


Objective-C学习

 一个明智地追求快乐的人，除了培养生活赖以支撑的主要兴趣之外，总得设法培养其他许多闲情逸致。—— 罗素

消息传递机制

继承自Smalltalk的消息传递模型（message passing）

在Objective-C，类别与消息的关系比较松散，调用方法视为对对象发送消息，所有方法都被视为对消息的回应。

所有消息处理直到运行时（runtime）才会动态决定，并交由类别自行决定如何处理收到的消息。也就是说，一个类别不保证一定会回应收到的消息，如果类别收到了一个无法处理的消息，程序只会抛出异常，不会出错或崩溃。

Objective-C

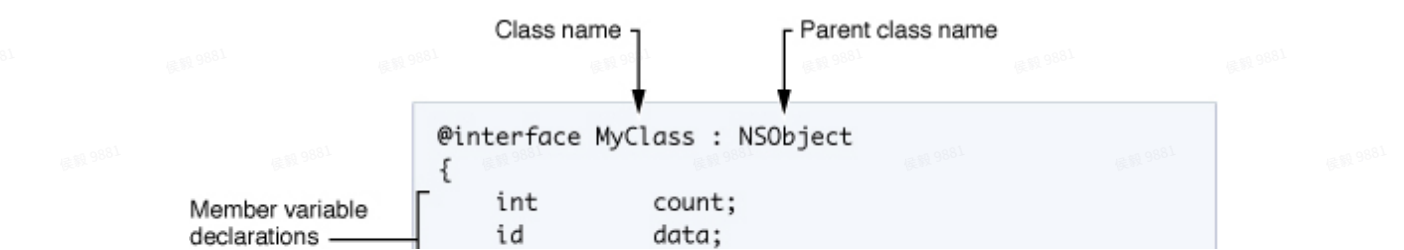
1 [car fly];

典型的C++意义解读是"调用car类别的fly方法"。若car类别里头没有定义fly方法，那编译肯定不会通过。但是Objective-C里，我们应当解读为"发送一个fly的消息给car对象"，fly是消息，而car是消息的接收者。car收到消息后会决定如何回应这个消息，若car类别内定义有fly方法就运行方法内之代码，若car内不存在fly方法，则程序依旧可以**通过编译**，**运行期则抛出异常**。

C++强制要求所有的方法都必须有对应的动作，且编译期绑定使得函数调用非常快速。缺点是仅能借由virtual关键字提供有限的动态绑定能力。Objective-C天生即具备**鸭子类型**之动态绑定能力，因为**运行期才处理消息**，**允许发送未知消息给对象**。可以送消息给整个对象集合而不需要一一检查每个对象的类型，也具备消息转送机制。同时**空对象nil接收消息后默认为不做事**，**所以发消息给nil也不用担心程序崩溃**。

类

类声明图



Method declarations

```
NSString* name;
}
- (id)initWithString:(NSString*)aName;
+ (MyClass*)createClassWithString:(NSString*)aName;
@end
```

方法的声明

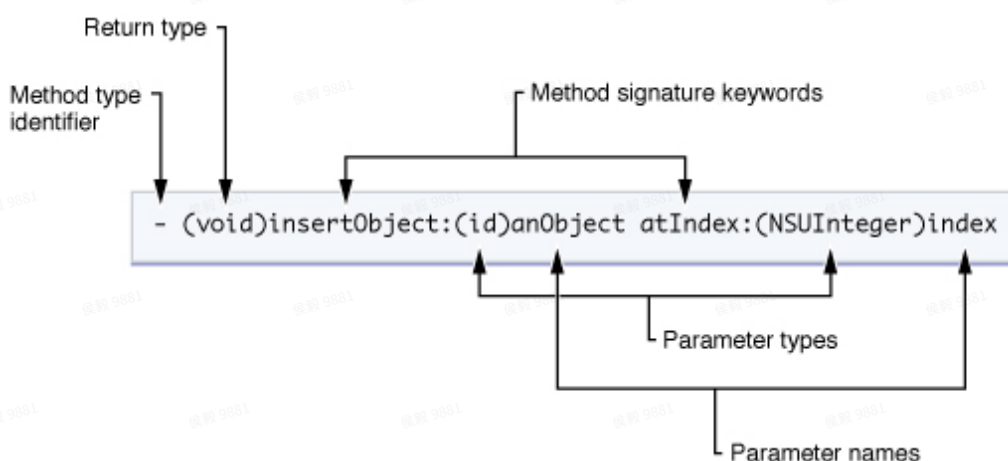
方法前面的 +/- 号代表函数的类型：加号 (+) 代表类方法（class method），不需要实例就可以调用，与C++ 的静态函数（static member function）相似。减号 (-) 即是一般的实例方法（instance method）。

Objective-C定义一个新的方法时，名称内的冒号 (:) 代表参数传递，Objective-C方法使得参数可以夹杂于名称中间，不必全部附缀于方法名称的尾端，可以提高程序可读性。

Objective-C

```
1 - (void) setColorToRed: (float)red Green: (float)green Blue:(float)blue; /* 方法声明 */
2 [myColor setColorToRed: 1.0 Green: 0.8 Blue: 0.2]; /* 方法调用 */
```

方法的名字 (setColorToRed:Green:Blue:) 是所有方法标识关键字的级联，包含了冒号。冒号表明了参数的出现。如果方法没有参数，可以省略第一个 (也是唯一的) 方法标识关键字后面的冒号。



当你想调用一个方法，你传递消息到对应的对象。这里消息就是方法标识符以及传递给方法的参数信息。发送给对象的所有消息都会动态分发，这样有利于实现Objective-C类的多态行为。也就是说，如果子类定义了跟父类的具有相同标识符的方法，那么子类首先收到消息，然后可以有选择的把消息转发（也可以不转发）给他的父类。

Objective-C允许使用嵌套消息，每个嵌套消息的返回值可以作为其他消息的参数或者目标。

Objective-C

```
1 [[myAppObject getArray] insertObject:[myAppObject getObjectToInsert] atIndex:0];
```

类成员变量定义的位置

不只Interface区块可定义实体变量，Implementation区块也可以定义实体变量，两者的差别在于访问权限的不同，Interface区块内的实体变量默认权限为protected，定义于implementation区块的实体变量则默认为private，故在Implementation区块定义私有成员更匹配面向对象之封装原则，因为如此类别之私有信息就不需曝露于公开interface（.h文件）中。

动态类型

一个对象收到消息之后，他有三种处理消息的可能手段，第一是回应该消息并运行方法，若无法回应，则可以转发消息给其他对象，若以上两者均无，就要处理无法回应而抛出的异常。只要进行三者之其一，该消息就算完成任务而被丢弃。若对"nil"（空对象指针）发送消息，该消息通常会被忽略，取决于编译器选项可能会抛出异常。

鸭子类型

虽然Objective-C具备动态类型的能力，但编译期的静态类型检查依旧可以应用到变量上。以下三种声明在运行时的效果是完全相同的，但是三种声明提供了一个比一个更明显的类型信息，附加的类型信息让编译器在编译时可以检查变量类型，并对类型不符的变量提出警告。

下面三个方法，差异仅在于参数的类型：

Objective-C

```
1 - setMyValue:(id) foo;
```

id类型表示参数"foo"可以是任何类的实例。

Objective-C

```
1 - setMyValue:(id <aProtocol>) foo;
```

id<aProtocol>表示"foo"可以是任何类的实例，但必须采纳"aProtocol"协议。

Objective-C

```
1 - setMyValue:(NSNumber*) foo;
```

该声明表示"foo"必须是"NSNumber"的实例。

注意：实际传递到方法中的参数类型可以是任意的，编译器只会给出警告，不会报错。错误发生在运行期。

动态类型相比于静态类型的一个优势

动态类型是一种强大的特性。在缺少泛型的静态类型语言（如Java 5以前的版本）中实现容器类时，程序员需要写一种针对通用类型对象的容器类，然后在通用类型和实际类型中不停的强制类型转换。无论如何，类型转换会破坏静态类型，例如写入一个"整数"而将其读取为"字符串"会产生运行时错误。这样的问题被泛型解决，但容器类需要其内容对象的类型一致，而对于动态类型语言则完全没有这方面的问题。

Objective-C

```
1      NSArray *array=[NSArray arrayWithObjects:@"0909",@"houyi",@"today",[NSAr  
y arrayWithObjects:@"2021",@"1018",nil], nil];
```

容器中可以同时存储不同类型的变量（NSString，NSArray等），静态类型的范型很难实现这一特性。

转发

Objective-C允许对一个对象发送消息，不管它是否能够响应之。除了响应或丢弃消息以外，对象也可以将消息转发到可以响应该消息的其他对象。

Objective-C是消息型语言，通过重写系统函数，在运行时实现消息的转发：

Objective-C

```
1  - (id) forwardingTargetForSelector:(SEL)aSelector
```

转发的原理

当调用一个函数的时候，分为两个阶段：

- 1.传递消息（一般我们写的正常函数，在消息传递时期就可以正确实现）。
- 2.消息传递失败时，消息转发。

1. 传递消息与objc_msgSend方法

例如当调用如下函数时：

Objective-C

```
1  id returnValue = [someObject messageName: parameter];
```

编译器会把它转换成一个 C 语言函数：

Objective-C

```
1 // void objc_msgSend(id self, SEL cmd, ...)
2 id returnValue = objc_msgSend(someObject, @selector(messageName:), parameter);
```

该方法在someObject类中搜寻"方法列表", 如果找不到, 就沿继承体系依次向上在父类们中找, 到顶层父类还找不到, 就会进入第二步：消息转发。

如果找到, 匹配结果会缓存到每个类的"快速映射表", 以提高下一次执行相同方法的速度。

2. 消息转发

若传递消息失败（查找至顶层父类仍未找到方法），则会进入消息转发阶段。消息转发也分为两个阶段：a.动态方法解析：征询接受者，看是否能动态添加方法。b.基于消息转发机制：i.让接收者看看有没有其他备选对象处理该消息；ii.启动完整的消息转发机制。

a. 动态方法解析

消息转发首先进入动态方法解析阶段，询问消息的接受者，能否动态添加方法。根据尚未实现的方法是类方法还是实例方法，调用其所属类的两个方法之一：

Objective-C

```
1 //尚未实现的方法是实例方法
2 + (BOOL)resolveInstanceMethod:(SEL)sel
3 //尚未实现的方法是类方法
4 + (BOOL)resolveClassMethod:(SEL)sel
```

返回值表示这个类能否新增一个方法来处理。

此方案常用来实现@dynamic属性，具体实现方法暂未学习。

b. 基于消息转发机制

若 a 阶段没有做，则进入 b 阶段，基于消息转发机制。此阶段又可进一步分为两个阶段：

i. 备选接收者

这一阶段，系统会询问，能不能把消息转给其他备选接受者来处理。

Objective-C

```
1 - (id)forwardingTargetForSelector:(SEL)aSelector
2 {
3     NSLog(@"在此函数中返回备选的消息接收者");
4     return _recipient;
5 }
```

重写该方法，返回一个备援对象，来处理aSelector方法。

ii. 启动完整的消息转发机制

如果上一步没处理，就会进入最后这一步，启用完整的消息转发机制。生成NSInvocation对象，存储了那条未处理的消息的全部细节：方法、目标、参数等。

重写下面两个函数，可实现消息的转发：

Objective-C

```
1 -(NSStringSignature*)methodSignatureForSelector:(SEL)aSelector
2 {
3     if(aSelector==@selector(sayHello))
4     {
5         return [NSStringSignature signatureWithObjCTypes:"v@:"];
6     }
7     return nil;
8 }
```

Objective-C

```
1 - (void)forwardInvocation:(NSInvocation *)anInvocation
2 {
3     if (anInvocation.selector == @selector(sayHello))
4     {
5         NSLog(@"在此处处理消息");
6     }
7 }
```

重写 forwardInvocation 函数，可以在里面做很多事，比如修改目标、修改方法、修改参数等等。

如果目标子类没有重写 methodSignatureForSelector 方法，或 methodSignatureForSelector 方法返回 nil，则它会层层向父类传递调用，直到传递给 NSObject 的该方法，里面会调用 "doesNotRecognizeSelector:"，然后抛出异常。

注意一个细节：若 b i. 阶段已经将消息转发给了备选接收者，且备选接收者无法处理该转发的消息，则 b ii. 阶段调用的是**备选接收者**的 methodSignatureForSelector 及 forwardInvocation 函数，而不是原消息转发者的 methodSignatureForSelector 及 forwardInvocation 函数。

分类 (Category)

在Objective-C的设计中，一个主要的考虑是大型代码框架的维护。结构化编程的经验显示，改进代码的一种主要方法是将其分解为更小的片段。Objective-C借用并扩展了Smalltalk实现中的"分类"概念，用以帮助达到**分解代码**的目的。

分类根据有无分类名称，可以分为 分类 和 类扩展（匿名分类）。

分类

分类可以将方法的实现分解进一系列分离的文件。程序员可以将一组相关的方法放进一个分类，使程序更具可读性。

1. 分类中，只能使用 getter、setter 或者 self 点语法访问成员变量，不能直接使用变量名访问。
2. 只在原类 .m 文件中定义的方法（私有方法未在 .h 文件中声明）是不能在分类中访问的。
3. 在使用分类的扩展方法时，需要导入分类的.h文件，才能让执行代码知道分类扩展方法的存在。实际上，由于在分类文件中已经导入了原始类的头文件，所以在使用分类时，仅仅导入分类头文件即可。
4. 分类能够对实例方法，类方法进行扩展，但不能够添加类的属性及实例变量。

进一步的，分类中的方法是在运行时被加入类中的，这一特性允许程序员向现存的类中增加方法，而无需持有原有的代码，或是重新编译原有的类。

若分类声明了与类中原有方法同名的函数，则分类中的方法会被调用。因此分类不仅可以增加类的方法，也可以代替原有的方法。这个特性可以用于修正原有代码中的错误，更可以从根本上改变程序中原有类的行为。若两个分类中的方法同名，则被调用的方法是不可预测的。

Objective-C 允许动态的、按需的加载分类；若不需要某一分类提供的功能，可以简单的不编译之。

类扩展

类扩展其实是一种特殊的分类，即匿名分类

类扩展与分类的区别如下：

1. 在类扩展中可以扩展类的属性，而在分类中仅能够扩展实例方法和类方法。
2. 类扩展仅能够在原始类中声明（.h或.m中均可，在.m中声明的类扩展其定义的属性和方法均是私有的）
3. 类扩展的实现仅能够在原始类的.m中编写。

分类（Category）的一些细节

1. 对于Cocoa中的类，我们也可以进行分类扩展，特别是对于NSObject类，我们对其扩展，那么所有的类均可以调用我们的扩展方法！
2. 分类扩展可以继承。
3. 对于分类或扩展中声明的方法，我们并不要必须实现，而是在必要时，才会有某个类来实现，这点和协议很像。
4. 对于类中的同名方法，分类扩展会覆盖其实现。
5. 对于类的私有方法，我们可以在分类扩展中将其声明为可见的（而不实现），这样在类外部就可以调用该类的私有函数了。即，使用分类来暴露原始类中的私有方法，使其不再私有。
6. 通常在原始类的 .m 文件中为原始类添加类扩展，在类扩展中可以使用 @property 语法，从而可用来定义私有变量。类外不能访问 .m 文件中的类扩展中定义的私有变量。

内存管理

ARC:

基本类型用 assign

delegate 用 weak

NSArray, NSDictionary 和 NSString, block 用 copy

其他用 strong

block 中 self 用 weak 打破循环引用