一些NSArray,NSDictionary,NSSet相关的算法知识

iOS编程当中的几个集合类：NSArray，NSDictionary，NSSet以及对应的Mutable版本，应该所有人都用过。只是简单使用的话，相信没人会用错，但要做到高效（时间复杂度）精确（业务准确性），还需要了解其中所隐藏的算法知识。

在项目当中使用集合类几乎是不可避免的，集合类的使用场景其实可以进行抽象的归类。大多数时候我们需要将若干个对象（object）暂时保存起来，以备后续的业务逻辑进行操作，「保存和操作」，或者说「存与取」，对应到计算机世界的术语就是读和写。最初保存的时候我们Insert，下次进行更新的时候我们再Get，不再需要的时候我们调用Delete，所以你看集合类的操作场景其实就那么多，关键在于我们存的方式，和取的方式不同。

最初我们学习数据结构和算法的时候，知道数据的组织方式不同，比如Array, List, Stack, Heap, Tree，其对应的读和取效率（时间复杂度）也不同。如果insert的效率高，下次get的时候效率就低，比如无序的Array，插入的时候O（1），查找的时候就变O（N）。如果想要查找的速度快，比如排序过的Array，查找的速度在O（logN），插入的时候就必须要保持Array有序这一特性O（N）。所以插入和查找是鱼与熊掌，想要下次快速的找到

一本书，就必须在整理书架的时候多花些心思分门别类。或者我们跳出时间的维度，用更多的空间来做弥补，使用哈希表或者Dictionary来存储数据，查找的速度可以快至O（1），缺点是牺牲了更多的空间。

当我们预先存好Array之后，使用的时候大多是以下几种场景：

场景一

```
for (NSObject* obj in self.arr) {
    //update each object
}
```

场景二

```
if ([self.arr containsObject:obj] == false) {
    [self.arr addObject:obj];
}
```

场景三

```
if ([self.arr containsObject:obj] == true) {
    [self.arr removeObject:obj];
}
```

第一种场景没有多少可发掘的，一次干净利索的遍历费时O（N）。唯一需要注意的是切忌在遍历的时候改变集合对象，比如：

```
for (NSObject* obj in self.arr) {
    if(obj.isInvalid){
        [self.arr removeObject:obj];
    }
}
```

如果要在遍历的时候删除可以换种写法，比如：

```
for (int i = self.arr.count-1; i > 0; i --) {
    NSObject* obj = self.arr[i];
    if (obj.isInvalid) {
        [self.arr removeObject:obj];
    }
}
```

场景二和场景三需要特别留意，containsObject，removeObject都涉及到一个集合当中的重要概念，即**相等性**。

值的相等性很简单，不用思索就能得出直观的答案，比如1==1，2.0f==2.0f。

对象的相等性就不那么简单了。什么时候我们认为两个对象是相等的呢？我们可以从两个维度去理解相等性。

同一对象相等：

理论上说两个对象的指针如果是指向同一块内存区域，那么他们一定是相等的，一定是指向同一个对象。这种情况下我们判断相等性是通过

```
if (obj1 == obj2)
```

业务属性相等：

两个对象即使不指向同一块内存区域，但他们的所有（或者部分关键的）property是相等的，我们也可以认为这两个对象是相等的，比如连个UserProfile对象，他们的name，gener，age属性都相等，在业务层面，我们可以认为他们是相等的，此时我们不能用==来判断相等性了，需要重载isEqual，或者自己实现isEqualToXXX：

```
@implementation MyObject

- (BOOL)isEqual:(id)object
{
    if (self == object) {
        return true;
    }
    if ([object isKindOfClass:[self class]] == false) {
        return false;
    }

    MyObject* myObject = object;
    if ([self.name isEqualToString:myObject.name]) {
        return true;
    }

    return false;
}

@end
```

所以当我们判断两个集合当中对象是否相等时，一定要心中明确是那种相等。当调用containsObject，removeObject的时候，如果我们重载了isEqual，系统就通过我们的isEqual方法来判断相等性，如果没有重载，那么系统就会通过判断内存地址来判断相等性了。

有些架构model layer的设计会允许同一个业务对象在应用层存在多份拷贝，此时在Array当中使用相等性的时候尤其要注意重载isEqua方法。当然有些mode layer只允许一份拷贝，一个业务对象永远只对应一个内存地址，isEqual方法就变得多余了。

和isEqual配套的另一个方法hash也经常被提起，官方文档甚至规定isEqual和hash必须被同时实现。学习过hash表之后，我们知道如果两个对象业务上相等，那么他们的hash值一定是相等的，hash方法的用处还是在于判断相等性，系统默认的hash方法实际上返回的就是对象的内存地址。问题是我们已经有isEqual方法来判断相等性了，为什么还需要一个hash呢?

答案是hash可以更加高效快速的判断一个对象是否存在集合当中，在NSArray当中我们需要遍历Array，调用N次isEqual才能知道对象是否存在集合当中，时间复杂度是O（N）。在调用isEqual之前，可以通过调用hash来判断是否相等，如果hash值不等就没有进一步调用isEqual的必要了，如果相等必须再调用一次isEqual来确认是否真正相等。但是hash为什么会比isEqual的效率要高呢？看下hash的声明就明白了。

```
- (NSUInteger)hash
{
    return [_name hash];
}
```

hash方法的返回值是一个NSUInteger，这个值往往和对象在内存当中的存储位置直接相关，也就是说我们可以通过这个值以O（1）的复杂度快速读取到某个对象来判断相等性，和Array O（N）的复杂度相比快了太多了，Array显然不具备这种特性，Array当中的元素是在一片内存空间当中连续排放的，和hash的返回值没任何关系。

但这种使用hash的便捷性有一个前提：对象在集合当中是唯一的，也就是说集合当中不允许存在重复的元素，比如NSDictionary，NSSet。我们在使用下列方法的时候：

```
[dictionary objectForKey:key];
```

```
[set addObject:object];
```

为了保证唯一性，都需要先判断对象是否存在集合当中，此时一个高效的判断机制十分重要，这也就是hash发挥作用的地方，这也是为什么使用NSArray的时候只会调用isEqual，而使用NSDictionary，NSSet的时候会频繁调用hash的原因。

所以当我们使用NSDictionary，NSSet的时候，同时重载isEqual和hash方法对性能至关重要。hash方法的选择并不需要过分挑剔，对关键的property做下运算，保证绝大部分场景下hash值不同即可，毕竟hash调用之后还是会调用isEqual做进一步判断，并不会对我们业务的准确性产生影响。

Objective C当中的几个关键集合类：NSArray，NSDictionary，NSSet要高效的使用并没有看起来那么简单，当集合类中的元素到达一定量级之后，考虑下背后的算法效率很有必要，这也是为什么一直强调算法对于程序员的重要性。

欢迎关注公众号：

上一篇
iOS开发之玩转蓝牙CoreBluetooth
(/blog/ios-bluetooth/)

下一篇
iOS端数据库解决方案分析
(/blog/ios-database/)