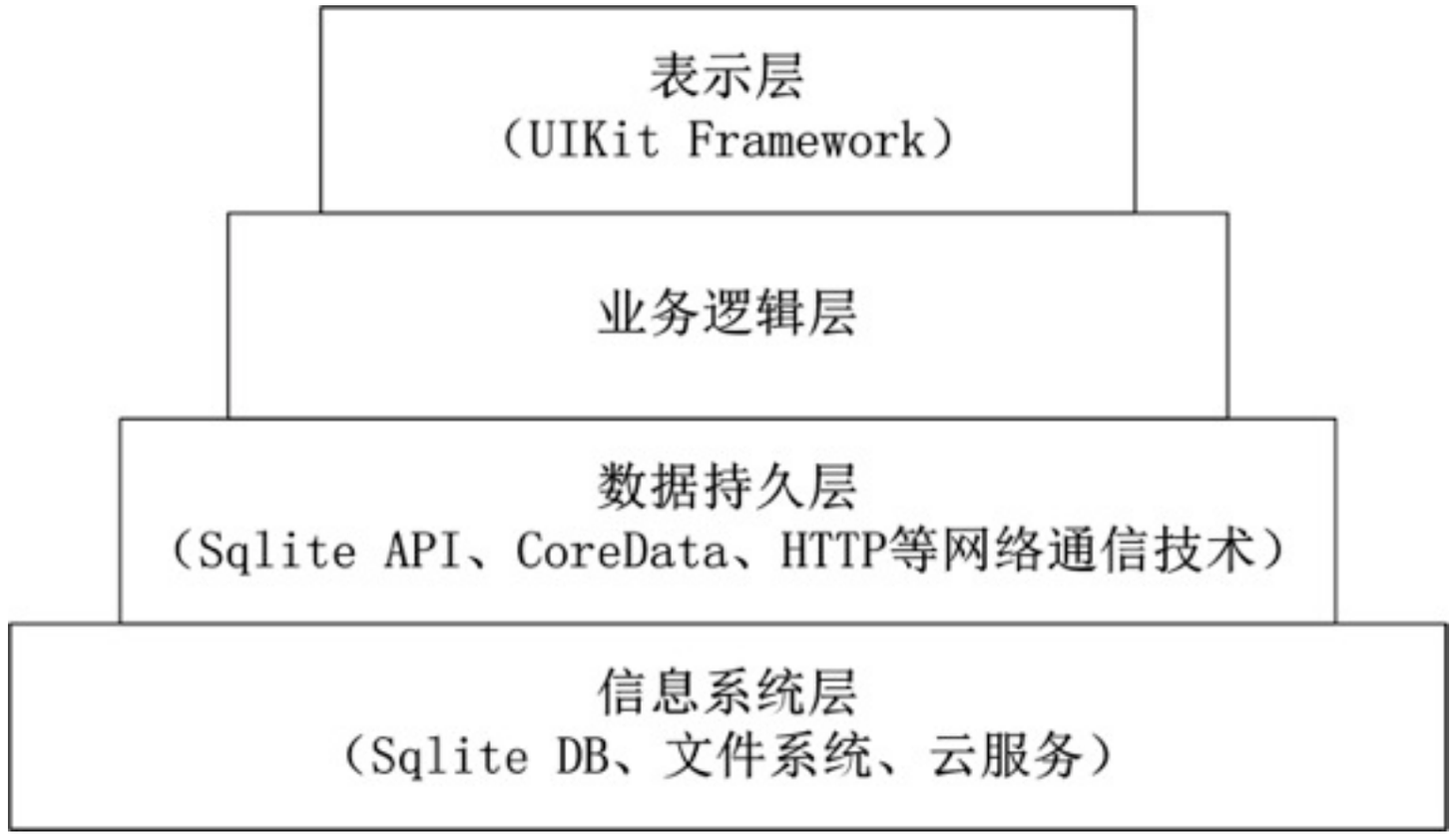
智捷iOS课堂

移动平台的分层架构设计

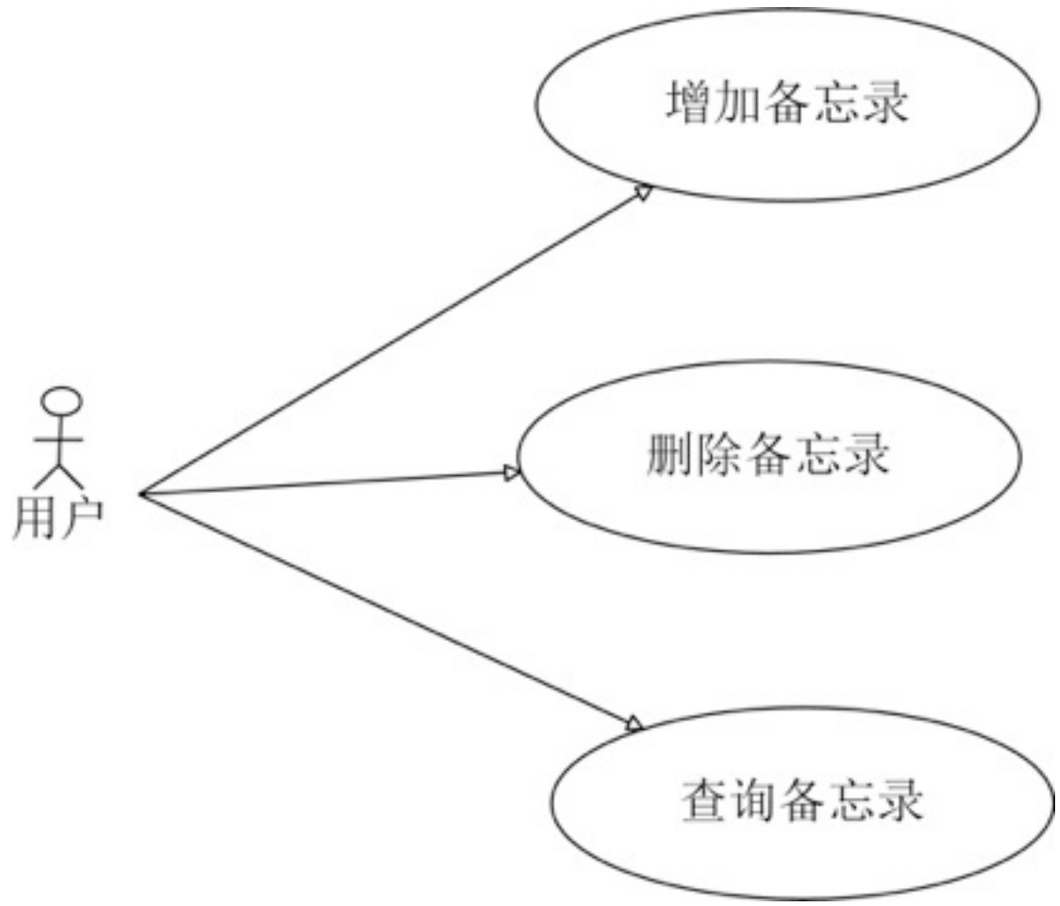
低耦合企业级系统架构设计

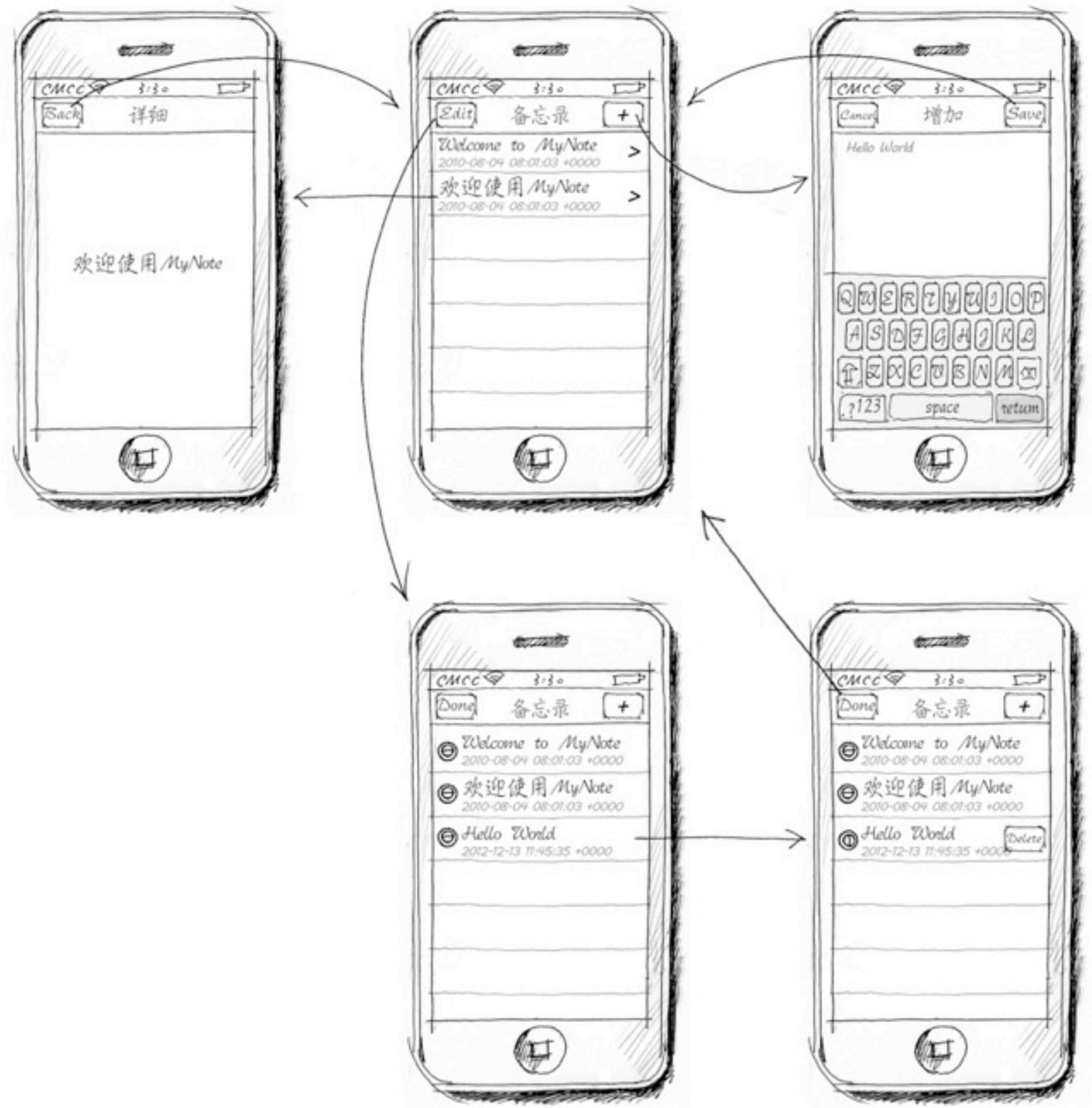

表示层
业务逻辑层
数据持久层
信息系统层

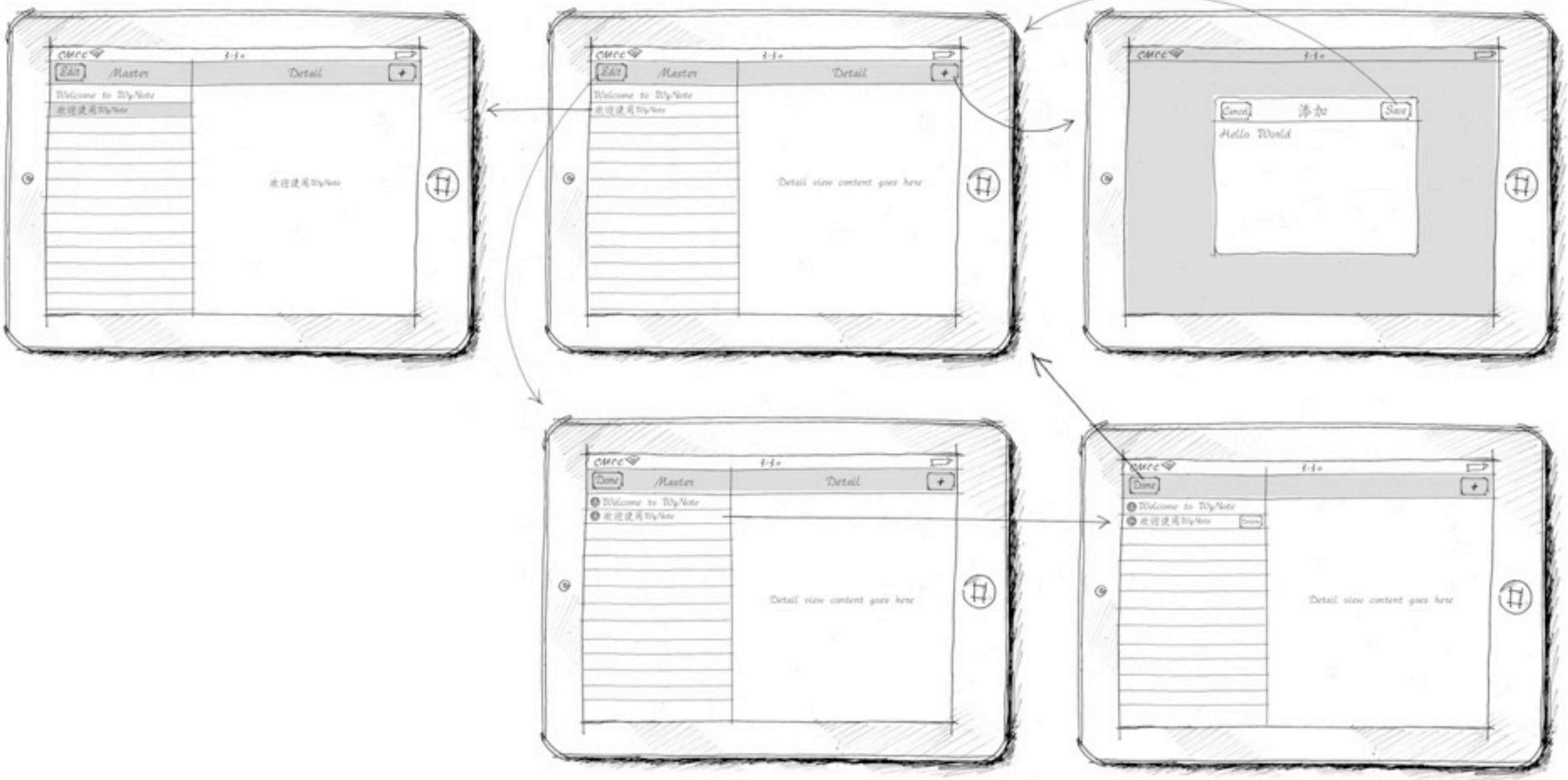
移动平台的分层架构设计

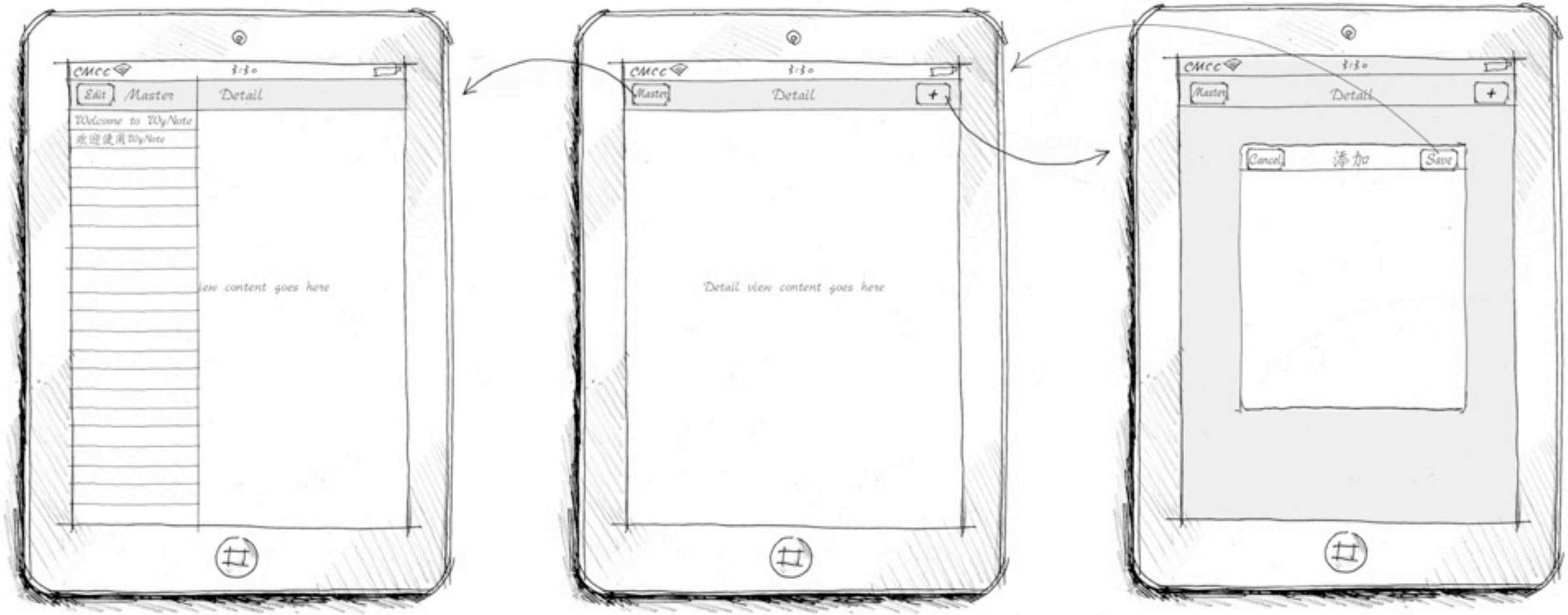

表示层
（UIKit Framework）
业务逻辑层
数据持久层
（Sqlite API、CoreData、HTTP等网络通信技术）
信息系统层
（Sqlite DB、文件系统、云服务）

“备忘录”应用

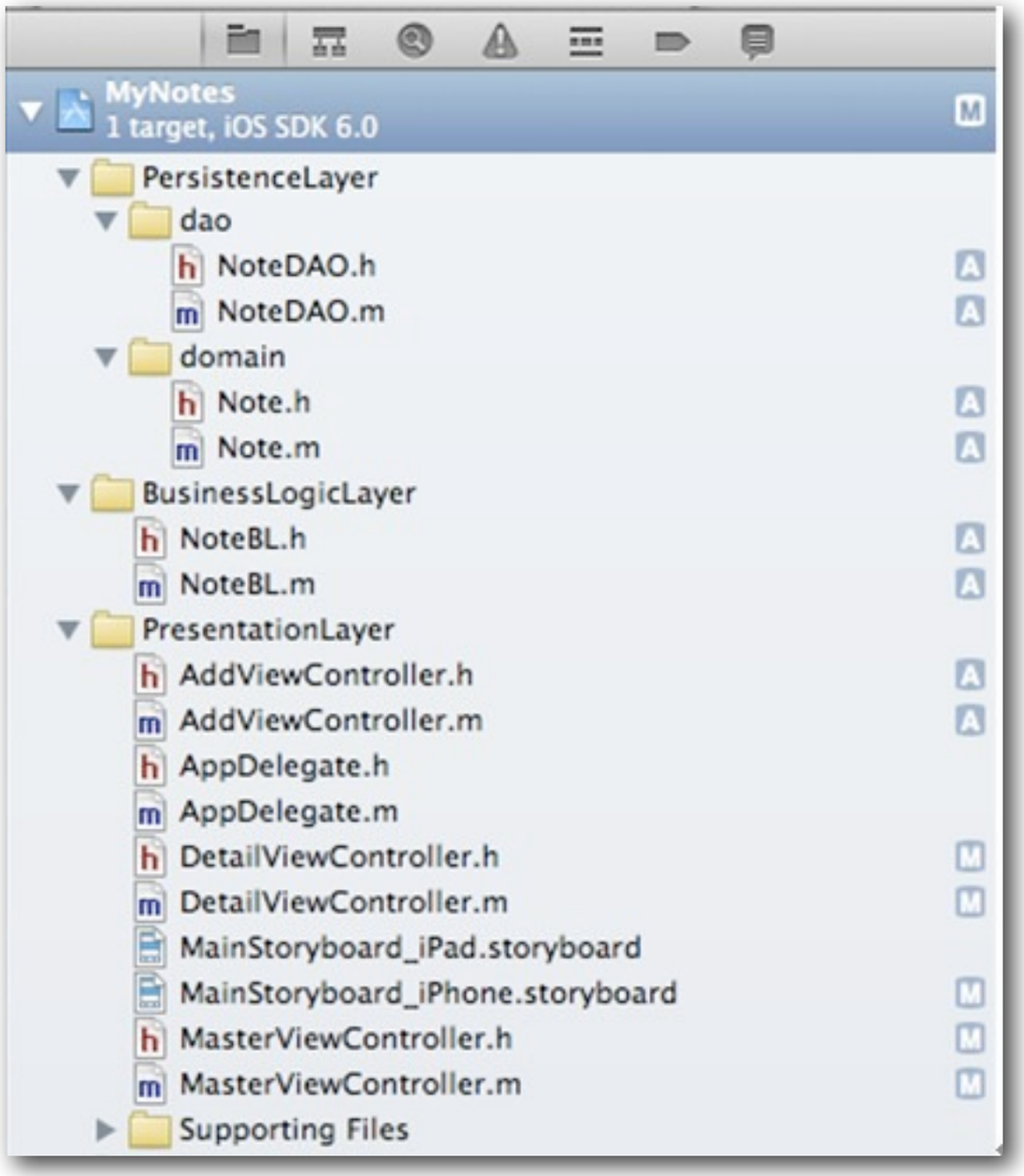

增加备忘录
删除备忘录
查询备忘录
用户


CMCC 3:30
Back
详细
欢迎使用MyNote
CMCC 3:30
Edit
备忘录
+
Welcome to MyNote
2010-08-04 08:01:03 +0000
欢迎使用MyNote
2010-08-04 08:01:03 +0000
CMCC 3:30
Cancel
增加
Save
Hello World
.?123
space
return
CMCC 3:30
Done
备忘录
+
Welcome to MyNote
2010-08-04 08:01:03 +0000
欢迎使用MyNote
2010-08-04 08:01:03 +0000
Hello World
2012-12-13 11:45:35 +0000
CMCC 3:30
Done
备忘录
+
Welcome to MyNote
2010-08-04 08:01:03 +0000
欢迎使用MyNote
2010-08-04 08:01:03 +0000
Hello World
2012-12-13 11:45:35 +000
Delete


Edit
Master
Detail
Welcome to WyNote
欢迎使用WyNote
Detail view content goes here
Cancel
添加
Save
Hello World
Done


CMCC
Edit
Master
Detail
Welcome to WyNote
欢迎使用WyNote
Detail view content goes here
Master
Detail
Detail view content goes here
Master
Detail
Cancel
添加
Save

基于同一工程的分层实现


MyNotes
1 target, iOS SDK 6.0
PersistenceLayer
dao
NoteDAO.h
NoteDAO.m
domain
Note.h
Note.m
BusinessLogicLayer
NoteBL.h
NoteBL.m
PresentationLayer
AddViewController.h
AddViewController.m
AppDelegate.h
AppDelegate.m
DetailViewController.h
DetailViewController.m
MainStoryboard_iPad.storyboard
MainStoryboard_iPhone.storyboard
MasterViewController.h
MasterViewController.m
Supporting Files

Persistence Layer（数据制持久层）

DAO和Domain

- dao是放置数据访问对象的，该对象中有对数据访问的CRUD四类方法，为了降低耦合度dao一般要设计成为协议（或Java接口），然后根据不同的数据来源采用不同的实现方式。
- domain是实体类，实体是应用中的“人”、“事”、“物”等，也叫“业务领域对象”。

NoteDAO.h代码

```
@interface NoteDAO : NSObject
//保存数据列表
@property (nonatomic,strong) NSMutableArray* listData;
+ (NoteDAO*)sharedManager;
//插入Note方法
-(int) create:(Note*)model;
//删除Note方法
-(int) remove:(Note*)model;
//修改Note方法
-(int) modify:(Note*)model;
//查询所有数据方法
-(NSMutableArray*) findAll;
//按照主键查询数据方法
-(Note*) findById:(Note*)model;
@end
```

单例实现DAO

```
static NoteDAO *sharedManager = nil;
+ (NoteDAO*)sharedManager {
    static dispatch_once_t once;
    dispatch_once(&once, ^{
        sharedManager = [[self alloc] init];

        NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
        [dateFormatter setDateFormat:@"yyyy-MM-dd HH:mm:ss"];
        NSDate *date1 = [dateFormatter dateFromString:@"2010-08-04 16:01:03"];
        Note* note1 = [[Note alloc] init];
        note1.date = date1;
        note1.content = @"Welcome to MyNote.";
        NSDate *date2 = [dateFormatter dateFromString:@"2011-12-04 16:01:03"];
        Note* note2 = [[Note alloc] init];
        note2.date = date2;
        note2.content = @"欢迎使用MyNote。";
        sharedManager.listData = [[NSMutableArray alloc] init];
        [sharedManager.listData addObject:note1];
        [sharedManager.listData addObject:note2];
    });
    return sharedManager;
}
```

插入和删除方法

```
//插入Note方法
-(int) create:(Note*)model
{
    [self.listData addObject:model];
    return 0;
}

//删除Note方法
-(int) remove:(Note*)model
{
    for (Note* note in self.listData) {
        //比较日期主键是否相等
        if ([note.date isEqualToDate:model.date]){
            [self.listData removeObject: note];
            break;
        }
    }
    return 0;
}
```

修改方法

```
//修改Note方法
-(int) modify:(Note*)model
{
    for (Note* note in self.listData) {
        //比较日期主键是否相等
        if ([note.date isEqualToDate:model.date]){
            note.content = model.content;
            break;
        }
    }
    return 0;
}
```


查询方法

```
//查询所有数据方法
-(NSMutableArray*) findAll
{
    return self.listData;
}

//按照主键查询数据方法
-(Note*) findById:(Note*)model
{
    for (Note* note in self.listData) {
        //比较日期主键是否相等
        if ([note.date isEqualToDate:model.date]){
            return note;
        }
    }
    return nil;
}
```

Domain中Note

```
//
//  Note.h

#import <Foundation/Foundation.h>

@interface Note : NSObject

@property(nonatomic, strong) NSDate* date;
@property(nonatomic, strong) NSString* content;

@end

//
//  Note.m
#import "Note.h"

@implementation Note

@end
```

BusinessLogic Layer（业务逻辑层）

NoteBL.h

```
@interface NoteBL : NSObject
//插入Note方法
-(NSMutableArray*) createNote:(Note*)model;

//删除Note方法
-(NSMutableArray*) remove:(Note*)model;

//查询所有数据方法
-(NSMutableArray*) findAll;

@end
```

业务逻辑层中的类的设计一般是按照业务模块设计的，它的方法是业务处理方法。之所以定义三个方法是根据我的业务需求决定的。

NoteBL.m

```
//插入Note方法
-(NSMutableArray*) createNote:(Note*)model
{
    NoteDAO *dao = [NoteDAO sharedManager];
    [dao create:model];
    return [dao findAll];
}

//删除Note方法
-(NSMutableArray*) remove:(Note*)model
{
    NoteDAO *dao = [NoteDAO sharedManager];
    [dao remove:model];
    return [dao findAll];
}

//查询所有数据方法
-(NSMutableArray*) findAll
{
    NoteDAO *dao = [NoteDAO sharedManager];
    return [dao findAll];
}
```

Presentation Layer（表示层）

- AppDelegate.h和AppDelegate.m，应用程序委托对象；
- MasterViewController.h和MasterViewController.m，Master视图控制器；
- DetailViewController.h和DetailViewController.m，Detail视图控制器；
- AddViewController.h和AddViewController.m，Add视图控制器；
- MainStoryboard_iPhone.storyboard，iPhone版的故事板文件；
- MainStoryboard_iPad.storyboard，iPad版的故事板文件。

基于一个工作空间不同工程的分层

由于某些原因不想提供源代码，我们可以将业务逻辑层和数据持久层编写成为静态链接库（Static Library或Static Link Library）。

静态链接库

所谓“库”（Library）是一些没有main函数的程序代码的集合，除了静态链接库还有动态链接库。静动态链接的区别是：静态链接库可以编译到你的执行代码中，应用程序可以在没有静态链接库环境下运行；而动态链接库不能编译到你的执行代码中，应用程序必须在有链接库文件环境下运行。

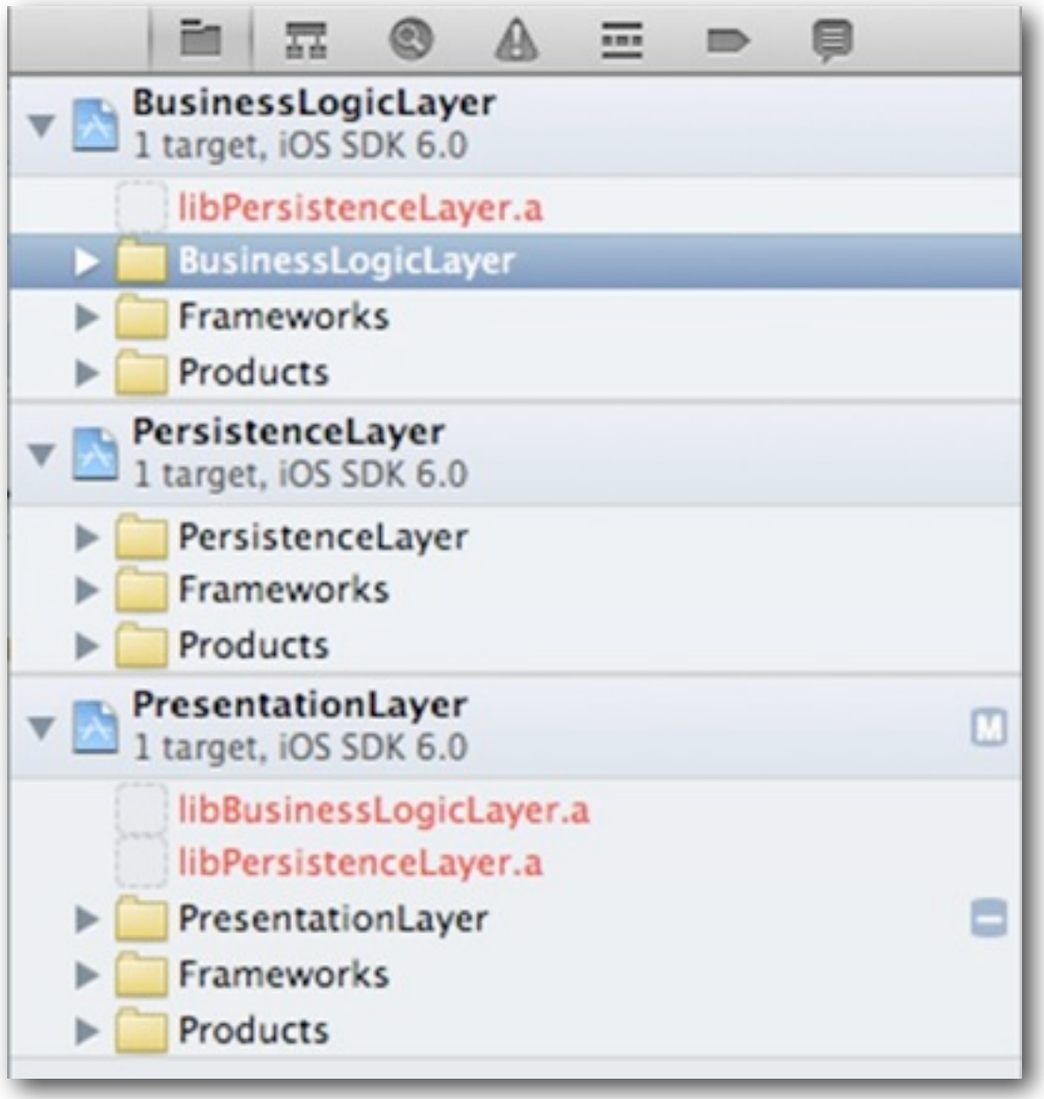
创建的过程

- 创建工作空间
- PersistenceLayer静态连接库工程
- BusinessLogicLayer静态连接库工程

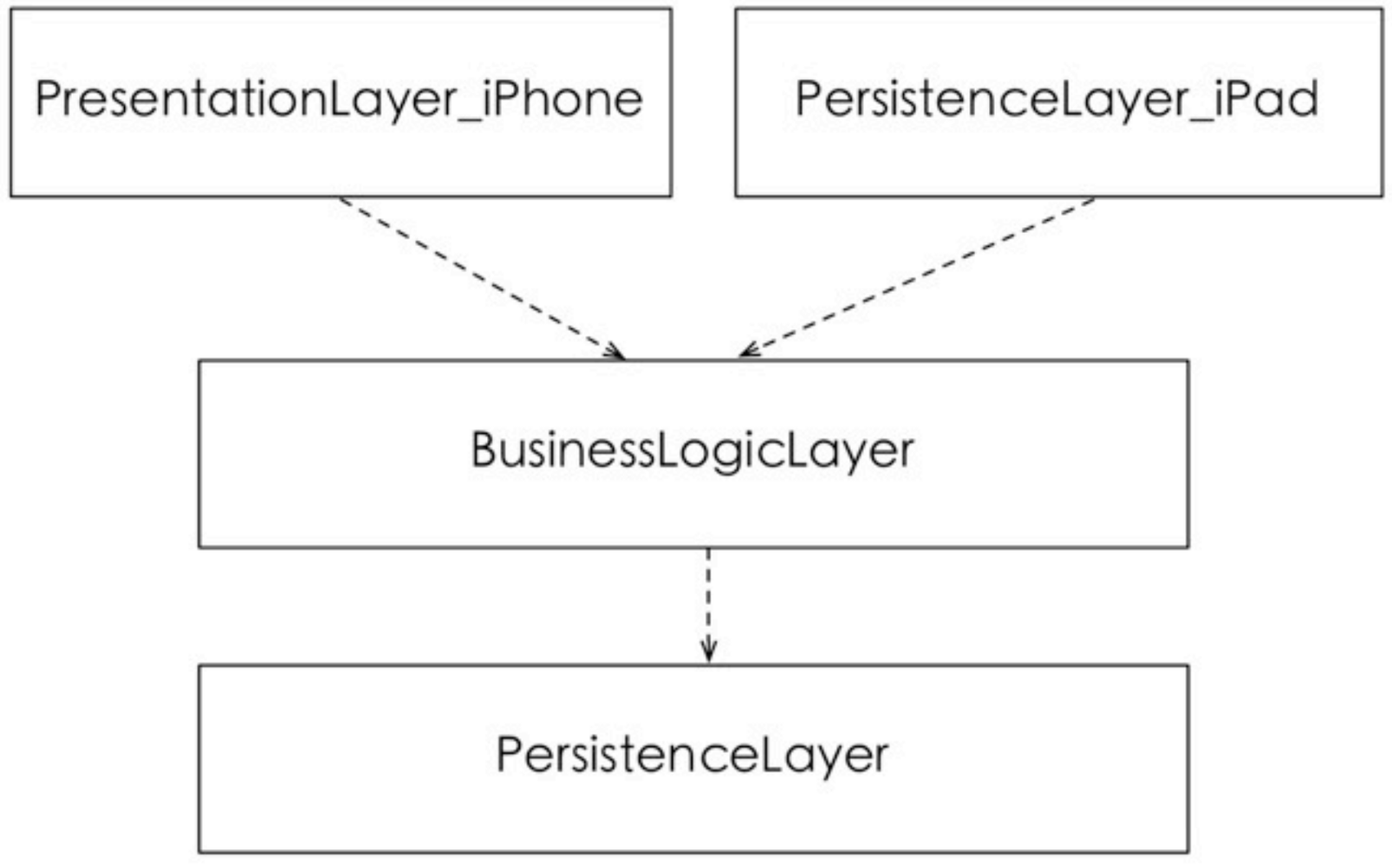
创建静态连接库工程

- 静态链接库工程
- 静态链接库工程拷贝头文件
- 添加依赖关系
- 添加头文件搜索设置

完成之后工作空间


BusinessLogicLayer
1 target, iOS SDK 6.0
libPersistenceLayer.a
BusinessLogicLayer
Frameworks
Products
PersistenceLayer
1 target, iOS SDK 6.0
PersistenceLayer
Frameworks
Products
PresentationLayer
1 target, iOS SDK 6.0
M
libBusinessLogicLayer.a
libPersistenceLayer.a
PresentationLayer
Frameworks
Products

依赖关系


PresentationLayer_iPhone
PersistenceLayer_iPad
BusinessLogicLayer
PersistenceLayer

谢谢