应该尽可能设计出容易复用的代码接口，让自己或者其他使用者使用方便。

第十五条：用前缀避免命名空间的冲突

OC中没用其他语言中的命名空间的概念，所以我们要自己约定自己的命名空间。为所有的类加上自己的前缀，这也是OC所提倡的做法，比如NS，UI等都是独立的前缀。

不管是自己写的类还是应用的第三方库，都提倡为其添加前缀，以区分为不同的的类，否则，极容易在编译链接时导致“重复符号”错误（duplicate symbol error）。

要点：

1. 选择与你的公司、应用程序或者二者皆关联的名称作为类名的前缀，并在所有代码中均使用这一前缀；
2. 若自己所开发的程序库中使用了第三方库，则应为其加上前缀。

第十六条：提供全能初始方法

如果创建的类的初始化方法不止一个，那么就要在其中选定一个作为全能初始化方法，并令其他初始化方法都调用它。若后续程序的底层设计改变了，就会很容易的维护和统一。

子类首先要调用超类的对应初始化方法，逐层调用。实现“initWithCoder:“也要这样，先调用超类的相关方法，再调用自身方法。

要点：

1. 在类中提供一个全能初始化方法，并于文档中指明。其他初始化方法均应该调用此初始化方法；
2. 若全能初始化方法与超类不同，那么应该覆写超类的对应方法；
3. 如果超类的初始化方法不适用子类，那么应该覆写这个超类方法，并在其中抛出异常。

第十七条：实现description方法

在description方法中可以这样写，以返回有用信息，而不是只返回内存地址和类名等信息：

```
-(NSString *)description {
    return [NSString stringWithFormat:@"<%@:%p,%@>",
                [self class], self,
                @{@"title":_title,
                    @"name":_name,
                      ……}
                ];
}
```

这样便可以很方便的打印出对开发者有用 的调试信息。

要点：

1. 实现description方法返回一个有意义的字符串，用以描述该实例；
2. 若想在调试时打印出更加详细的对象信息，则应该实现debugDescription方法。

第十八条：尽量使用不可变对象

要点：

1. 尽量创建不可变的对象；
2. 若某属性仅可于对象内部修改，则在“class-continuation"中有readonly属性扩充为readwrite属性；
3. 不要把可变的collection作为属性公开，而应提供相关方法，依次修改对象中的可变collection。

第十九条：使用清晰而协调的命名方式

要点：

1. 起名时应该昨从标准的objectIve-C命名规范，这样创建出来的接口更容易为开发者理解；
2. 方法名要言简意赅，从左到右读起来要像日常用语中的句子才好；
3. 方法名不要使用省略后的类型名称；
4. 给方法起名时的第一要务是确保其风格和自己的代码或者要集成的框架相符。

第二十条：为私有方法添加前缀

要点：

1. 给私有方法的名称加上前缀，这样会很容易地将其与公开方法区分开；
2. 不要单用一个下划线做私有方法的前缀，因为这种做法是苹果公司自己用的。

第二十一条：理解objective-C错误模型

在iOS中，无法将某个类标识为抽象类，要想达到同样的效果，可以在那些子类必须覆写父类方法中抛出异常。

```
-(void)mustOverrideMethod{
    @throw [NSException exceptionWithName:NSExceptionNotInstanceException reason:
}
```

用NSError或者nil/0代表那些并不是致命性的错误，表明在某个操作中发生了错误：

NSError中封装了三条信息：

- Error domain（错误范围，类型为字符串）错误发生的范围，也就是错误的根源，通常用一个特有的全局变量来定义。比方说“处理URL的子系统在从URL中解析或者取得数据时出错了，那么就会使用NSURLErrorDomain来表示错误。
- Error code（错误码，类型为整数）一种独有的代码，表示某个范围的某种错误。
- User info（用户信息，类型为字典）有关错误的额外信息，其中可能包含一段本地化描述。

最好能为您自己的程序库张所发生的错误指定一个专有的”错误范围“字符串，使用此字符来创建您的NSError对象，并将其返回给库的使用者。

要点：

1. 只有发生了可使整个程序奔溃的严重错误时，才应该使用异常；
2. 在错误不那么严重的情况下，可以实拍“委托方法”来处理错误，也可以吧错误信息放在NSError对象中，经由“输出参数”返回给调用者。

第二十二条：理解NSCoping协议

iOS中的内置类基本都支持copy方法，但是默认都是“浅拷贝”。

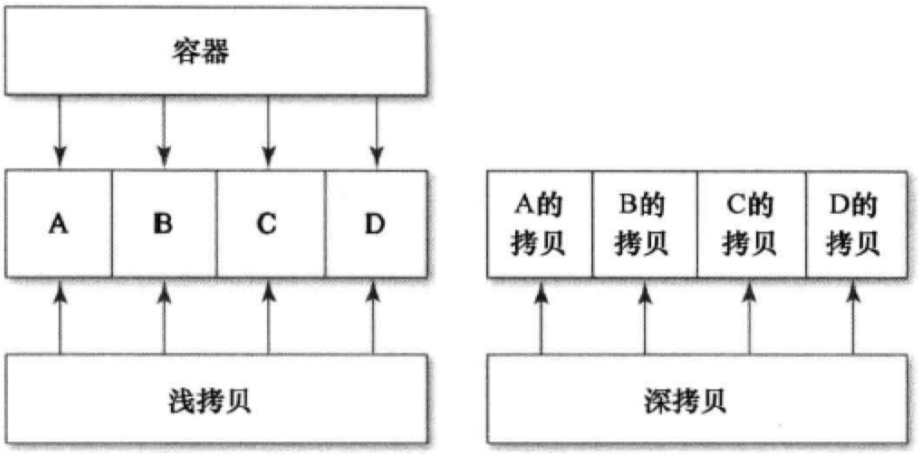

图 3-2　浅拷贝与深拷贝对比图。浅拷贝之后的内容与原始内容均指向相同对象。而深拷贝之后的内容所指的对象是原始内容中相关对象的一份拷贝

- 浅拷贝：拷贝得到的数据和被拷贝的数据指向同一块内存地址，修改其中的一个，另一个也会感受到改变；
- 深拷贝：拷贝得到的数据和被拷贝的数据是独立的，其内存地址没有指向同一块。

如果要让自己的类支持copy操作，那就必须让自己的类实现NSCopying协议中的方法：

copyWithZone:zone

zone指的区（磁盘的分区所致），这里对于同一个程序都是默认区。

```
-(SomeClass *)copyWithZone:zone{
    SomeClass someCopyClassInstance = [SomeClass alloc] initWith::::]
    return comeCopyClassInstance;
}
```

要实现真正的深拷贝，一般通过序列化和反序列化，来得到真正相互独立的对象。

```
@implementation ClassA
- (id)copyWithZone:(NSZone*)zone{
    NSData *buffer;
    buffer = [NSKeyedArchiver archivedDataWithRootObject:self];
    ClassA *copy = [NSKeyedUnarchiver unarchiveObjectWithData: buffer];
    return copy;
}
@end
```

要点：

1. 若想令自己所写的类具有copy功能，则需要实现NSCopying协议。
2. 如果自定义的对象分为可变版本和不可变版本，需要同时实现NSCopying和NSMutableCopying协议。

3. 复制对象时需要决定浅拷贝还是深拷贝,一般情况下应该尽量执行浅拷贝。
4. 如果你所写的对象需要深拷贝，那么可考虑新增一个专门执行深拷贝的方法。