# 第十四章 数据的持久化

数据持久化就是将内存中的数据模型转换为存储模型，以及将存储模型转换为内存中的数据模型的统称.。应用进程的许多数据需要存储在硬盘中进行持久化存储，如游戏进度存档、用户历史信息等。

使用数据持久化的好处：

1、持久化技术封装了数据访问细节，为大部分业务逻辑提供[面向对象](http://www.so.com/s?q=%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1&ie=utf-8&src=internal_wenda_recommend_text" \t "/Users/ADMIN\\x/_blank)的API。

2、通过持久化技术可以减少访问数据库数据次数，增加应用程序执行速度。

3、代码重用性高，能够完成大部分数据库操作。

4、松散耦合，使持久化不依赖于底层数据库和上层业务逻辑实现，更换数据库时只需修改配置文件而不用修改代码。

iOS中使用的数据持久化方式有：1、property list（属性列表）2、Preference（偏好设置）3、NSKeyedArchiver（归档）4、SQLite3/FMDB(嵌入式数据库) 5、CoreData(面向对象的嵌入式数据库)

本章主要介绍对property list的创建、读取、添加、修改、删除和写入以及如何对CoreData进行数据的存储、查找、编辑和删除操作。

## 14.1 plist文件

plist文件全名为property list，即属性列表文件。它以类似于键值对(key-value)的形式来存储项目的一些属性或配置信息。在新建一个项目时，项目文件夹中会自动包含一个名为[info.plist]文件，该文件存储了该项目的各项配置信息，包括该项目版本号、语言环境、主故事版等信息的配置。plist文件的存储格式为xml格式。plist文件内容及其xml源代码如图所示。

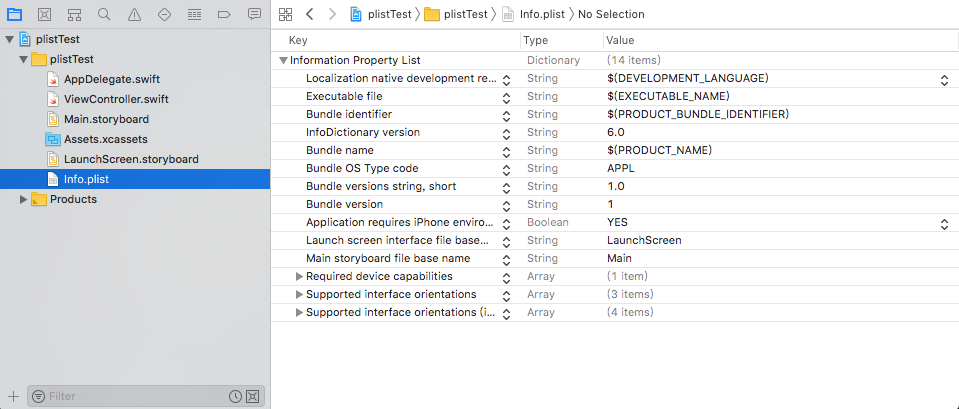


图1 plist文件

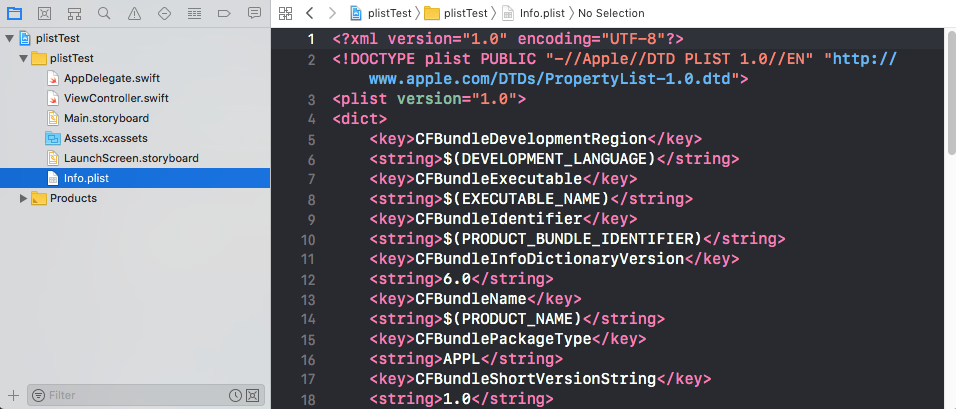


图2 plist文件xml源代码

### 14.1.1 plist文件创建和解析

创建一个基于【Single View Application】模板的空项目，创建完毕后在项目名称文件夹上点击鼠标右键，在弹出的菜单栏中选择【New File】，接着在弹出的模板窗口中在iOS栏选择【Resource】模板下的【Property List】，创建plist文件，如图3所示。

创建的plist文件在项目文件夹中可以看到。点击该文件，会在右侧窗口显示该plist文件内容。在右侧窗口处点击右键，在弹出的菜单栏选择【Add Row】，这样就可以在plist文件中添加一个新的键值对，如图4所示。

双击键对相应的Key和Value位置可对其进行编写，对选中的键对点击Key栏的‘+’号可以添加一行数据，点击‘-’号可以删除该行数据。点击键值对的Type栏可以在弹出的菜单栏选择该行数据的的数据类型。plist文件的数据类型有Array、Dictionary、Boolean、Date、Data、String、Number。其中前两个为集合类型。plist文件的根数据类型即解析plist文件得到的数据类型，只有两种NSArray或者NSDictionary。

如图所示在plist文件中添加两行数据。

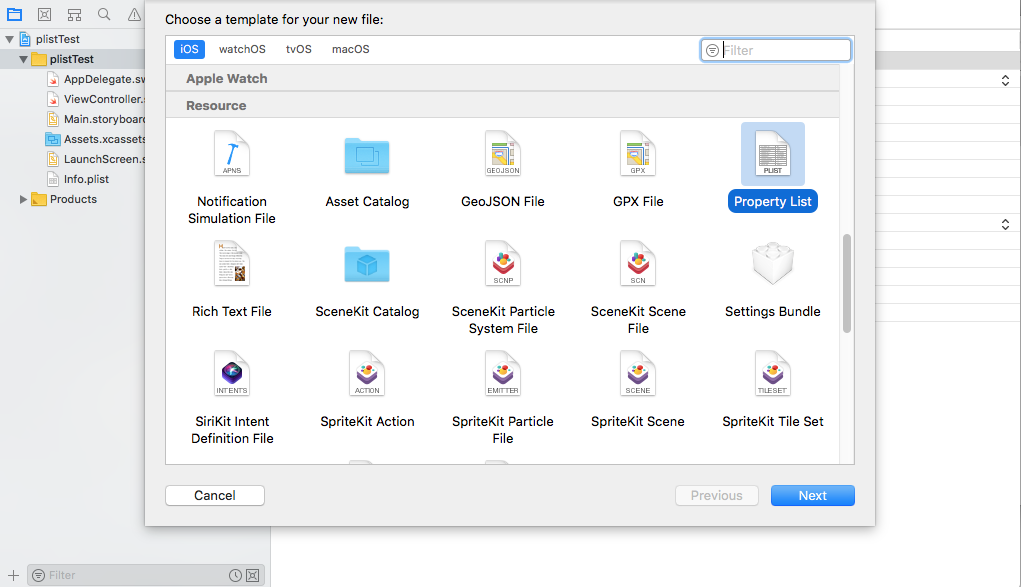


图3 plist文件创建

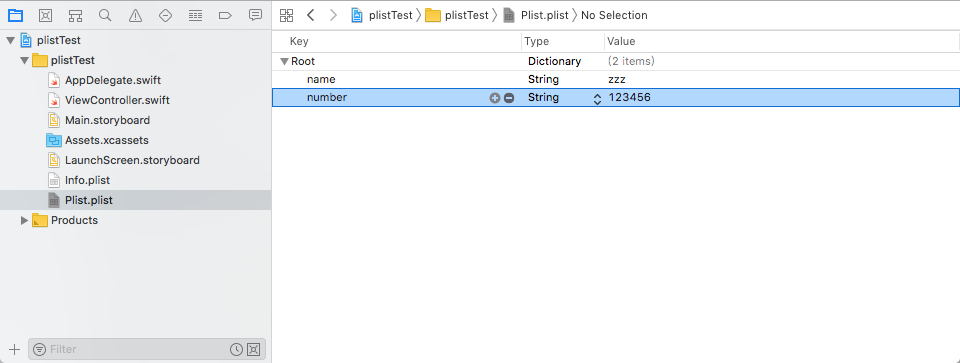


图4 在plist文件添加键对

编辑【ViewController.swift】文件，输入以下代码，实现解析plist文件并在日志区输出plist文件内容的操作。

1. import UIKit
2. class ViewController: UIViewController {
3. override func viewDidLoad() {
4. super.viewDidLoad()
5. // Do any additional setup after loading the view, typically from a nib.
6. //获得plist文件在沙箱中的路径
7. let path=Bundle.main.path(forResource: "Plist", ofType: "plist")
8. //根据该路径加载plist文件,并将其转换为可变字典类型对象
9. let plistDate:NSMutableDictionary=NSMutableDictionary.init(contentsOfFile: path!)!
10. //获得字典对象的描述信息
11. let info=plistDate.description
12. print(info)
13. }
14. }

日志区输出结果如图5所示。

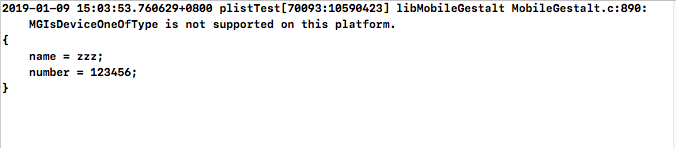


图5 日志区输出结果

### 14.1.2 plist文件的修改、删除和写入

编辑【ViewController.swift】文件，输入以下代码，实现plist文件的修改、删除和写入操作并在日志区输出修改后的plist文件内容。

1. import UIKit
2. class ViewController: UIViewController {
3. override func viewDidLoad() {
4. super.viewDidLoad()
5. // Do any additional setup after loading the view, typically from a nib.
6. //获得plist文件在沙箱中的路径
7. let path=Bundle.main.path(forResource: "Plist", ofType: "plist")
8. //根据该路径加载plist文件,并将其转换为可变字典类型对象
9. let plistDate:NSMutableDictionary=NSMutableDictionary.init(contentsOfFile: path!)!
10. //对Key为name的键值对进行修改，以下两个操作结果相同
11. //plistDate.setObject("changed", forKey: "name" as NSCopying)
12. plistDate["name"]="abc"
13. //添加一行数据
14. plistDate.setObject("123456", forKey: "password" as NSCopying)
15. //删除一行数据
16. plistDate.removeObject(forKey: "number" as NSCopying)
17. //将修改后的内容写入文件
18. plistDate.write(toFile: path!, atomically: true)
19. //获得字典对象的描述信息并输出
20. let info=plistDate.description
21. print(info)
22. //输出plist文件路径
23. print(path!)
24. }
25. }

日志区输出结果如图6所示。

打开Finder，使用快捷键command+shift+g，复制日志输出的路径来找到plist文件所在文件夹，打开该文件，文件内容如图7所示。



图6 日志输出结果

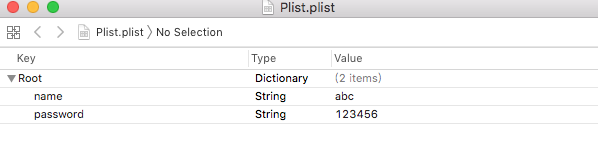


图7 沙箱内的plist文件

这里需要注意的是，在plist文件写入时，会先清空整个plist文件，然后重新将修改后的内容写进去。如果执行以下操作：

let data:NSMutableDictionary=NSMutableDictionary()

data.setObject("ccc", forKey: "name" as NSCopying)

data.write(toFile: path!, atomically: true)

即创建一个新的可变字典类型对象并写入plist文件，那么plist文件的内容只有data的内容。所以要修改plist文件内容要先将其内容赋值给一个对象，然后对该对象进行修改操作，再写入plist文件。

## 14.2 CoreData

CoreData是Apple官方为iOS提供的一个数据持久化方案，其本质是一个通过封装底层数据操作，让程序员以面向对象的方式存储和管理数据的ORM框架（Object-Relational Mapping：对象-关系映射，简称ORM）。虽然底层支持SQLite、二进制数据、xml等多种文件存储，但是主要还是用来操作SQLite数据库。

开发者不需要学习或者使用SQL语句，只需要使用CoreData框架提供的对象和接口以及图形化工具，即可完成SQLite数据库的创建、表关系、增删改查等一系列操作，在一定程度上降低了开发人员的学习成本并增加了代码的统一性和可阅读性。

CoreData主要包含以下几个类：

1、NSManagedObjectModel：托管对象模型，映射实体类和数据库数据的关系，本质是一个XML文件

2、NSManagedObject：托管对象，对应数据库数据的实体

3、NSManagedObjectContext：托管对象上下文，管理托管对象

4、NSPersistentStoreCoordinator：持久化存储调度器，用来处理磁盘持久化数据和实体类对象的相互转化

5、NSPersistentStore：持久化存储器，负责磁盘持久化数据存取

CoreData的总体框架如图所示。

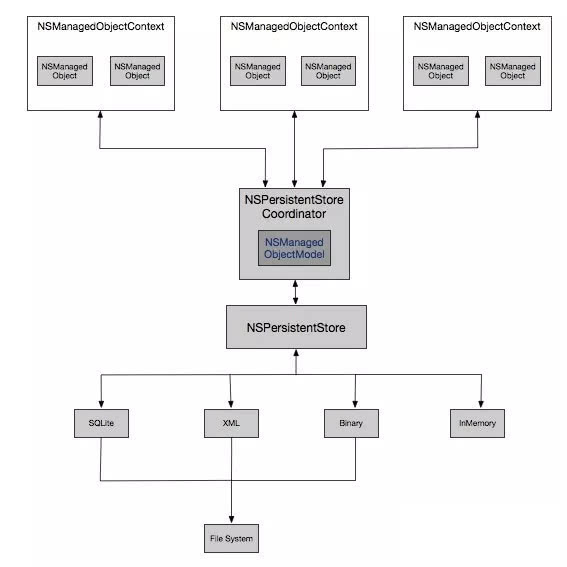


图8 CoreData框架

本节将要介绍的是CoreData的创建、插入、查找、修改和删除操作。

### 14.2.1 CoreData实体的创建和数据的插入

若想在项目中使用CoreData，需要引入CoreData框架，最简单的方法是在创建项目时勾选【Use Core Data】，然后创建好的项目会自动生成一个后缀名为.xcdatamodeld的文件。该文件可以使用图形化的方式编辑数据类型，如图所示。打开AppDelegate.swift文件会发现在文件的末尾多出了关于CoreData的内容，如图所示。

点击底部的【Add Entity】，添加一个新的实体，将实体名称改为Student。在【Attribute】设置区，点击+号图标为实体添加3个属性并设置其数据类型，如图9所示。注意，实体名称的首字母必须大写，而属性名称首字母必须小写。

编辑【ViewController.swift】文件，输入以下代码，实现对CoreData实体插入数据的操作。

1. import UIKit
2. //引入CoreData框架
3. import CoreData
4. class ViewController: UIViewController {
5. override func viewDidLoad() {
6. super.viewDidLoad()
7. // Do any additional setup after loading the view, typically from a nib.
8. //获得当前项目的AppDelegate对象，进而取出其中的托管对象上下文
9. let appDel:AppDelegate = UIApplication.shared.delegate as! AppDelegate
10. let context=appDel.persistentContainer.viewContext
11. //在托管对象上下文中插入一个Student实体对象
12. let stu=NSEntityDescription.insertNewObject(forEntityName: "Student", into: context) as! Student
13. //给实体对象各属性赋值
14. stu.name="zzz"
15. stu.id="123456"
16. stu.department="computer"
17. //将实体对象保存到托管对象上下文中
18. do {
19. try context.save()
20. print("save data successfully.")
21. } catch  {
22. print("save data failed.")
23. }
24. }
25. }

运行该项目，若保存成功会在日志区输出“save data successfully.”。

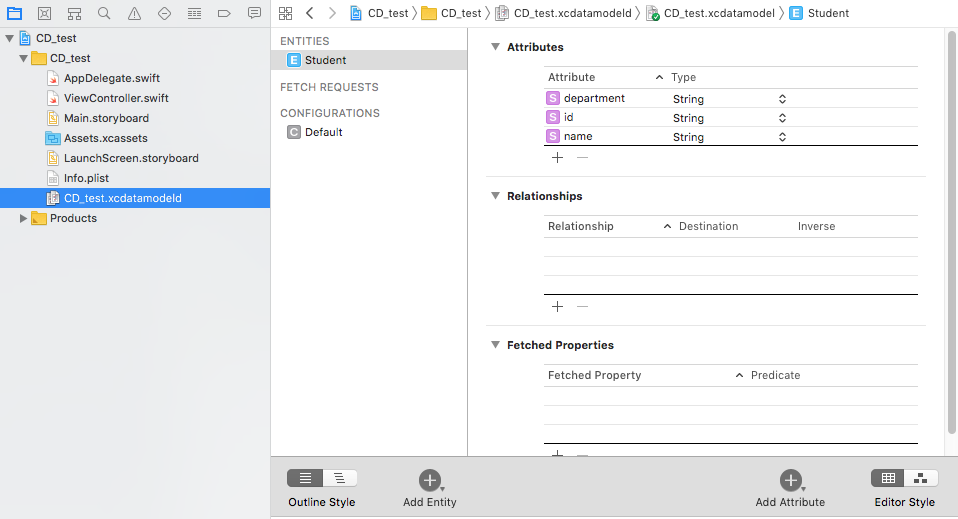


图9 CoreDate实体属性

### 14.2.2 CoreData的查找操作

要查找CoreData的数据需要使用NSPredicate谓词和NSFetchRequest类。使用托管对象上下文来查找数据时，要先创建请求对象，再编写查询条件，请求对象根据查询条件在托管上下文中进行数据查找。

在上一节的代码上进行修改，此时的托管对象上下文已经存储了上一节中的实体。修改11行以后的代码。

1. //创建Student类型实体
2. let entity=NSEntityDescription.entity(forEntityName: "Student", in: context)
3. //创建请求对象和查询谓词，设置请求对象的实体类型、查询谓词、偏移量和查询结果数量
4. let req=NSFetchRequest<Student>(entityName: "Student")
5. req.entity=entity
6. //创建查询谓词，设置请求对象的查询谓词
7. let pred=NSPredicate.init(format: "id='123456'", "")
8. req.predicate=pred
9. //设置请求对象的查询偏移量和查询数量
10. req.fetchLimit=20
11. req.fetchOffset=0
12. //查询与谓词相对应的实体数据并输出
13. do {
14. let datas:[Student]?=try context.fetch(req)
15. for stu in datas!{
16. print("name=\(stu.name!)")
17. print("id=\(stu.id!)")
18. print("department=\(stu.department!)")
19. }
20. } catch  {
21. print("fetch failed.")
22. }

运行项目，结果如图10所示。



图10

对于18行，如果req.predicate=nil,那么以上代码会输出在该托管对象上下文中Student的全部实体。

### 14.2.3 CoreData修改和删除操作

如果要对CoreData中的数据进行修改和删除，首先要查找数据库中所需要修改的对应实体，通过托管对象上下文对实体对应的托管对象进行操作，然后将更改后的结果保存到持久化存储中。

假设在CoreData中已经存储了3个Student实体，内容如下：

Entity1：name=zzz，id=123456，department=computer

Entity2：name=abc，id=248650，department=math

Entity3：name=xyz，id=112233，department=computer

编辑【ViewController.swift】文件，输入以下代码，实现对CoreData实体修改数据的操作。

1. import UIKit
2. import CoreData
3. class ViewController: UIViewController {
4. override func viewDidLoad() {
5. super.viewDidLoad()
6. // Do any additional setup after loading the view, typically from a nib.
7. let appDel:AppDelegate = UIApplication.shared.delegate as! AppDelegate
8. let context=appDel.persistentContainer.viewContext
9. let entity=NSEntityDescription.entity(forEntityName: "Student", in: context)
10. let req=NSFetchRequest<Student>(entityName: "Student")
11. req.entity=entity
12. let pred=NSPredicate.init(format: "id='123456'", "")
13. req.predicate=pred
14. req.fetchLimit=20
15. req.fetchOffset=0
16. do {
17. //查找到要修改的实体，通过托管对象上下文进行修改，然后保存
18. let changedDatas:[Student]?=try context.fetch(req)
19. for stu in changedDatas!{
20. stu.department="computer"
21. }
22. try context.save()
23. //输出CoreData中的所有Student实体
24. req.predicate=nil
25. let datas:[Student]?=try context.fetch(req)
26. for stu in datas!{
27. print("name=\(stu.name!)")
28. print("id=\(stu.id!)")
29. print("department=\(stu.department!)")
30. }
31. } catch  {
32. print("fetch failed.")
33. }
34. }
35. }

运行项目，日志区输出结果如图11所示。

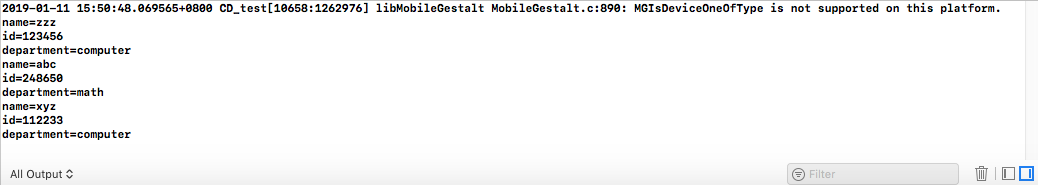


图11日志输出结果

删除CoreData的数据使用delete(\_ object: [NSManagedObject](apple-reference-documentation://hszrY8Gi1t))方法。将上述代码的22行修改为

1. for stu in changedDatas!{
2. context.delete(stu)
3. }
4. try context.save()

就成功删除了谓词所对应的实体。运行修改后的项目，日志区输出结果如图12所示。



图12 日志输出结果

## 总结