ithero_zhou的专栏

个人资料

访问我的空间

忘了怎么哭

访问： 6557次
积分： 102
等级： BLOG 2
排名： 千里之外

原创： 4篇 转载： 0篇
译文： 0篇 评论： 1条

objective-c 关键字和概念

分类： Objective-C 2012-02-29 23:23 5167人阅读 评论(0) 收藏 举报

apple interface iphone setter getter object

目录(?) [+]

关键字

@

看到这个关键字，我们就应该想到，这是Object-C对C语言的扩展，例如@interface XXX。

@interface

声明类

@implementation

实现类

@protocol

声明协议

@optional

与@protocol配合使用，说明协议中的某个或者某几个方法可以不实现

@required

与@protocol配合使用，说明协议中的某个方法或者某几个方法必须实现

@end

与@interface ,@implementation,@protocol配合使用，代表声明或者实现结束

@encode

@encode为编译器宏，它可以将类型转换为相应的字符串。

id

id是指向Objective-C类对象的指针，它可以声明为任何类对象的指针，当在Objective-C中使用id时，编译器会假定你知道，id指向哪个类的对象。与void*是不同的是，void*编译器不知道也不假定指向任何类型的指针。

nil

定义为一个常量，如果一个指针的值为nil,代表这个指针没有指向任何对象。

self

在Objective-C中，关键字self与c++中this是同一概念，就是类对象自身的地址，通过self可以调用自己的实例变量和方法

Super

当子类需要调用父类的方法时，会用到Super关键字. Super指向的是父类的指针，子类重写父类的方法时，调用父类的方法是一个比较好的习惯。因为当我们不知道父类在该方法中实现的功能时，如果不调用父类的方法，有可能

* 【ShaderToy】开篇
* FFmpeg源代码简单分析：avio_open2()
* 技能树之旅: 从模块分离到测试
* Qt5官方demo解析集36——Wiggly Example
* Unity3d HDR和Bloom效果（高动态范围图像和泛光）
* Android的Google官方设计指南

最新评论

错误处理
foxshy: 不错

我们重写的方法会失去该功能，这是我们不愿意看到的情况。

NSNull

NSNull是没有的意思，如果一个字典的值为NSNull,那说明与该值对应的Key是没有值的，例如Key为address，说明与address对应的是值是没有。

self super class public protected private id

[self class] [super class] selector

objective-c runtime reference

标准用法

self = [super init]

new

1 Objective-C有一个特性，就是可以把类当成对象来发送消息，这种用法通常用于新建对像时，例如 XXX *object = [XXX new];

类方法 +

如果想声明属于类而不属于类对象的方法，用+。 + 用来修饰类的方法，使用+修饰的类方法，是整个类的方法，不属于哪一个类对象，这与C++中的static在类中使用的概念一样，

%@

在NSLog中，使用%@表示要调用对象的description方法。

概念

类

是一种结构，它表示对象的类型，就像int与 char 一样，也可以声明类的变量(对像)

实例化

为类的对象分配内存和初始化，达到可以使用该 类对象的目的。

对象(实例)

类的实例化后的产物

消息

在Object-C中，类的对象执行的操作，是通过给该类或者该类对象发送消息实现，如：［object func］；就是给object对象发送func消息，类似C++中的方法调用。给object对象发送func消息后，object对象查询所属类的func方法执行。

方法调度

当向一个对象发送消息时(调用方法)，这个方法是怎么被调用的呢？这就依赖于方法高度程序，方法调度程序查找的方法如下：

在本类的方法中，找被调用的方法，如果找到了，就调用，如果找不到被沿着继承路径去查找，从哪个类找到，就调用哪个类的方法，如果到最根上的类还是没有找到，那编译就会出错。

继承与复合

在Objective-C中支持继承，但只是支持单一继承(有且只有一个父类有)，如果想使用多继承的特性，可以使用分类和协议技术。

继承是is-a,复合是has-a。复合是通过包含指向对象的指针实现的，严格意义上讲，复合是针对于对象间来说，对于基本数据类型来说，它们被认为是对象的一部分。

装箱与拆箱

由于NSArray,NSDirectory等类不能直接存储基本数据类型，所以要想在NSArray＼NSDirectory中使用基本数据类型，就得使用装箱与拆箱。

在Objective-C中，可以使用NSNumber和NSValue来实现对数据类型的包装，NSNumber可以实现对基本数据类型的包装，NSValue可以实现对任意类型数据的包装。

将基本类型封装成对象叫装箱，从封装的对象中提取基本类型叫拆箱(取消装箱)，其它语言如Java原生支持装箱与拆箱，Ojbective-C不支持自动装箱与拆箱，如果需要得需要自己来实现装箱与拆箱。

存取方法

在使用类对象的实例变量(成员数据)时，不要直接使用对象中的实例，要使用存以方法来获取或者修改实例，既setter和getter,在Cocoa中，存取方法有命名习惯，我们得符合这种习惯，以便于与其它团队成员合作。setter方法是修改或者设置实例值，命名习惯为set+实例名，例有一个类有path实例变量，那setter命名为setPath,getter命名为Path,为什么不是getPath,因为get在Cocoa中有特殊的含义，这个含义就是带有get的方法就意味着这个方法通过形参指针(传入函数的参数指针)来返回值。我们要遵守这个命名习惯或者说规则。

在Objective-C 2.0中加入了@property和@synthesize来代替setter和getter，这两个关键字为编译器指令。 还有点表达式，存取类成员的值时，可以使用点表达式。

Object.attribute,当点表达式在=号左边时，调用的是setter方法，在=号右边时，调用的是getter方法。

@property 语法为:@property (参数) 类型 变量名.

在这里主要说明一下参数.

参数分为三种:

第一种:读写属性包括(readonly/readwrite/)

第二种:setter属性(assign,copy,retain),assign是简单的赋值，copy是释放旧成员变量，并新分配内存地址给成员变量，将传入参数内容复制一份，给成员变量。retain是将传 入　参数引用计数加1，然后将原有的成员变量释放，在将成员变量指向该传入参数。

第三种:与多线程有关(atomic,nonatomic).当使用多线程时，使用atomic,在不使用多线程时使用nonatomic

对象创建与初始化

在Objective-C中创建对象有两种方法　一种是[类 new];另一种是[[类 alloc] init],这两种方法是等价的，但按惯例来讲，使用[[类 alloc] init];

alloc操作是为对象分配内　　对象的数据成员都初始，int 为0，BOOL 为NO, float 为0.0等。

初始化，默认的初始化函数为init,init返回值为id,为什么回返回id呢，因为要实现链式表达式，在Objective-C中叫嵌套调用。

为什么要嵌套调用？？因为初始化方法init返回值可能与alloc返回的对象不是同一个？为什么会发生这种情况？基于类簇的初始化，因为init可以接受参数，在init内部有可能根据不同的参数来返回不同种类型的对象，所以最会发生上面说的情况。

在初始化时，建议使用if (self = [super init])

便利初始化

当一个类需要根据不同的情况来初始化数据成员时，就需要便利初始化函数，与init初始化不同的是，便利初始化函数有参数，参数个数可以有1到Ｎ个，Ｎ是类数据成员个数。

指定初始化函数：什么是指定初始化函数？在类中，某个初始化函数会被指定为指定的初始化函数，确定指定初始化函数的规则是初始化函数中，参数最多的为指定初始化函数，

其它未被指定为指定初始化函数的初始化函数要调用指定初始化函数来实现。对于该类的子类也是一样，只要重写或者直接使用父类的指定初始化函数。上述文字有些绕，来个例子吧

```
@interface A{
int x;
int y;
}

-(id) init;
-(id) initWithX:(int) xValue;
-(id) initWithY:(int) yValue;
```

```
-(id) initWithXY:(int) xValue
      yVal:(int) yValue;
@end
```

这里initWithXY被确定为指定初始化函数。

```
-(id) initWithXY:(int) xValue
yVal:(int) yValue{
  if (self = [super init]){
x = xValue;
y = yValue;
}
return self;
}

-(id) init{
if (self = self initWithXY:10
yVal:20){
}
return self;
}
.......
@interface B: A{
int z;
}
-(jd) initWithXY......;
@end
@implementation B

-(id) initWithXY:(int) xValue
yVal:(int) yValue{
  if (self = [super initWithXY:10
yVal=20]){
z= 40;
}
return self;
}
@end
```

自动释放池

内存管理是软件代码中的重中之重，内存管理的好坏，直接影响着软件的稳定性。在Cocoa中，有自动释放池，这类似于C++中的智能指针。

NSObject有一个方法是autorelease，当一个对象调用这个方法时，就会将这个对象放入到自动释放池中。

drain,该方法是清空自动释放池，不是销毁它。drain方法只适用于Mac OS X 10.4以上的版本，在我们写的代码中要使用release，release适用于所有版本。

自动释放池是以栈的方式实现，当创建一个自动释放池A时，A被压入栈顶，这时将接入autorelease消息的对象放入A自动释放池，这时创建一个新的B自动释放池，B被压入栈顶，创建完成后删除B,这个接收autorelease消息的对象依然存在，因为A自动释放池依然存在。

引用计数

每个对象都有一个与之相应的整数，称它为引用计数，当该 引用计数为0时，Objective-C自动向该对象发送dealloc,以销毁该对向，与该引用计数相关的方法(消息)有下面几个

1 增加引用计数：通过alloc,new,copy创建一个对象时，该对象的引用计数加1(其实就是1，因为之前为0)

2 增加引用计数: retain

3 减少引用计数: release

局部分配内存(临时对象):

1 如果使用alloc,new,copy创建对象，则需要主动调用对象的release方法

2 如果使用非alloc,new,copy创建对象，我们认为该 对象引用计数为1，并已经加入了自动释放池，我们不需要主动的调用对象的release方法。

拥有对象(在类中以成员的方法存在):

1 如果使用alloc,new,copy创建对象，则需要在dealloc方法中，释放该对象

2 如果使用非alloc,new,copy创建对象，则在拥有该对象时，保留该对象(执行retain方法)，在dealloc方法中，释放该对象。

dealloc

当对象的引用计数为0时，Objective-C会自动发送对象的dealloc消息(自动调用对象的dealloc方法，类似于C++的析构函数)，所以我们可以自己重写dealloc方法，来实现类里的对其它使用资源的释放工作。

注意：不要直接在代码中显示调用dealloc方法。

垃圾回收

在Objective-C 2.0中引入了垃圾回收机制(自动管理内存)，在工程设置里设置Objective-C Garbage Collection为Required[-fobjc-gc-only]就可以使用垃圾回收机制。

启用垃圾回收机制后，通常的内存管理命令都变成了空操作指令，不执行任何操作。

Objective-C的垃圾回收机制是一种继承性的垃圾回收器，垃圾回收器定期检查变量和对象以及他们之间的指针，当发现没有任何变量指向对象时，就将该对象视为被丢弃的垃圾。所以在不在使用一个对象时，将指针他的指针设置为nil,这时垃圾回收器就会清理该对象。

注意：如果开发iPhone软件，则不能使用垃圾回收。在编写iPhone软件时，Apple公司建议不要在自己的代码中使用autorelease方法，并且不要使用创建自动释放对象的函数。

类别

什么是类别？类别是一种为现有类添加新方法的方式。

为什么使用类别或者说使用类别的目的是什么？有以下三点：

第一，可以将类的实现分散到多个不同的文件或多个不同的框架中。

如果一个类需要实现很多个方法，我们可以将方法分类，把分好的类形成类别，可以有效的管理和驾驭代码。

第二，创建对私有方法的前向引用。

第三，向对象添加非正式协议。

委托

委托的意思就是你自己想做某事，你自己不做，你委托给别人做。

在Ojbective-C中，实现委托是通过类别(或非正式协议)或者协议来实现。

举个例子：Apple要生产iPhone,Apple自己不生产(种种原因，其中之一就是在中国生产成本低，他们赚的银子多),Apple委托富士康来生产,本来富士康原来不生产iPhone,现在要生产了,所以他得自己加一个生产iPhone的生产线(类别，增加生产iPhone方法)，这就是通过类别来实现委托。下面用代码来说明这个例子。

```
.....
Apple *apple = [[Apple alloc ] init];
Foxconn *fox = [[Foxconn alloc] init];
[apple setDelegate:fox];
[apple produceIPhone];
........

@implementation Apple
-(...) setDelegate:(id) x{
delegate  = x; //! 将委托的生产对象指定为x
}

-(...) produceIPhone{
[delegate produceIPhone]; //! 委托对象生产iPhone
}

@interface Foxconn : NSObject
```

```
...
@end
@interface NSObject(ProduceIPhone) //! Foxconn之前就可以生产其它产品，有过声明和定义
-(...) produceIPhone  //! 增加生产iPhone能力
@end

@implementation NSObject(ProduceIPhone)
//! 生产iPhone
-(...) produceIPhone{
......
}
@end
```

非正式协议

创建一个NSObject的类别， 称为创建一个非正式协议。为什么叫非正式协议呢?
也就是说可以实现，也可以不实现被委托的任务。
拿上面的例子来说，Apple要求Foxconn除了能生产iPhone外，还有一个要求是在一定时间内完成.由于双方没有签合同，所以时间要求和生产要求规格都是非正式协议

选择器

选择器就是一个方法的名称。选择器是在Objective-C运行时使用的编码方式，以实现快速查找。可以使用@selector预编译指令，获取选择器@selector(方法名)。NSObject提供了一个方法respondsToSelector:的方法，来访问对象是否有该方法(响应该消息)。
拿上面的Apple请Foxconn生产iPhone为例，Apple怎么知道Foxconn有没有生产iPhone的能力呢?Apple就通过respondsToSelector方法询问Foxconn，是否可以生产iPhone(是否可以响应produceIPhone)，询问结果是可以，那Apple就委托Foxconn生产，Foxconn就生产出来了人们比较喜欢的iPhone产品。

正式协议

与非正式协议比较而言，在Ojbective-C中，正式协议规定的所有方法必须实现。在Ojbective-C2.0中，Apple又增加了两个关键字，协议中的方法也可以不完全实现，是哪个关键字见关键字部份的@optional,@required。
正式协议声明如下:

```
@protocol XXX
-(...) func1;
-(...) func2;
@end
```

使用协议:

```
@interface Object : NSObject<XXX> //! Object从NSObject派生，并遵循XXX协议，要实现func1,func2函数。
...
@end
```

习惯用法

分配内存和初始化

```
self = [super init];
```

对象间交互

在Objective-C中，所有对象间的交互都是通过指针实现。

快速枚举

```
for (Type *p in array)
```

注意:

Objective-C不支持多继承

上一篇 Unicode

主题推荐 objective-c 内存管理 智能指针 编译器 多线程

猜你在找

准备好了么？跳吧！

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC
coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap