个性化需求订制方案。这些存储方案侧重不同，支持的形式和方式也各不相同，在不同的使用场景下表现也是各有优劣。但万变不离其宗，无论什么方案都可以用下图来解释。

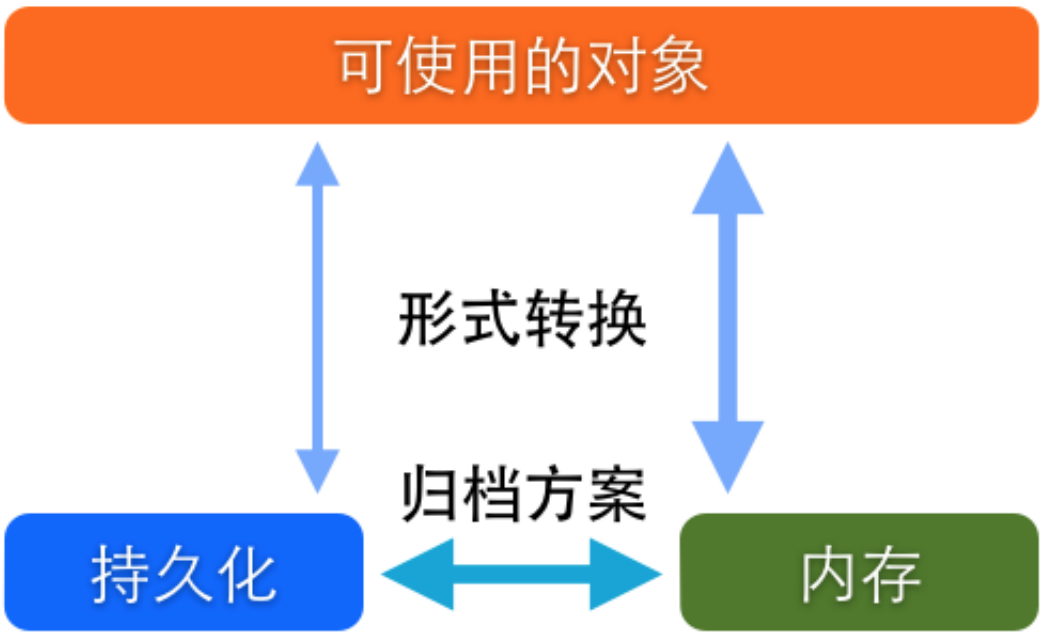

图1，存储方案示意图

以下将对四种存储方式进行详细的介绍：

- 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

用NSUserDefaults存储配置信息

NSUserDefaults被设计用来存储设备和应用的配置信息，它通过一个工厂方法返回默认的、也是最常用到的实例对象。这个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。

```
{
    ATOutputLevel = 7;
    AppleICUForce24HourTime = 1;
    AppleITunesStoreItemKinds =     (
        eBook, document, "software-update", booklet, "itunes-u",
        newsstand, artist, podcast, "podcast-episode", software
    );
    AppleKeyboards =     (
        "en_US@hw=US;sw=QWERTY",
        "zh_Hans-Pinyin@sw=Pinyin;hw=US",
        "emoji@sw=Emoji"
    );
    AppleKeyboardsExpanded = 1;
    AppleLanguages =     ( en, "zh-Hans", fr, de, ja, nl, it, es,
        pt, "pt-PT", da, fi, nb, sv, ko, "zh-Hant", ru, pl, tr, uk,
        ar, hr, cs, el, he, ro, sk, th, id, ms, "en-GB", ca, hu, vi
    );
    AppleLocale = "en_US";
    NSInterfaceStyle = macintosh;
    NSLanguages =     ( en, "zh-Hans", fr, de, ja, nl, it, es, pt,
        "pt-PT", da, fi, nb, sv, ko, "zh-Hant", ru, pl, tr, uk, ar,
        hr, cs, el, he, ro, sk, th, id, ms, "en-GB", ca, hu, vi
    );
    key = alkdjfkladsjfmm;
}
```

图2，笔者手机中[NSUserDefaults standardUserDefaults]内容

NSUserDefaults把配置信息以字典的形式组织起来，支持字典的项包括：字符串或者是数组，除此之外还支持数字等基本格式。一句话概括就是：基础类型的小数据的字典。操作方法几乎与NSDictionary的操作方法无异，另外还可以通过指定返回类型的方法获取到指定类型的返回值。

```
- (NSString *)stringForKey:(NSString *)defaultName;
- (NSArray *)arrayForKey:(NSString *)defaultName;
- (NSDictionary *)dictionaryForKey:(NSString *)defaultName;
- (NSData *)dataForKey:(NSString *)defaultName;
- (NSArray *)stringArrayForKey:(NSString *)defaultName;
- (NSInteger)integerForKey:(NSString *)defaultName;
- (float)floatForKey:(NSString *)defaultName;
- (double)doubleForKey:(NSString *)defaultName;
- (BOOL)boolForKey:(NSString *)defaultName;
```

图3，NSUserDefaults提供的指定返回类型的方法列表

NSUserDefaults的所有数据都放在内存里，因此操作速度很快，并还提供一个归档方法：+ (void)synchronize。开发者自定义的配置项（如图2中的最后一项 key:alkdjfkladsjfmm）会以plist格式的文件归档在相应应用目录的/Library/Preferences/[App_Bundle_Identifier].plist文件。再次初始化获得实例对象后，框架会把用户自定义的这个配置和系统配置合并得到完整数据。

用SQLite存储查询需求较多的数据

iOS的SDK里预置了SQLite的库，开发者可以自建SQLite数据库。SQLite每次写入数据都会产生IO消耗，把数据归档到相应的文件。

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。

每一个数据库的句柄都会在内存中都会被分配一段缓存，用于提高查询效率。另一个方面，由于查询缓存，当产生大量句柄或数据量较大时，会出现缓存过大，造成内存浪费。

SQLite的使用起来要比NSUserDefaults复杂的多，因此建议开发者使用SQLite要搭配一个操作控件使用，可以简化操作。笔者开发的SQLight是一款对SQLite操作的封装，把相对复杂的SQLite命令封装成对象和方法，可以供大家参考。大家可以在Github上获取这个工程的代码进一步了解。

用CoreData规划应用中对象

官方给出的定义是，一个支持持久化的，对象图和生命周期的自动化管理方案。严格意义上说CoreData是一个管理方案，他的持久化可以通过SQLite、XML或二进制文件储存。如官方定义所说，CoreData的作用远远不止储存数据这么简单，它可以把整个应用中的对象建模并进行自动化的管理。

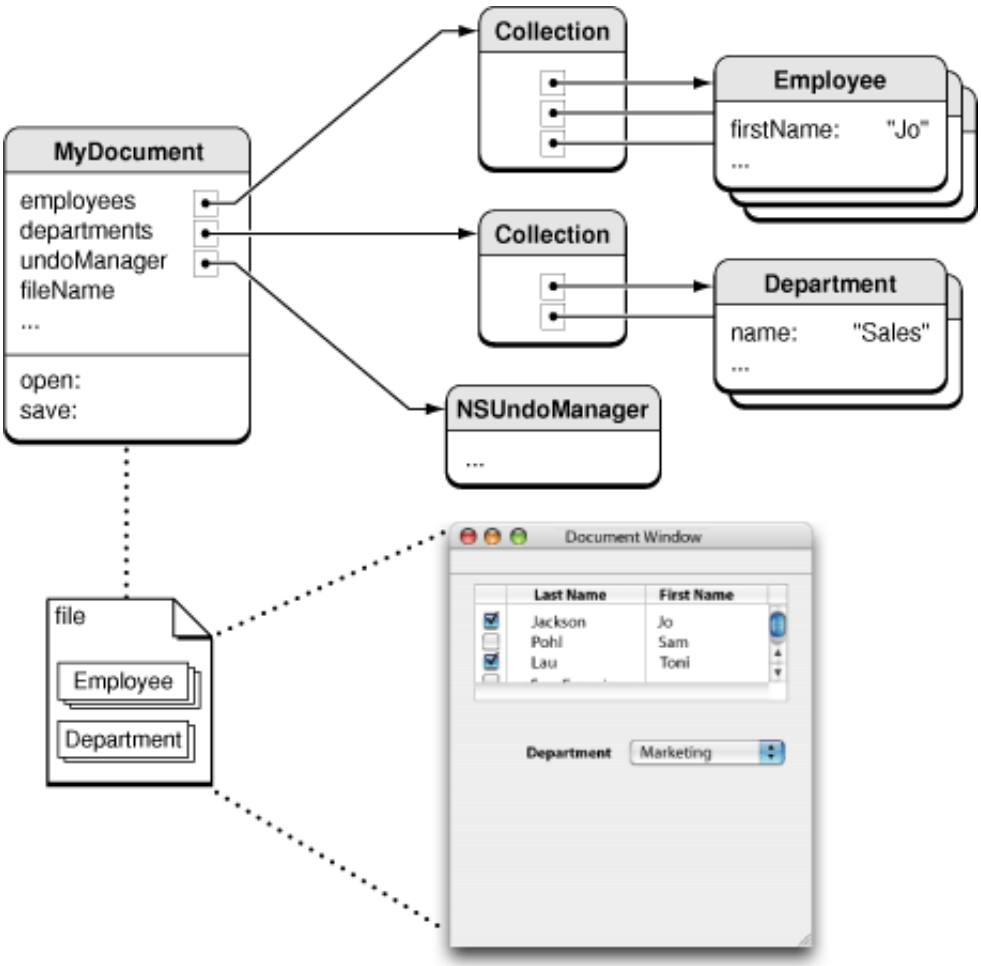

图4，官方文档中解释CoreData给出的对象图示例

正如上图所示，MyDocument是一个对象实例，有两个Collection：Employee和Department，存放各自的对象列表。MyDocument、Employee和Department三个对象以及他们之间的关系都通过CoreData建模，并可以通过save方法进行持久化。

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。

苹果官方给出的一个示例代码，结构相对简单，可以帮助大家入门CoreData。

使用基本对象类型定制的个性化缓存方案

之前提到的NSUserDefaults和SQLite适合存储基础类型的小数据，而CoreData则不适合存储单一的数据，那么对于类似图片这种较大的数据要用什么方式储存呢？我给出的建议就是：自己实现一套存储方案。说到订制存储方案大家非常容易质疑，这是不是又在重新发明轮子。我可以非常明确的告诉大家，这绝不是在重新发明轮子。首先要明确，这个所谓的定制方案适用于互联网应用中对远程数据的缓存，几个限制条件缺一不可。

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。从这些需求入手设计一个缓存方案并不十分复杂，Kache是笔者根据开发应用的需求开发的一套缓存组件，通过分析Kache希望可以给大家一个思路。

图5，Kache架构图

如上图所示，Kache扮演的是一个典型缓存角色。应用加载远程数据生成应用数据对象的同时，通过Kache把数据缓存起来，再次请求则直接通过Kache获取数据。

缓存对象可以是NSDictionary、NSArray、NSSet或NSData这些可直接归档的类型，每个缓存对象对应一个Key。缓存对象包括数据和过期时间，内存中存放在一个单例字典中，闪存中每个对象存为一个文件。Key空间按照各种顺序存放缓存对象的Key集合，Pool为固定大小的数组，当数量达到上限，最早过期的一个Key将被删除，对应的缓存对象也被清除。Queue也是固定大小的数组，以先进先出的规则管理Key的增删。 每一次有新的缓存对象存入，自动检测Key空间中过期的集合并清除。

此外，控件提供save和load方法支持持久化和重新载入。

Kache最初设计为存放图片缓存，之后也曾用于缓存文本数据，由于使用了过期和归档相结合的逻辑，可以保证大部分命中的缓存对象都在内存中，从而获取了较高的效率。读者可以从Github上获取Kache源码了解更多。

以上介绍了几种iOS开发中经常会遇到的储存数据方法，从其存储原理、使用方式和适用场景几方面进行进了简单的对比。事实上每一款应用都很难采用一种单一的方案完成整个应用的数据储存任务，需要根据不同的数据类型，选择最合适的方案，以便整个应用获得良好的运行时性能。

标签：ios

添加新评论

称呼 *

Email *

网站

http://

内容 *

提交评论