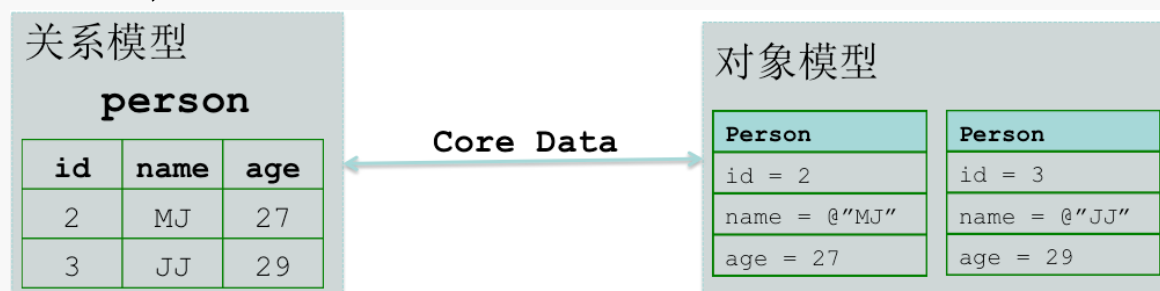


# Core Data By 熊孩子

## 简介

大概10年前，在2005年4月，Apple发布了OS X 10.4版本，第一次引入了Core Data框架。那时YouTube也刚发布。

Core Data提供了对象-关系映射(ORM)的功能，即能够将OC对象转化成数据，保存在SQLite数据库文件中，也能够将保存在数据库中的数据还原成OC对象。简单地用下图描述下它的作用：



左边是关系模型，即数据库，数据库里面有张person表，person表里面有id、name、age三个字段，而且有2条记录；右边是对象模型，可以看到，有2个OC对象；利用Core Data框架，我们就可以轻松地将数据库里面的2条记录转换成2个OC对象，也可以轻松地将2个OC对象保存到数据库中，变成2条表记录，而且不用写一条SQL语句。

Core Data是模型层的技术。Core Data帮助你构建代表程序状态的模型层。Core Data也是一种持久化技术，它可以将模型的状态持久化到磁盘。但它更重要的特点是：Core Data不只是一个加载和保存数据的框架，它也能处理内存中的数据。

如果你曾接触过Object-relational mapping(O/RM)，Core Data不仅是一种O/RM。如果你曾接触过SQL wrappers, Core Data也不是一种SQL wrapper。它确实默认使用SQL，但是，它是一种更高层次的抽象概念。如果你需要一个O/RM或者SQL wrapper，那么Core Data并不适合你。

Core Data提供的最强大的功能之一是它的对象图形管理。为了更有效的使用Core Data, 你需要理解这一部分内容。

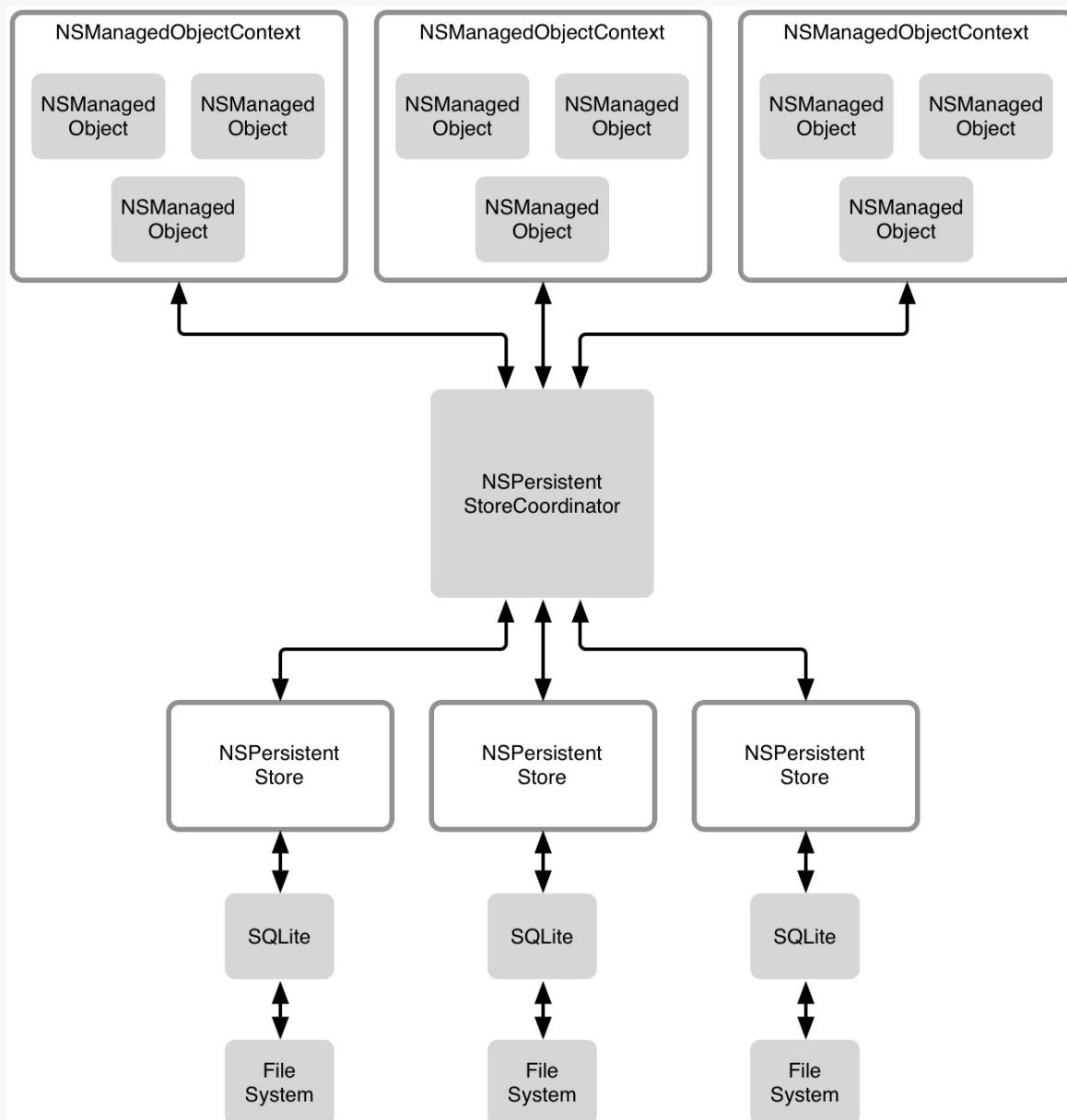
还有一点需要注意：Core Data完全独立于任何UI层的框架。从设计的角度来说，它是完全的模型层的框架。

## 堆栈The Stack

Core Data中有不少组件，它是一种非常灵活的技术。在大多数使用情况里，设置相对来说比较简单。

当所有组件绑定在一起，我们把它们称为Core Data Stack. 这种堆栈有两个主要部分。一部分是关于对象图管理，这是你需要掌握好的部分，也应该知道怎么使用。第二部分是关于持久化的，比如保存模型对象的状态和再次恢复对象的状态。

在这两部分的中间，即堆栈中间，是持久化存储协调器(Persistent Store Coordinator, PSC)，也被朋友们戏称做中心监视局。通过它将对对象图管理部分和持久化部分绑在一起。当这两部分中的一部分需要和另一部分交互，将通过PSC来调节。

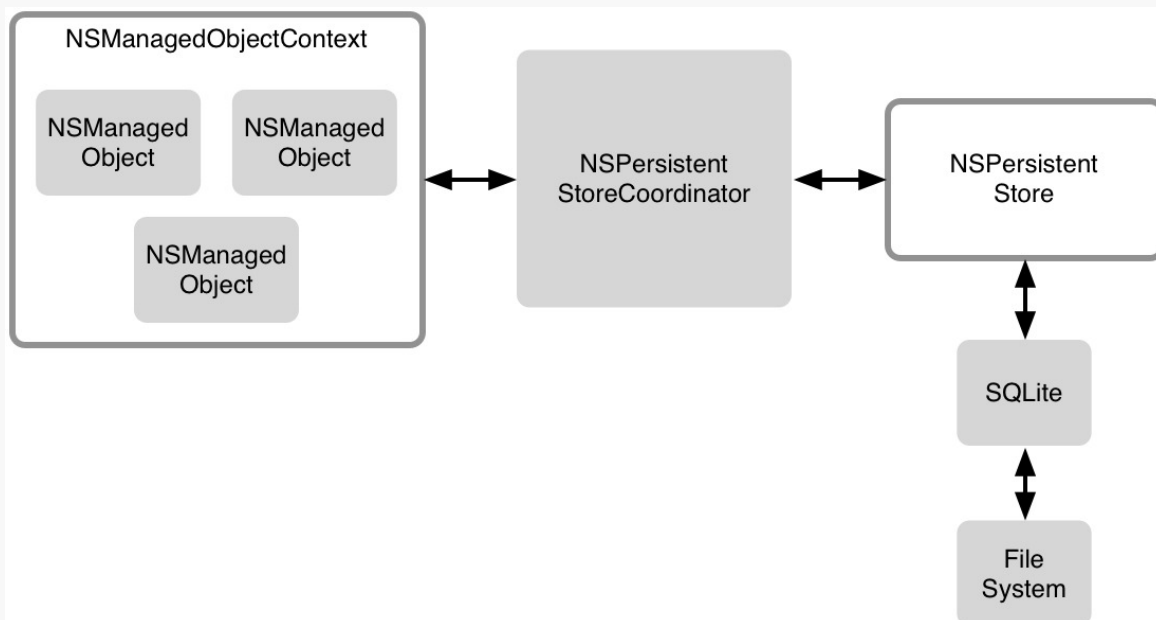


对象图管理是你的应用中模型层逻辑存在的地方。模型层对象存在于一个context里。在大多数设置中，只有一个context，所有的对象都放在这个context中。Core Data支持多个context，但是是针对更高级的使用情况。需要注意的是，每个context和其他context区分都很清楚。有个重要的事需要记住，对象和他们的context绑定在一起。每一个被管理的对象都知道它属于哪个context，每一个context也知道它管理着哪些对象。

堆栈的另一部分是持久化发生的地方，比如Core Data从文件系统读或写。在所有情况下，持久化存储协调器(PSC)有一个属于自己的持久化存储器(persistent store)，这个store在文件系统

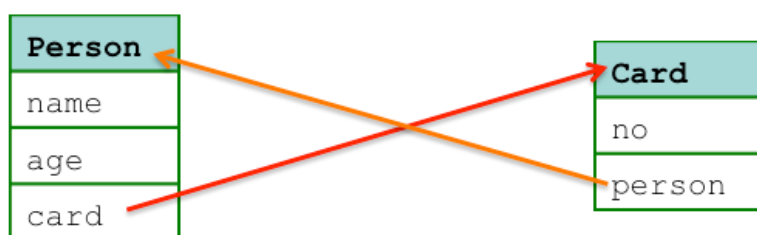
和SQLite数据库交互。为了支持更高级的设置，Core Data支持使用多个存储器附属于同一个持久化存储协调器，并且除了SQL，还有一些别的存储类型可以选择。

一个常见的解决方案，看起来是这个样子的：



## 模型文件

在Core Data，需要进行映射的对象称为实体(entity)，而且需要使用Core Data的模型文件来描述app中的所有实体和实体属性。这里以Person(人)和Card(身份证)2个实体为例子，先看看实体属性和实体之间的关联关系：

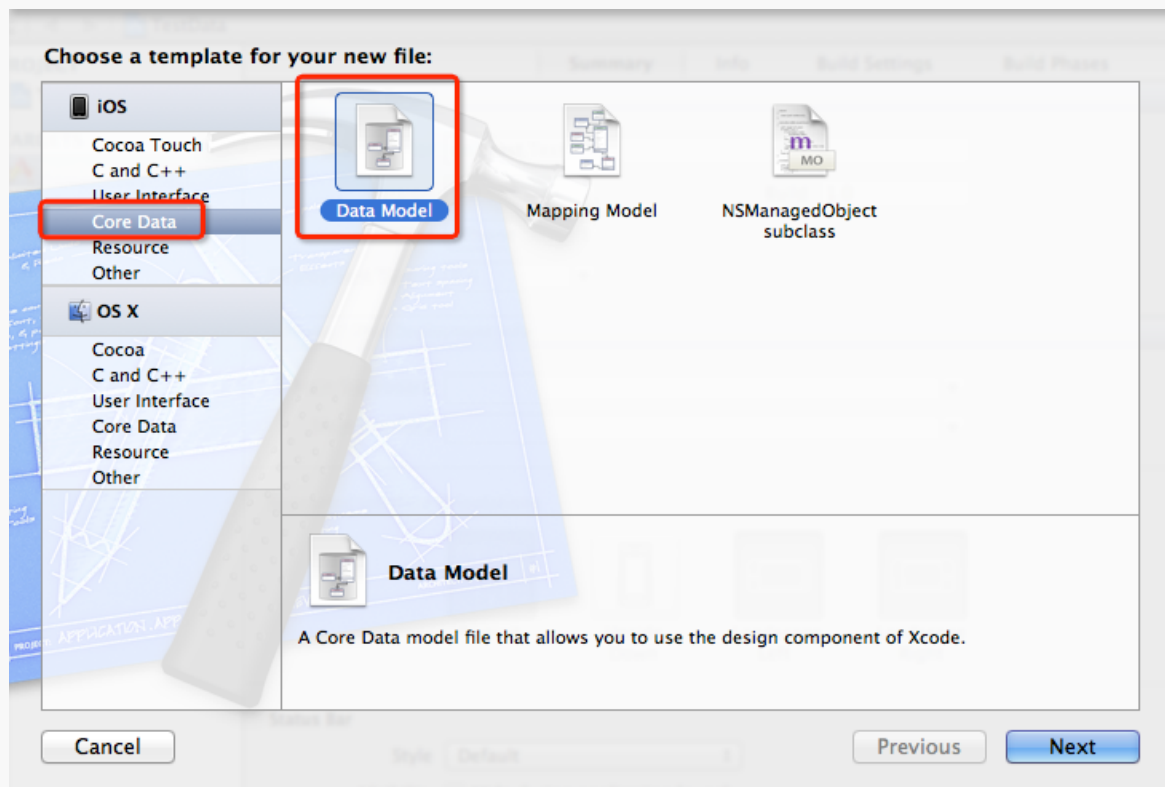


- Person中有个Card属性，Card中有个Person属性
- 属于一对一双向关联

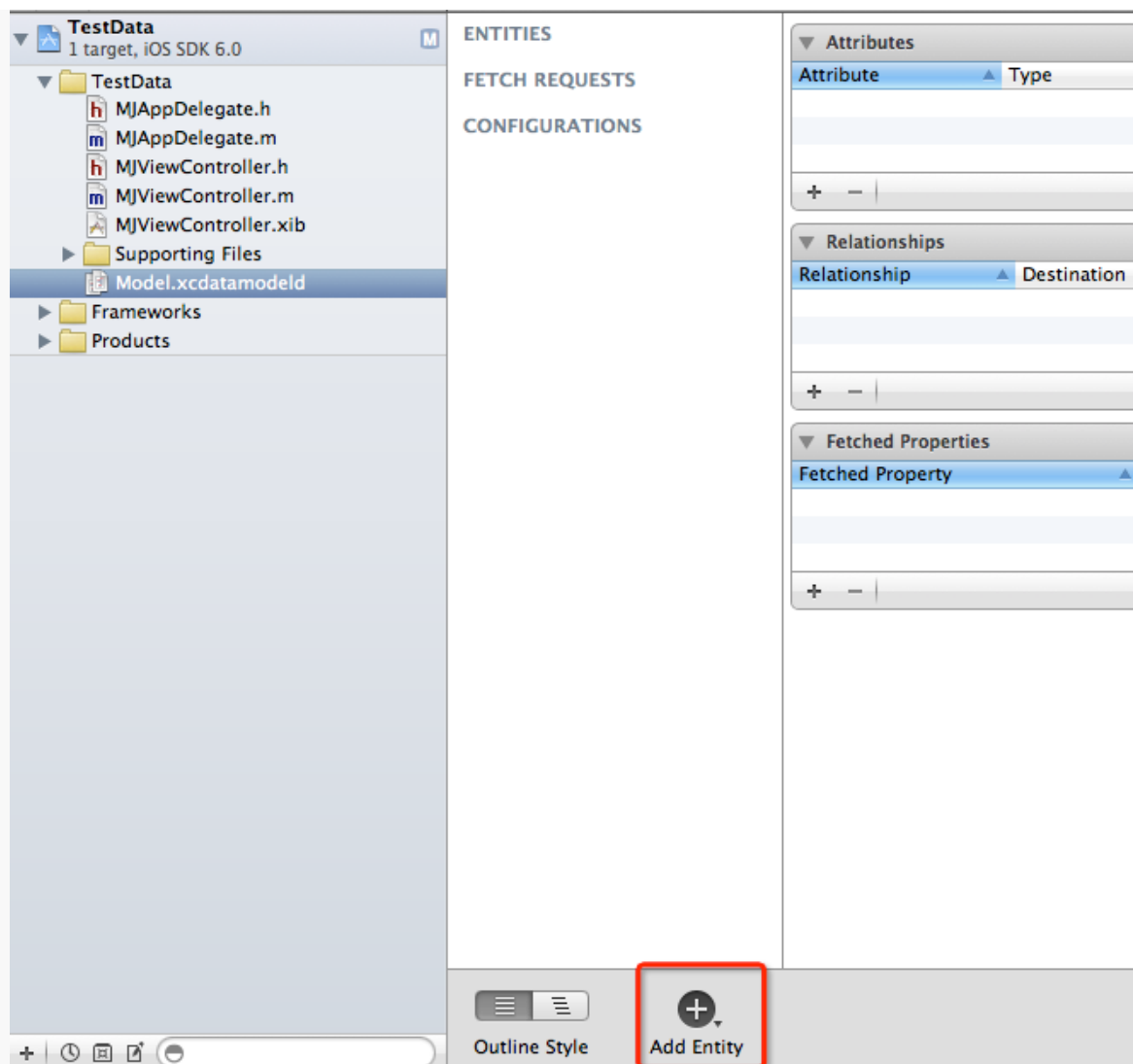
Person实体中有：name（姓名）、age（年龄）、card（身份证）三个属性 Card实体中有：no（号码）、person（人）两个属性

接下来看看创建模型文件的过程：

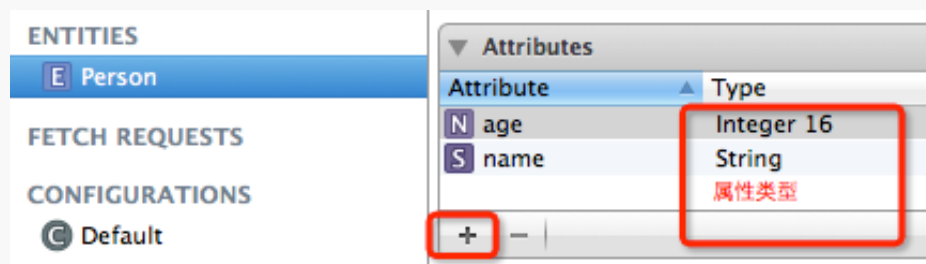
## 1. 选择模板



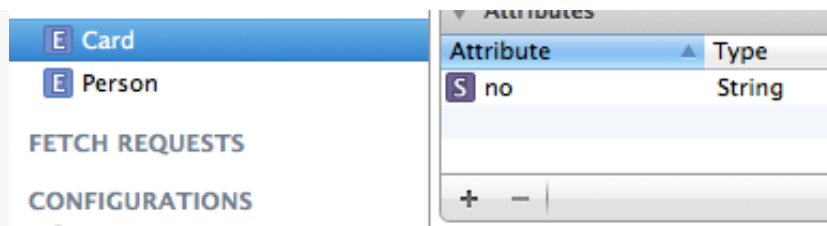
## 2. 添加实体



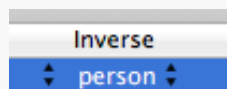
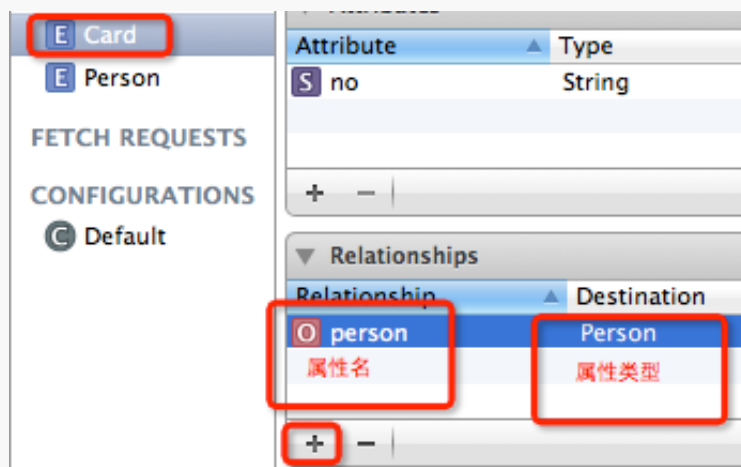
3.添加Person的2个基本属性




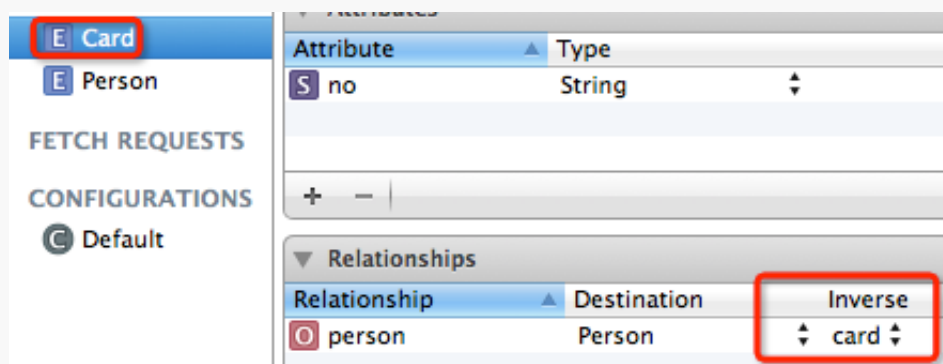
4.添加Card的1个基本属性



## 5.建立Card和Person的关联关系

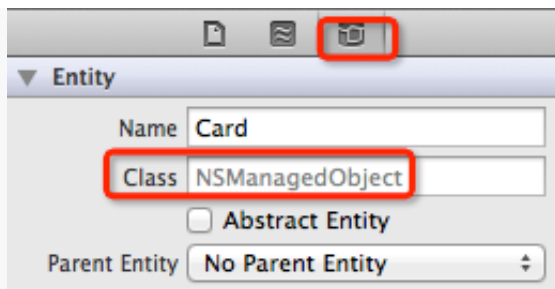


上图中的  表示Card中有个Person类型的人属性，目的就是建立Card跟Person之间的一对一关联关系(建议补上这一项)，在Person中加上Inverse属性后，你会发现Card中Inverse属性也自动补上了



## 了解NSManagedObject

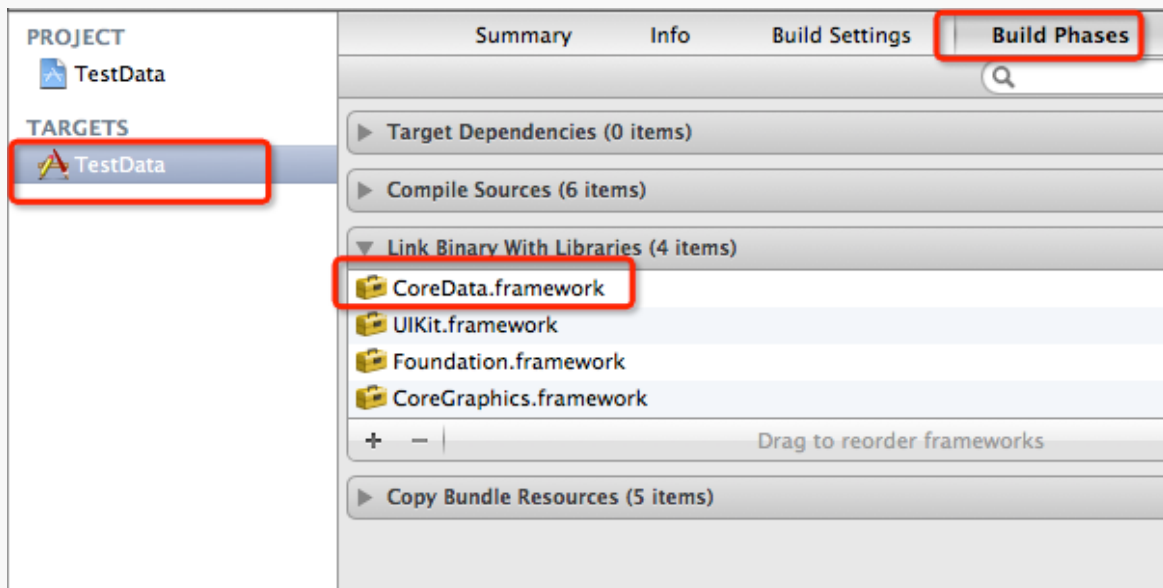
1.上面创建的模型，默认情况下都是NSManagedObject类型，对应的Core Data中的模型对象都是NSManagedObject类的对象



2.NSManagedObject的工作模式有点类似于NSDictionary对象，通过键-值对来存取所有的实体属性 1> setValue:forKey:存储属性值(属性名为key) 2> valueForKey:获取属性值(属性名为key)

## 代码实现

先添加CoreData.framework和导入主头文件<CoreData/CoreData.h>



1.搭建上下文环境

```

// 从应用程序包中加载模型文件
NSManagedObjectModel *model = [NSManagedObjectModel mergedModelFromBundles:nil];
// 传入模型对象, 初始化NSPersistentStoreCoordinator
NSPersistentStoreCoordinator *psc = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:model] autorelease];
// 构建SQLite数据库文件的路径
NSString *docs = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) lastObject];
NSURL *url = [NSURL fileURLWithPath:[docs stringByAppendingPathComponent:@"person.data"]];
// 添加持久化存储库, 这里使用SQLite作为存储库
NSError *error = nil;
NSPersistentStore *store = [psc addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:url options:nil error:&error];
if (store == nil) { // 直接抛异常
    [NSException raise:@"添加数据库错误" format:@"%@", [error localizedDescription]];
}
// 初始化上下文, 设置persistentStoreCoordinator属性
NSManagedObjectContext *context = [[NSManagedObjectContext alloc] init];
context.persistentStoreCoordinator = psc;

```

## 2. 添加数据到数据库



```
// 传入上下文，创建一个Person实体对象
NSManagedObject *person = [NSEntityDescription insertNewObjectForEntityForName:@"Person" inManagedObjectContext:context];
// 设置Person的简单属性
[person setValue:@"xhz" forKey:@"name"];
[person setValue:[NSNumber numberWithInt:27] forKey:@"age"];
// 传入上下文，创建一个Card实体对象
NSManagedObject *card = [NSEntityDescription insertNewObjectForEntityForName:@"Card" inManagedObjectContext:context];
[card setValue:@"4414241933432" forKey:@"no"];
// 设置Person和Card之间的关联关系
[person setValue:card forKey:@"card"];
// 利用上下文对象，将数据同步到持久化存储库
NSError *error = nil;
BOOL success = [context save:&error];
if (!success) {
    [NSException raise:@"访问数据库错误" format:@"%@", [error localizedDescription]];
}
// 如果是想做更新操作：只要在更改了实体对象的属性后调用[context save:&error]，就能将更改的数据同步到数据库
```

### 3.从数据库中查询数据

```

// 初始化一个查询请求
NSFetchRequest *request = [[[NSFetchRequest alloc] init] autorelease];
// 设置要查询的实体
request.entity = [NSEntityDescription entityWithName:@"Person" inManagedObjectContext:context];
// 设置排序（按照age降序）
NSSortDescriptor *sort = [NSSortDescriptor sortDescriptorWithKey:@"age" ascending:NO];
request.sortDescriptors = [NSArray arrayWithObject:sort];
// 设置条件过滤(搜索name中包含字符串"Itcast-1"的记录，注意：设置条件过滤时，数据库SQL语句中的%要用*来代替，所以%Itcast-1%应该写成*Itcast-1*)
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"name like %@", @"*Itcast-1*"];
request.predicate = predicate;
// 执行请求
NSError *error = nil;
NSArray *objs = [context executeFetchRequest:request error:&error];
if (error) {
    [NSException raise:@"查询错误" format:@"%@", [error localizedDescription]];
}
// 遍历数据
for (NSManagedObject *obj in objs) {
    NSLog(@"name=%@", [obj valueForKey:@"name"]);
}

```

注：Core Data不会根据实体中的关联关系立即获取相应的关联对象，比如通过Core Data取出Person实体时，并不会立即查询相关联的Card实体；当应用真的需要使用Card时，才会再次查询数据库，加载Card实体的信息。这个就是Core Data的延迟加载机制

#### 4.删除数据库中的数据

```

// 传入需要删除的实体对象
[context deleteObject:managedObject];
// 将结果同步到数据库
NSError *error = nil;
[context save:&error];
if (error) {
    [NSException raise:@"删除错误" format:@"%@", [error localizedDescription]];
}

```

## 打开CoreData的SQL语句输出开关

1. 打开Product, 点击EditScheme...
2. 点击Arguments, 在ArgumentsPassed On Launch中添加2项
  - -com.apple.CoreData.SQLDebug
  - 1

Product

Window

Help

Run

⌘R

Test

⌘U

Profile

⌘I

Analyze

⇧⌘B

Archive

Build For



Perform Action



Build

⌘B

Clean

⇧⌘K

Stop

⌘.

Generate Output



Debug



Debug Workflow



Attach to Process

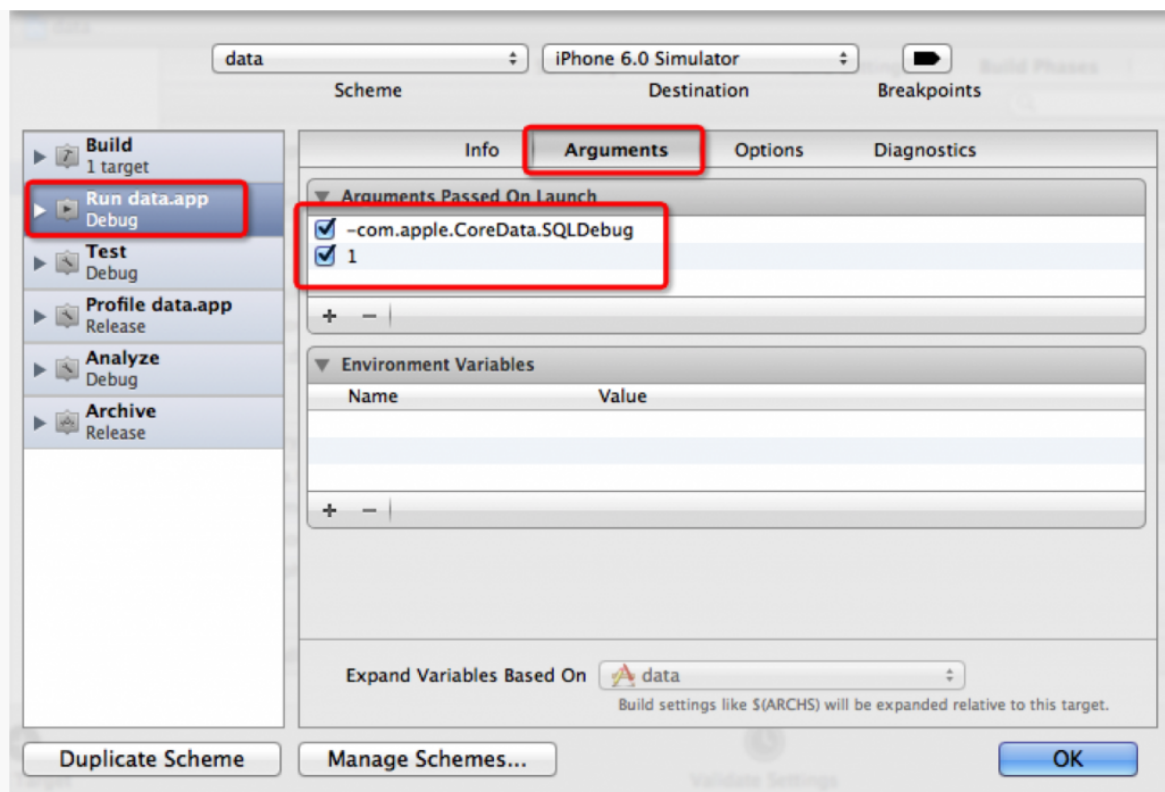


Edit Scheme...

⌘<

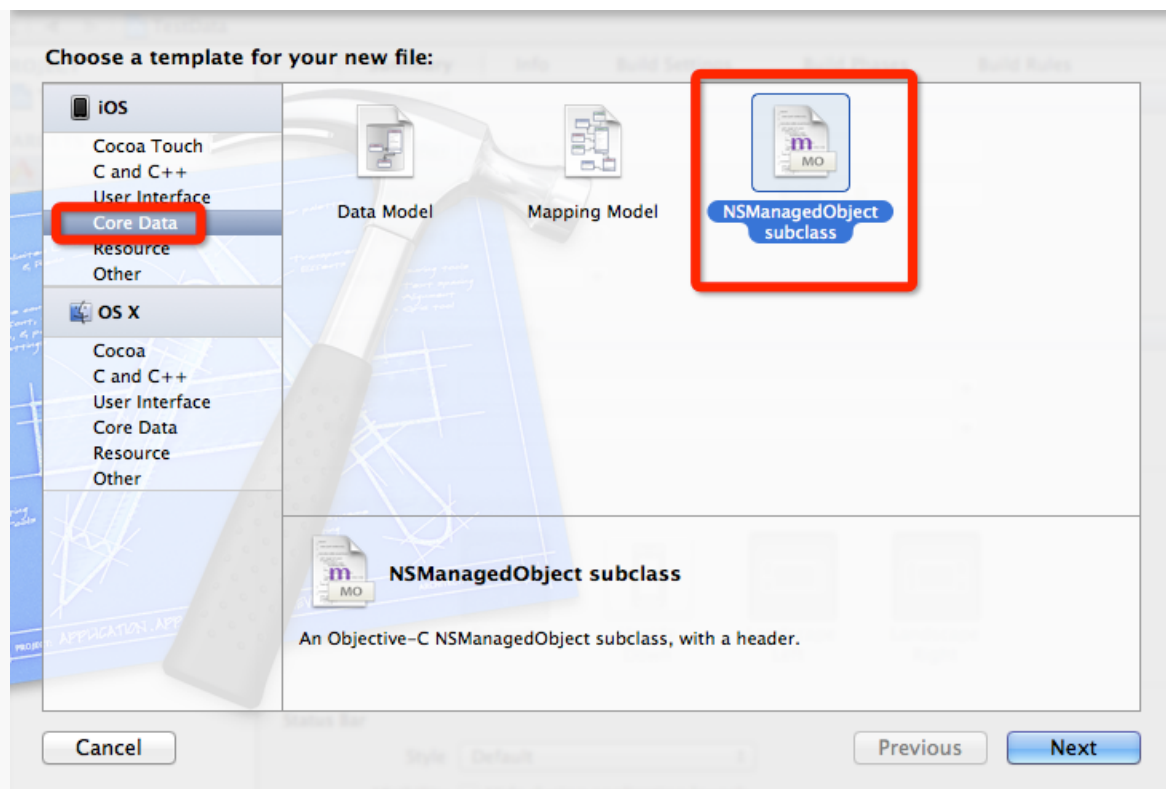
New Scheme...

Manage Schemes...

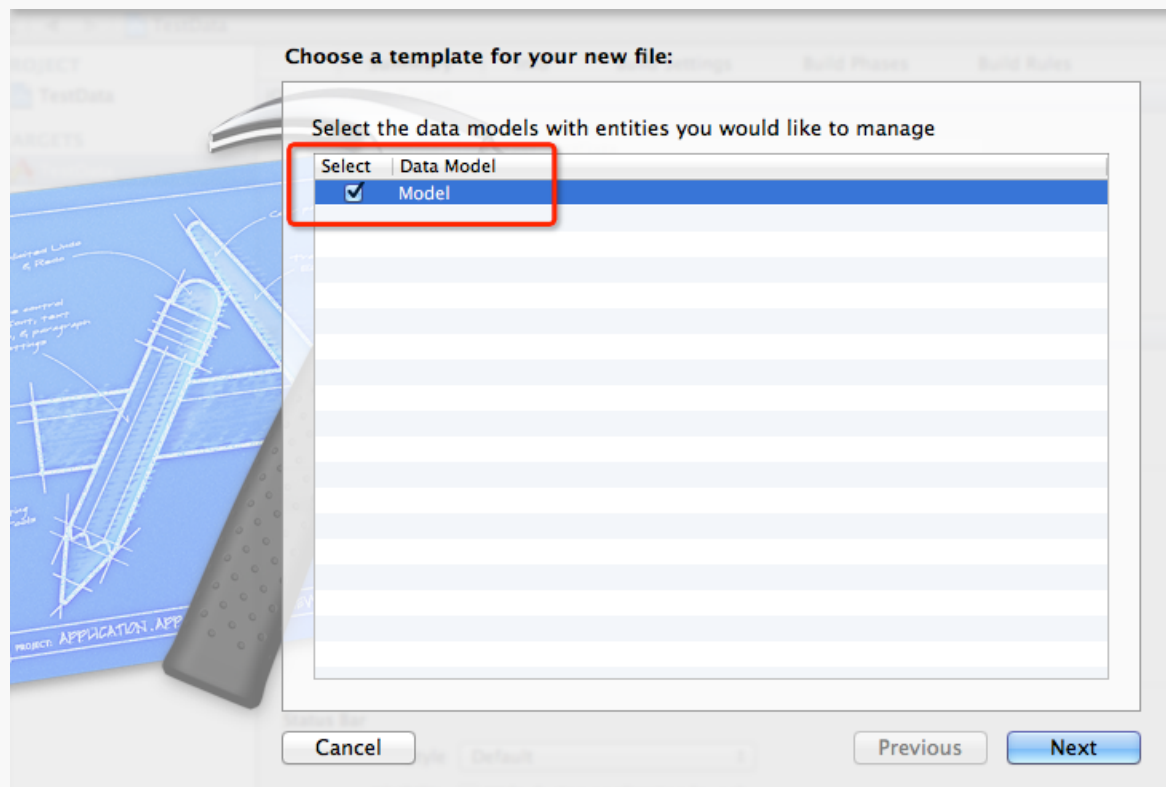


## 创建NSManagedObject的子类

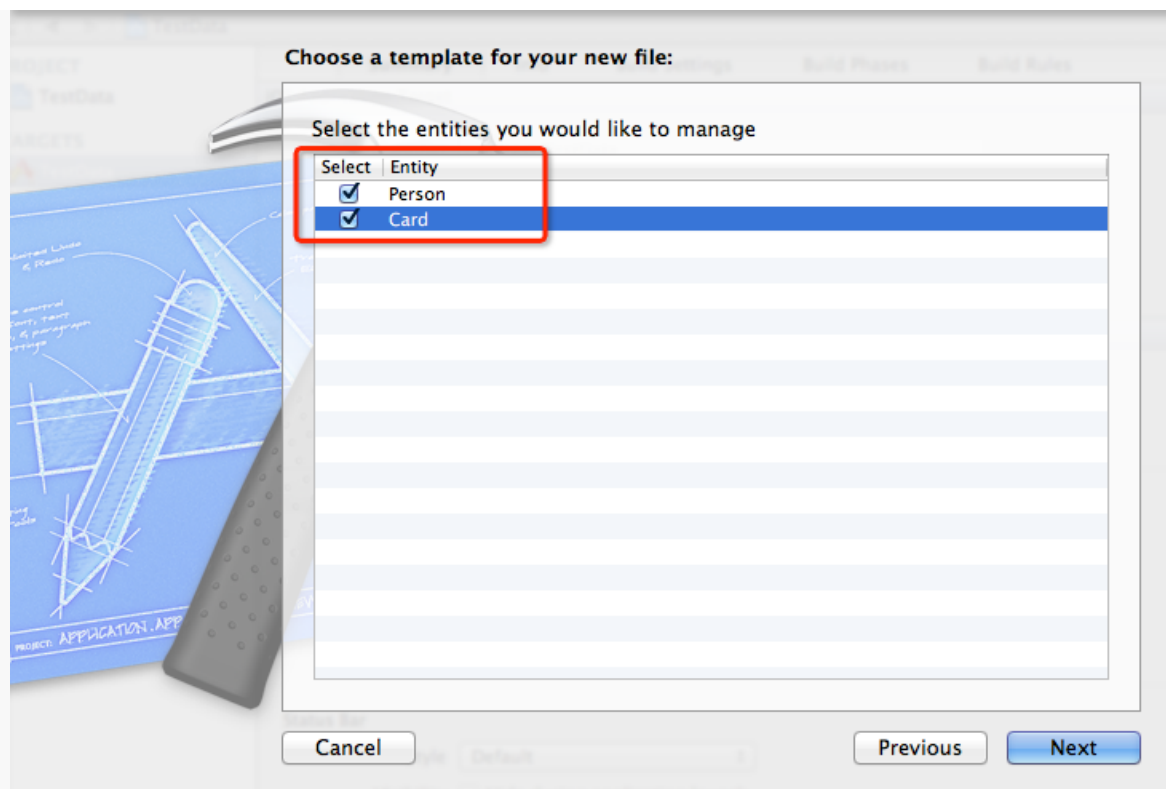
默认情况下，利用Core Data取出的实体都是NSManagedObject类型的，能够利用键-值对来存取数据。但是一般情况下，实体在存取数据的基础上，有时还需要添加一些业务方法来完成一些其他任务，那么就必須创建NSManagedObject的子类



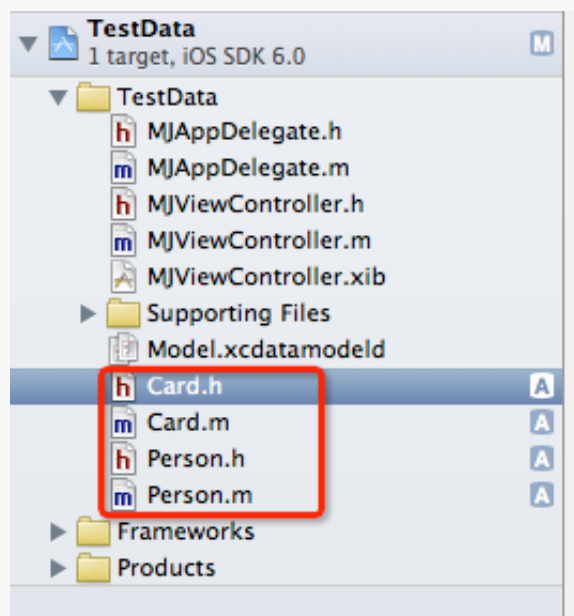
选择模型文件



选择需要创建子类的实体



创建完毕后，多了2个子类



文件内容展示：

Person.h

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class Card;

@interface Person : NSManagedObject

@property (nonatomic, retain) NSString * name;
@property (nonatomic, retain) NSNumber * age;
@property (nonatomic, retain) Card *card;

@end
```

#### Person.m

```
#import "Person.h"

@implementation Person

@dynamic name;
@dynamic age;
@dynamic card;

@end
```

#### Card.h

```
#import <Foundation/Foundation.h>
#import <CoreData/CoreData.h>

@class Person;

@interface Card : NSManagedObject

@property (nonatomic, retain) NSString * no;
@property (nonatomic, retain) Person *person;

@end
```

#### Card.m



```
#import "Card.h"
#import "Person.h"

@implementation Card

@dynamic no;
@dynamic person;

@end
```

那么往数据库中添加数据的时候就应该写了：

```
Person *person = [NSEntityDescription insertNewObjectForEntityForName:@"Person" inManagedObjectContext:context];
person.name = @"XHZ";
person.age = [NSNumber numberWithInt:27];

Card *card = [NSEntityDescription insertNewObjectForEntityForName:@"Card" inManagedObjectContext:context];
card.no = @"4414245465656";
person.card = card;
// 最后调用[context save&error];保存数据
```