# 点语法

## 利用点语法替换set方法和get方法

* 方法调用

Student \*stu = [Student new];

[stu setAge:100];

int age = [stu age];

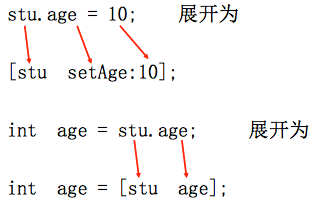
* 点语法

stu.age = 100;

int age = stu.age;

## 点语法的本质

* 其实点语法的本质还是方法调用
* 当使用点语法时，编译器会自动展开成相应的方法



## 死循环注意

- (void) setAge:(int)age {

// 下面的代码会引发死循环

self.age = age;

}

- (int) age {

// 下面的代码会引发死循环

return self.age;

}

# 成员变量的作用域

## 基本概念

局部变量、全局变量都有自己的作用域，成员变量也不例外

## 类型

* @private：只能在当前类的对象方法中直接访问
* @protected：可以在当前类以及子类的对象方法中直接访问
* @public：任何地方都可以直接访问
* @package：同一个“体系内”（框架）可以访问，介于@private和@public之间

## 继承补充

* 专业术语
* 父类\超类 superclass
* 子类 subclass\subclasses
* 单继承

## @implementation补充

没有@interface，只有@implementation，也可以开发一个类

# @property和@synthesize

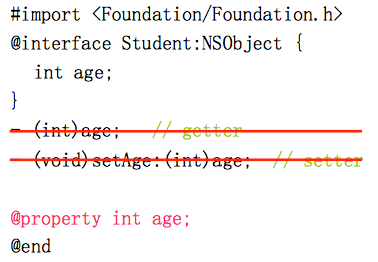
## @property

* 用在@inteface中
* 用来自动生成setter和getter的声明
* 用@property int age;就可以代替下面的两行

- (int)age; // getter

- (void)setAge:(int)age; // setter

* 示例



## @synthesize

* 用在@implementation中
* 用来自动生成setter和getter的实现
* 用@synthesize age = \_age;就可以代替

- (int)age{

return \_age;

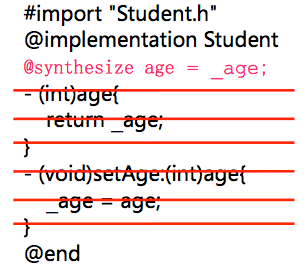
}

- (void)setAge:(int)age{

\_age = age;

}

* 示例



## @synthesize的细节

* @synthesize age = \_age;
* setter和getter实现中会访问成员变量\_age
* 如果成员变量\_age不存在，就会自动生成一个@private的成员变量\_age
* @synthesize age;
* setter和getter实现中会访问成员变量age
* 如果成员变量age不存在，就会自动生成一个@private的成员变量age
* 手动实现
* 若手动实现了setter方法，编译器就只会自动生成getter方法
* 若手动实现了getter方法，编译器就只会自动生成setter方法
* 若同时手动实现了setter和getter方法，编译器就不会自动生成不存在的成员变量

## @property新特性

* 自从Xcode 4.x后，@property就独揽了@synthesize的功能。也就是说，@property可以同时生成setter和getter的声明和实现
* 默认情况下，setter和getter方法中的实现，会去访问下划线 \_ 开头的成员变量

# id

## 简介

* 万能指针，能指向任何OC对象，相当于NSObject \*
* id类型的定义

typedef struct objc\_object {

Class isa;

} \*id;

## 使用

// 注意：id后面不要加上\*

id p = [Person new];

## 局限性

调用一个不存在的方法，编译器会马上报错

# 构造方法

## 对象创建原理

* new的拆分两部曲
* 分配内存（+alloc）
* 初始化（-init）

Person \*p1 = [Person alloc];

Person \*p1 = [p1 init];

合成一句后：

Person \*p = [[Person alloc] init];

## init方法的重写

* 想在对象创建完毕后，成员变量马上就有一些默认的值
* init方法的重写过程

- (id)init

{

if (self = [super init])

{

\_age = 10;

}

return self;

}

## 自定义构造方法

* 构造方法的一些规范
* 返回值是id类型
* 方法名都以init开头

- (id)initWithAge:(int)age {

if (self = [super init]) {

\_age = age;

}

return self;

}

* 传递多个参数进行初始化

- (id) initWithAge:(int)age andNo:(int)no;

# .h和.m文件的抽取

* 每个类分布在不同文件中
* 类的声明放在.h文件，类的实现放在.m文件
* 若想使用某个类，就包含某个类的.h声明文件