LRU在寻梦记账中的应用

在记账的时候，为了用户更快捷的记录名称和备注，寻梦记账提供了一种「快捷备注」的功能，目前快捷名称/备注主要根据「出现次数」来决定优先级，但是出现次数并不是用户需要的，而时效性，一段时间内最常用的才是用户真正需要的，这和LRU非常相似，所以我们决定对快捷备注进行优化

1. 原理

快捷备注优化：二级缓存+LRU

- LRU优化展示顺序
- 缓存优化重复加载

2. 技术选型

- 大量set/get?
  - 只在insert/update的时候set，每次记账（加上关闭页面重新打开的时间）通常需要2-6秒，非大量set
  - 只在记账/修改页面打开时候get，操作间隔时间较长
- 并发
  - 需要加锁，选择NSLock，无递归锁场景
  - 无大批量场景不需要高性能锁，NSLock可读性，易用性高于pthread_mutex_t，且小批量场景性能差距不大
- 磁盘容器
  - 不使用NSFileManager或SQLite，使用更加轻量级的NSUserDefaults
  - 存储数据结构自适应NSUserDefaults(NSDictionary、NSArray、NSString)
- 数据结构

  存储在NSUserDefaults里的数据将会是这样的

代码块

```
{
    @"categoryId_1" : {
        @"早餐",
        @"拌面",
        @"馕",
    },
    @"categoryId_2" : {
```

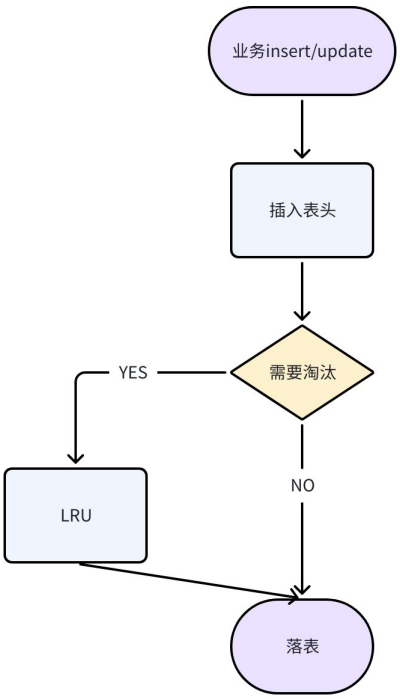
```
        @"饮料",
        @"咖啡",
        @"红牛",
    }
}
```

- 内部数据结构使用NSMutableDictionary + NSMutableOrderedSet：Dictionary保证O(1)级别的存取，OrderedSet保证数据单一，有序，实现LRU
- 对外返回NSArray，业务层无感知

3. 为什么不用YYCache，而是要自研?

- YYCache更加适合大对象，图片等高性能场景，仅存储100+字符串的数组不需要引入新的sdk
- 数据量不大且存储的是Foundation基础类型的时候，使用NSUserDefaults足够了，不需要存储到NSFileManager/SQLite
- 场景不搭配
  - 寻梦记账的场景：一次valueForKey:需要categoryId对应的完整的数组，而不是一个对象，如果setValue:forKey:的时候set数组又不能支持LRU了
  - 如果把categoryId映射成YYCache里面的name，也就是filePath，虽然支持了LRU，但是读取的时候需要getAllValues，YYCache不支持这样的方法

4. 流程图



5. 关键代码

代码块

```
@implementation _XMDiskCacheManager {
    NSString *_userDefaultsKey;
}

- (NSMutableOrderedSet *)orderedSetForKey:(NSString *)key {
    NSDictionary *dictionary = [self _dictionary];
    NSArray *array = dictionary[key];
    if ([array isKindOfClass:[NSArray class]]) {
        return [NSMutableOrderedSet orderedSetWithArray:array];
    }
    return nil;
}

- (void)setOrderedSet:(NSMutableOrderedSet *)orderedSet forKey:(NSString *)key
{
    NSMutableDictionary *dictionary = [self _dictionary].mutableCopy;
    if (CollectionsUtils.isNotEmpty(orderedSet)) {
        dictionary[key] = [orderedSet array];
    } else {
        dictionary[key] = nil;
    }
```

```
    [[NSUserDefaults standardUserDefaults] setValue:dictionary forKey:_userDefaultsKey];
}

@end
```

- 职责单一原则：只负责本地持久化相关操作，不和CacheManager耦合
- 解藕/变动：后期如果需要其它容器存储，只需要修改diskManager，业务/cacheManager无感知

代码块

```

@implementation XMCacheManager

- (void)setObject:(id)object forKey:(NSString *)key {
    if (key == nil || object == nil) {
        // object是set中的对象，所以不需要remove
        return;
    }
    [_lock lock];
    NSMutableOrderedSet *set = _cache[key];
    if (set == nil) {
        set = [_diskCache orderedSetForKey:key] ?: [NSMutableOrderedSet orderedSet];
        self.cache[key] = set;
    }
    // NSOrderedSet插入一个已存在的数据，会无操作
    [set removeObject:object];
    [set insertObject:object atIndex:0];

    [self _trimToLimit:set];

    [_diskCache setOrderedSet:set forKey:key];
    [_lock unlock];
}

- (NSArray *)objectsForKey:(NSString *)key {
    if (!key) return nil;

    [_lock lock];
    NSMutableOrderedSet *set = _cache[key];
    if (!set) {
        set = [_diskCache orderedSetForKey:key];
        if (!set) {
```

```
            if ([self.delegate respondsToSelector:@selector(defaultObjectsForKey:)]) {
                NSArray *objects = [self.delegate defaultObjectsForKey:key];
                set = [[NSMutableOrderedSet alloc] initWithArray:objects];
                [_diskCache setOrderedSet:set forKey:key];
            }
        }
        _cache[key] = set;
    }
    [_lock unlock];
    return [set array];
}

- (void)_trimToLimit:(NSMutableOrderedSet *)orderedSet {
    while (orderedSet.count > _limitCount) {
        [orderedSet removeObjectAtIndex:orderedSet.count - 1];
    }
}

@end
```

- 对外使用统一使用NSArray，隐藏内部数据结构，业务层不关心内部数据结构的实现，也不要做额外的类型转换
- XMCacheManagerDelegate：不同类型业务返回各自的默认值

其它

- CacheManager要同时管理cache的逻辑，还需要负责cache和disk联动，不符合单一职责原则，为什么不继续分层?
  - 拆分主要是为了解藕，单一性原则，代码复用
  - 可以继续拆分，但是没必要，业务定制型sdk，后期变动可能性不大，且cache, disk都不会单独使用
  - 如果后续需要，只需要拆分出memoryCache，把memoryCache的逻辑单独抽象成一个class

实战

- 业务代码量减少百分之50，删除了没有必要的class，代码可读性增加

代码块

```
static NSString * const kXMFastNotesUserDefaultsKey =
```

```
@"com.lannastudio.fast.notes.kXMFastNotesUserDefaultsKey";

@interface FastNotesManager () <XMCacheManagerDelegate>

@property (nonatomic, strong) XMCacheManager *cacheManager;

@end

@implementation FastNotesManager

+ (instancetype)sharedInstance {
    static FastNotesManager *sharedInstance = nil;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        sharedInstance = [[self alloc] init];
    });
    return sharedInstance;
}

- (instancetype)init {
    self = [super init];

    if (self) {
        _cacheManager = [[XMCacheManager alloc]
initWithUserDefaultsKey:kXMFastNotesUserDefaultsKey];
        _cacheManager.delegate = self;
        _cacheManager.limitCount = 25;
    }
    return self;
}

- (void)cacheNotes:(NSString *)notes categoryId:(NSInteger)categoryId {
    if (StringUtils.isBlank(notes)) {
        return;
    }
    [_cacheManager setObject:notes forKey:@(categoryId).stringValue];
}

- (NSArray *)fastNotesArrayWithCategoryId:(int64_t)categoryId {
    return [_cacheManager objectsForKey:@(categoryId).stringValue];
}

#pragma mark - cache delegate

- (NSArray *)defaultObjectsForKey:(NSString *)key {
    int64_t categoryId = key.longLongValue;

```

```
    return [self _notesSortedByCountWithCategoryId:categoryId];
}

- (NSArray *)_notesSortedByCountWithCategoryId:(int64_t)categoryId {
    NSArray *filterBills = [[[BillManager sharedInstance] billsInCurrenBook]
kt_select:^BOOL(Bill *bill) {
        return bill.categoryId == categoryId &&
StringUtils.isNotBlank(bill.notes);
    }];

    if (CollectionsUtils.isEmpty(filterBills)) {
        return @[];
    }

    NSMutableDictionary *dictionary = [NSMutableDictionary dictionary];
    [filterBills xm_each:^(Bill *bill) {
        if (dictionary[bill.notes] == nil) {
            dictionary[bill.notes] = @1;
        } else {
            NSNumber *count = dictionary[bill.notes];
            dictionary[bill.notes] = @(count.integerValue + 1);
        }
    }];

    return [dictionary keysSortedByValueUsingComparator:^NSComparisonResult(id
obj1, id obj2) {
        return [obj2 compare:obj1];
    }];
}

@end

```