第四十七条：熟悉系统框架

要点：

1. 许多系统框架都可以直接使用，其中最重要的是Foundation与CoreFoundation，这两个框架提供了构建应用程序所必须的许多核心功能。可以通过无缝桥接将它们互相转换。
2. 许多常见任务都能用框架来做，例如音频和视频处理，网络通信和数据管理。
3. 请记住：用纯C写成的框架与Objective-C写成的一样重要，若想成为优秀的Objective-C开发者，应该熟悉C语言的核心概念。

第四十八条：多使用block枚举，少用for循环

可以使用NSEnumerator枚举所有的集合对象，对于没有下标的NSSet同样适用。

```
NSArray *array = @{……};
NSEnumerator *enumerator = [array objectEnumerator];
id object;
while ((object = [enumerator nextObject]) != nil) {
    //do someThing.
}
```

对于NSEnumerator还有keyEnumerator：遍历字典key，reverseObjectEnumerator：反向遍历

基于block的遍历方式

```
// 这是针对数组最基本的block遍历方式。
-(void)enumeratorObjectUsingBlock:(void(^)(id object,NSUInteget idx,BOOL *stop)bl
// 针对字典
-(void)enumeratorKeysAndObjectsUsingBlock:(void(^)(id key,id object,BOOL *stop)bl
```

此方式也可以实现反向遍历，只需要在另一个版本中传入一些参数NSEnumerationOptions.

要点：

1. 遍历collection有四种方法：for循环，NSEnumeator，快速遍历和block枚举法；
2. block枚举法本身可以通过GCD实现并行执行遍历操作，无需另行编写代码，采用其他形式无法轻易实现这一点；
3. 若提前知道待遍历的collections含有何种对象，则应该修改block签名，指出对象的具体类型。

第四十九条：对自定义其内存管理语义的collection使用无缝桥接

在Foundation框架下的对象可以和CoreFoundation框架下的对象实现平滑转换，这种技术叫做“无缝桥接toll-free-bridging”，比如Foundation下的NSArray和CoreFoundation下的CFArray可以实现无缝桥接。

下面的代码显示了这一技术

```
NSArray *anNSArray = @[@1,@2,@3,@4];
CFArrayRef aCFArray = (__bridge CFArrayRef)anNSArray;
NSLog(@"size of array is %li",CFGetArrayCount(aCFArray));//output 4.
```

要点：

1. 通过无缝桥接技术，可以在Foundation下的NSObejct和CoreFoundation下的C语言数据结构之间来回转换；
2. 在CoreFoundation层面创建的collection时，可以指定许多回调函数用来处理collection中的每个元素，然后用无缝桥接技术转成Objective-C对象。

第五十条：构建缓存时选用NSCache而非NSDictionary

要点：

1. 实现缓存时应选用NSCache而非NSDictionary对象，因为NSCache可以提供优雅的自动删减功能，而且是线程安全的，此外，它与字典不同，并不会拷贝；
2. 可以给NSCache对象设置上限，用以限制缓存中的对象的总数和总成本，而这些尺度则定义了缓存删减其中对象的时机；
3. 将NSPurgeableData与NSCache搭配使用，可以实现自动清空数据的功能，也就是说，NSPurgeableData所占据的内存被系统丢弃时，该对象自身也会从缓存中移除；
4. 如果缓存使用得当，那么应用程序的响应速度就能提高。只有那种重新计算起来很费事的数据才值得放入缓存，比如从网络上获取的或者从磁盘中读取的数据。

第五十一条：精简initialize与load的实现代码

在类的初始化方法中，load和initialize方法会最先调用，但是也有不同。

要点：

1. 在加载阶段，如果实现了load方法，那么系统就会调用它。分类里面也可以定义此方法，类的load方法会比分类的load方法先调用。**与其他方法不同，load方法不参与覆写机制**；
2. 首次使用某个类之前，系统会向其发送initialize方法。**由于此方法遵从普通的覆写规则**，所以通常应该在里面判断当前要初始化的是那个类。
3. load方法与initialize方法都应该实现的精简一点，这有助于保持应用程序的响应能力，也能减少引入“依赖循环”（interdependency cycle）的几率；
4. 无法在编译器设定的全局变量，可以放在initialize方法中初始化。

第五十二条：别忘了NSTimer会保留其目标对象

要点：

1. NSTimer对象会保留其目标，直到计时器本身失效为止，调用invalidate方法可以使计时器失效，另外，一次性计时器在触发任务之后也会失效；
2. 反复执行任务的计时器，很容易引起循环计数问题，如果这种计时器的目标对象保留了计时器本身，那么肯定会引起循环引用，这种问题可能是直接发生的，也可能是通过对象图里的其他对象间接发生的；
3. 可以扩充NSTimer的功能，用block来打破这种循环引用问题，不过，除非NSTimer将来在公共接口中提供此功能，否则必须创建分类，将相关实现代码加入其中。