

第2回

「 CoreData なにそれ死ぬわ」

toru.inoue@kissaki.tv

タグは#TechHUBJP

Agenda 1/2

- ・今日やる事：CoreDataという保存の為の機構について、解説と実践。

主に端折って解説しますが、Appleのドキュメントは見た方が良いっていうか見る癖付けるとすぐステキ。

<http://developer.apple.com/jp/documentation/cocoa/Conceptual/CoreData/cdProgrammingGuide.html>

- ・効能：iOSで保存したり削除したり、データ形式バージョンのUpの仕方とかまで、理解出来るようになります。
もう何も怖くない。 というのは嘘。
- ・今日やらない事：
編集の競合、ロールバックについてはやりません。

Agenda 2/2

- 1.“保存”についての基礎知識
- 2.そこで CoreData
- 3.バージョンを超えていけ

Drive I/64

1.“保存”についての基礎知識

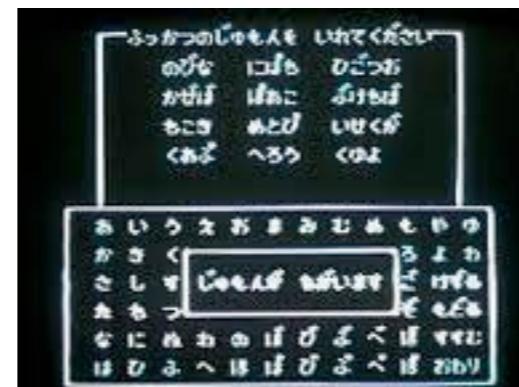
Drive2/64

1-0-0.保存(Persistent)ってどういう事？

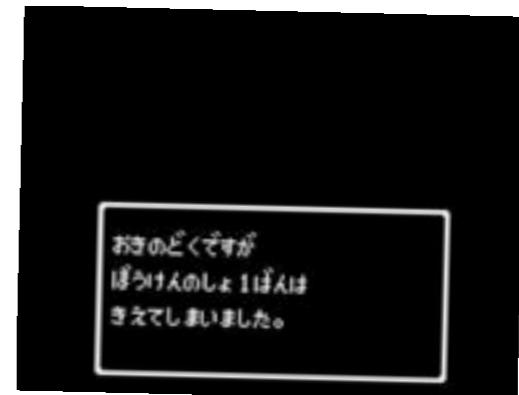
アプリケーションを終了して、もう一回起動してもデータが残ってる状態実現するには？

いろんな方法があります。俺はなじみ深いゲームからアプローチ。

- ・ ふっかつのじゅもんがちがいます
→もの凄い文字を一杯打つことで、ソレ自体が実はデータ、という。
→実は、、ソフトは全くユーザーの事を覚えてない。つまり人力セーブ。



- ・ 冒険の書が消えました
→ソフトに保存！ということが出来るようになった
→人力ではない、セーブ、と、ロードの誕生



Drive3/64

1-0-1.保存(Persistent、 persistency)

英語だと、永続化(=いつまでたっても不屈で不变であるさま)を指して、
Persistentとかそういう名前が着く。

The screenshot shows a Japanese dictionary application window titled 'すべての辞書ソース' (All Dictionary Sources). The search bar contains the character '永'. The results are displayed under the '国語辞典' (Kotoba Jiten) tab.

えい-ぞく【永続】
〔名〕スル ながく続くこと。長続き。「一する政権を目指す」

連続・継続

えいぞく【永続】
permanence; permanency

派生語 永続する | last (for a long time, (米国用法) forever, (英國用法) for ever); remain permanently

派生語 永続的な | permanent; perpetual; lasting
両国の友好関係は永続しなかった | The friendly ties between the two nations were short-lived.

Drive4/64

1-0-2.身の回りによく有る永続化

昨今の世の中では、データベース、
データストアと呼ばれるモノ達がこれらを実践しています。

MySQLとか、**PostgreSQL**とか、**SQLite**とか、**BigTable**とか。

- ・上のは永続化に関わる機構、機能、プログラムの名前です。

時々概念の名前の奴が混じってますが、、

- ・内容は様々ですが、用途は一つ、「保存に関する事」。

特にサーバで、ユーザーの情報とかを溜め込むモノとして、**MySQL**とか
そのあたりが最大にメジャーなんじゃなかろうか。

こいつらについて、特に細かい解説はしません。

Drive5/64

1-0-3.iOSでの永続化は、CoreData(SQLite他)

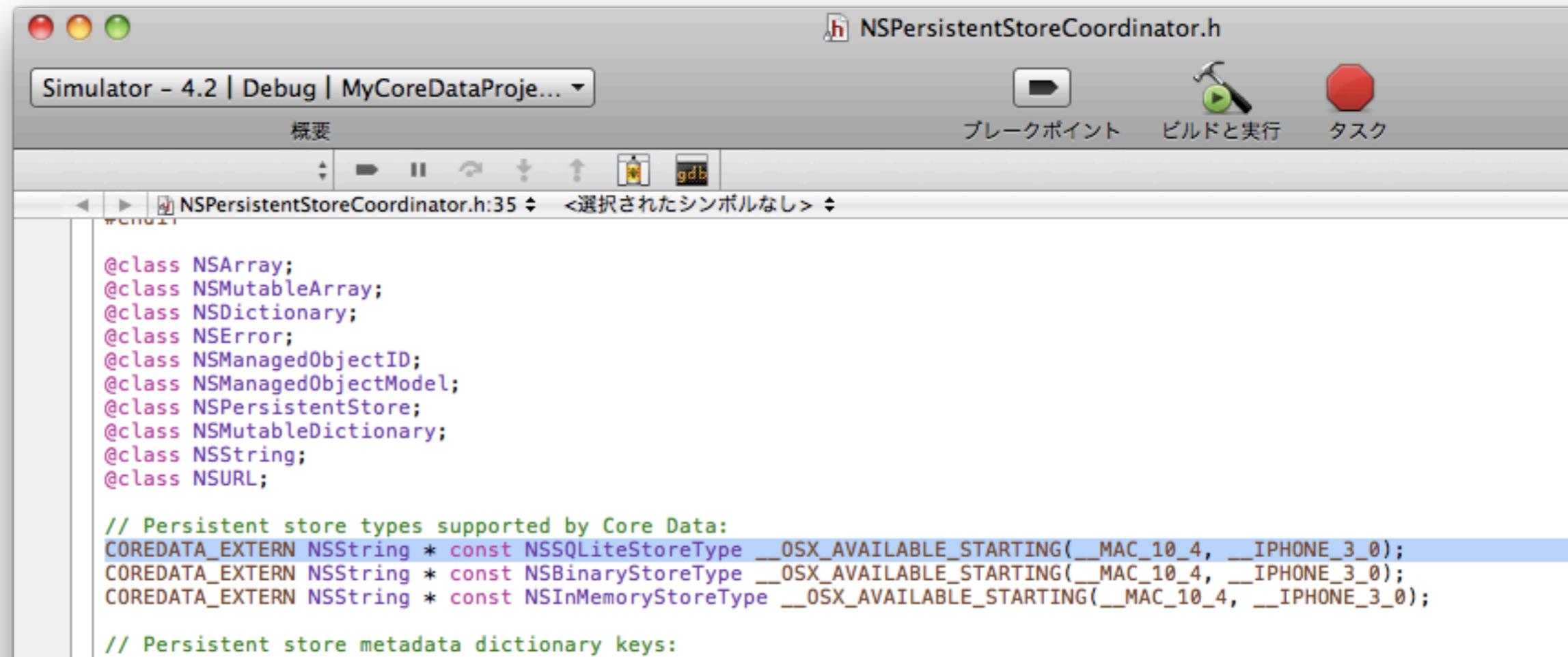
iOSでの永続化には、CoreDataというAppleが用意したフレームワークを使う事が可能。もちろん自力で実装してもいいんだぜ？

CoreDataが採用してる保存方式として、**SQLite**と、**その他2種**の方式がある。

今回、その他2種についてはスルーしますが、

CoreData = SQLiteだとか言ってると(ワラw)ってなるので気を付けてね！

資料：



The screenshot shows the Xcode IDE interface with the title bar "Simulator - 4.2 | Debug | My CoreData Proj..." and the file "NSPersistentStoreCoordinator.h" open in the editor. The code editor displays the following content:

```
@class NSArray;
@class NSMutableArray;
@class NSDictionary;
@class NSError;
@class NSManagedObjectID;
@class NSManagedObjectModel;
@class NSPersistentStore;
@class NSMutableDictionary;
@class NSString;
@class NSURL;

// Persistent store types supported by Core Data:
COREDATA_EXTERN NSString * const NSSQLiteStoreType __OSX_AVAILABLE_STARTING(__MAC_10_4, __IPHONE_3_0);
COREDATA_EXTERN NSString * const NSBinaryStoreType __OSX_AVAILABLE_STARTING(__MAC_10_4, __IPHONE_3_0);
COREDATA_EXTERN NSString * const NSInMemoryStoreType __OSX_AVAILABLE_STARTING(__MAC_10_4, __IPHONE_3_0);

// Persistent store metadata dictionary keys:
```

Drive6/64

1-0-4.SQLite?

SQLiteは、SQLという種類のデータベース言語(問い合わせ言語)を使って、
保存してあるデータを取り出したり、また書き込んだりできるソフトウェア。
.sqliteというファイルの形で、データを保存しておくことが出来る。

ちなみに**SQL**は、

Structured English Query Language

(問い合わせのための言語として構築された英語で書かれた言語)

なんだってさ。へー。



The screenshot shows a Mac OS X browser window displaying the SQLite Home Page at <http://www.sqlite.org/>. The page features the SQLite logo (a blue feather quill icon next to the word "SQLite") and the slogan "Small. Fast. Reliable. Choose any three." Below the logo is a navigation bar with links: About, Sitemap, Documentation, Download, License, News, Support, and a search bar labeled "Search SQLite Docs... Go". The main content area has two columns: "Welcome." on the left and "Current Status" on the right. The "Welcome." section describes SQLite as a self-contained, serverless, zero-configuration, transactional SQL database engine. The "Current Status" section highlights Version 3.7.6.1 as recommended for new development.

SQLite Home Page

[KISSAKI.Inc] Sign in Q Quix ドキュメント 見てるなう! Gito github kissaki-blog Maps KISSAKI Flight Gerrit local gb KissakiServer

 SQLite Small. Fast. Reliable. Choose any three.

About Sitemap Documentation Download License News Support Search SQLite Docs... Go

Welcome.

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

Current Status

- Version 3.7.6.1 of SQLite is recommended for all new development. Upgrading from

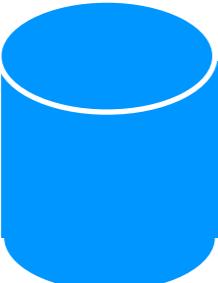
Drive7/64

1-0-5. 永続化の形式の例

例えば、運動会とかデートとかそういう"イベント"みたいなものを保存しておきたいとして、

イベント名、開催日、場所、とかが保存出来ると良いとしよう。

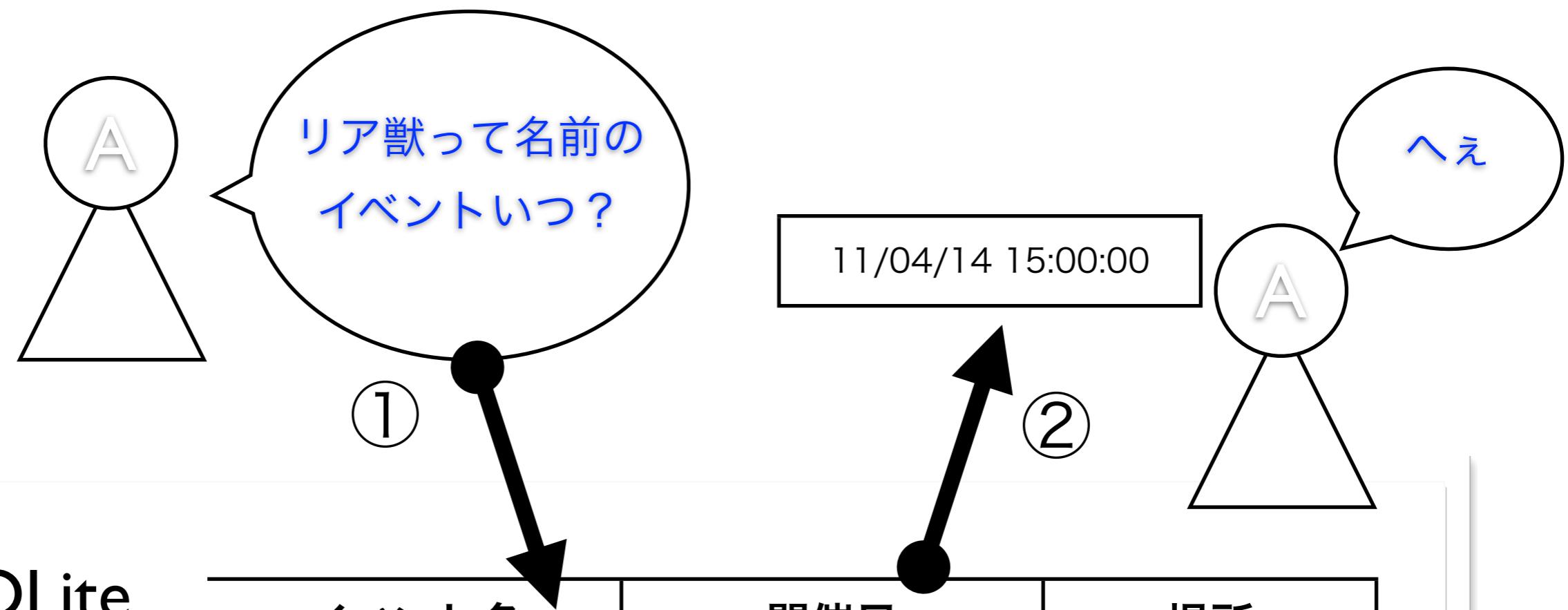
SQLite



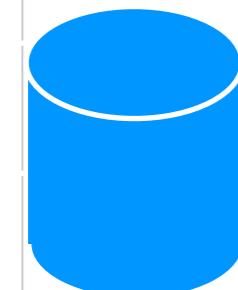
イベント名	開催日	場所
リア獣の会	11/04/14 15:00:00	お台場
リア充の会	11/04/12 11:00:00	六本木ヒルズ
...

Drive8/64

1-0-6. こういうデータを、沢山ぶち込んで、
全部くれ、とか、最近の奴だけくれとか、このイベント名の奴だけくれ、とか
言えばいいわけですね。



SQLite



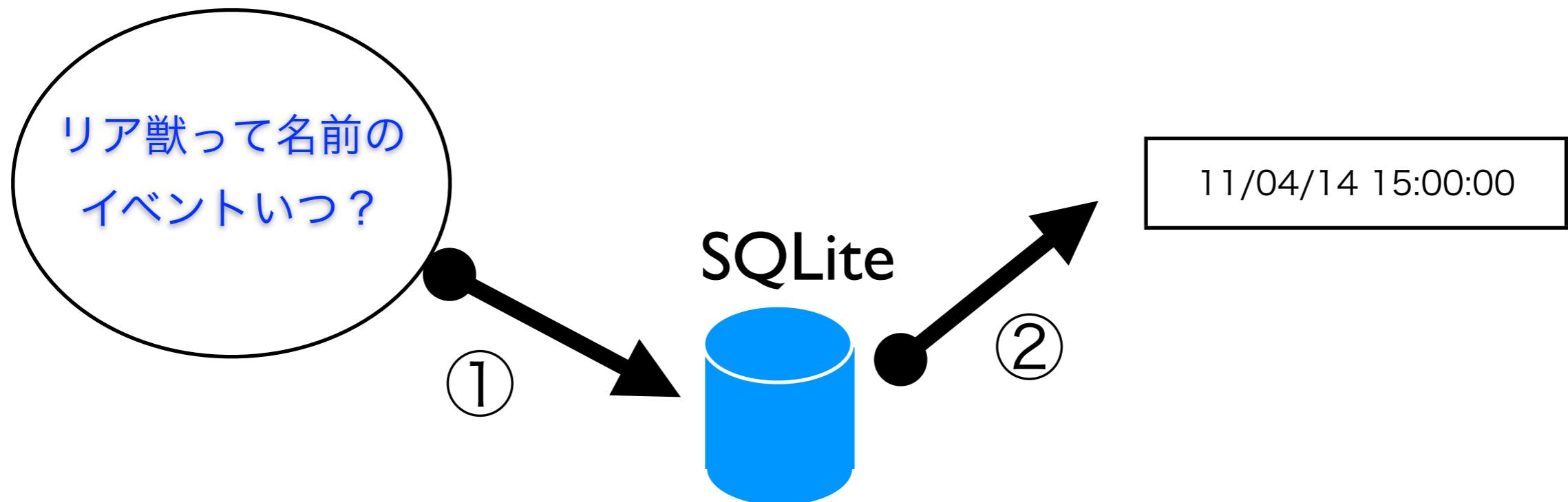
イベント名	開催日	場所
リア獣の会	11/04/14 15:00:00	お台場
リア充の会	11/04/12 11:00:00	六本木ヒルズ
...

Drive9/64

1-0-7. 実際どういうシーンで使う？

iOSの中で考えよう。

コレちょーだい、って言ったら、、、



こんな風にゲット出来る。文字列とか、Byteとかだね。

でも、コレだと、使いづれー。

だって更新とかする時、「リア獣って名前のイベントの日時を変えたい」って

言わなきゃ、変更する箇所が特定出来ない。

日時変えたいだけなのに。

Drive | 0/64

1-0-8. データを扱い易く→オブジェクト化

「なんていうかオブジェクトで貰えると、嬉しいんじゃね？」
って考えた人たちが居た。

というか、実はオブジェクト指向は、データを固めて扱う、という意味では、
現在のデータベースと非常に相性がいいように作られている。
そういうふうに進化して来た。

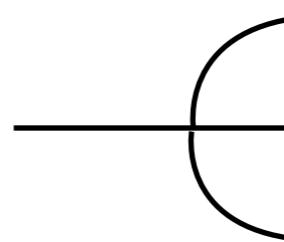
で、

"イベント"

= イベント名、開催日、場所 の情報を含んでるオブジェクトと考えよう。



"イベント"



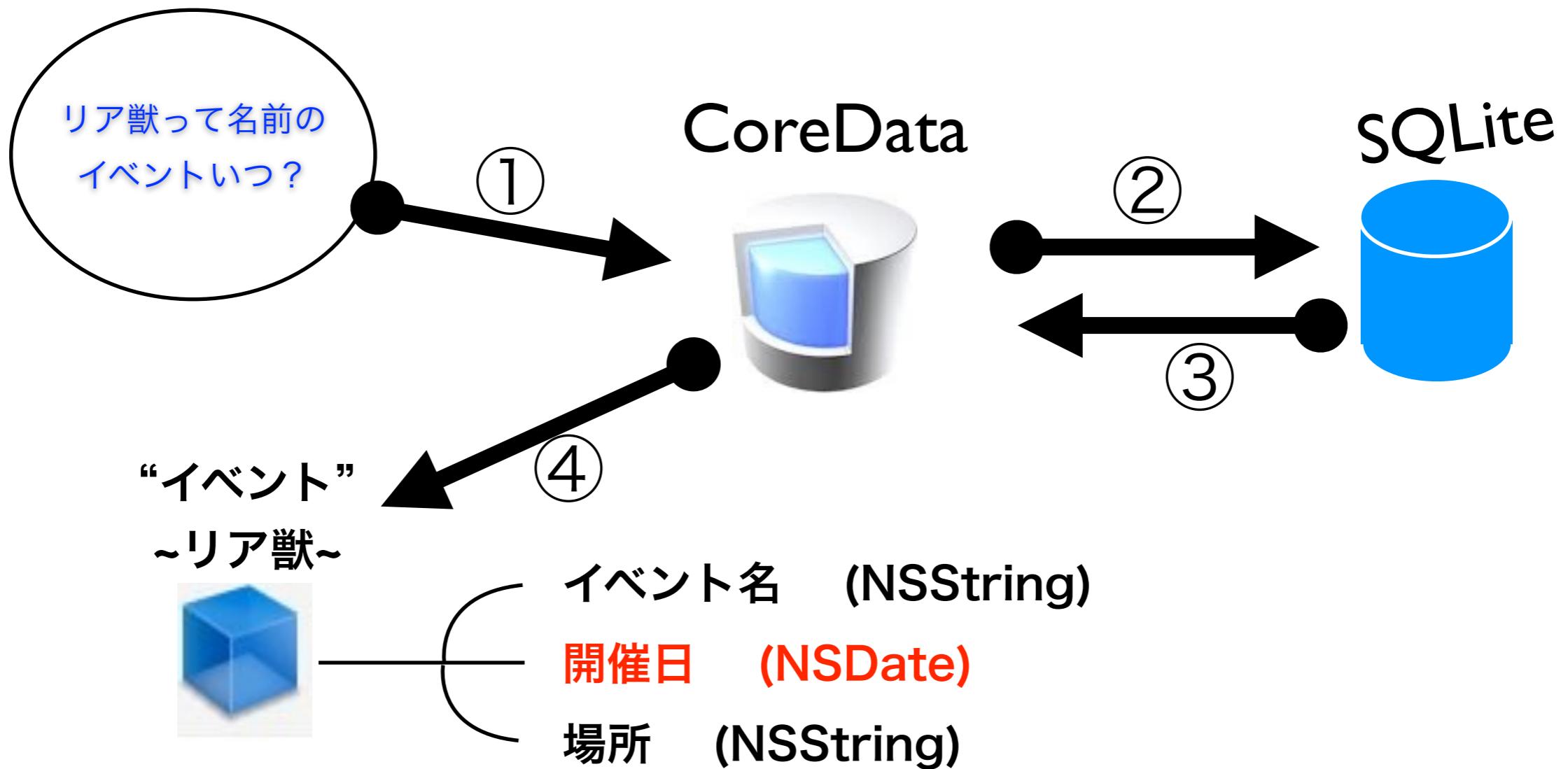
イベント名 (NSString)
開催日 (NSDate)
場所 (NSString)

Drive II/64

1-0-9. そこでCoreData

CoreDataというiOSのフレームワークは、まさにその、

SQLiteからオブジェクトを引っ張り出したり
SQLiteにオブジェクトを保存したり、という部分を請け負ってくれる。



Drive | 2/64

2. そこで CoreData

Drive | 3/64

2-0-0.実際に使うCoreData

先ずは、全体像を把握する。

NSManagedObjectModel

保存したいオブジェクトの定義がしてある["モデル"の情報]を扱うクラス。

▪ xcdata model ファイル

[保存したいオブジェクトの定義がしてある"モデル"の情報]が入ってるファイル。

NSPersistentStoreCoordinator

SQLiteのデータファイルを扱うクラス。

▪ sqlite ファイル

SQLiteの形式で保存されているデータファイル、それ自体。

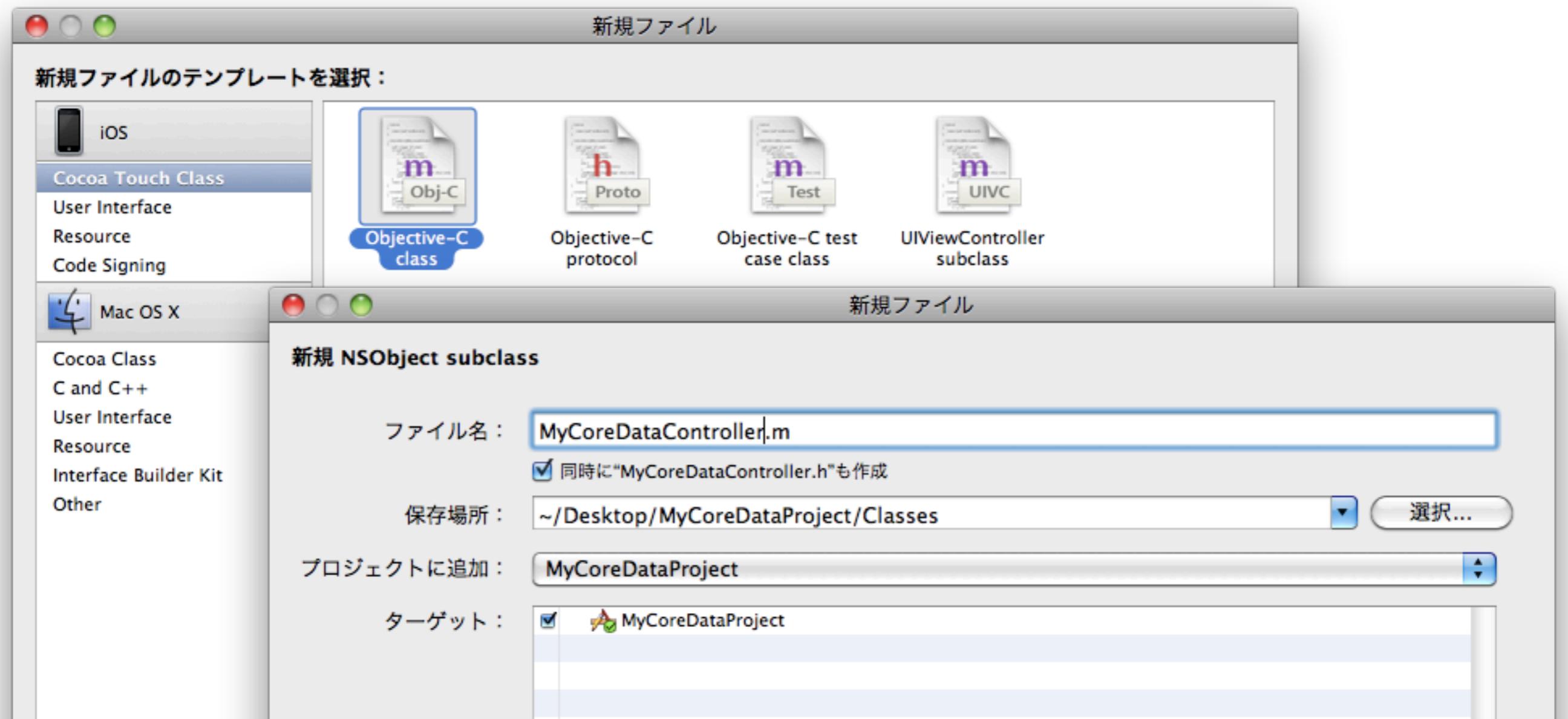
NSManagedObjectContext

コンテキスト。現在実行している事象のポイント、
プログラム中の"今、ここ"という視点を扱うクラス。

Drive | 4/64

2-0-1.やってみよう
プロジェクト制作
MyCoreDataProject

今回の主役、MyCoreDataControllerクラスを作る。



Drive | 5/64

2-0-2. CoreDataを扱う為のクラスを作る

NSManagedObjectModel

.xcdatamodelファイル

NSPersistentStoreCoordinator

.sqliteファイル

NSManagedObjectContext

こいつらを作ろう。

Drive | 6/64

2-0-3. 手始め

NSManagedObjectContext

と

NSManagedObjectModel

と

NSPersistentStoreCoordinator

のオブジェクトを、定義する。



The screenshot shows the Xcode IDE interface with the following details:

- Title Bar:** Shows "MyCoreDataController.h" as the active file.
- Toolbar:** Includes icons for Simulator (red), Run (yellow), Stop (green), Breakpoint (blue), Build and Run (black hammer), and Task (red octagon).
- Document Window:** Displays the code for `MyCoreDataController.h`. The code defines an interface for a Core Data controller, specifying three properties: `NSManagedObjectContext`, `NSManagedObjectModel`, and `NSPersistentStoreCoordinator`.

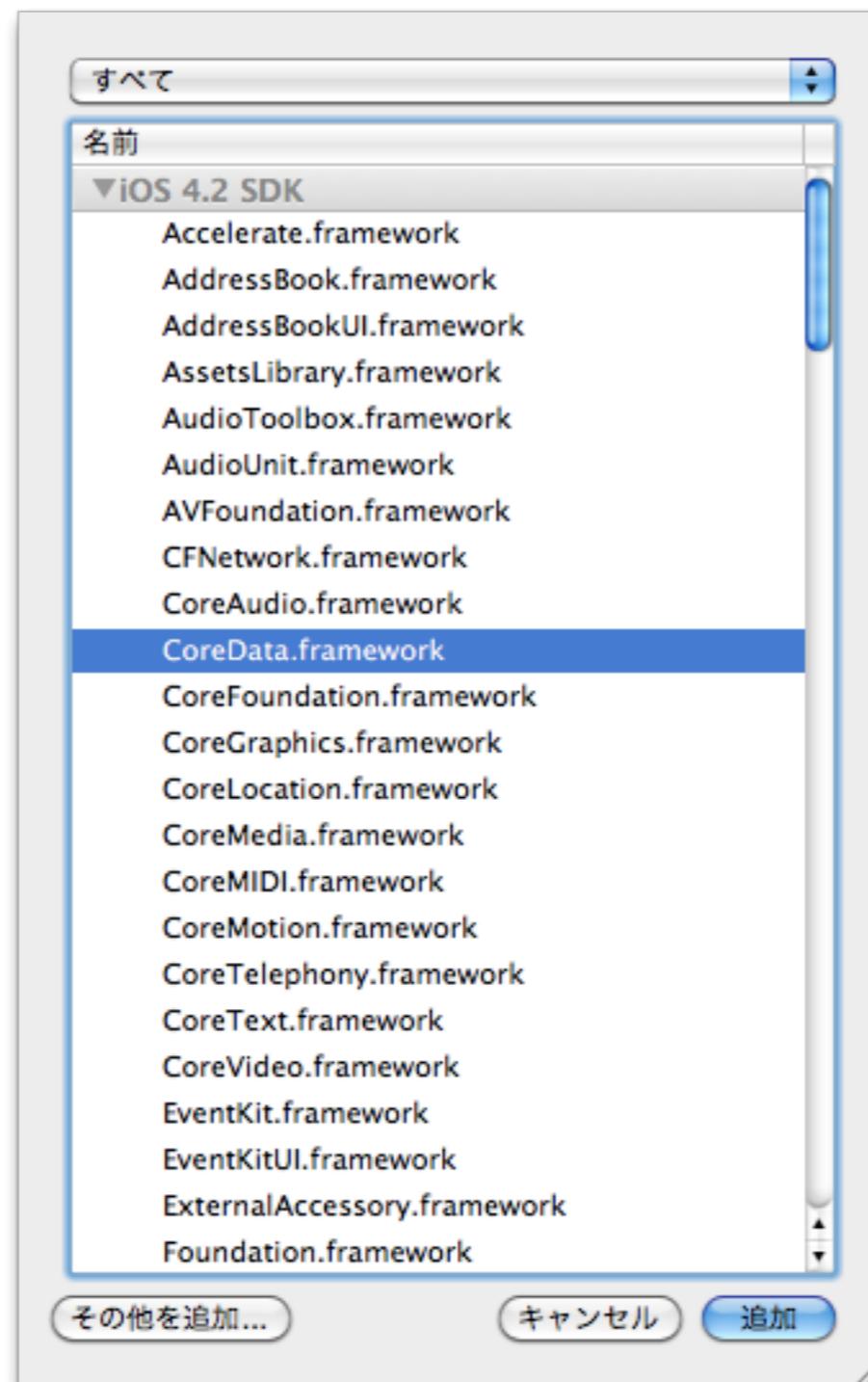
```
// MyCoreDataController.h
// MyCoreDataProject
//
// Created by Toru Inoue on 11/04/14.
// Copyright 2011 KISSAKI. All rights reserved.

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

@interface MyCoreDataController : NSObject {
    NSManagedObjectContext * myNSManagedObjectContext;
    NSManagedObjectModel * myNSManagedObjectModel;
    NSPersistentStoreCoordinator * myNSPersistentStoreCoordinator;
}
@end
```

Drive 17/64

2-0-4.CoreDataFrameworkをプロジェクトに読み込む



Drive | 8/64

2-0-5.CoreData関連の、.xcdatamodelファイルを作る

Resourceフォルダの下にでも、作ろう。

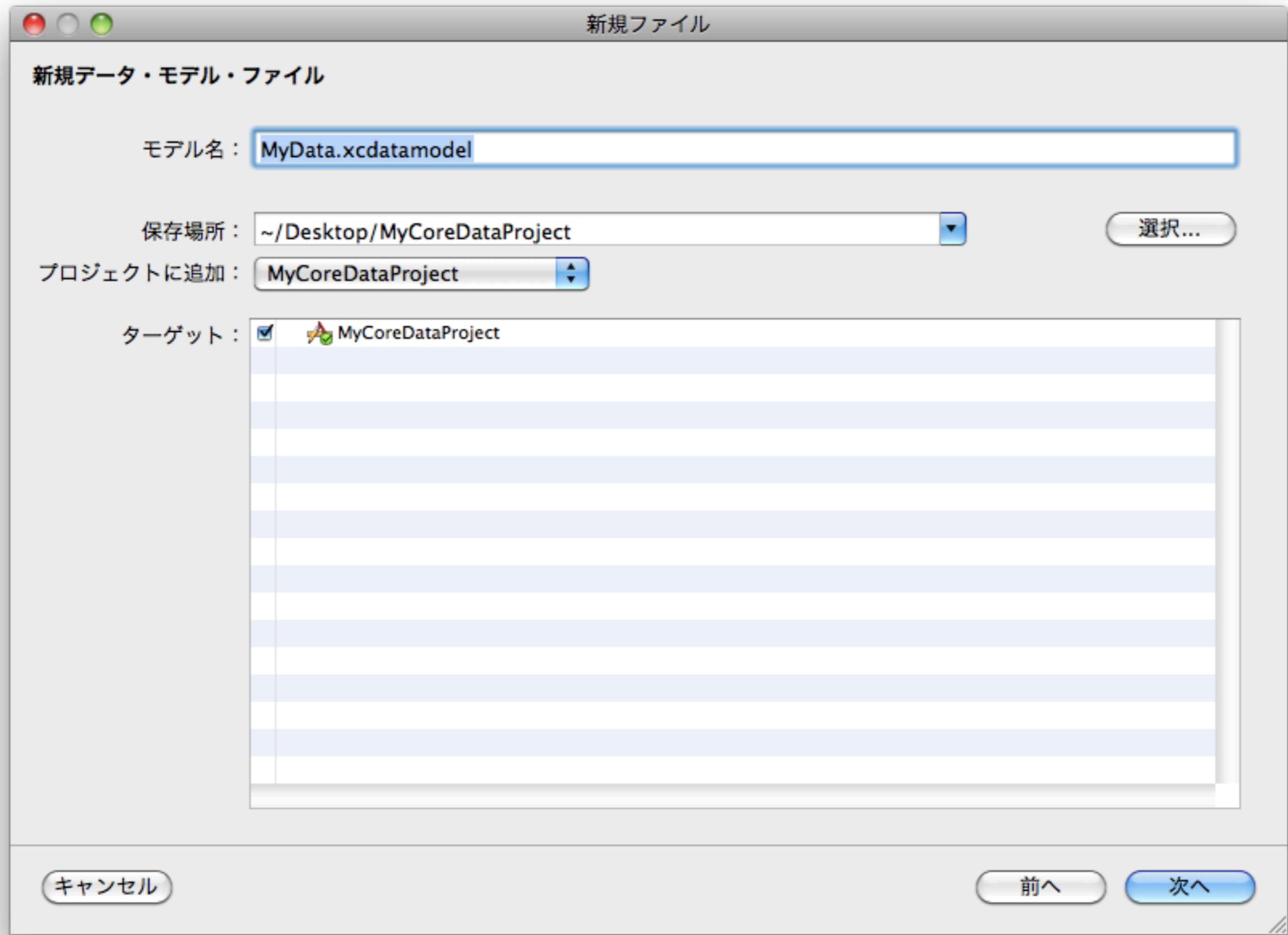
名前は、MyData.xcdatamodel とする。

なんか何画面か出てくるけどハイハイって感じで次へ。



Drive | 9/64

2-0-6.. xcdatamodel ファイル(2)



Drive20/64

2-0-7.出来上がったMyData.xcdatamodelファイルを開く

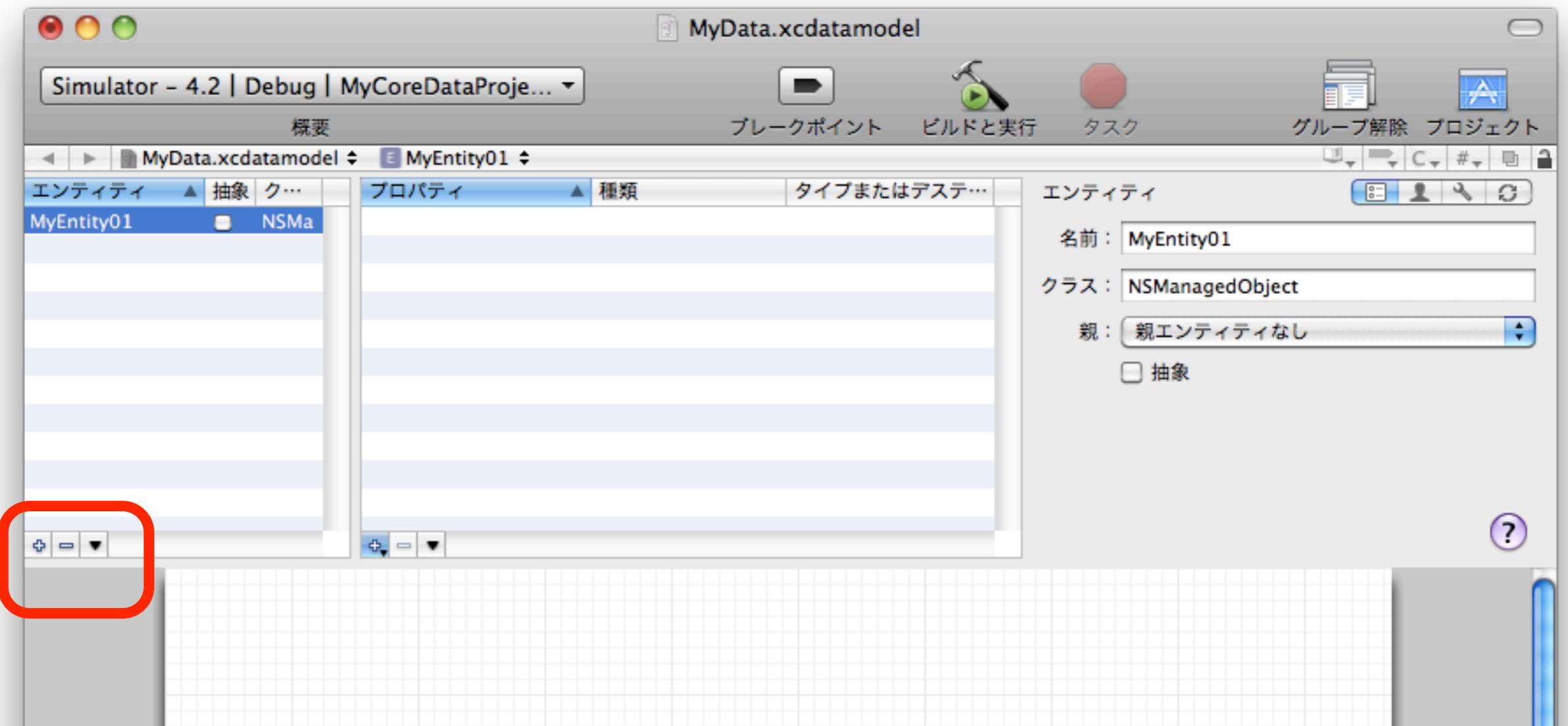
こんな画面が出てくる筈。

いきなりだけど、エンティティを作る。

+を押す。

出来上がったやつを改名しよう。

名前は、**MyEntity01**とかで。



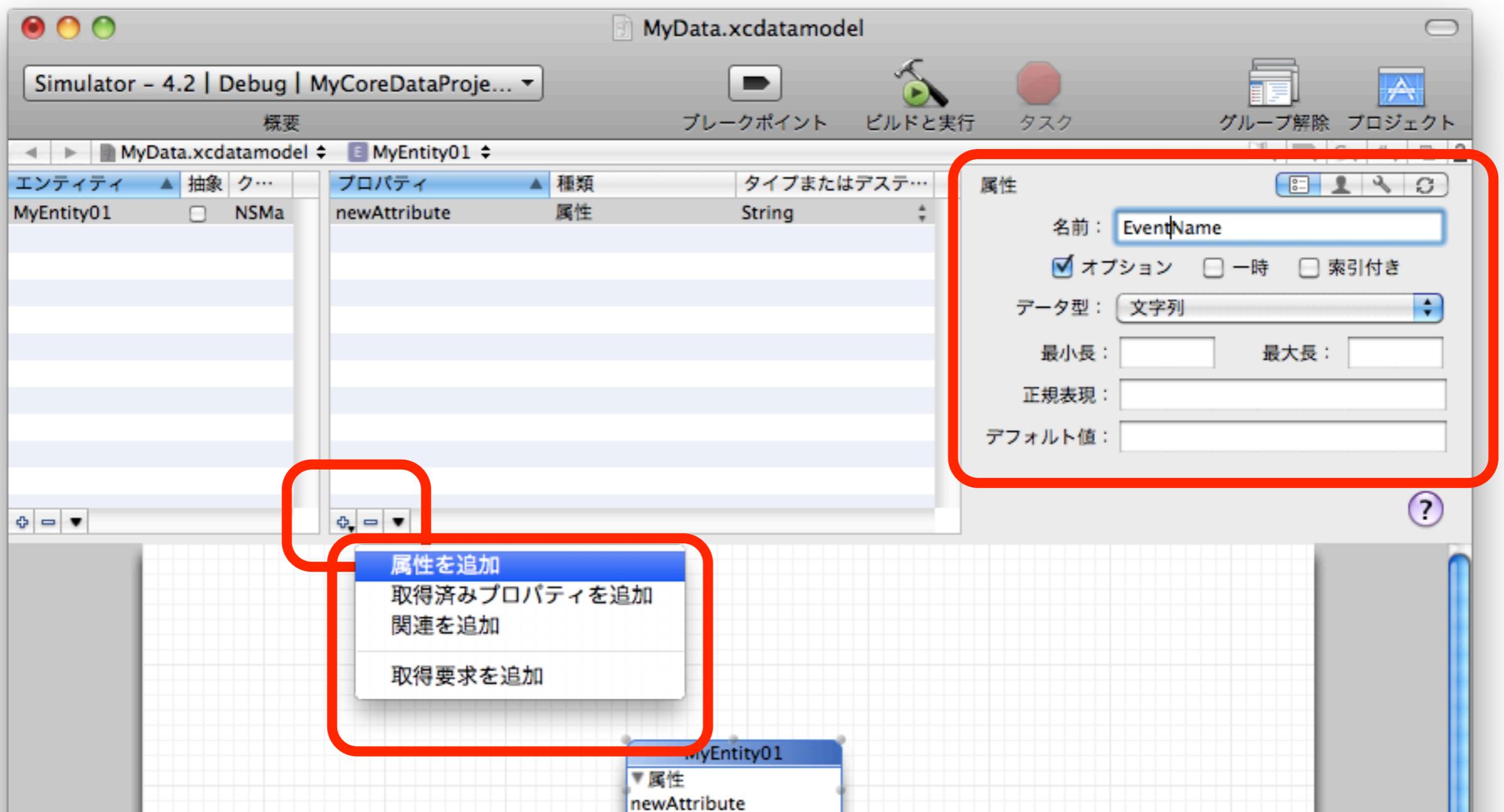
Drive2 | /64

2-0-8. プロパティを足す

一つ右の、プロパティのウインドウ、で、

+を押して、属性を足す。

名前は、EventNameにしこう。データ型は文字列。



Drive22/64

2-0-9.このエンティティ情報を持つ、クラスファイルを作る！

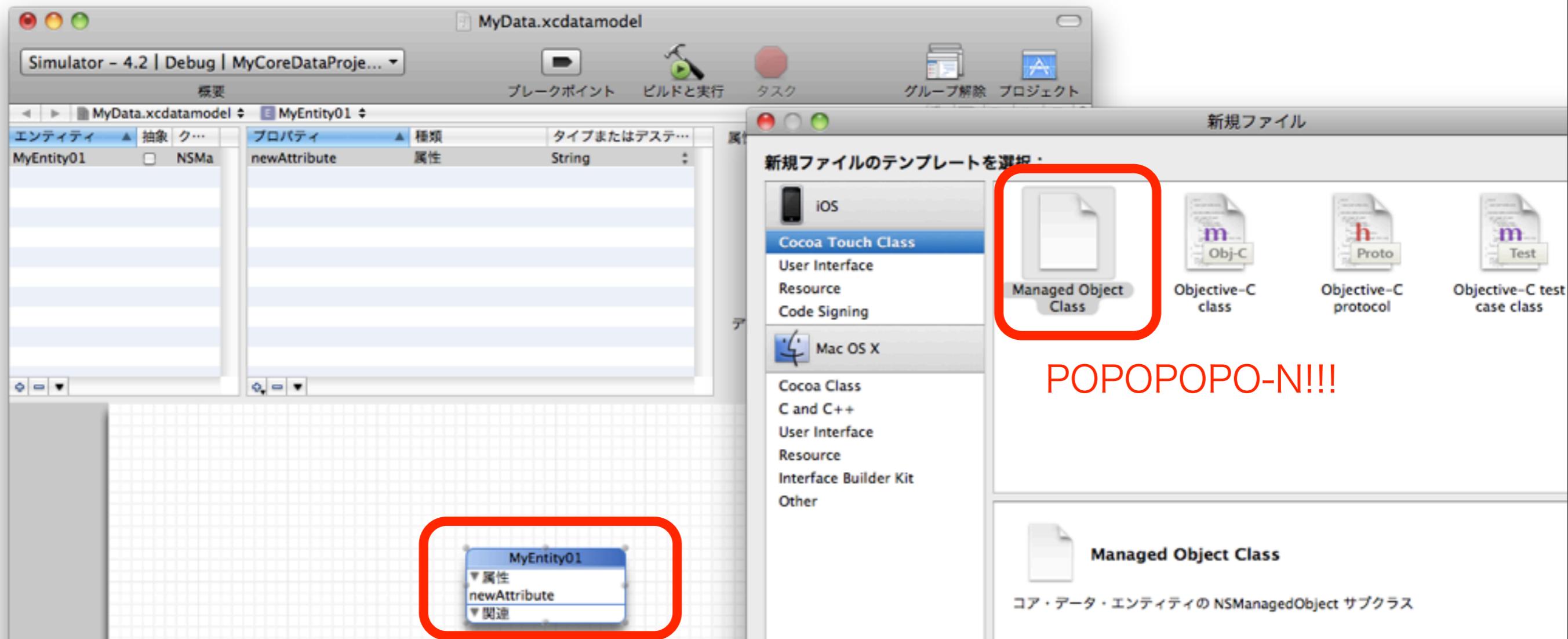
一番の鬼門。MyEntityを選択した状態で、

ファイル > 新規ファイル > CocoaTouchClass > ManagedObjectClass

or

⌘+n (!)

ってやると、新規作成ウィンドウにManagedObjectってのが！！



Drive23/64

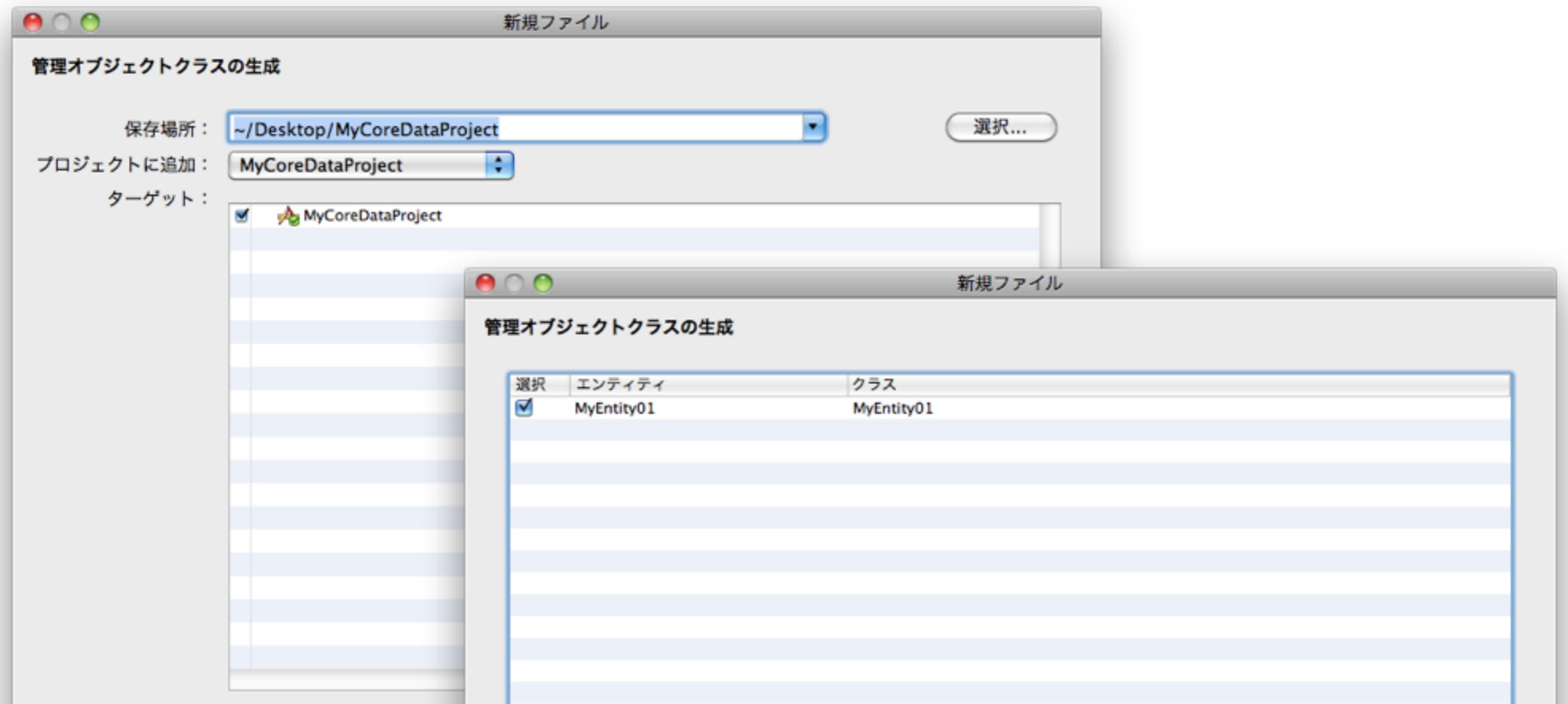
2-0-10. クラスファイル生成

詳細を決定、次々OK。

すると、Resourceフォルダに、

MyEntity01.h と MyEntity01.m が出来てる筈。

このクラスが、CoreDataで永続化を行う際の、生命線になります。



Drive24/64

2-1-0.CoreDataをコントロールする用のオブジェクトを生成するメソッド書く
NSManagedObjectContext

と

NSManagedObjectModel
と

NSPersistentStoreCoordinator
について、それぞれインスタンス化するメソッドを書き足す。

まず**NSPersistentStoreCoordinator**

Drive25/64

2-1-1. まずNSPersistentStoreCoordinator

```
MyCoreDataController.m
MyCoreDataController.m:31 -myNSPersistentStoreCoordinator

#import "MyCoreDataController.h"

@implementation MyCoreDataController

- (NSPersistentStoreCoordinator *) myNSPersistentStoreCoordinator {
    if (myNSPersistentStoreCoordinator != nil) {
        return myNSPersistentStoreCoordinator;
    }

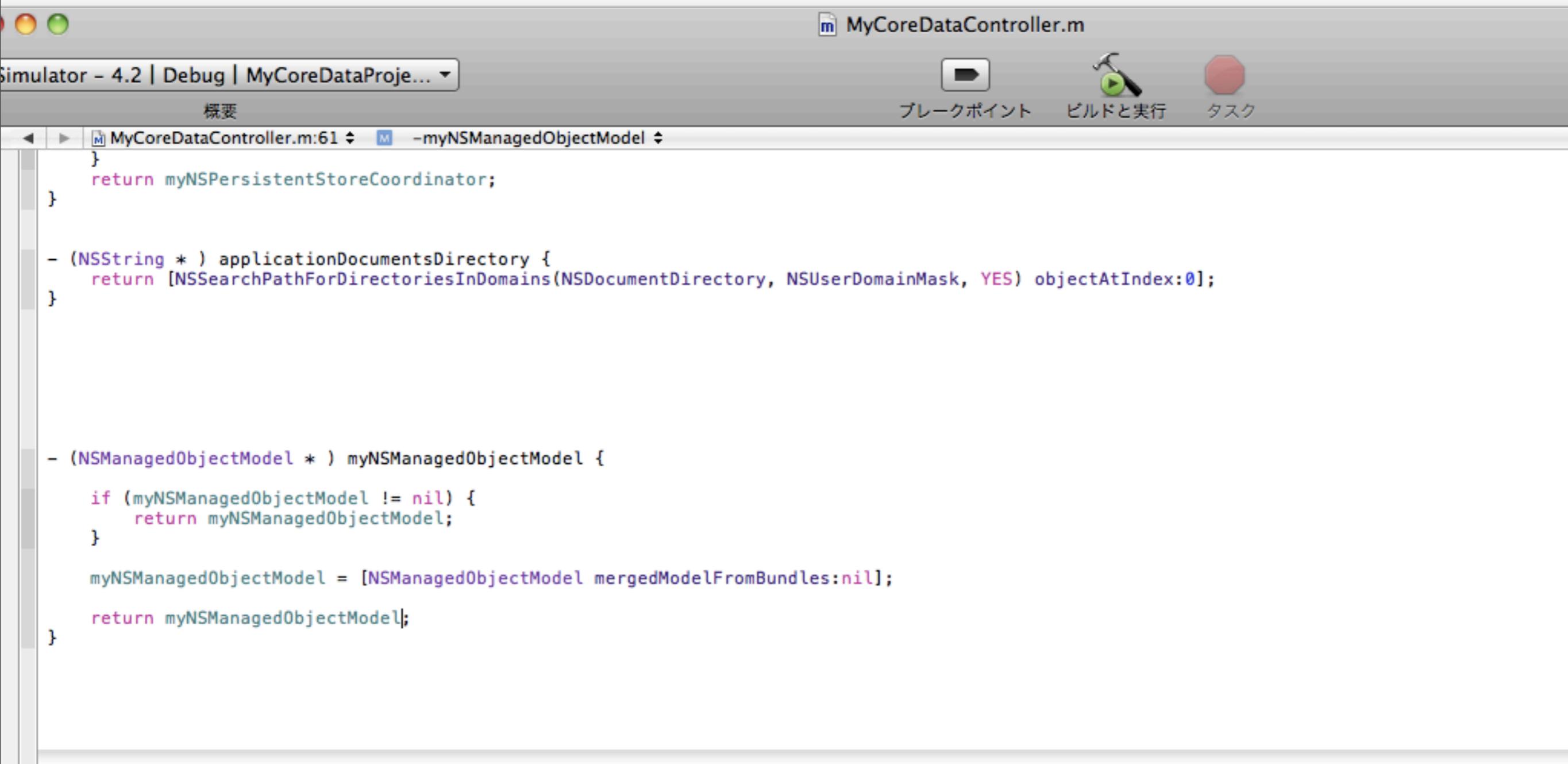
    //sqliteファイルを取得
    NSURL * storeUrl = [NSURL fileURLWithPath:
        [
            [self applicationDocumentsDirectory] stringByAppendingPathComponent:
            [NSString stringWithFormat:@"MySaveData.sqlite"]
        ]
    ];

    NSError * storeSettingError = nil;
    myNSPersistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:[self myNSManagedObjectModel]];
    if (![myNSPersistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:storeUrl options:nil error:&storeSettingError]) {
        if (storeSettingError) {
            NSLog(@"storeSettingError %@", storeSettingError);
        }
        NSAssert(FALSE, @"wrong pass");
    }
    return myNSPersistentStoreCoordinator;
}

- (NSString *) applicationDocumentsDirectory {
    return [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];
}
```

Drive26/64

2-1-2.NSManagedObjectModelを生成するメソッド書く



The screenshot shows the Xcode IDE interface with the following details:

- Title Bar:** Shows "MyCoreDataController.m" as the active file.
- Toolbar:** Includes icons for Stop, Run, Breakpoint, Build & Run, and Task.
- Document Area:** Displays the code for `MyCoreDataController.m`. The code includes methods for returning persistent store coordinators and managed object models.

```
MyCoreDataController.m:61 -myNSManagedObjectContext
}
return myNSPersistentStoreCoordinator;
}

- (NSString * ) applicationDocumentsDirectory {
    return [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];
}

- (NSManagedObjectModel * ) myNSManagedObjectModel {
    if (myNSManagedObjectModel != nil) {
        return myNSManagedObjectModel;
    }
    myNSManagedObjectModel = [NSManagedObjectModel mergedModelFromBundles:nil];
    return myNSManagedObjectModel;
}
```

Drive27/64

2-1-3.NSManagedObjectContextを生成するメソッド書く

The screenshot shows the Xcode IDE with the MyCoreDataController.m file open in the editor. The code defines a class method `+ (NSManagedObjectContext *) myNSManagedObjectContext` that returns an `NSManagedObjectContext`. It first checks if `myNSManagedObjectContext` is `nil`. If it is, it creates a new `NSManagedObjectContext`, sets its persistent store coordinator to `myNSPersistentStoreCoordinator`, and then returns it. If `myNSManagedObjectContext` is not `nil`, it simply returns it.

```
myNSManagedObjectContext = [NSManagedObjectContext mergedModelFromBundles:nil];
return myNSManagedObjectContext;
}

+ (NSManagedObjectContext *) myNSManagedObjectContext {
    if (myNSManagedObjectContext != nil) {
        return myNSManagedObjectContext;
    }

    NSPersistentStoreCoordinator * coordinator = [self myNSPersistentStoreCoordinator];
    if (coordinator != nil) {
        myNSManagedObjectContext = [NSManagedObjectContext new];
        [myNSManagedObjectContext setPersistentStoreCoordinator:coordinator];
    }
    return myNSManagedObjectContext;
}
```

Drive28/64

2-1-4.各メソッドをヘッダにも書いたら、一段落。
さて、次は実際に動かすコード。

データをcreateしてsaveするメソッドを書く
MyEntity01が出てくるので、importしどう。



The screenshot shows the Xcode IDE interface with the following details:

- Title Bar:** Shows "Simulator - 4.2 | Debug | MyCoreDataProj..." and the file name "MyCoreDataController.m".
- Toolbar:** Includes standard Xcode icons for Run, Stop, Breakpoint, Build & Run, and Task.
- Code Editor:** Displays the implementation of the `- (void) create` method in `MyCoreDataController.m`. The code uses Objective-C syntax and imports `"MyCoreDataController.h"` and `"MyEntity01.h"`.

```
// Created by Toru Inoue on 11/04/14.
// Copyright 2011 KISSAKI. All rights reserved.

#import "MyCoreDataController.h"
#import "MyEntity01.h"

@implementation MyCoreDataController

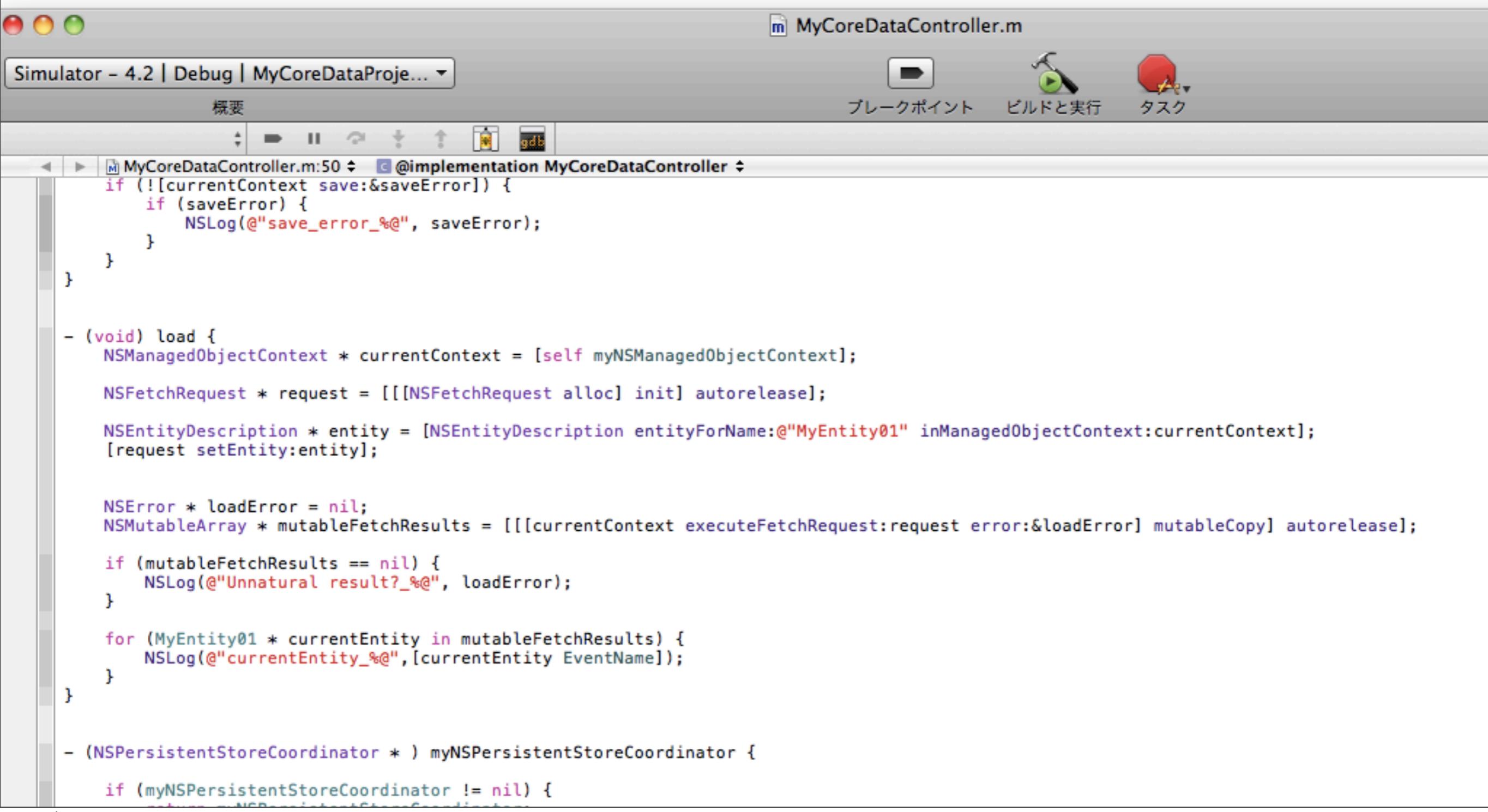
- (void) create {
    NSManagedObjectContext * currentContext = [self myNSManagedObjectContext];
    MyEntity01 * newEntity = (MyEntity01 *) [NSEntityDescription insertNewObjectForEntityForName:@"MyEntity01" inManagedObjectContext:currentContext];
    [newEntity setEventName:@"アンパンマン新しい顔よ！"];

    NSError * saveError = nil;
    if (![currentContext save:&saveError]) {
        if (saveError) {
            NSLog(@"save_error %@", saveError);
        }
    }
}
```

- Status Bar:** Shows "MyCoreDataProjectを起動" and "問題" (Issues).

Drive29/64

2-1-5. 続けて、ロードするメソッドを書く
(でないとセーブされたかどうか判らん)



```
MyCoreDataController.m
Simulator - 4.2 | Debug | MyCoreDataProj...
概要 プレークポイント ビルドと実行 タスク
MyCoreDataController.m:50 @implementation MyCoreDataController
if (![currentContext save:&saveError]) {
    if (saveError) {
        NSLog(@"save_error_%@", saveError);
    }
}

- (void) load {
    NSManagedObjectContext * currentContext = [self myNSManagedObjectContext];
    NSFetchedResultsController * request = [[[NSFetchRequest alloc] init] autorelease];
    NSEntityDescription * entity = [NSEntityDescription entityForName:@"MyEntity01" inManagedObjectContext:currentContext];
    [request setEntity:entity];

    NSError * loadError = nil;
    NSMutableArray * mutableFetchResults = [[[currentContext executeFetchRequest:request error:&loadError] mutableCopy] autorelease];

    if (mutableFetchResults == nil) {
        NSLog(@"Unnatural result?_%@", loadError);
    }

    for (MyEntity01 * currentEntity in mutableFetchResults) {
        NSLog(@"currentEntity_%@", [currentEntity EventName]);
    }
}

- (NSPersistentStoreCoordinator * ) myNSPersistentStoreCoordinator {
    if (myNSPersistentStoreCoordinator != nil) {
        ...
    }
}
```

Drive30/64

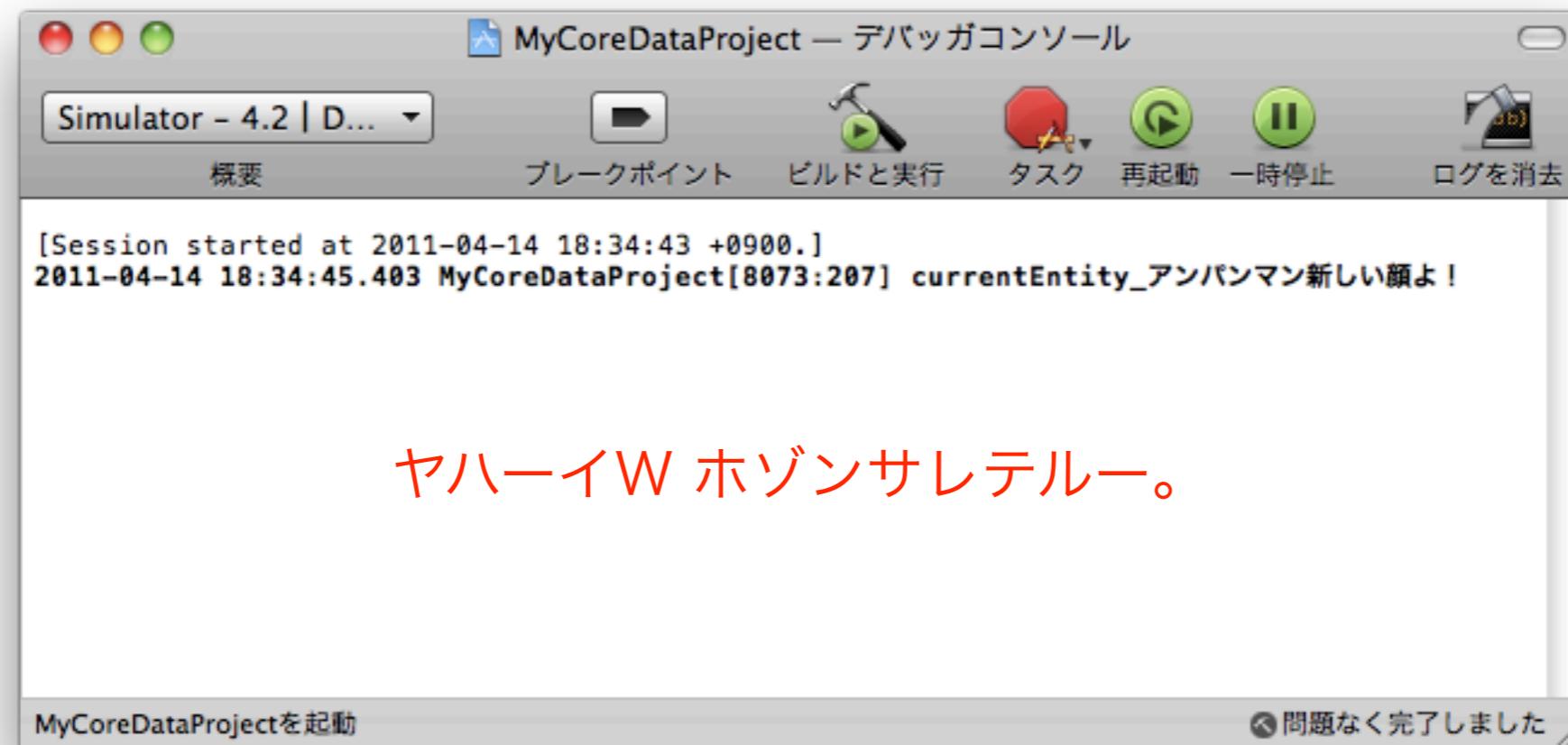
2-1-6.Delegateからcreateとload

Delegateの

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
メソッドに、

MyCoreDataProject オブジェクトを初期化して、
createとloadをするコードを書く。

うまく行くと、、



Drive3 | /64

2-1-7. 軽い解説

createメソッドでオブジェクトを作って、保存して、
loadメソッドで保存されたものを表示してます。

Drive32/64

2-2-0.解説、オブジェクトとソースとファイルの関係性を図にすると

NSManagedObjectModel * myNSManagedObjectModel

参照するよ → XXXXX.xcdatamodel

```
myNSManagedObjectModel = [NSManagedObjectModel mergedModelFromBundles:nil];
```

NSPersistentStoreCoordinator * myNSPersistentStoreCoordinator

参照するよ → YYYYY.sqlite

```
NSURL * storeUrl = (中略) [NSString stringWithFormat:@"MySaveData.sqlite"]
[myNSPersistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType
                                                 configuration:nil
                                                   URL:storeUrl
                                                 options:nil
                                                error:&storeSettingError]
```

NSManagedObjectContext

Drive33/64

2-2-1.意味、内容について解説

`NSManagedObjectModel * myNSManagedObjectModel`

初期化時に参照するよ

`XXXXX.xcdatamodel`

= 何てクラスをどんな風に入れるかという
ルールを記録したファイル。
取り出す時とか入れる時に使う。

`NSPersistentStoreCoordinator * myNSPersistentStoreCoordinator`

初期化時に参照するよ

`YYYYYY.sqlite`

=

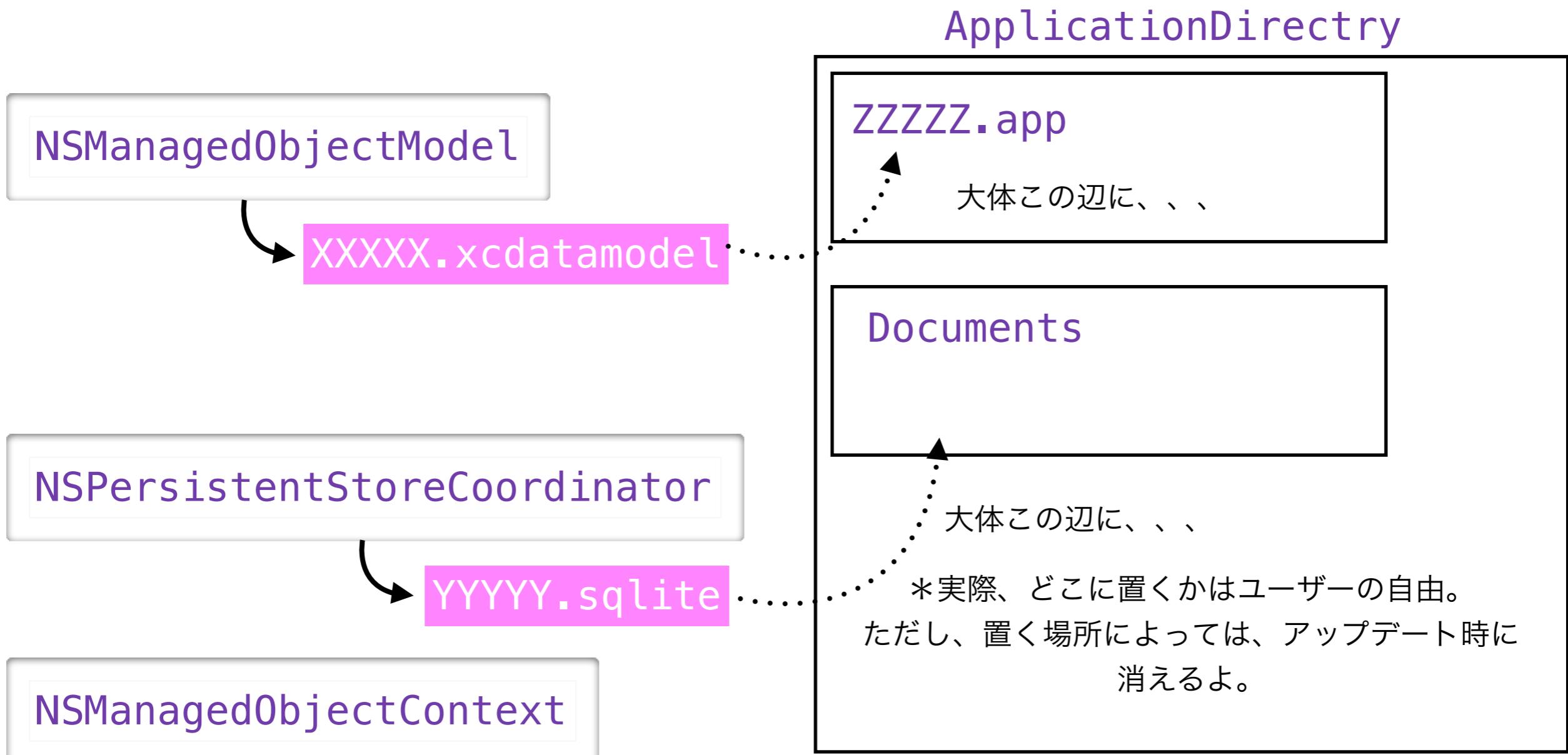
セーブデータの実体

`NSManagedObjectContext`

Drive34/64

2-2-2. 実際の配置

各ファイルは、アプリケーションのフォルダの中にこんな風に置かれてる。
で、前ページのメソッドで、myなんたらに関連づけられるように、
プログラムは動いている。



Drive35/64

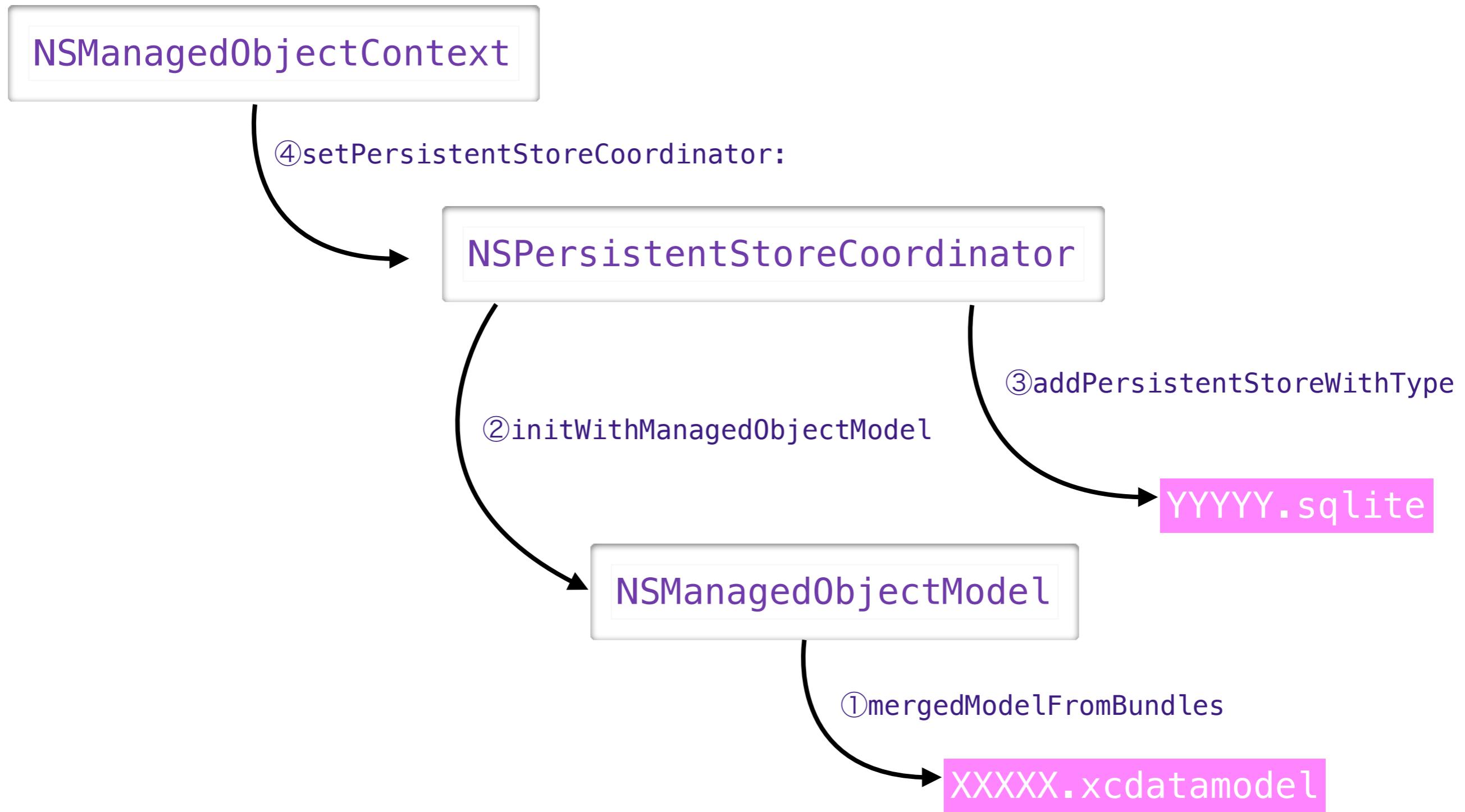
2-2-3. コンテキストから始まる処理

createも、**load**も、コンテキストの取得から始まっている。

これは、データストアについて、スゲー素敵な事が出来るようにするための配慮。
なんだが、動きを追おう。

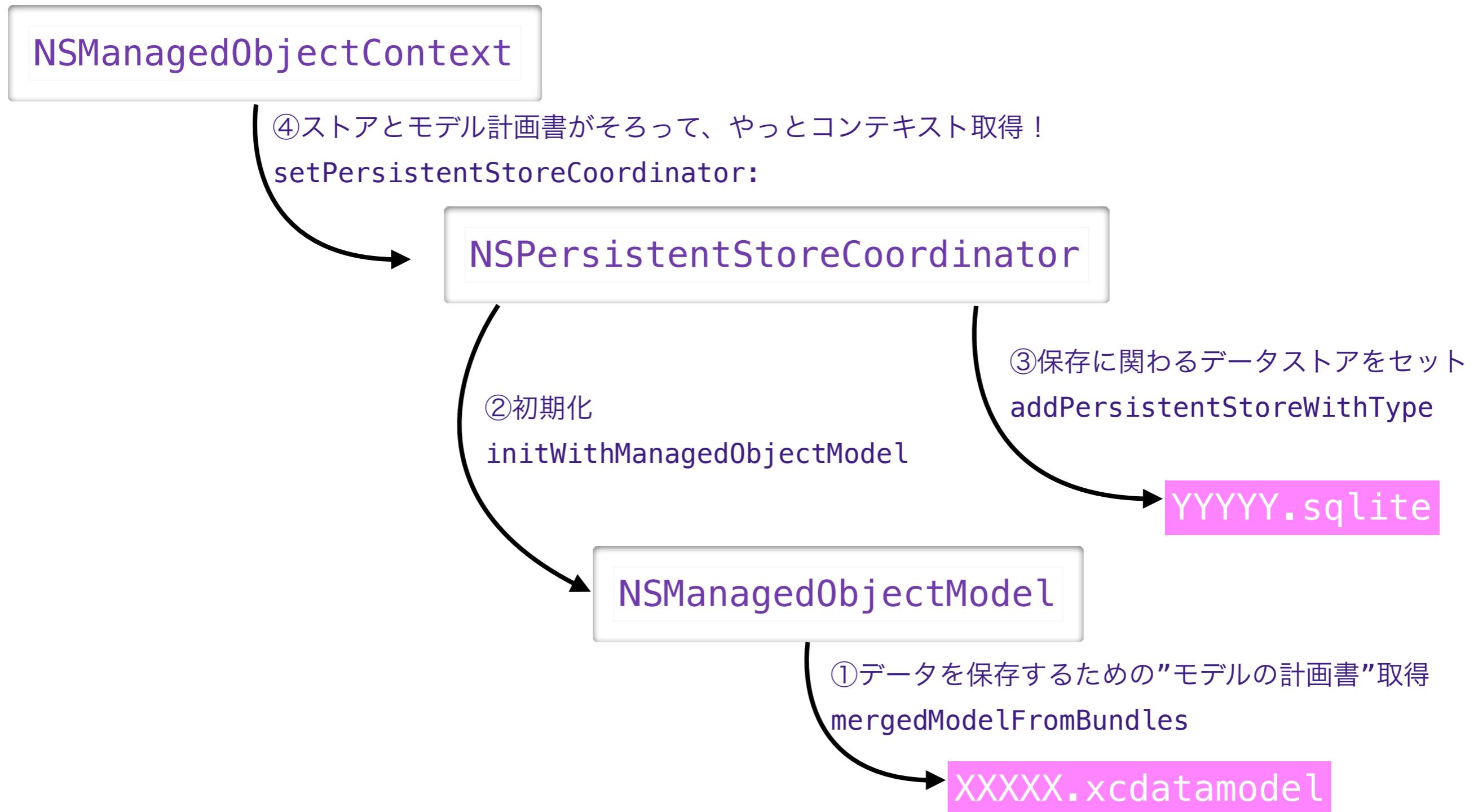
Drive36/64

2-2-4. myNSManagedObjectContextが作られるまでの実際の動き



Drive37/64

2-2-5. myNSManagedObjectContextが作られるまでの実際の動き(のさらに中身)



Drive38/64

2-2-6. createの続き

コンテキストは取得出来た。なので、

```
MyEntity01 * newEntity = (MyEntity01 *) [NSEntityDescription  
insertNewObjectForEntityForName:@"MyEntity01"  
inManagedObjectContext:currentContext];
```

で、新規に保存するオブジェクトを作り出す。

この時、

コンテキストが①の計画書を知っているので、その計画書にある、**MyEntity01**を指定して、

コンテキストから、新規に保存するオブジェクトを作り出す。

Drive39/64

2-2-7. コンテキストとは何か

→この新オブジェクト、コンテキストと密接に関係してて、
新オブジェクトに変化を加えてから
コンテキストに対して **save** を行う、という形になっている。

ここで、コンテキストは、"～に変化を加えた"という[状況]を現している。

現在のcreateメソッドの中なら、

- ・新しいオブジェクトを作った
 - ・オブジェクトの内容を弄った
- という、[状況]をまず作りだして、

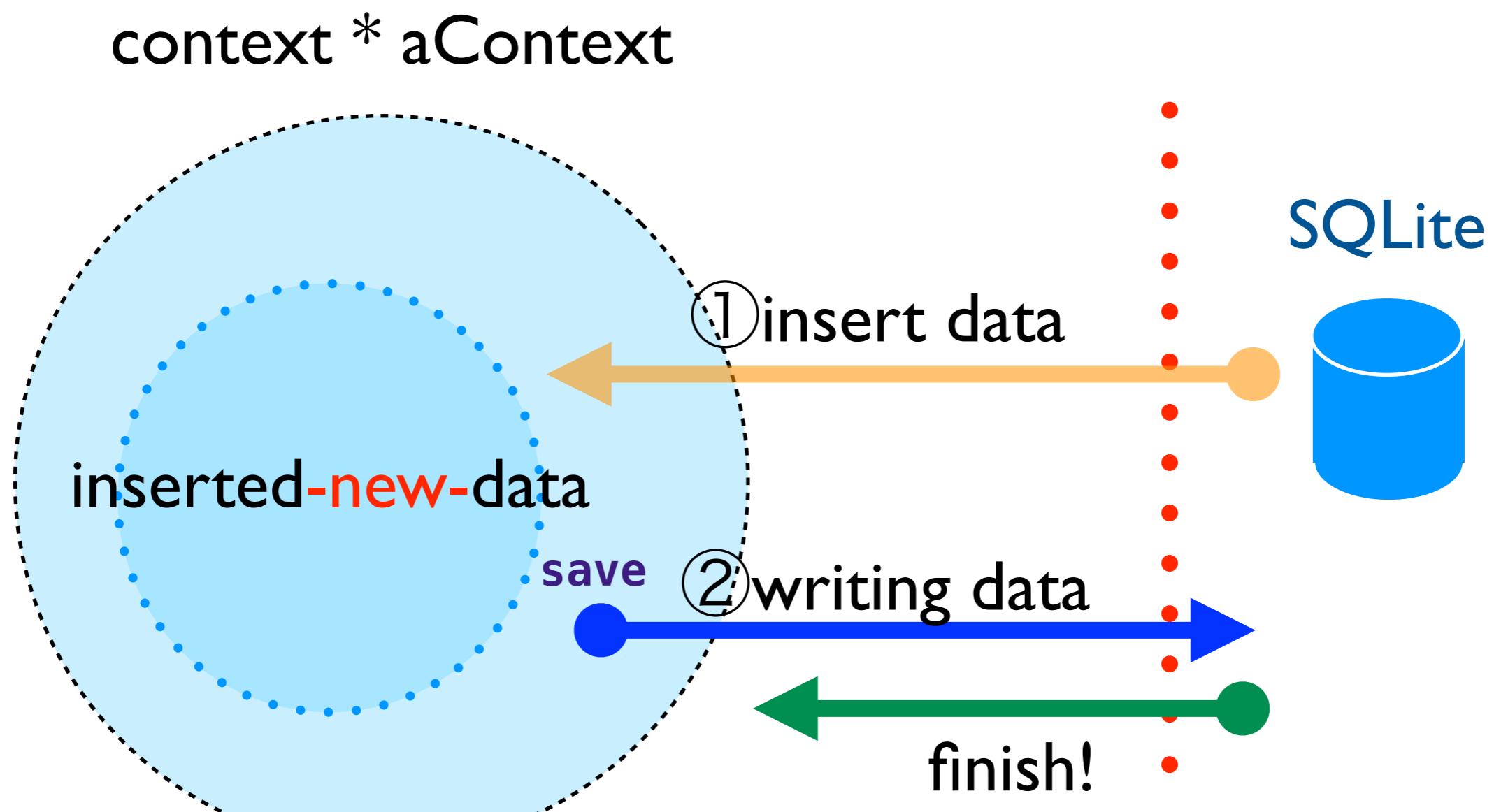
最後に、「この状況、アリ！」なら、**save** で、実際にデータを書き換えている。

Drive40/64

2-2-8. コンテキストとは何か(2)

→データベースに加える「変更点」を、メモリ上で一時的に扱う"机上データベース"

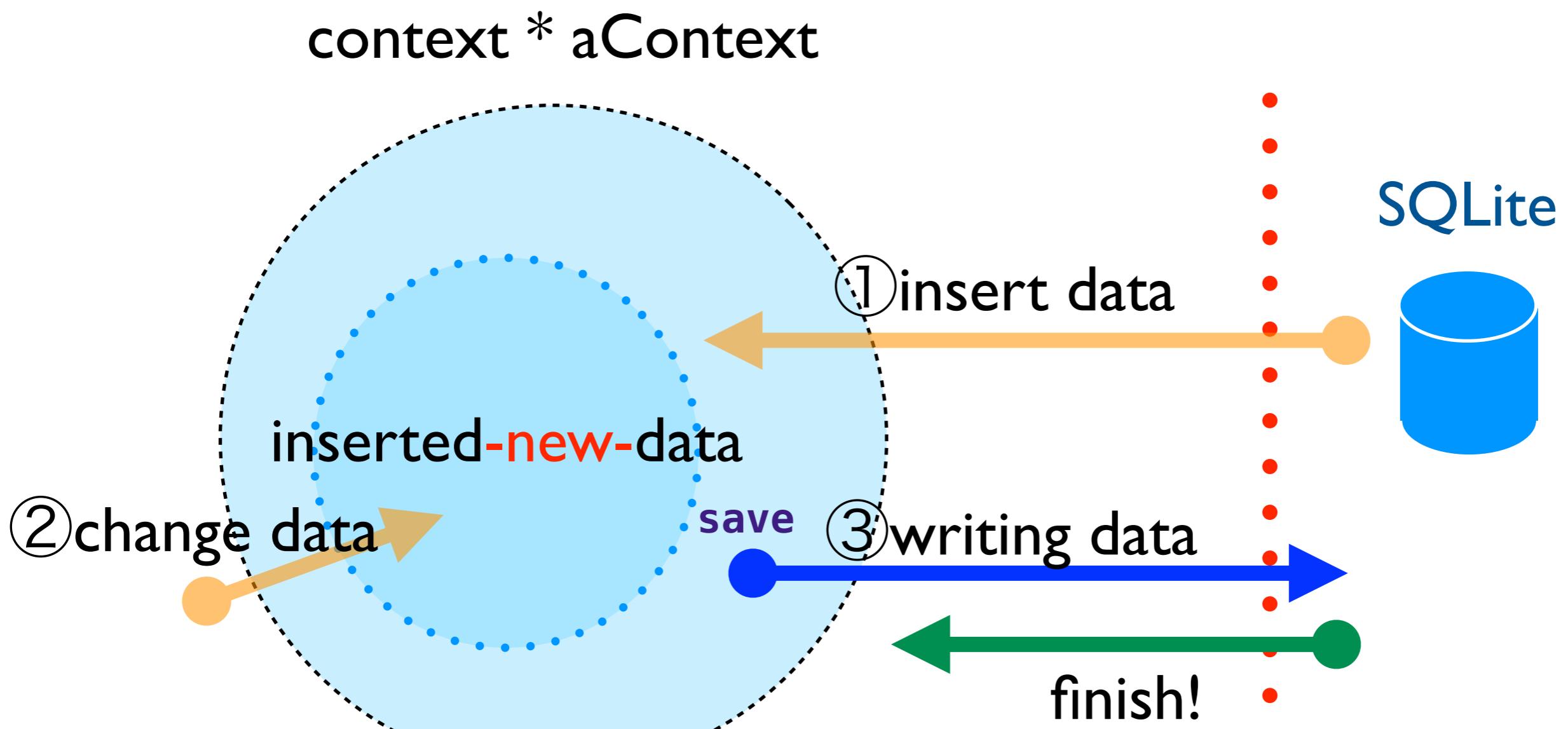
→**save**で、「変更点」を実際のデータベースに対して発生させる。



Drive4 | /64

2-2-8. コンテキストとは何か(3)

特に、さっき書いた create では、取得したオブジェクトに
“イベント名をセットする”という事をしている。



Drive42/64

2-3-0. loadはどうなっているのか

やっぱりコンテキストから**MyEntity01**を指定、

executeFetchRequest

で取得している。

ってことはここで内容を改変して**create**すると、保存されてる内容は変わらぬか？

→オブジェクトの内容の改変と、末尾に**create(セーブ代わり)**を書き足してみよう。

```
m MyCoreDataController.m
Simulator - 4.2 | Debug | MyCoreDataProj...
概要 ブレークポイント ビルドと実行 タスク グループ
MyCoreDataController.m:51 -load
- (void) load {
    NSManagedObjectContext * currentContext = [self myNSManagedObjectContext];
    NSFetchedResultsController * request = [[[NSFetchRequest alloc] init] autorelease];
    NSEntityDescription * entity = [NSEntityDescription entityForName:@"MyEntity01" inManagedObjectContext:currentContext];
    [request setEntity:entity];

    NSError * loadError = nil;
    NSMutableArray * mutableFetchResults = [[[currentContext executeFetchRequest:request error:&loadError] mutableCopy] autorelease];
    if (mutableFetchResults == nil) {
        NSLog(@"Unnatural result? %@", loadError);
    }

    for (MyEntity01 * currentEntity in mutableFetchResults) {
        NSLog(@"currentEntity %@", [currentEntity EventName]);
        [currentEntity setEventName:@"元気百倍って凄くね"];
    }
}

[self create];
```

Drive43/64

2-3-1.loadダッシュ

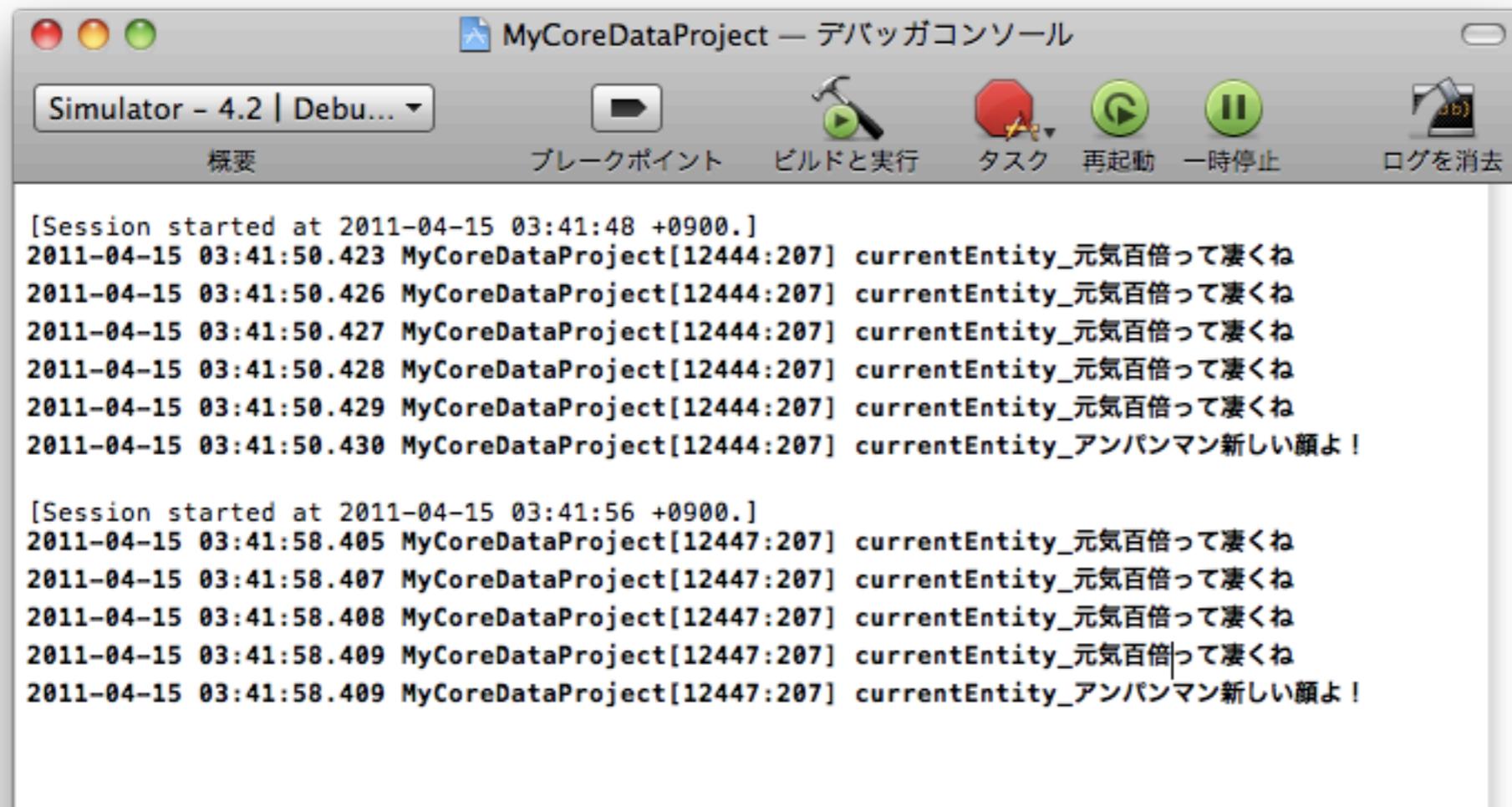
→変わった。

以前のアなんとか～ から、 元気～ に、 以前有った物が変わる。

NSLogで出しているのがsave直後のモノなのと、

このあとまたcreateが行われているので、

次の起動時にアなんとか～ が2つ新たに出る。

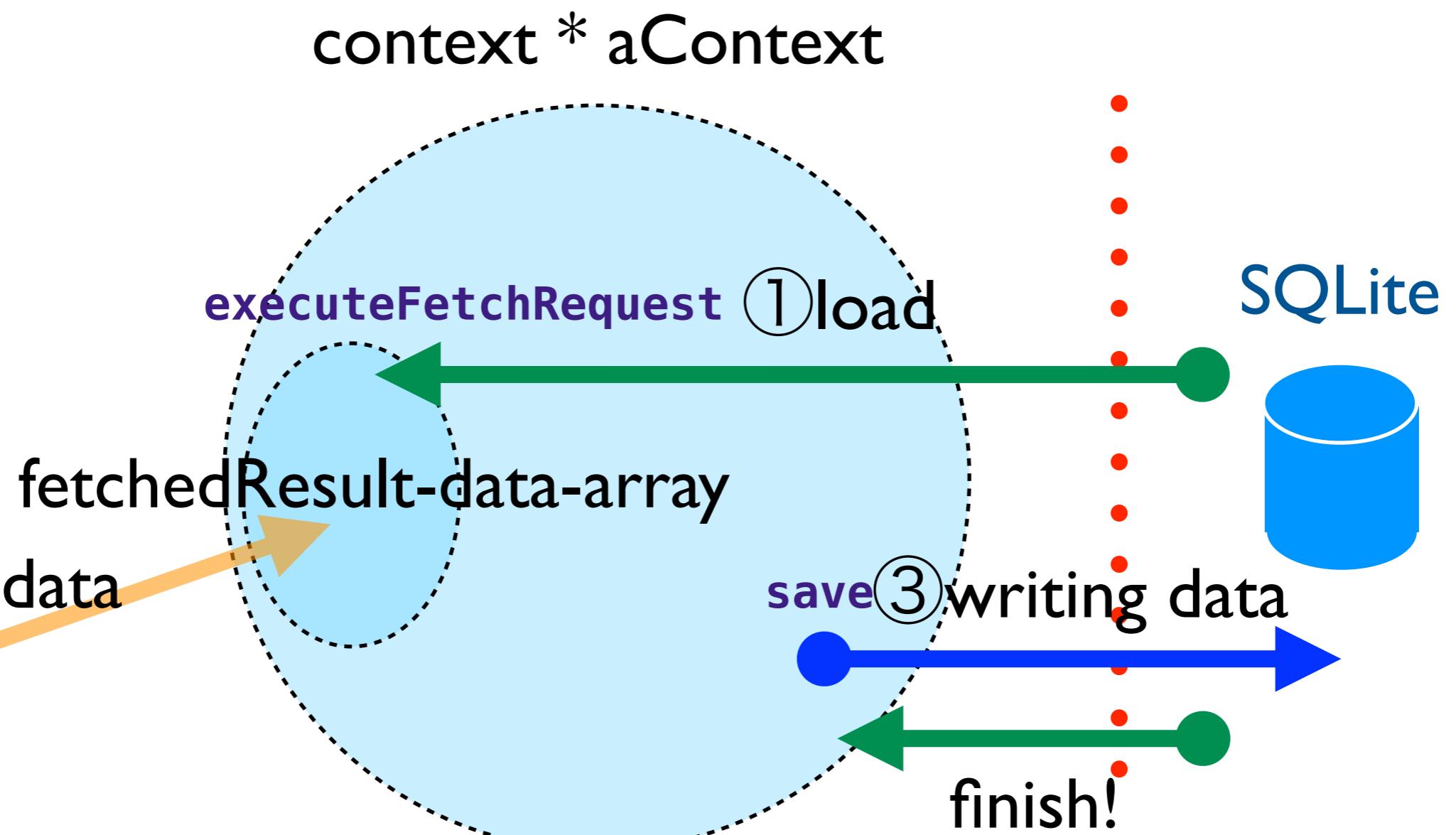


The screenshot shows the Xcode Debug Navigator window titled "MyCoreDataProject — デバッガコンソール". The window has a toolbar with various icons: Simulator - 4.2 | Debug, Breakpoint, Build & Run, Task, Restart, Suspend, and Clear Log. Below the toolbar, there are two sections of log output:

```
[Session started at 2011-04-15 03:41:48 +0900.]  
2011-04-15 03:41:50.423 MyCoreDataProject[12444:207] currentEntity_元気百倍って凄くね  
2011-04-15 03:41:50.426 MyCoreDataProject[12444:207] currentEntity_元気百倍って凄くね  
2011-04-15 03:41:50.427 MyCoreDataProject[12444:207] currentEntity_元気百倍って凄くね  
2011-04-15 03:41:50.428 MyCoreDataProject[12444:207] currentEntity_元気百倍って凄くね  
2011-04-15 03:41:50.429 MyCoreDataProject[12444:207] currentEntity_元気百倍って凄くね  
2011-04-15 03:41:50.430 MyCoreDataProject[12444:207] currentEntity_アンパンマン新しい顔よ!  
  
[Session started at 2011-04-15 03:41:56 +0900.]  
2011-04-15 03:41:58.405 MyCoreDataProject[12447:207] currentEntity_元気百倍って凄くね  
2011-04-15 03:41:58.407 MyCoreDataProject[12447:207] currentEntity_元気百倍って凄くね  
2011-04-15 03:41:58.408 MyCoreDataProject[12447:207] currentEntity_元気百倍って凄くね  
2011-04-15 03:41:58.409 MyCoreDataProject[12447:207] currentEntity_元気百倍って凄くね  
2011-04-15 03:41:58.409 MyCoreDataProject[12447:207] currentEntity_アンパンマン新しい顔よ!
```

Drive44/64

2-3-2.loadダッシュ解説



Drive45/64

2-4-0.物理的にデータを消す方法

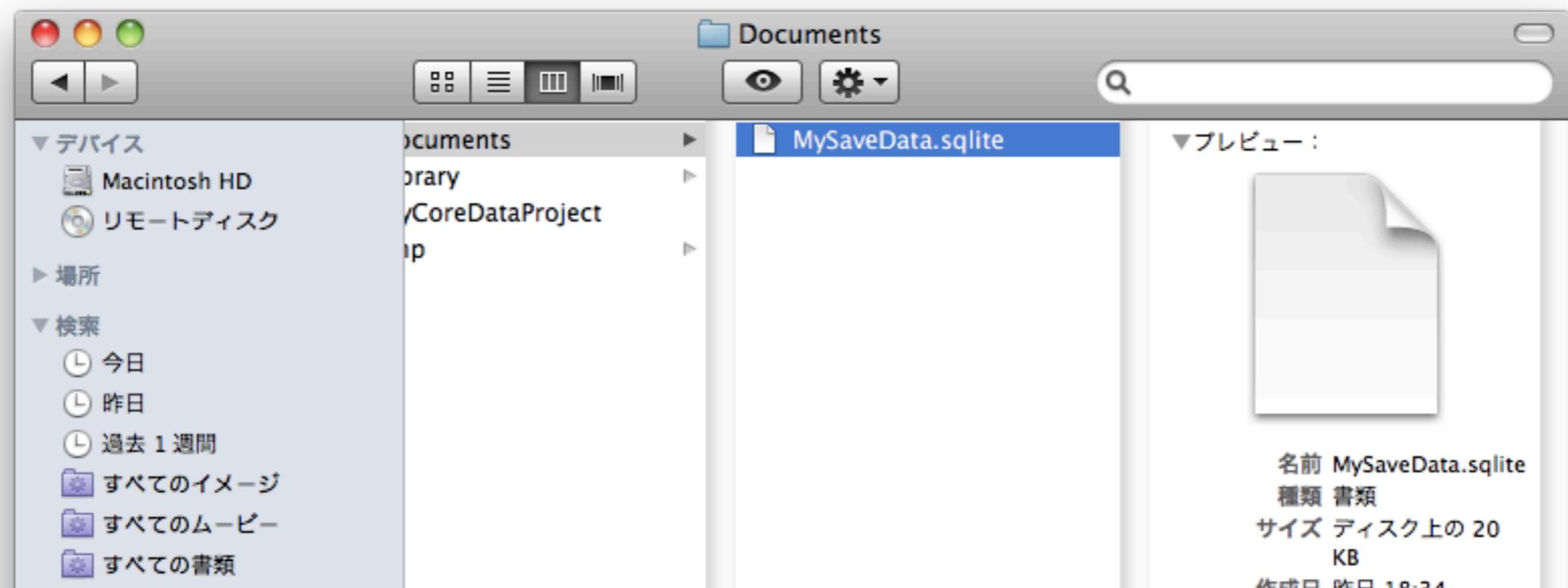
/Users/ユーザー名/Library/Application Support/iPhone Simulator/4.2/
Applications/

に行くと、シミュレータで動いてるアプリケーションの実ファイルがある。

→実際にiPhoneとかの中でもこうなっているんだぜ。

> Documents > ~.sqlite が有るから、ゴミ箱に捨てたり、このフォルダから出したりすると、

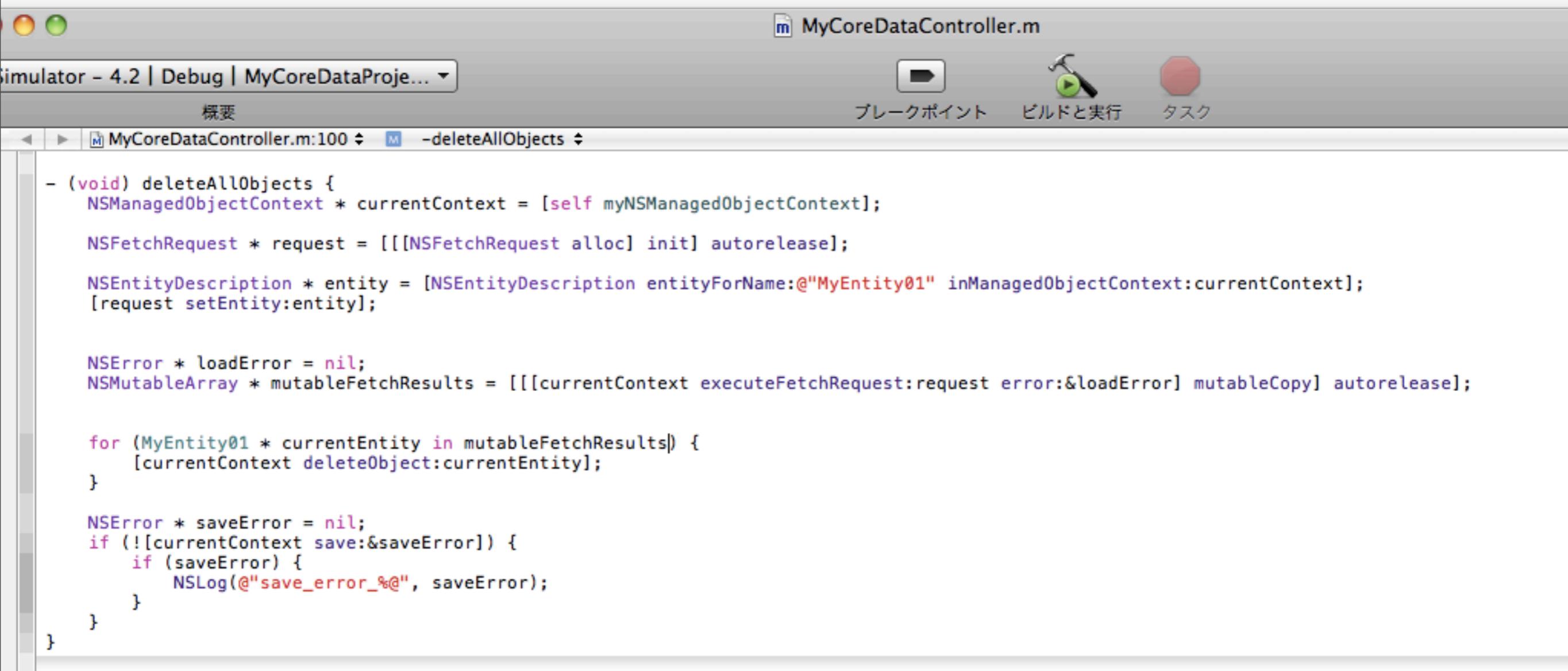
セーブデータはきれいに消える。



Drive46/64

2-4-1.手続き的にデータを消す

ロードしたものに対して全部消すメソッドを書くと、こんなカンジ。
Delegateに書き足すと、書き足したとこから先が消える。
メソッドのあとに、**save**を忘れないように。



```
MyCoreDataController.m
Simulator - 4.2 | Debug | MyCoreDataProj...
概要 ブレークポイント ビルドと実行 タスク
MyCoreDataController.m:100 -deleteAllObjects
- (void) deleteAllObjects {
    NSManagedObjectContext * currentContext = [self myNSManagedObjectContext];
    NSFetchedResultsController * request = [[[NSFetchRequest alloc] init] autorelease];
    NSEntityDescription * entity = [NSEntityDescription entityForName:@"MyEntity01" inManagedObjectContext:currentContext];
    [request setEntity:entity];

    NSError * loadError = nil;
    NSMutableArray * mutableFetchResults = [[[currentContext executeFetchRequest:request error:&loadError] mutableCopy] autorelease];

    for (MyEntity01 * currentEntity in mutableFetchResults) {
        [currentContext deleteObject:currentEntity];
    }

    NSError * saveError = nil;
    if (![currentContext save:&saveError]) {
        if (saveError) {
            NSLog(@"save_error %@", saveError);
        }
    }
}
```

Drive47/64

2-5-0. 取得するのをスマートに

SQLiteを使ってるのに、毎回全データを取得しないと行けないのか？

否、SQLiteなんだから、SQLが使える。

load時のrequestに、下記を加える事で、

エンティティ内のプロパティ(名前や日付)から、

50音の昇順だったりなんたりで並べられた物を

取得出来る。loadに下記を足すと、絞られた結果が得られる。

```
NSSortDescriptor * sortDescriptor = [[NSSortDescriptor alloc]
initWithKey:@"name" ascending:NO];
NSArray * sortDescriptorsArray = [[NSArray alloc]
initWithObjects:sortDescriptor, nil];

[request setSortDescriptors:sortDescriptorsArray];
```

他に、.xcdatamodelファイルに、SQLを登録する事も出来る。

ここでは扱わないけど。

Drive48/64

3. バージョンを超えていけ

Drive49/64

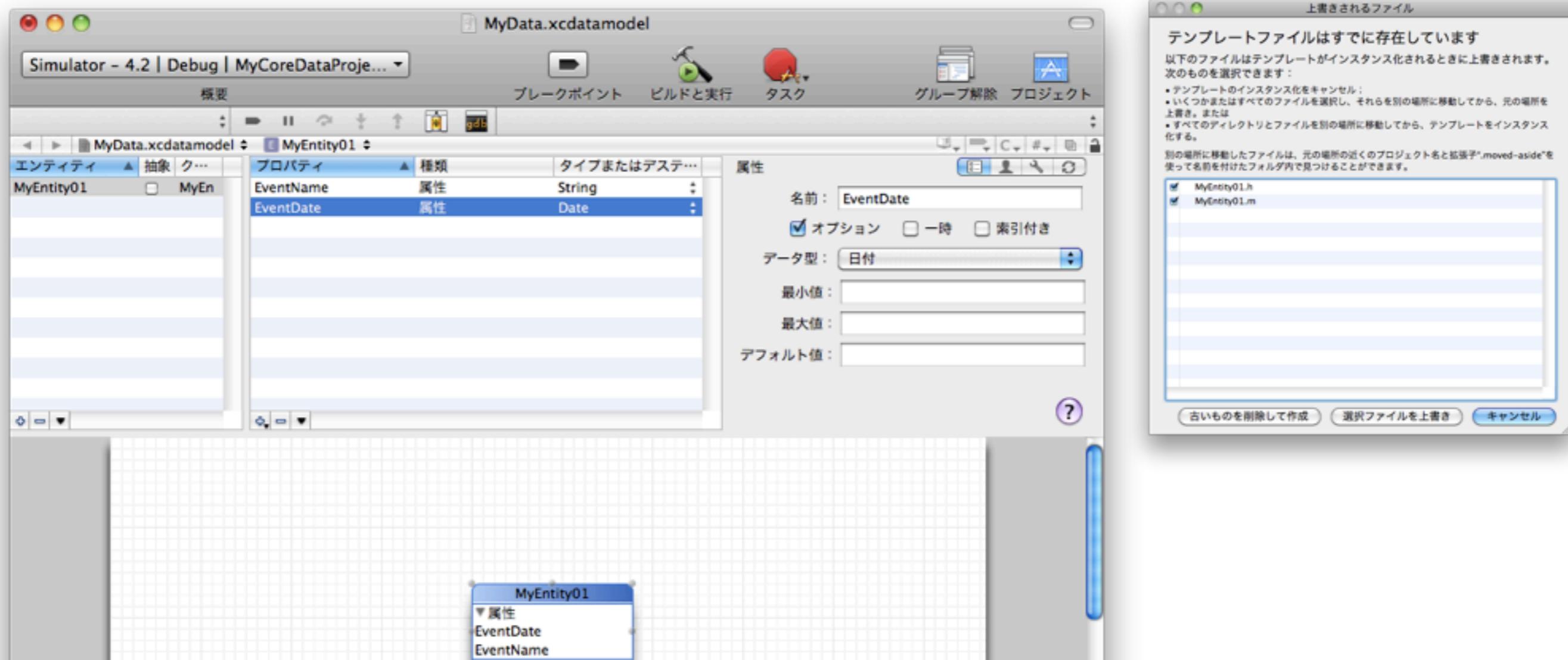
3-0-0.エンティティに情報を足す

イベント名だけじゃ足りね。日付入れようか。

resource > MyData.xcdatamodel開いて、
EventDateという名前のプロパティを追加。

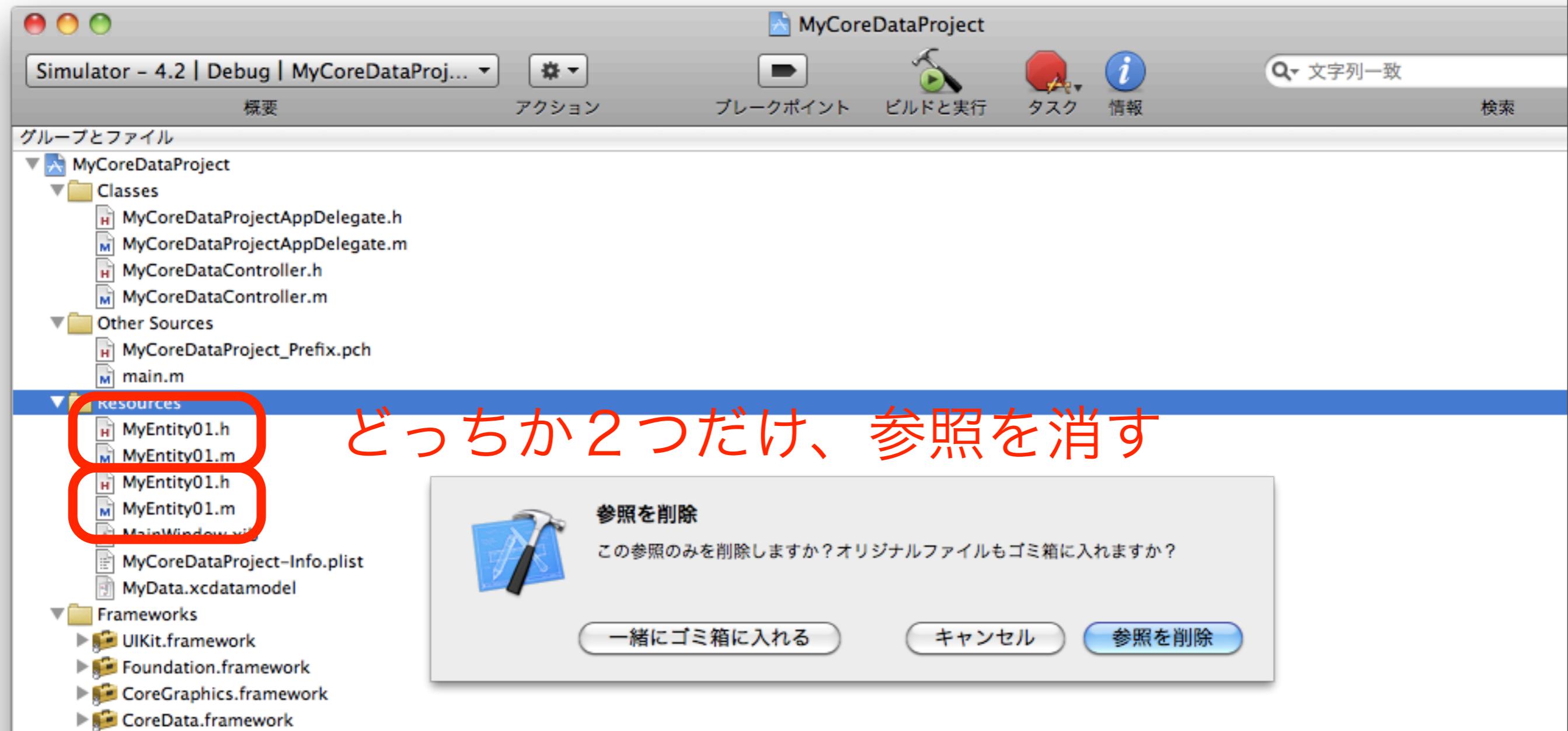
データ型は日付にしよう。

→再度、クラスを作成、、、ここで、問題が。



Drive50/64

3-0-1. 削除して作成or既にあるクラスファイルを上書き
どっちを選んでも、参照がこんなかんじになる。
どっちも同じファイルをみてるので、選択して、「参照を削除」。



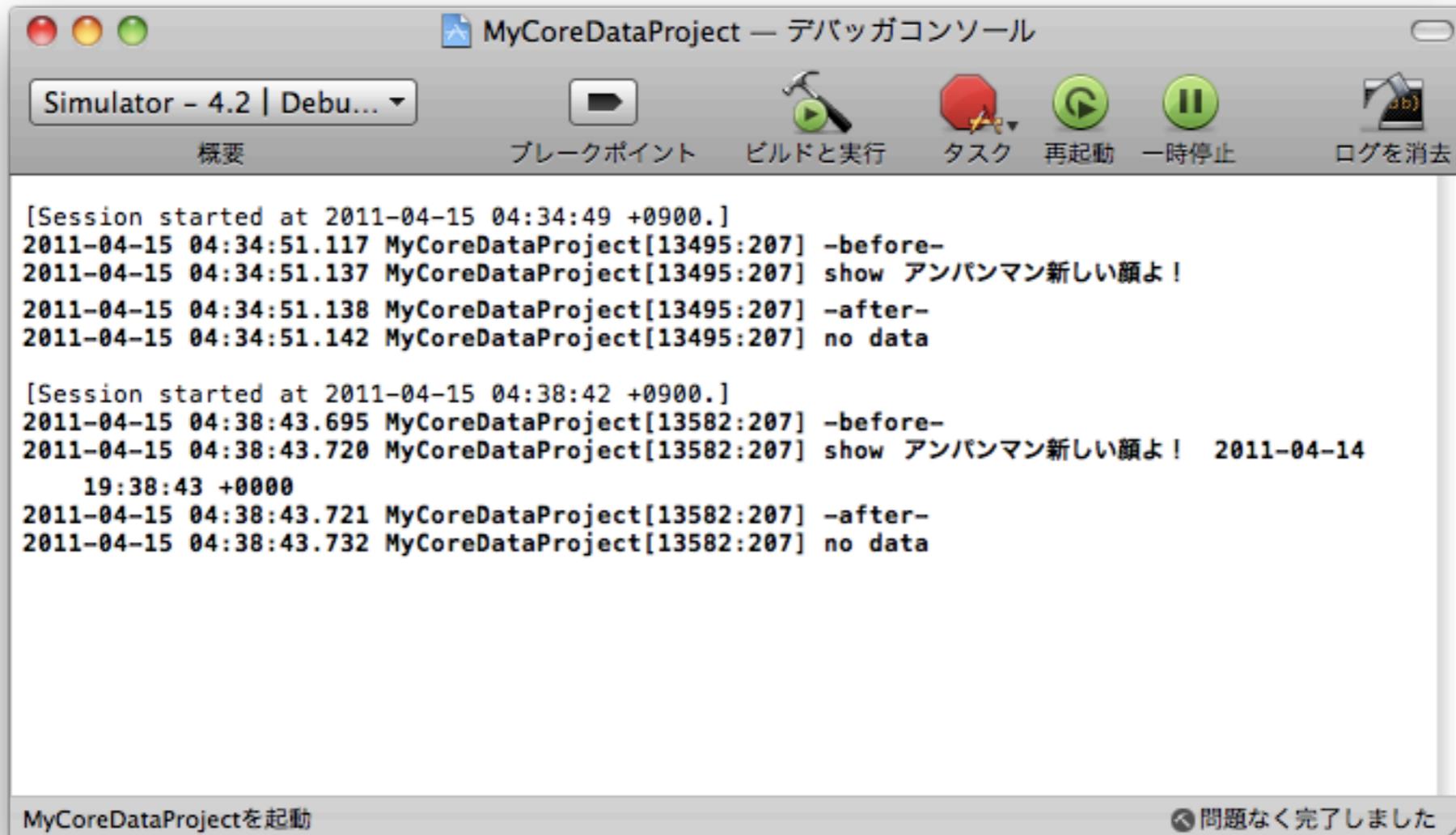
Drive5 | /64

3-0-2.エンティティを変更したときの処理

コレで実行、、！　すると、落ちる。

原因是、古い情報で作られた.xcdatamodelの情報が、.sqliteファイル内にあるから。
データ自体は一件も無いのにね。

→物理的にデータを消してリトライ、loadメソッドにて、
EventDateもログに表示するようにしてみよう。



The screenshot shows the Xcode Debug Navigator window titled "MyCoreDataProject – デバッガコンソール". It displays two log sessions:

```
[Session started at 2011-04-15 04:34:49 +0900.]  
2011-04-15 04:34:51.117 MyCoreDataProject[13495:207] -before-  
2011-04-15 04:34:51.137 MyCoreDataProject[13495:207] show アンパンマン新しい顔よ!  
2011-04-15 04:34:51.138 MyCoreDataProject[13495:207] -after-  
2011-04-15 04:34:51.142 MyCoreDataProject[13495:207] no data  
  
[Session started at 2011-04-15 04:38:42 +0900.]  
2011-04-15 04:38:43.695 MyCoreDataProject[13582:207] -before-  
2011-04-15 04:38:43.720 MyCoreDataProject[13582:207] show アンパンマン新しい顔よ! 2011-04-14  
19:38:43 +0000  
2011-04-15 04:38:43.721 MyCoreDataProject[13582:207] -after-  
2011-04-15 04:38:43.732 MyCoreDataProject[13582:207] no data
```

The bottom status bar indicates "MyCoreDataProjectを起動" and "問題なく完了しました" (Completed without errors).

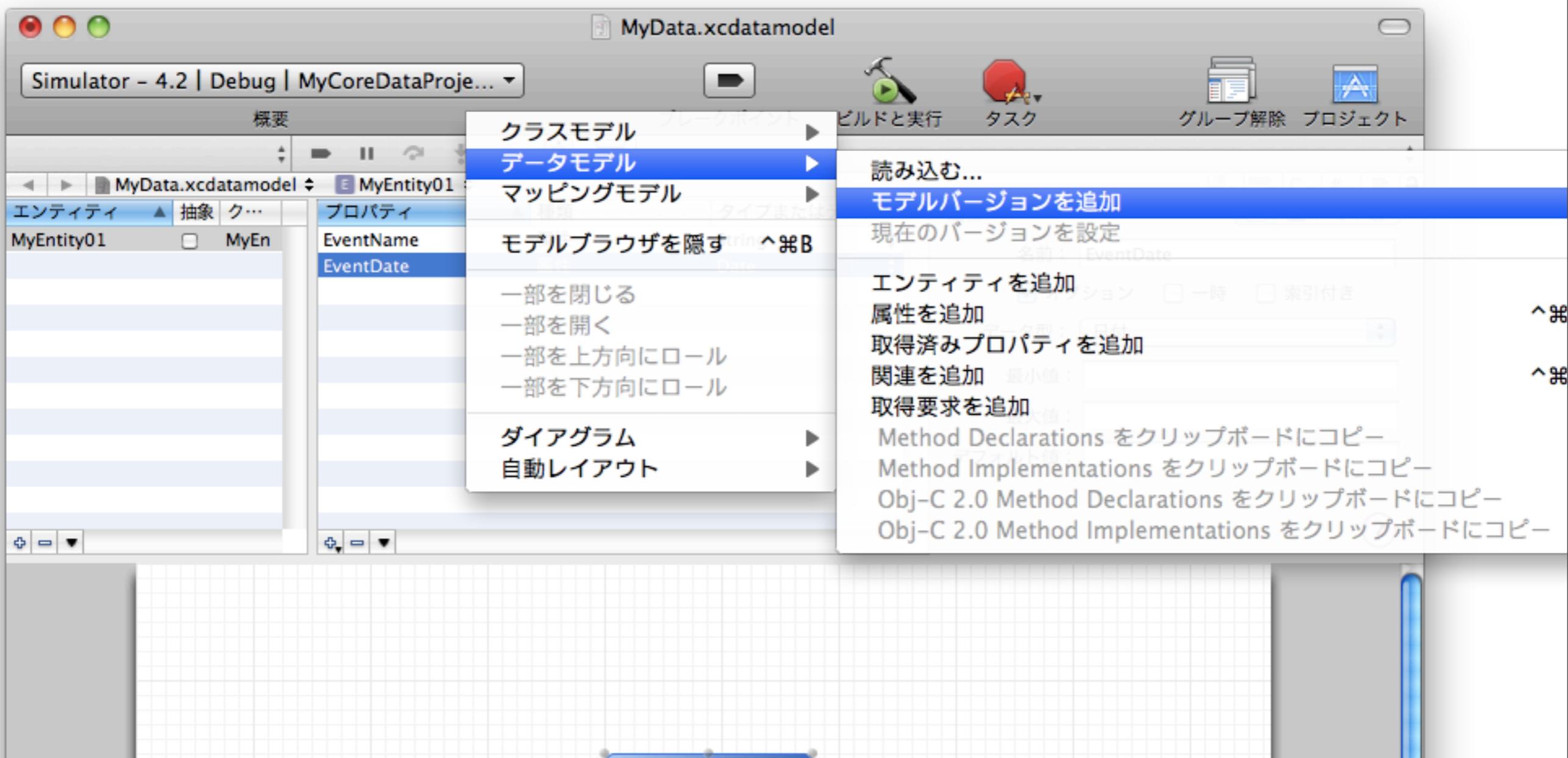
Drive52/64

3-1-0.バージョンアップ

この操作、実際にアプリでどんなときに起こるかというと、
後から保存したい値が増えたりとか減ったりとか、形が変わったりとか、
そういうときに発生する。
で、正直必ずと言っていいほど有る。

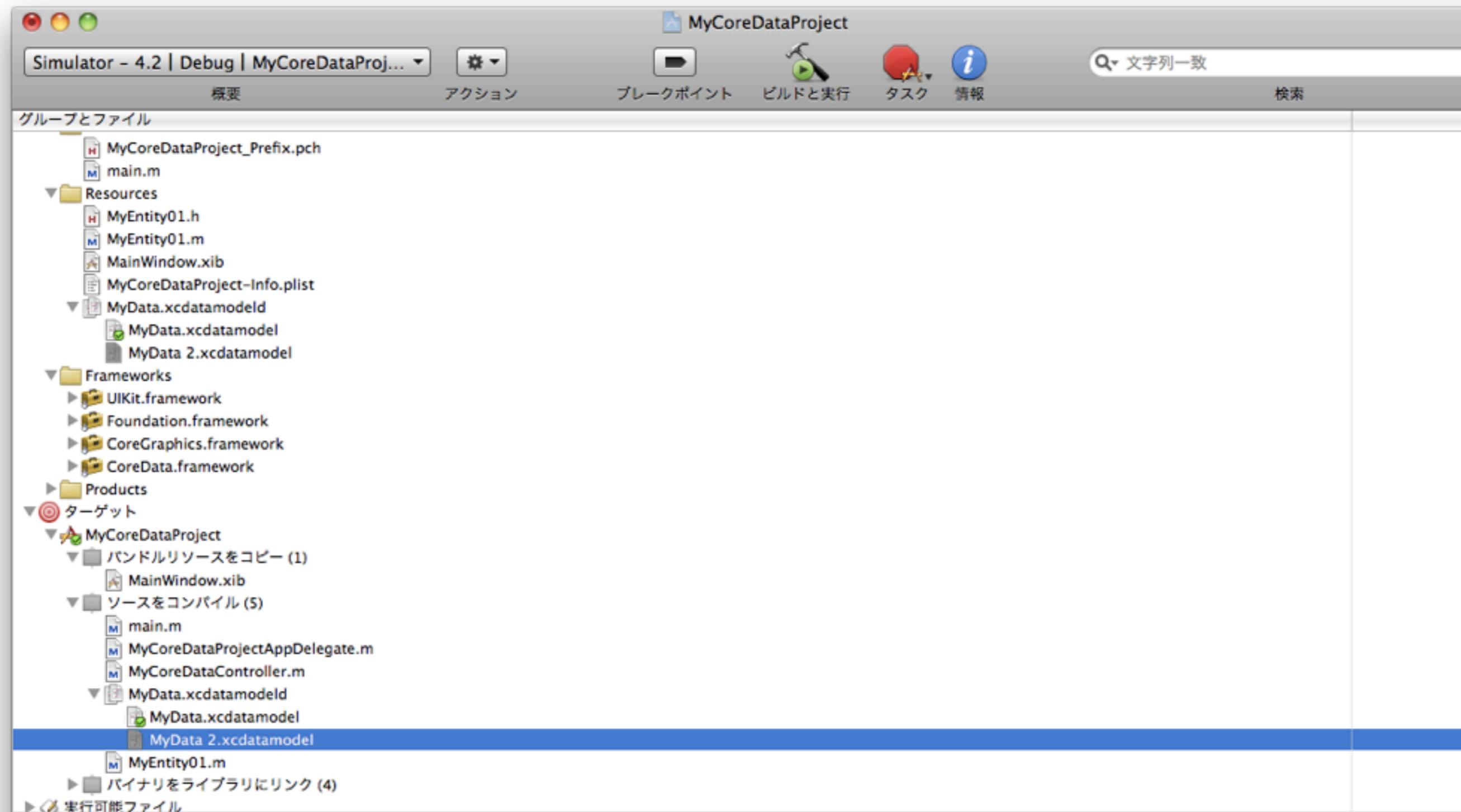
Drive53/64

3-1-1. .xcdatamodelを選択した状態で、
設計 > データモデル > モデルバージョンを追加 とすると、
.xcdatamodelへとモデルバージョンが追加出来る。



Drive54/64

3-1-2. →なんか、みどりのマークが着いてる”前から有る.xcdatamodel”と、”～2.xcdatamodel”が出来たと思う。

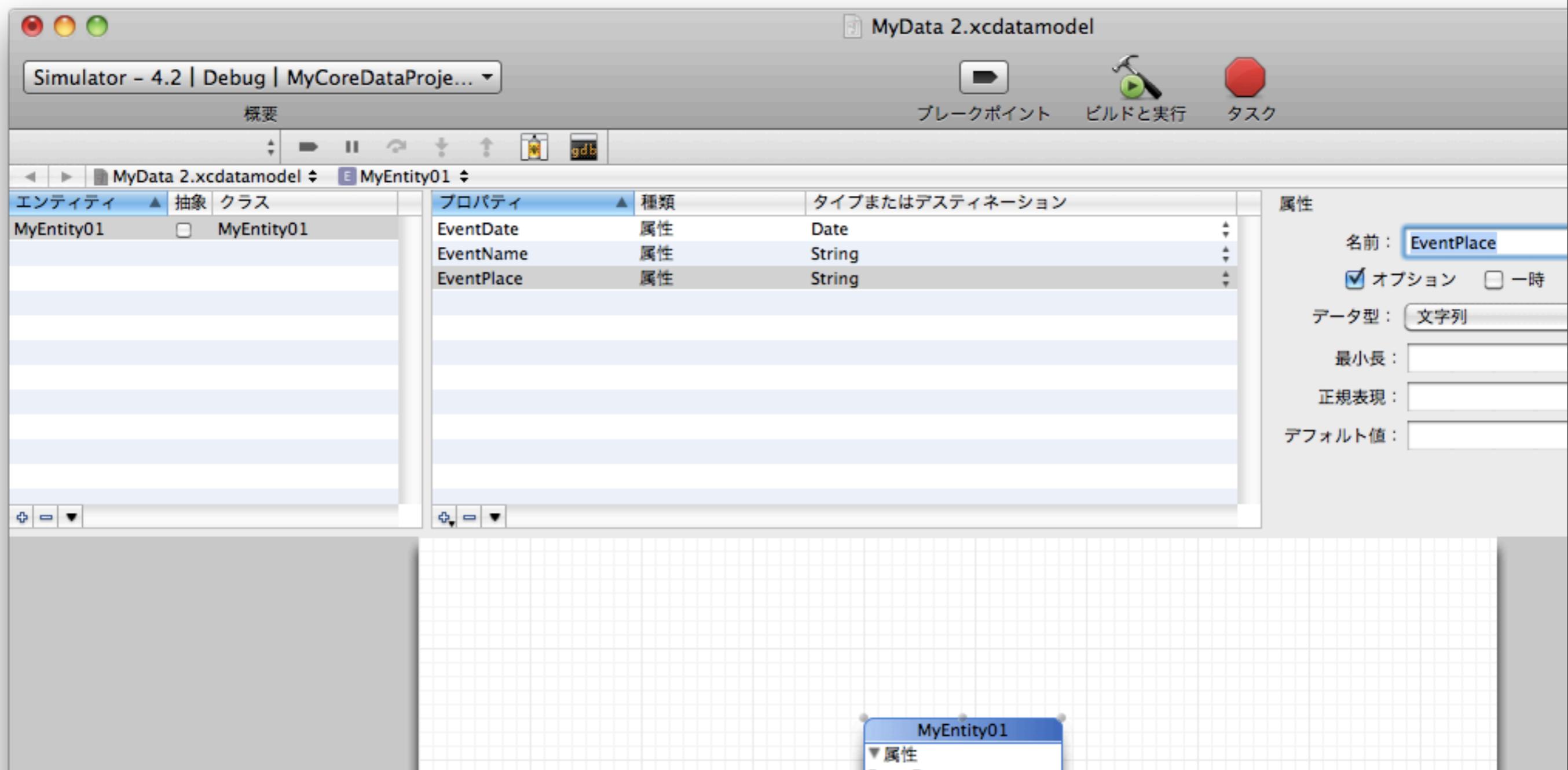


Drive55/64

3-1-3.新エンティティにプロパティを追加

～2.xcdatamodelを開いて、

MyEntity01にプロパティとして”EventPlace”を追加する。

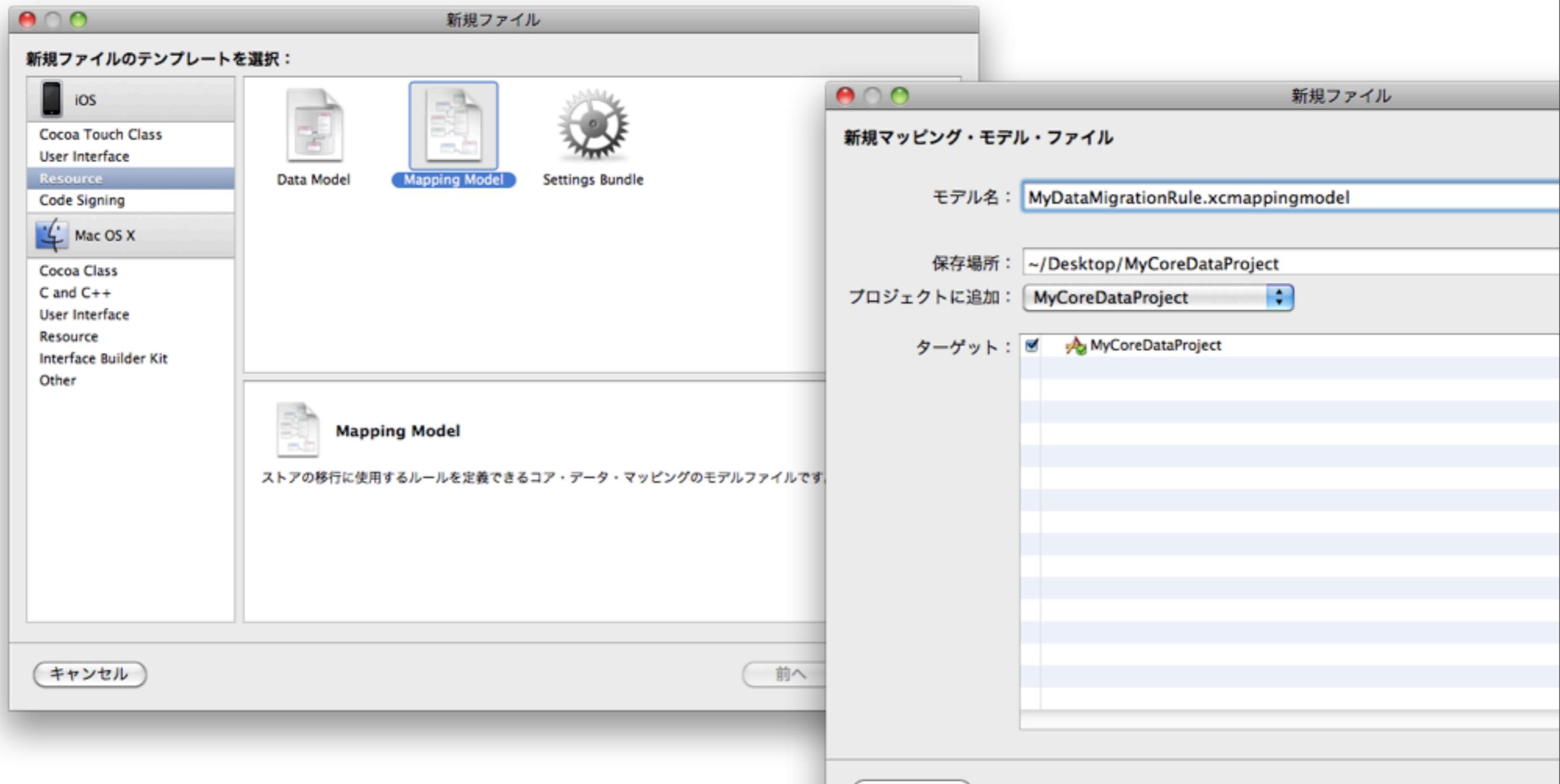


Drive56/64

3-1-4. 移行ファイルを作る

ファイル > 新規 > Resource > Mapping Model を選ぶ。

名前は、MyDataMigrationRule.xcmappingmodel としよう。



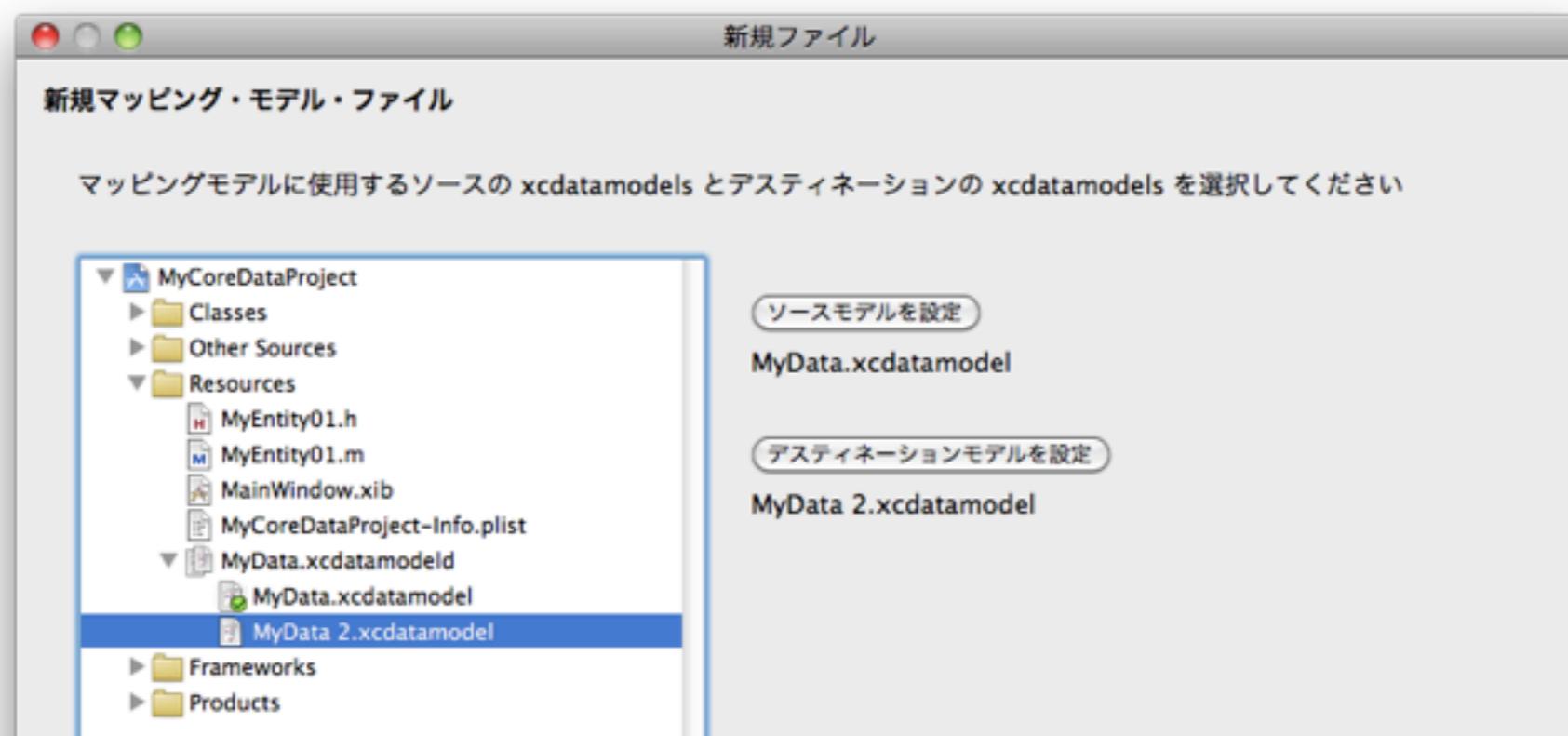
Drive57/64

3-1-5.Migration

バージョンOld→Newへのデータの移行 を、 Migration(移行)という
データ形状のバージョンが変化した時、この Mapping Model ファイルの記述を元に、
そのときアプリに入ってるデータを、自動的に最新版に適応させる事が出来る。

MyDataMigrationRule.xcmappingmodel

ソースに、みどりのマークが着いてる”前から有る.xcdatamodel”、
デスティネーションに、”～2.xcdatamodel”を指定する。
ま、ビフォアアフターの関係性を考えるだけ。

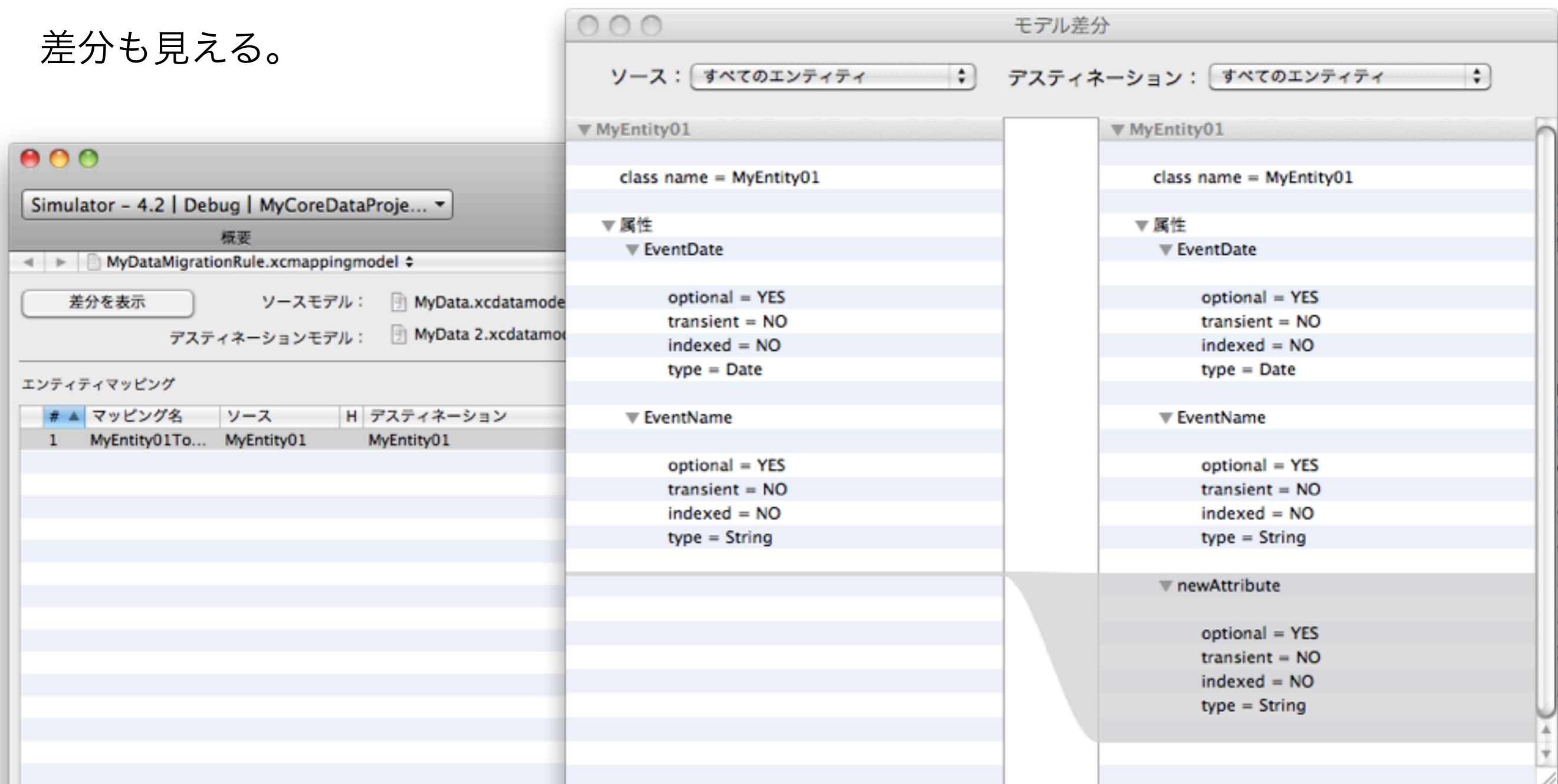


Drive58/64

3-1-6.判り易く

MyDataMigrationRule.xcmappingmodelを開くと、
エンティティに項目[EventPlace]が増えてて、

差分も見える。

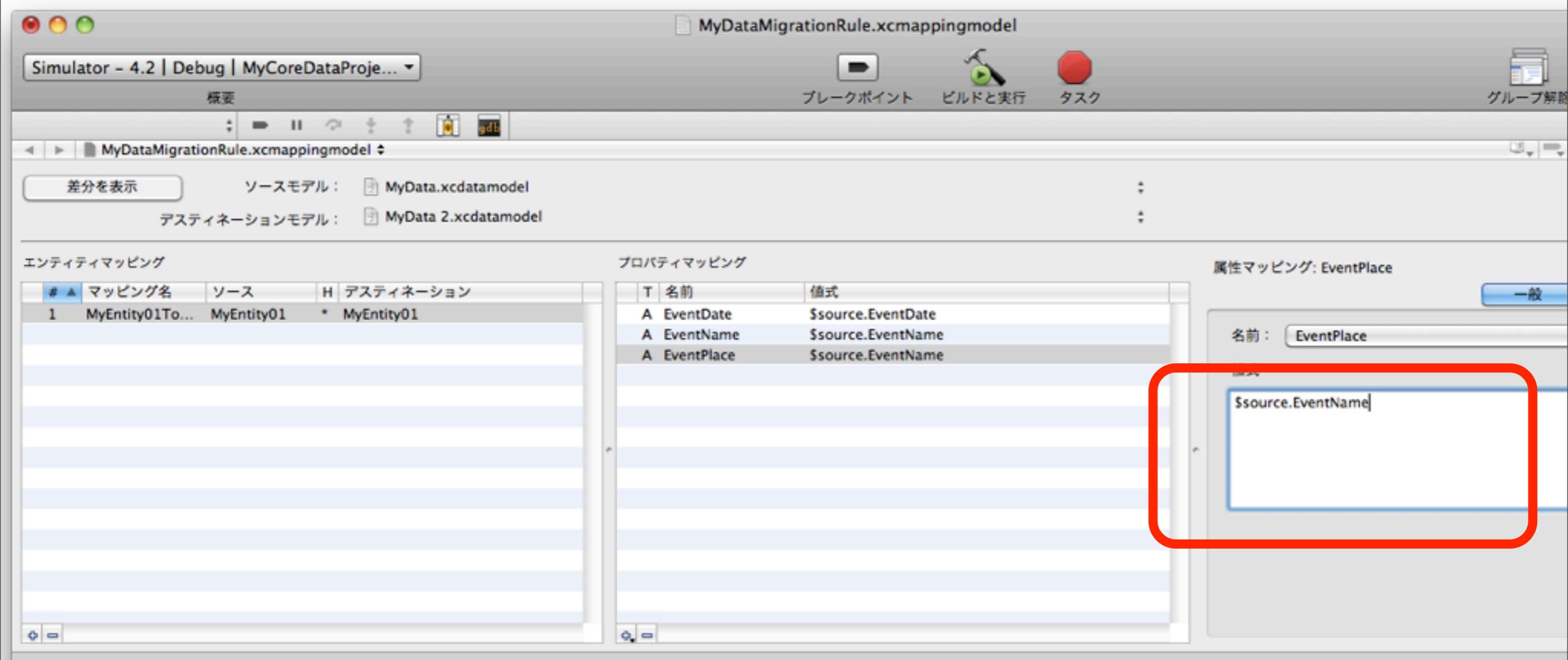


Drive59/64

3-1-7.差分をどう処理するか

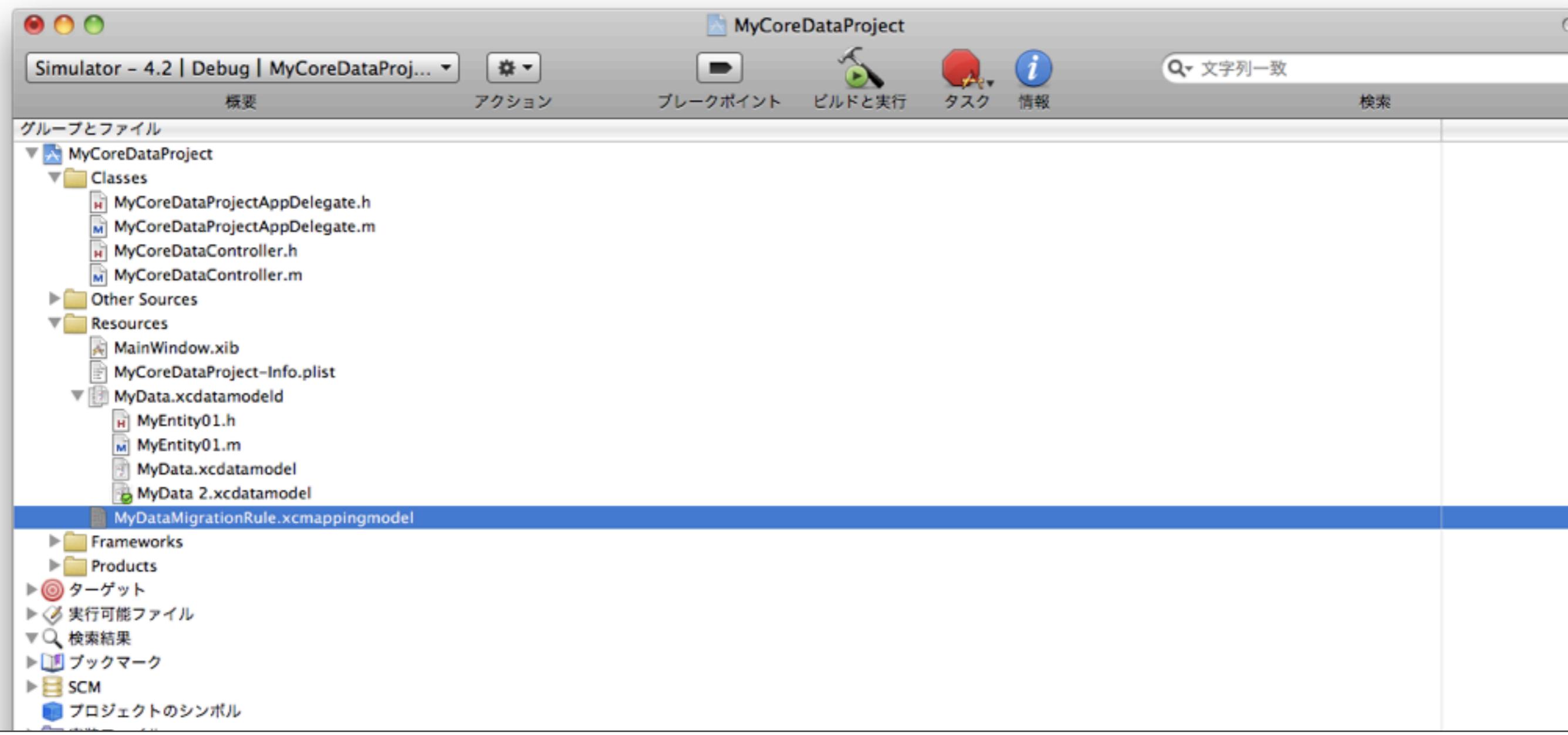
ここで、バージョン間の移動時にどんな値が入って欲しいか、記述出来る。

EventNameと一緒にしてみた。



Drive60/64

3-1-8.データベースの現在のバージョンを、～2.xcdatamodelにする
～2.xcdatamodelを選択した状態で、
設計 > データモデル > 現在のバージョンを設定、とする。
緑のマークがついた筈。



Drive6 | /64

3-1-9. 移行用のオプションコードを書き足す

```
MyCoreDataController.m
Simulator - 4.2 | Debug | MyCoreDataProj...
概要 ブレークポイント ビルドと実行 タスク グループ解除
MyCoreDataController.m:155 -myNSPersistentStoreCoordinator
- (NSPersistentStoreCoordinator * ) myNSPersistentStoreCoordinator {
    if (myNSPersistentStoreCoordinator != nil) {
        return myNSPersistentStoreCoordinator;
    }

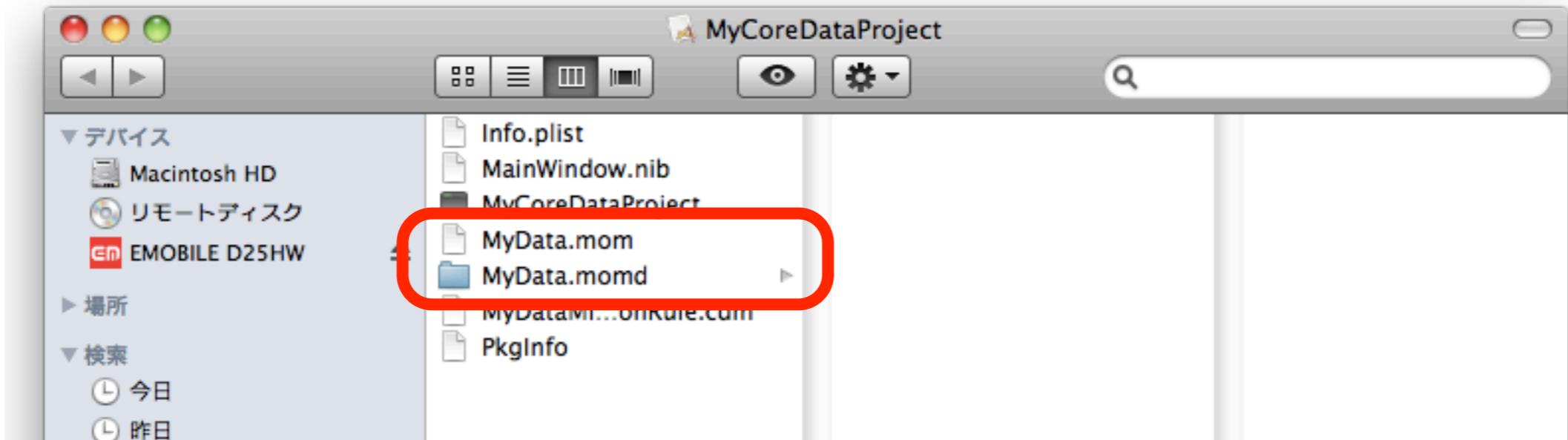
    NSURL * storeUrl = [NSURL fileURLWithPath:
        [
            [self applicationDocumentsDirectory] stringByAppendingPathComponent:
                [NSString stringWithFormat:@"MySaveData.sqlite"]
        ]
    ];

    NSError * storeSettingError = nil;
    myNSPersistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:[self myNSManagedObjectModel]];
    NSDictionary * options = [NSDictionary dictionaryWithObjectsAndKeys:
        [NSNumber numberWithBool:YES], NSMigratePersistentStoresAutomaticallyOption,
        [NSNumber numberWithBool:YES], NSInferMappingModelAutomaticallyOption,
        nil];
    if (![myNSPersistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType
        configuration:nil
        URL:storeUrl
        options:options
        error:&storeSettingError]) {
        if (storeSettingError) {
            NSLog(@"storeSettingError %@", storeSettingError);
        }
        NSAssert(FALSE, @"wrong pass");
    }
    return myNSPersistentStoreCoordinator;
}
```

Drive62/64

3-1-10.起動すると、落ちる。

MyCoreDataProject.appのパッケージの中身を見てみると、



～2.xcdatamodelを作成した時、.xcdatamodelのファイル位置が、
変化してたと思う。

あの影響で、.monファイルの下に、.mondフォルダができる。
→中には、.monファイルがある。

バージョンを追加した事で、
.mondフォルダが作られた。そのお陰で、フォルダ構造が狂ったんだわな。

Drive63/64

3-1-11. この1回だけ、削除

.monファイルをこのフォルダ外に取り出して起動すると、動く。

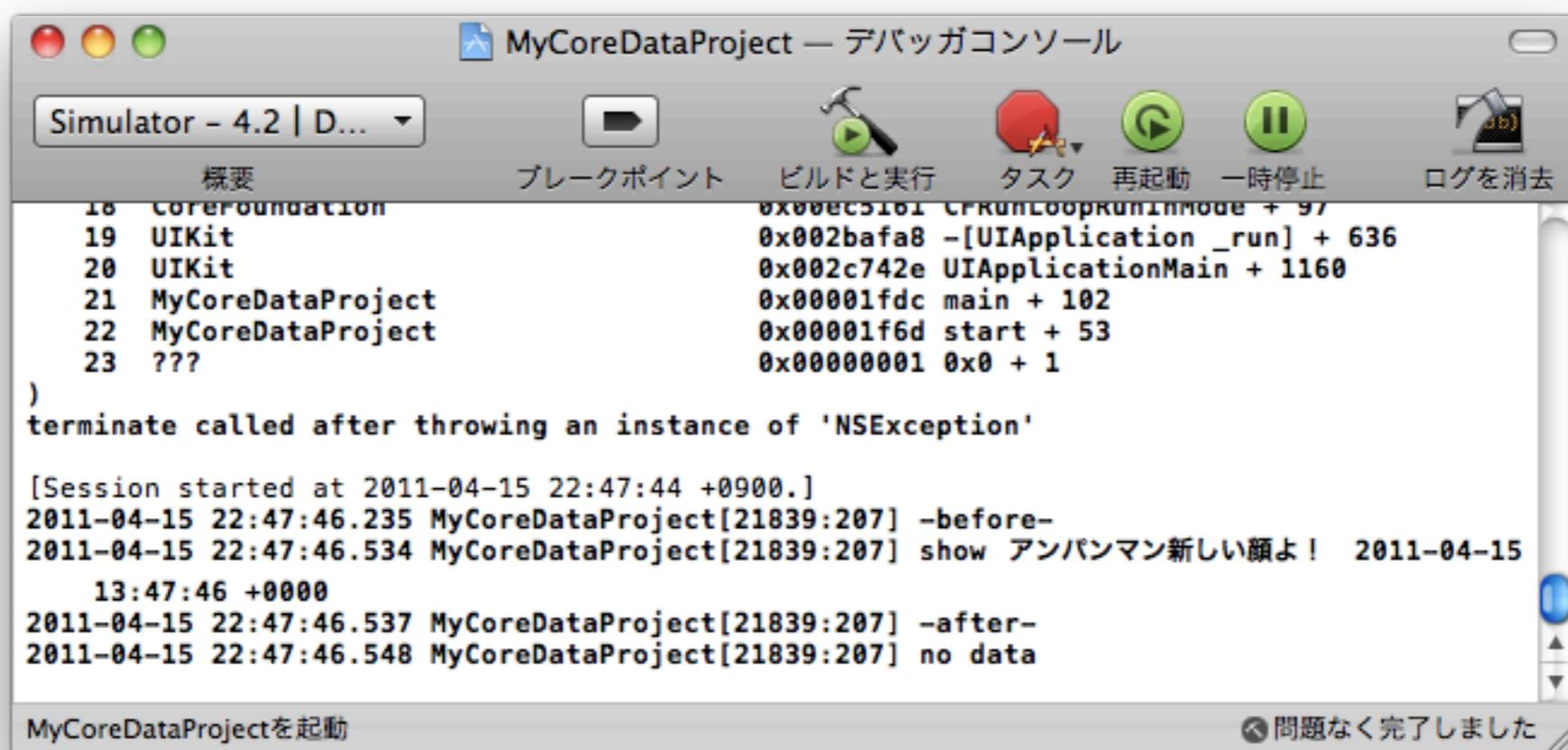
フォルダの構成が変わるのは、

Before:バージョンを一つも付加していなかった

→

After:バージョンが付加されるようになった

この時の、一回だけなので、なんていうか始めからバージョン振っておこう。



The screenshot shows the Xcode Debug Console window titled "MyCoreDataProject — デバッガコンソール". The console output is as follows:

```
Simulator - 4.2 | D...
MyCoreDataProject — デバッガコンソール

概要 ブレークポイント ビルドと実行 タスク 再起動 一時停止 ログを消去

18 CoreFoundation
19 UIKit
20 UIKit
21 MyCoreDataProject
22 MyCoreDataProject
23 ???
)
terminate called after throwing an instance of 'NSException'

[Session started at 2011-04-15 22:47:44 +0900.]
2011-04-15 22:47:46.235 MyCoreDataProject[21839:207] -before-
2011-04-15 22:47:46.534 MyCoreDataProject[21839:207] show アンパンマン新しい顔よ！ 2011-04-15
    13:47:46 +0000
2011-04-15 22:47:46.537 MyCoreDataProject[21839:207] -after-
2011-04-15 22:47:46.548 MyCoreDataProject[21839:207] no data

MyCoreDataProjectを起動 問題なく完了しました
```

Drive64/64

3-1-12.バージョンを自由に変える

別途バージョンを追加して、値を変えたりしてみよう。

- ・バージョンを追加する
- ・プロパティを足すとか変更を加える
- ・移行ルールである.xcmappingmodelを足す
- ・指定のバージョンを変える

と行うだけで、あとは、勝手にデータベースの移行が発生する。

(ちなみに、何時この移行が実際にどのような形で発生しているのか、
確かめたければドキュメントを見よう！Notification系があるぞ。)

ココまでで、終わり！

LookBack

- 1.“保存”についての基礎知識
- 2.そこでCoreData
- 3.バージョンを超えていけ

プログラムの質問、提出はこっちな！！
自分の名前と宛名忘れんなよ！！



toru.inoue@kissaki.tv

Continue?

次回、

第3回

「サウンド鳴らす、
メッセージングについて」