Object C 基础

一、什么是Object C?

1、 它是一种面向对象编程语言

2、 它是ANSI版本C编程语言的超集，支持C的基本语法。

2007年发布了版本2.0

Objective-C头文件使用的文件名如下

扩展名	内容类型
.h	头文件。头文件包含类，类型，函数和常数的声明。
.m	源代码文件。这是典型的源代码文件扩展名，可以包含Objective-C和C代码。
.mm	源代码文件。带有这种扩展名的源代码文件，除了可以包含Objective-C和C代码以外还可以包含C++代码。仅在你的Objective-C代码中确实需要使用C++类或者特性的时候才用这种扩展名。

二、数据类型

1、通用数据类型(与c语言通用)

整型：int, short, long ,unsigned int,
unsigned short, unsigned long.
浮点型: float, double, long double
字符型: char

二、数据类型

2、特有数据类型

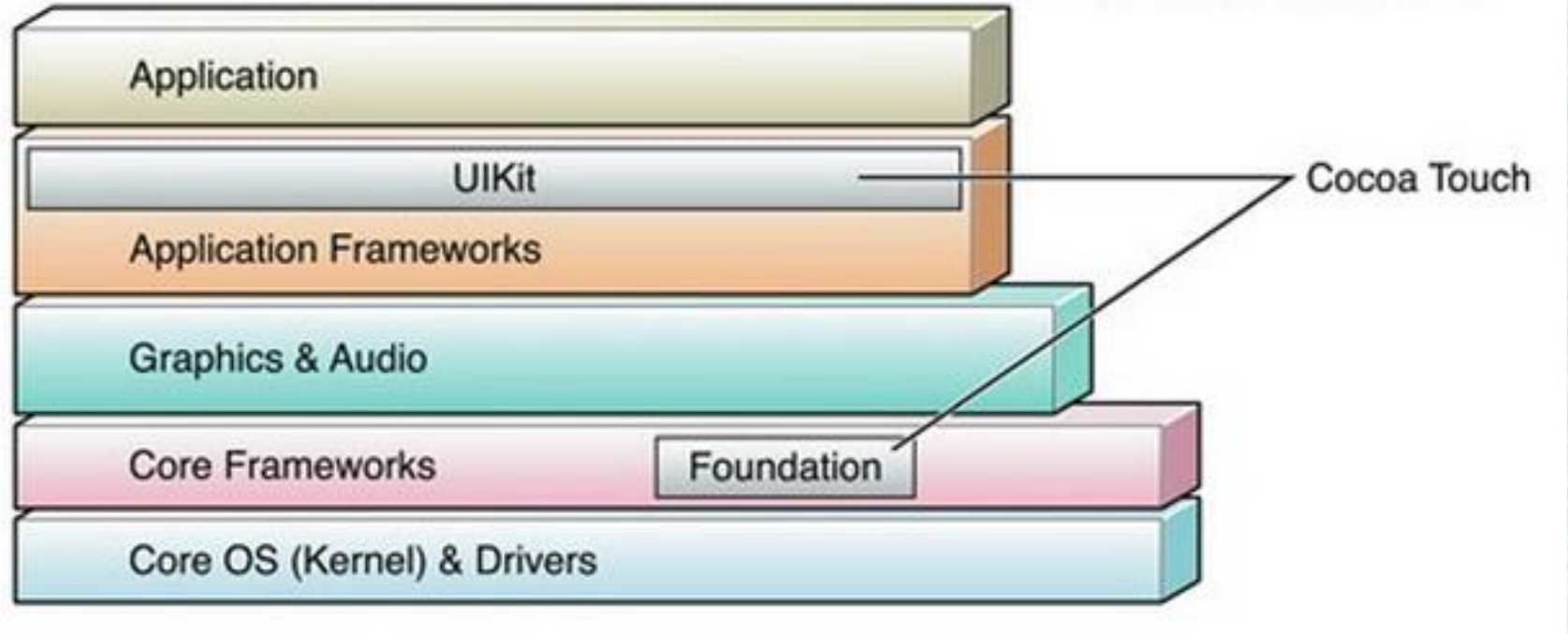
id ： 动态类型
BOOL: 布尔型（0,1）
enum: 枚举类型
SEL: 选择器类型
Class: 类的类型
Nil(nil)：空对象

二、数据类型

3、Cocoa 类型

- Cocoa是苹果公司为Mac OS X所创建的原生面向对象的API。 Cocoa包括两个方面：即运行环境方面和开发方面。
- Cocoa 是一整套集成的面对对象的软件模块(类)，它允许开发人员快速创建稳定的功能完善的OS X和OS 应用程序。这些类是可以重用和修改的软件构造模块；开发人员可以直接使用它们或者也可以根据他们自己的需求扩展这些类。
- Cocoa的类满足几乎所有可以想像到的开发需求。

iOS中两个Cocoa核心子框架



二、数据类型

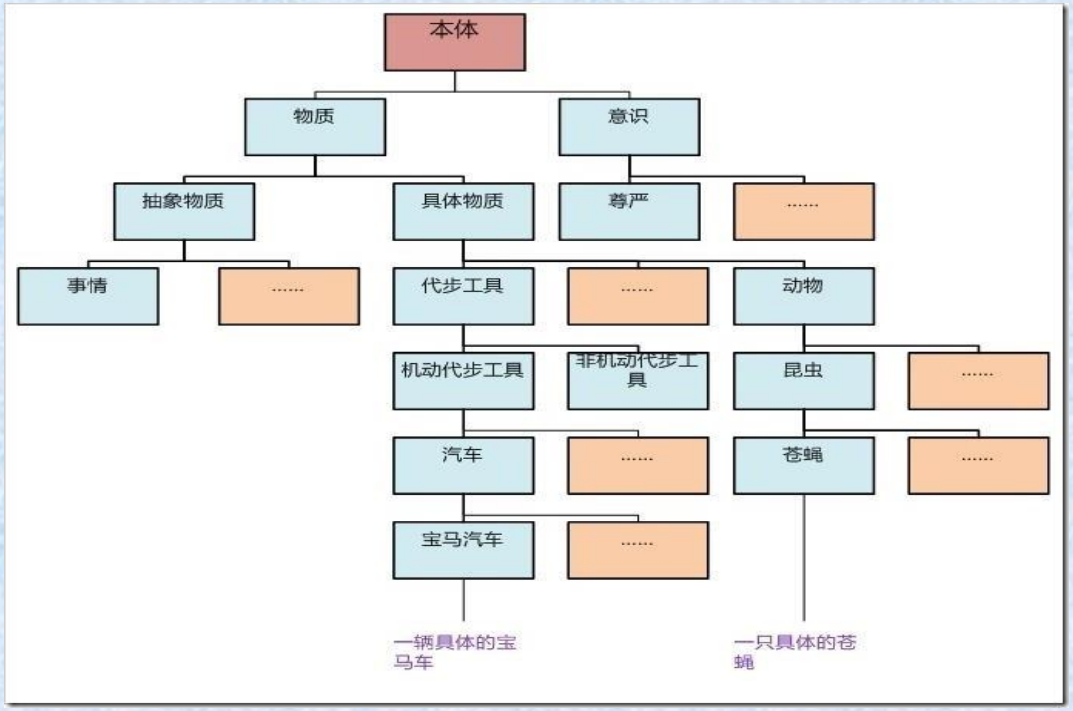
一个Foundation框架中的NSString 的例子

```
NSString *result = [[NSString alloc] initWithString:@"This is a demo."];
```

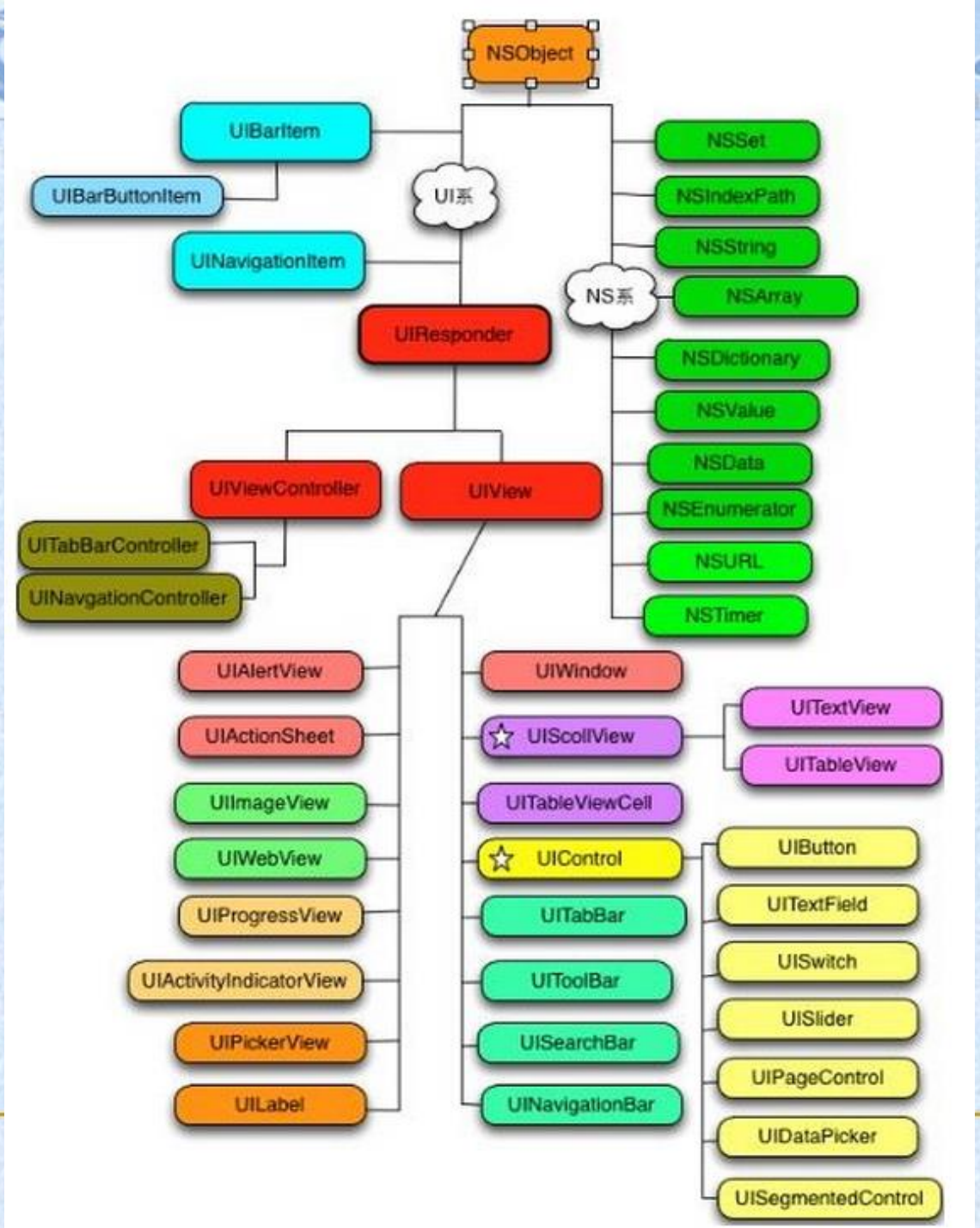
三、类与对象

1、根类NSObject

抽象层次树



本体即万物之源、万物之本，是哲学层面上最高层次的抽象



NSObject为基本父类

NSObject根类和它采纳的NSObject协议以及其它“根”协议一起，为所有不作为代理对象的Cocoa对象指定了如下的接口和行为特征：

1、分配、初始化和复制

eg: alloc, init, new

```
myClass *obj=[[ myClass alloc] init]
myClass *obj=[myClass new]
```

2、对象的保持和清理

eg: retain,release

3、内省与比较

eg: isKindOfClass方法

```
if ([teacher isKindOfClass:[Teacher class]])
{ …… }
```

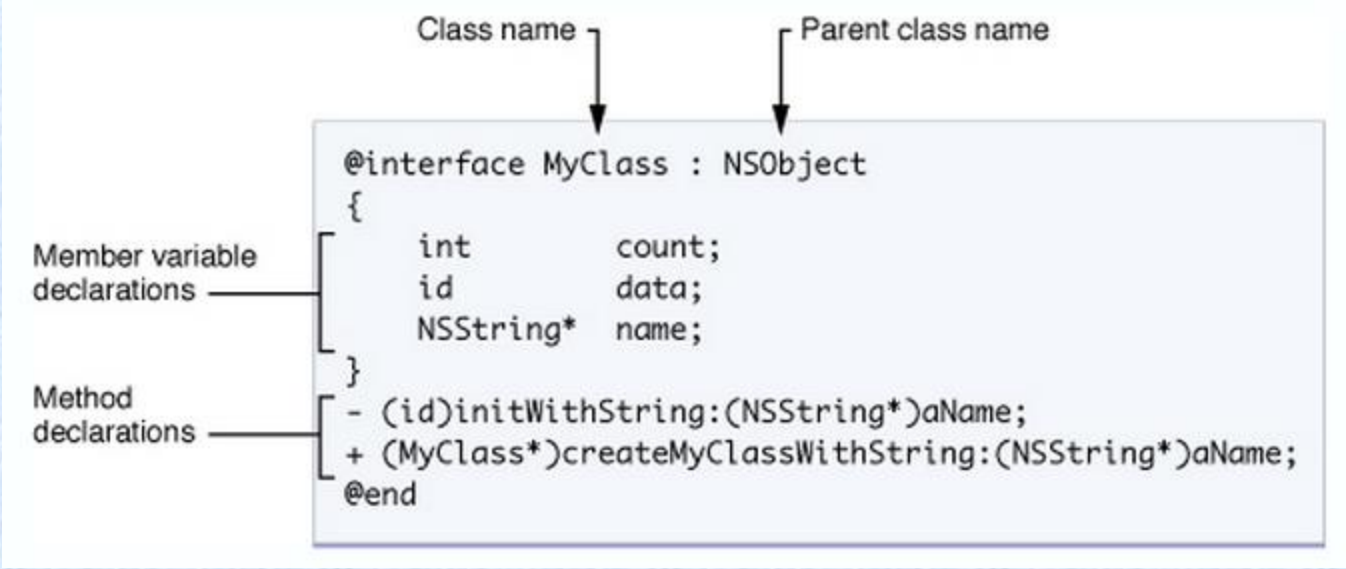
4、对象的编码和解码

5、消息转发

6、消息派发

2、类的声明、实现

类声明总是由@interface编译选项开始，由@end编译选项结束。类名之后的（用冒号分隔的）是父类的名字。类的实例（或者成员）变量声明在被大括号包含的代码块中。实例变量块后面就是类声明的方法的列表。每个实例变量和方法声明都以分号结尾。



类实现的位置也由两个编译选项确定，@implementation和@end。这些选项给编译器提供了要将方法和对应类联系起来，所需的范围信息。

```
@implementation MyClass

- (id)initWithString:(NSString *) aName
{
    if (self = [super init]) {
        count count = 0;
        data = nil;
        name = [aName copy];
        return self;
    }
}

+ (MyClass *)createMyClassWithString: (NSString *) aName
{
    return [[[self alloc] initWithString:aName] autorelease];
}
@end
```

3、.h与.m文件

头文件.h和代码文件.m可分别存放声明和实现。当你需要在源代码中包含头文件的时候，你可以使用标准的include编译选项，但是Objective-C提供了更好的方法#import选项。它和#include选项完全相同，只是它可以确保相同的文件只会被包含一次。Objective-C的例子和文档都倾向于使用#import。

eg: #import <Foundation/Foundation.h>

```
//  Copyright (c) 2014年 sse. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface helloworld : NSObject


@end
```

```
//  Copyright (c) 2014年 sse. All rights reserved.
//

#import "helloworld.h"
@interface helloworld ()

@end

@implementation helloworld
```

注意：在.h中声明的方法和属性可以被其它的类访问，如果需要定义仅供本类访问的属性和方法可以在.m中进行声明。声明也是由@interface编译选项开始，由@end编译选项结束。

4、强类型和弱类型

当用变量保存对象的时候，始终应该使用指针类型。Objective-C对变量包含的对象支持强弱两种类型。强类型(静态类型)指针的变量类型声明包含了类名。弱类型(动态类型)指针使用id作为对象的类型。弱类型指针常用于类的集合，在集合中对象精确的类型可以是未知的。

```
MyClass*   myObject1;     // Strong typing
id         myObject2;     // Weak typing
```

注意：id类型前没有*,但它还是一个指针

在对象类型未知的情况下，id类型非常有用，例如NSArray

5、类方法与实例方法

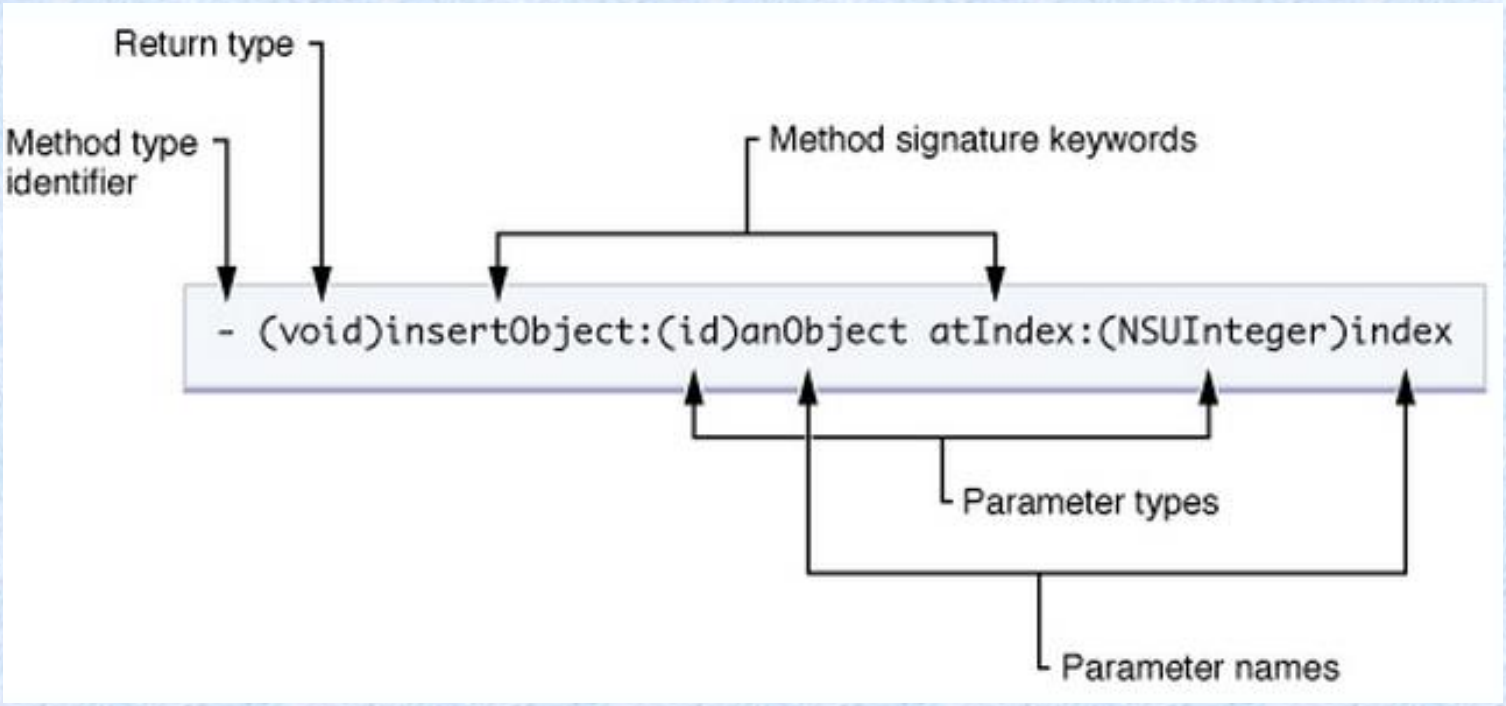
Object-C中的方法同其它面向对象语言一样，分两种方法：实例方法(-)和类方法(+)(静态方法)。实例方法需要通过类的实例去调用，而静态方法可直接通过类名去调用。

```
//  Copyright (c) 2014年 sse. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface helloworld : NSObject

+ (void) sayHelloWorld;
- (void) sayHello:(NSString *) greeting;

@end
```

```
//  Copyright (c) 2014年 sse. All rights reserved
//

#import "helloworld.h"
@interface helloworld ()


@end

@implementation helloworld

- (void) sayHello:(NSString *) greeting
{

}

+ (void) sayHelloWorld
{

}
@end
```

6、方法的声明与消息发送

方法声明包括方法类型标识符，返回值类型，一个或多个方法标识关键字，参数类型和名信息。



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。

消息被中括号([和])包括。中括号中间，接收消息的对象在左边，消息（包括消息需要的任何参数）在右边。例如，给myArray变量传递消息insertObject:atIndex:消息，你需要使用如下的语法：

```
[myArray insertObject:anObj atIndex:0];
```

为了避免声明过多的本地变量保存临时结果，Objective-C允许你使用嵌套消息。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。例如，你可以用任何获取这种值的消息来代替前面例子里面的任何变量。所以，如果你有另外一个对象叫做myAppObject拥有方法，可以访问数组对象，以及插入对象到一个数组，你可以把前面的例子写成如下的样子：

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

7、属性

可以使用@interface定义属性，属性是用来代替声明存取方法的便捷方式。属性不会在你的类声明中创建一个新的实例变量。他们仅仅是定义方法访问已有的实例变量的速记方式而已。暴露实例变量的类，可以使用属性记号代替getter和setter语法。

getter和setter方法由系统自动生成。如果想同时手动设置getter和setter方法需要使用@synthesize. @synthesize通知编译器在没有setter和getter方法时，由编译器生成这两个方法

```
@synthesize something=_something;

-(void) setSomething:(NSString *)something
{
    _something=something;

}


-(NSString *)something
{
    return _something;

}
```

注意setter和getter方法中的方法名和大小写

```objc
#import <Foundation/Foundation.h>

@interface helloworld : NSObject
@property (strong,nonatomic) NSString *something;
+ (void) sayHelloWorld;
- (void) sayHello:(NSString *) greeting;

@end
```

```objc
#import "helloworld.h"
@interface helloworld ()


@end


@implementation helloworld
@synthesize something=_something;

-(void) setSomething:(NSString *)something
{
    _something=something;

}


-(NSString *)something
{
    return _something;

}


- (void) sayHello:(NSString *) greeting
{

}

+ (void) sayHelloWorld
{

}
@end
```

也可以为getter和setter方法指定不同的名称

@property (nonatomic,getter=getsth) NSString *something

```objc
//
//  helloworld.h
//  lesson1
//
//  Created by ustcsse on 14-10-29.
//  Copyright (c) 2014年 sse. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface helloworld : NSObject
@property (strong,nonatomic) NSString *something;
+ (void) sayHelloWorld;
- (void) sayHello:(NSString *) greeting ;
- (void) saysomething;

@end
```

```objc
//
//  helloworld.m
//  lesson1
//
//  Created by ustcsse on 14-10-29.
//  Copyright (c) 2014年 sse. All rights reserved.
//

#import "helloworld.h"
@interface helloworld ()
{

}

@end

@implementation helloworld
@synthesize something=_something;

-(void) setSomething:(NSString *)something
{
    NSMutableString *mstr=[NSMutableString stringWithString:something];
    [mstr  appendString:@"😊"];

    _something=mstr;

}

-(NSString *)something
{
    return _something;

}

- (void) sayHello:(NSString *) greeting
{
    NSString *str=@"hello ";
    NSMutableString *mstr=[NSMutableString stringWithString:str];
    [mstr appendString:greeting];
    NSLog(@"%@",mstr);
}

+ (void) sayHelloWorld
{
    NSLog(@"Hellow world");
}

- (void) saysomething
{
    NSString *str=@"hello ";
    NSMutableString *mstr=[NSMutableString stringWithString:str];
    [mstr appendString:self.something];
    NSLog(@"%@",mstr);
}

@end
```

7、属性

可使用“.”来存储属性。

```
[helloworld sayHelloWorld];给类发消息，只能使用类方法
helloworld *h=[[helloworld alloc] init];
h.something=@"IOS world";
[h sayHello:@"IOS"];
```

```objc
#import <UIKit/UIKit.h>
#import "AppDelegate.h"
#import "helloworld.h"

int main(int argc, char * argv[]) {
    @autoreleasepool {
        // return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
        [helloworld sayHelloWorld];
        helloworld *h=[[helloworld alloc] init];
        h.something=@"IOS world";
        [h sayHello:@"IOS"];
        [h saysomething];
        return 0;
    }
}
```

8、nil空对象

在Object C中可以发送消息给空对象nil。

9、self与super

- self调用自己方法，super调用父类方法
- self是类，super是预编译指令
- [self class]和[super class]输出是一样的

10、异常处理

Object-C语言的异常处理符号和C++、JAVA相似。可使用NSException，NSError或者自定义的类。异常处理机制是由这个四个关键字支持的：@try，@catch，@thorw，@finally。

捕获异常：

```
@try {
    <#statements#>
}
@catch (NSException *exception
) {
    <#handler#>
}
@finally {
    <#statements#>
}
```

例子：

```
helloworld  *h = [[helloworld alloc] init];

@try {
    [h saysomething];
    } @catch (NSException *exception)
    {
       NSLog(@"main: Caught %@: %@", [exception name],
[exception reason]);
} @finally {
     [h release];
}
```

抛出异常

@throw

例子

```
NSException *exception = [NSException  exceptionWithName:
@“helloworldException"  reason: @”illegal words”  userInfo:nil];

@throw exception;
```

四、课后练习

利用Object C 实现数值表达式的求值