SYBDatabase对象数据库

概　述

SYBDatabase是基于FMDB的对象数据库。提供了OC对象的数据库操作接口。

使用说明

- **数据对象**

数据库的存储对象必须继承SYBManagedObject，每个存储对象都有一个唯一标识objectID，在数据定义时可以使用BIND_OBJECT_ID()方法将某个属性绑定到objectID上。若不绑定objectID，则插入数据时会自动生成一个类型为NSNumber的唯一标识。

```
#import "SYBManagedObject.h"

@interface SYBTestObject : SYBManagedObject // 必须继承SYBManagedObject

BIND_OBJECT_ID(name) // 将name设置为关键字，name必须唯一。也可以不用设置
@property (strong, nonatomic) NSString *name;
@property (nonatomic) NSInteger value;

@end
```

SYBTestObject最终保存到数据库中是一个名为SYBTestObject的数据表，带有两个列：name，value。

❗目前并不支持多级继承关系的数据对象，举个例子说明：

```
#import "SYBTestObject.h"

@interface AnotherTestObject : SYBTestObject

@property (strong, nonatomic) NSString *nickName;
@property (nonatomic) NSInteger secondValue;

@end
```

上面的AnotherTestObject在保存到数据库中时，只会保存nickName，secondValue这两个属性。目前的类解析还不支持父类属性。所以这里父类SYBTestObject的name和value属性不会被保存到数据库中。

- 支持的类型

Objective-C 类型	C类型
NSString	BOOL
NSNumber	short / unsigned short
NSArray	int / unsigned int
NSDictionary	long / unsigned long
NSValue	long long / unsigned long long
NSData	double
NSSet	float

SYBDatabase

SYBDatabase可以看作是数据库db文件的抽象，负责数据库的管理：

- 在本地目录下创建和删除db文件
- 删除数据库中的表
- 设置数据库基本参数
- 线程安全，支持跨线程使用

创建一个SYBDatabase实例并不会在本地创建数据库文件，只有在真正使用的时候才会在本地目录下面创建指定名称的数据库文件。

```
SYBDatabase *db  = [SYBDatabase databaseWithName:@"MyDatabase"];
```

上面的SYBDatabase实例在使用时会在App的Document目录下创建一个名为 MyDatabase.db 的数据库文件

SYBDatabaseContext

SYBDatabaseContext代表一个数据库表，是数据库操作的入口，实现了数据对象的添加，更新，查询，删除等基本操作。
SYBDatabaseContext是线程安全的，可以在不同线程之间传递使用。

```
SYBDatabase *db  = [SYBDatabase databaseWithName:@"MyDatabase"];
SYBDatabaseContext *context = [SYBDatabaseContext contextWithClass:
[SYBTestObject class] inDatabase:db];
```

上面代码会在数据库 MyDatabase 中创建名称为 SYBTestObject 的数据库表，按照SYBTestObject的property属性来定义表中的列名。

- **数据插入**

调用SYBDatabaseContext的 **insert:** 和 **insertObjects:** 方法。

```
SYBDatabaseContext *context = …

// 插入单个数据
SYBTestObject *object = [SYBTestObject new];
object.name = @"obj_1";
object.value = 12345;
[context insert:object];

// 批量插入多个数据
NSArray *testObjects = ... // 一组SYBTestObject对象
[context insertObjects:testObjects];
```

- **数据更新**

调用SYBDatabaseContext的 **update:** 和 **updateObjects:** 方法。

```
SYBDatabaseContext *context = …

// 更新单个数据
NSArray *result = [context queryAll];
for (SYBTestObject *obj in result) {
    obj.name = @"Updated Object";
    [context update:obj];
}

// 插入多个数据
NSArray *testObjects = ... // 一组SYBTestObject对象
[context updateObjects:testObjects];
```

❗如果对象没有objectID，则更新操作会返回失败。 因此更新操作的对象必须是之前查询出来的对象，通过修改之后再更新回去。

- **数据删除**

调用SYBDatabaseContext的 **remove:** ， **removeObjects:** 和**removeAllObjects:**方法。

```
SYBDatabaseContext *context = …
// 查询所有数据
NSArray *objects = [context queryAll];

// 删除单个数据
[context remove:objects[0]];

// 批量删除多个数据
[context removeObjects:objects];

// 删除所有数据
[context removeAllObjects];
```

❗如果对象没有objectID，则删除操作会返回失败。因此删除操作的对象必须是之前查询出来的对象。

• 批量操作

批量操作的接口有：**insertObjects: , updateObjects: , removeObjects:**

- 在操作数据的数量相同的情况下，**批量操作的效率要比普通操作高效10+倍。**
- 在批量操作过程中，若有一个数据操作失败，则数据库会回滚到这次操作之前的状态，方法返回NO。

• 普通查询

调用SYBDatabaseContext的 **queryAll** 和 **queryById:** 方法。

queryAll 查询当前表中的所有数据。

queryById:按照对象的objectID查询特定的对象。若没绑定objectID，则会按自动分配的objectID查找。

```
SYBDatabaseContext *context = …

// 查找所有
NSArray *objects = [context queryAll];

// 根据objectID查找
SYBTestObject *object = [context queryById:@"obj_1"]
```

• 条件查询

调用SYBDatabaseContext的 **queryByCondition:** 方按照指定条件进行查找。使用时传入一个SYBQueryCondition变量，变量中定义了具体的查找条件。

```
// 条件查询例子
SYBQueryCondition *condition = [SYBQueryCondition new];
// 设置查询条件
[condition setConditionWithFormat:@"name = %@",@"obj_1"];
// 设置排序
[condition setSortOrderWithProperties:@[@"name"] isAscending:YES];
// 设置返回结果的范围
[condition setRange:NSMakeRange(1, 10)];
NSArray *result = [context queryByCondition:condition];
```

设置查询条件:

- 支持的操作符有: **>, <, ==, >=, <=**
- 两个条件之间可以用 **&&** 或 **||** 组合起来
- 只支持Objective-C类型的对比，例如数字类参数必须转换成NSNumber

例子：

```
// 普通写法
[...Format:@"stringValue == %@", @"hello"];

// 多条件写法
[...Format:@"stringValue == %@ && numValue >= %@", @"hello", @(123)];

// 以NSArray, NSDictionary, NSValue, NSData等为参数
[...Format:@"array_property == %@", @[@"array1", @"array2"]];

// 若不使用%@，则在对比字符时须添加''
[...Format:@"stringValue == 'string' && intValue = 1"];
```

设置排序

properties: 排序属性数组，数组中字段的先后顺序进行排序

isAscending: 排序顺序 YES:升序 NO:降序

例子：

```
// 单属性升序排序
[condition setSortOrderWithProperties:@[@"name"] isAscending:YES];

// 多属性降序排序，这里先按name排序，在name相同的情况下按照value来排
[condition setSortOrderWithProperties:@[@“name”, @“value”]
                            isAscending:NO];
```

设置返回结果的范围

range.location 查询结果的起始位置,即偏移值，不需要时填写SYBNotUsed

range.length 查询结果长度，不需要时填写SYBNotUsed

例子：

```
// 第1条记录（包括第1条）后的10条记录
[condition setRange:NSMakeRange(1, 10)];

// 限制返回条数为20条
[condition setRange:NSMakeRange(SYBNotUsed, 20)];

// 从第3条记录开始返回
[condition setRange:NSMakeRange(3, SYBNotUsed)];

// 注意：这种写法的返回结果为0
[condition setRange:NSMakeRange(3, 0)];
```

- **删除数据库表**

调用SYBDatabase实例中的**removeTableForClass:** 删除该数据库中的表。删除表时，如果相关联的SYBDatabaseContext没有被销毁，则该Context会变成无效状态，这时候任何的数据库操作都会返回失败。

```
// 创建
SYBDatabase *db  = [SYBDatabase databaseWithName:@"TestDB"];
SYBDatabaseContext *context = [SYBDatabaseContext contextWithClass:
[SYBTestObject class] inDatabase:db];
...
// 从TestDB数据库中删除表SYBTestObject
[db removeTableForClass:context.tableClass];

// 这里的操作是无效的，其他在使用SYBTestObject这个表的Context也处于无效状态
[context insert:object];
```

- **删除数据库**

调用SYBDatabase类方法**removeDatabase:** 删除数据库，此操作会删除保存在本的数据库文件。删除时，如果有其他 SYBDatabase 或 SYBDatabaseContext 实例也在使用该数据库，那么这些实例会变成无效状态，这时候任何的数据库操作都会返回失败。
SYBDatabase实例释放时，依赖该实例的SYBDatabaseContext也会失效

```
// 创建
SYBDatabase *db  = [SYBDatabase databaseWithName:@"TestDB"];
SYBDatabaseContext *context = [SYBDatabaseContext contextWithClass:
[SYBTestObject class] inDatabase:db];
...
// 删除数据库TestDB
[SYBDatabase removeDatabase:db.name];

// 这里的db实例已经无法继续使用，当前的删除表操作会返回失败
[db removeTableForClass:context.tableClass];
```

- **数据库通知**

- 数据库删除通知 SYBDatabaseRemovedNotification，在调用到**removeDatabase:** 时发出
- 数据库关闭通知 SYBDatabaseClosedNotification 在SYBDatabase实例释放时发出
- 数据库表删除通知 SYBDatabaseRemoveTableNotification 在调用到**removeTableForClass:** 时发出

- 线程安全

- SYBDatabase，SYBDatabaseContext这两个类的实例都是线程安全的，可以在任意线程上传递调用。
- SYBDatabaseContext中的查询，添加，更新和删除等操作都是同步操作，即当执行这些操作时，当前线程会挂起，等数据库操作完成后再返回线程继续执行。由于数据库涉及到I/O操作，有可能会长时间挂起线程，**因此不推荐在主线程上面进行数据库操作。**

技术实现

SYBDatabase是基于FMDB的一个面向对象数据库。基本上这里进行的工作就是把一个OC对象进行解析，把里面的property属性的值转换成一条数据记录保存到数据库里面，取出时把数据库里面的每条记录转换成OC对象。

- 数据库与OC对象的转换

创建数据库表

1) SYBDatabaseContext初始化时会拿到一个NSObject类
2) 调用class_copyPropertyList( )方法获取类中的属性列表，解析列表获取每个属性的类型以及其对应的sqlite类型
3) 对与sqlite的关键字有冲突的表名或属性名进行转换
4) 以类名作为表名，以属性名为列名，在数据库中创建表

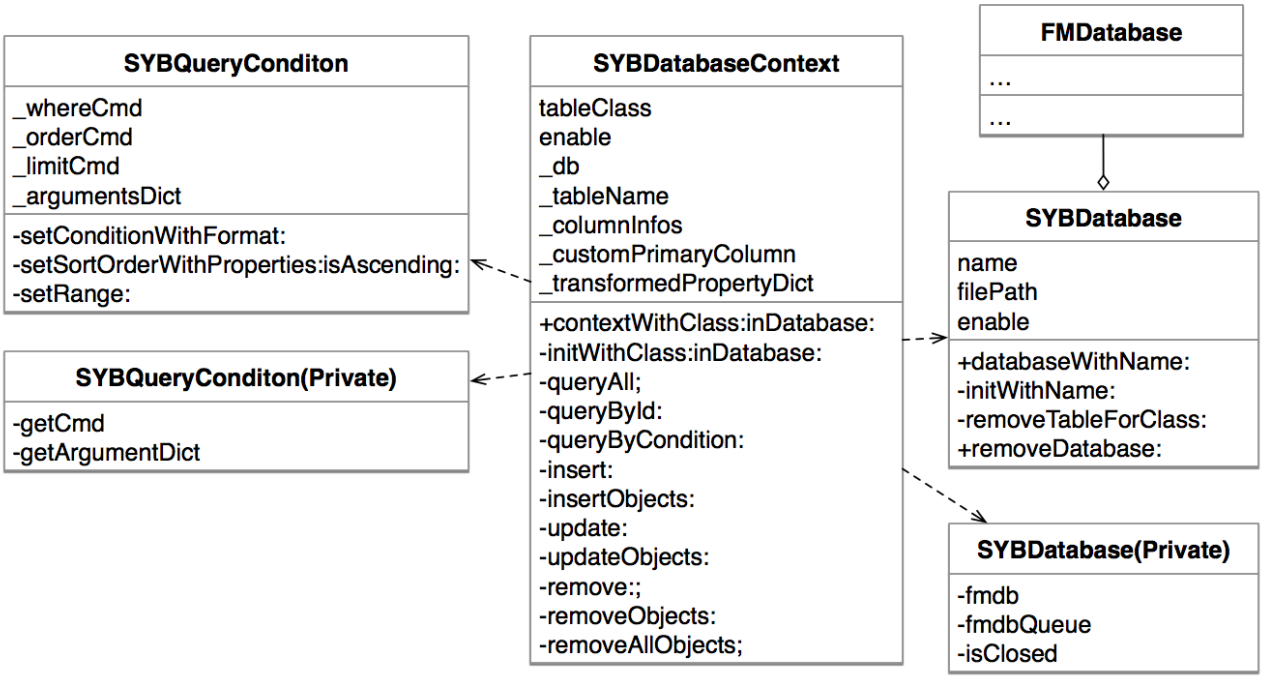
OC对象到数据库

1) 创建表时拿到了OC对象的属性信息。
2) 利用KVO特性，直接对插入对象调用**valueForKey:**方法获取相应的属性值。
3) 把NSArray，NSDictionary，NSSet，NSValue类型的属性转换成NSData
4) 调用fmdb接口把这条记录插入到数据库中

数据库到OC对象

1) fmdb查询接口返回一个FMResultSet对象
2) FMResultSet中调用**objectForColumnName:** 获取特定列名对应的值
3) 新建一个OC对象，调用**setValue:ForKey:** 的把值赋到对应的属性上面

- **UML**



- 多线程

- 每个数据库拥有一条独立的线程进行数据库操作
- 不同SYBDatabase实例，使用同一个数据库，共享同一条线程，当所有SYBDatabase实例都被释放后，该数据库线程也会被释放
- SYBDatabaseContext进行数据库操作时，会挂起当前线程，然后在数据库线程中执行操作，完成后返回到原来的线程继续执行

NEXT

- **SYBDatabase后续新增功能**

- 数据加密

- **SYBDatabase考虑优化点**

- 数据库表版本升级
- 实现数据对象之间的关联关系
- 数据库操作缓存
- 延迟数据对象的属性值获取