GYDB使用文档

# 描述(description)

1.采用分类的模式,方便基于NSObject的类或者对象的数据库操作.

2.关于线程安全:所有数据库操作都放在一个并发量为1的操作队列中,保证多线程使用的安全.

3.支持的数据类型有:

* char, unsigned char, short, unsigned short, int, unsigned int.
* long, unsigned long, long long, unsingned long long, float, double.
* NSString.
* NSNumber.
* NSDate.

如果没有搜索到GYDB,请删除本地缓存后再安装:

rm ~/Library/Caches/CocoaPods/search\_index.json

注意:不使用pod安装的小伙伴需要手动添加libsqlite3.tbd的依赖

**基本配置(config)**

**通用配置**

是否开启日志(默认开启):

//关闭日志

DBManager.openLog = NO;

设置异步回调时的队列,默认为manager的异步队列

//设置在主线程中回调block

DBManager.completionQueue = dispatch\_get\_main\_queue();

打开(创建)数据库:默认会打开(创建)Documents/gydb/gydb.sqlite路径下的数据库

[DBManager openDatabase:dbPath];

关闭数据库

[DBManager closeDatabase:dbPath];

**存储配置**

自定义主键值

- (NSString \*)gy\_customPrimaryKeyValue {

    return @"456";

}

自定义对象关联(返回自定义对象属性的类型)

+ (NSDictionary<NSString \*,Class> \*)gy\_customClass {

    return @{@"bestFriend":[Person class],

             @"favoritePet":[Pet class]};

}

对象数组关联(返回数组中元素的类型,不管是自定义类型还是支持的NS类型,都需要返回)

+ (NSDictionary<NSString \*,Class> \*)gy\_classInArray {

    return @{@"nickNames":[NSString class],

             @"favoriteNums":[NSNumber class],

             @"favoriteDates":[NSDate class],

             @"privateDatas":[NSData class],

             @"pets1":[Pet class],

             @"pets2":[Pet class]};

}

**表操作(table operate)**

检查表是否存在

BOOL exist = [Person gy\_tableExistsWithError:nil];

创建表

[Person gy\_createTable];

删除表

[Person gy\_dropTable];

更新表(只增加旧表没有的字段)

[Person gy\_updateTable];

**插入(insert)**

**插入单条数据**

同步方法:

[person gy\_insert];

异步方法:

[person gy\_insertWithCompletion:^(GYDBError \*error) {

        if (error) {

                //succeed

        }else {

                //failed

        }

}];

**插入多条数据**

同步方法:

GYDBError \*error = [[GYDatabaseManager sharedManager] insertObjs:persons];

异步方法:

[DBManager insertObjs:persons completion:^(GYDBError \*error) {

}];

**删除(delete)**

**根据对象删除**

同步方法:

[person gy\_delete];

异步方法:

[person gy\_deleteWithCompletion:^(GYDBError \*error) {

}];

**根据类删除**

同步方法:

//删除Person表中所有数据

[Person gy\_deleteAll]

//删除Person表中age小于79的行

[Person gy\_deleteObjsWithCondition:DBCondition.Where\_P(age).Lt(@79)];

异步方法:

//删除Person表中所有的行

[Person gy\_deleteAllWithCompletion:^(GYDBError \*error) {

}];

//删除Person表中height大于等于100的行

[Person gy\_deleteObjsWithCondition:DBCondition.Where\_P(height).GtOrEq(@100) completion:^(GYDBError \*error) {

}];

**查询(query)**

同步方法:

//查询Person表中主键大于456的行,根据age逆序排序

[Person gy\_queryObjsWithCondition:DBCondition.Where\_PK().Gt(@"456").OrderBy\_P(age).Descending() error:&error];

异步方法:

//查询Person表中所有的行

[Person gy\_queryObjsWithCondition:nil completion:^(NSArray \*result, GYDBError \*error) {

}];

**修改(update)**

同步方法:

//更新除age之外的所有属性

[person gy\_updateWithExcludeColumns:@[@"age"]];

异步方法:

//更新所有属性

[person gy\_updateWithExcludeColumns:nil completion:^(GYDBError \*error) {

}];

**保存(save)**

如果对象未入库,save方法等于insert方法.否则等于update方法.

同步方法:

[obj gy\_save];

异步方法:

[obj gy\_saveWithCompletion:^(GYDBError \*error) {

}];

**其它操作(other)**

获取当前打开的数据库路径,没有则返回nil

DBManager.databasePath;

获取表中数据行数

//查询Person表中的数据行数

NSInteger rowCount1 = [Person gy\_countWithCondition:nil error:&error];

//查询Person表中age大于等于24的数据行数

NSInteger rowCount2 = [Person gy\_countWithCondition:DBCondition.Where\_P(age).GtOrEq(@24) error:&error];

**链式条件配置(condition)**

使用链式语法能够方便地配置执行Sql操作时的条件.举例:

//查询name以Alone结尾或者age大于等于24,根据age降序排序的从索引0开始的10条数据...

[Person gy\_queryObjsWithCondition:DBCondition.Where\_P(name).Like(@"%Alone").Or\_P(age).GtOrEq(@24).OrderBy\_P(age).Descending().Limit(0,10) error:&error]

详细说明见下表:

| **条件** | **说明** | **举例** | **参数类型** |
| --- | --- | --- | --- |
| Where | 筛选条件.传入属性名. | Where(@"age"):根据age筛选 | NSString \* |
| Where\_P | 筛选条件.和Where相同,会自动匹配selector,方便输入. | Where\_P(age):根据age筛选 | NSString \* |
| Where\_PK | 筛选条件.匹配主键,等价Where(@"\_id"). | Where\_PK():根据主键筛选 | void |
| Eq | 比较条件.相等. | Where\_P(age).Eq(@24):age等于24 | NSString *NSNumber* |
| Nq | 比较条件.不相等. | Where\_P(name).Nq(@"ygy"):name不等于ygy | NSString *NSNumber* |
| Lt | 比较条件.小于. | Where\_P(age).Lt(@24):age小于24 | NSString *NSNumber* |
| Gt | 比较条件.大于. | Where\_P(age).Gt(@24):age大于24 | NSString *NSNumber* |
| LtOrEq | 比较条件.小于等于. | Where\_P(age).LtOrEq(@24):age小于等于24 | NSString *NSNumber* |
| GtOrEq | 比较条件.大于等于. | Where\_P(age).GtOrEq(@24):age大于等于24 | NSString *NSNumber* |
| Like | 通配符筛选条件. | Where\_P(name).Like(@"%Alone"):根据通配符'%Alone'查询 | NSString \* |
| And | 与条件. | Where\_P(name).Like(@"%Alone").And(@"age").Gt(@17):name满足通配符匹配,并且age大于17 | NSString \* |
| And\_P | 与条件.和And相同,会自动匹配selector,方便输入. | Where\_P(name).Like(@"%Alone").And\_P(age).Gt(@17):name满足通配符匹配,并且age大于17 | NSString \* |
| Or | 或条件. | Where\_P(name).Like(@"%Alone").Or(@"age").Gt(@24):name满足通配符匹配,或者age大于24 | NSString \* |
| Or\_P | 或条件.和Or相同,会自动匹配selector,方便输入. | Where\_P(name).Like(@"%Alone").Or(@"age").Gt(@24):name满足通配符匹配,或者age大于24 | NSString \* |
| OrderBy | 排序条件. | OrderBy(age):根据age排序,默认为升序排序 | NSString \* |
| OrderBy\_P | 排序条件,和OrderBy相同,会自动匹配selector,方便输入. | OrderBy(age):根据age排序,默认为升序排序 | NSString \* |
| Ascending | 升序排序条件 | OrderBy\_P(age).Ascending():根据age升序排序 | void |
| Descending | 降序排序条件 | OrderBy\_P(age).Descending():根据age降序排序 | void |
| Limit | 数量限制条件 | Limit(0,10):结果集合中从索引0开始的10条数据 | NSInteger(offset) NSInteger(len) |

更多详细介绍请参考NSObject+GYDB.h和GYDatabaseManager.h

github地址:<https://github.com/ygyalone>