ByteDance 字节跳动

20秋

MOSAD
现代操作系统应用开发

#2 Objective-C 基础

Content

- What's Objective-C
- Why Objective-C
- Objective-C Basic concepts
- Objective-C in action

What's Objective-C

- The primary programming language for OS X and iOS.

  Superset of the standard ANSI C

  Object-oriented (OO) capabilities

  A dynamic runtime
- Adds language-level support for object graph management and object literals while providing dynamic typing and binding, deferring many responsibilities until runtime.
- Compiler: gcc, clang

Historical note

- Created at the Stepstone company in the early 1980s by Brad Cox and Tom Love.
- Licensed by NeXT Computer Inc. in the late 1980s to develop the NeXTStep frameworks that preceded Cocoa.
- NeXT extended the language in several ways, e.g. with the addition of protocols.

Why Objective-C

- An OO language
- An extension of standard ANSI C

  Reuse existing C programs,

  free to use OO or procedural programming techniques.
- A fundamentally simple and well-organized OO language.

  much less difficult to become a proficient OO programmer.
- Compare to other OO languages, OC is very dynamic, compiler preserves a great deal of information about the objects themselves for use at runtime.

  Support an open style of dynamic binding

  Enables the construction of sophisticated development tools.

New language to learn!

- Strict superset of C
- Adds syntax for classes, methods, etc.
- A few things to "think differently" about

  Properties?

  Dynamic binding?

  ...

At a Glance

- An App Is Built from a Network of Objects

  Defining Classes, Working with Objects, Encapsulating Data
- Categories Extend Existing Classes

  Customizing Existing Classes
- Protocols Define Messaging Contracts
- Values and Collections Are Often Represented as Objective-C Objects
- Blocks Simplify Common Tasks
- Error Objects Are Used for Runtime Problems

  Dealing with Errors
- Objective-C Code Follows Established Conventions
- **References**

  https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/

Basic concept: Properties in a Class

- Use "properties" to access instance variables
- The combination of a getter method and a setter method

  The getter has the name of the property (e.g. "myValue")

  The setter's name is "set" plus capitalized property name (e.g. "setMyValue:")
- We always use a lowercase letter as the first letter of a property name.
- We just call the setter to store the value we want and the getter to get it. Simple!

Defining Classes

Student.h

```
#import "Person.h"

@interface Student : Person

@end
```

导入父类的头文件
常常是<UIKit/UIKit.h>

类名

父类

Student.m

```
#import "Student.h"

@implementation Student

@end
```

我们自己的头文件

这里不需要指定父类

Public or private

Student.h

```
#import "Person.h"


@interface Student : Person
// declaration of public methods



@end
```

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods

@end

@implementation Student

// implementation of public and
// private methods



@end
```

()是必需的；这里也不需要指定父类

Defining Methods

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods
```

函数全名是 learnUnit:ofLang:

```
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;
```

冒号对齐，漂亮

2个参数是 unit 和 aLang.

```
@end
```

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods

@end

@implementation Student

// implementation of public and
// private methods




-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
    // put the code here
}
@end
```

Getter and Setter

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods




-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;

-(void)setCredit:(double)credit;
-(double)credit;




@end
```

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods

@end

@implementation Student

// implementation of public and
// private methods

-(void)setCredit:(double)credit
{
      ???
}
-(double)credit
{
      ???
}
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
      // put the code here
}
@end
```

@property

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;


-(void)setCredit:(double)credit;
-(double)credit;




@end
```

@property自动声明下面的2个credit函数

nonatomic非线程安全.

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods

@end

@implementation Student

// implementation of public and
// private methods

-(void)setCredit:(double)credit
{
      ???
}
-(double)credit
{
      ???
}
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
      // put the code here
}
@end
```

@property only

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;



@end
```

不需要在头文件重复声明getter和setter函数，
使用@property足矣

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods

@end

@implementation Student

// implementation of public and
// private methods

-(void)setCredit:(double)credit
{
      ???
}
-(double)credit
{
      ???
}
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
      // put the code here
}
@end
```

@synthesize

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;



@end
```

大部分情况下，使用@synthesize即可实现@property的getter和setter函数，包括其值的存储变量。

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods

@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;
-(void)setCredit:(double)credit
{
      ???
}
-(double)credit
{
      ???
}
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
      // put the code here
}
@end
```

@property的存储变量名，约定：加下划线前缀

@synthesize

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;



@end
```

@synthesize实现的getter和setter函数是这样的.

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods

@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;
-(void)setCredit:(double)credit
{
      _credit = credit;
}
-(double)credit
{
      return _credit;
}
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
      // put the code here
}
@end
```

@synthesize

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;



@end
```

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods

@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
     // put the code here
}
@end
```

大部分情况下，让@synthesize实现getter和setter函数即可。

@synthesize

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;



@end
```

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods

@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;
-(void)setCredit:(double)credit
{
      if(credit>0) _credit = credit;
}
```

除非有特殊的需求，可自定义getter或setter函数。

```
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
      // put the code here
}
@end
```

Private @property

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;



@end
```

私有@property的声明.

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods
@property (nonatomic,strong) Progress *curProgress
@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;
-(void)setCredit:(double)credit
{
      if(credit>0) _credit = credit;
}




-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
      // put the code here
}
@end
```

Object @property

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;
```

这是一个对象的指针，指向Progress类的实例.
strong是强引用，需要时该对象的内存不会被释放.
weak是弱引用，随时可以被释放.

```
@end
```

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods
@property (nonatomic,strong) Progress *curProgress
@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;
-(void)setCredit:(double)credit
{
      if(credit>0) _credit = credit;
}
```

所有对象均在堆(heap)中分配，通过指针访问.

```
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
      // put the code here
}
@end
```

Object @property

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;


@end
```

@synthesize为新属性创建setter、getter.
注意，只分配指针内存，不分配对象内存.
对象初始化稍后讲.

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods
@property (nonatomic,strong) Progress *curProgress
@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;
@synthesize curProgress = _curProgress;
-(void)setCredit:(double)credit
{
      if(credit>0) _credit = credit;
}




-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
      // put the code here
}
@end
```

Objective-C

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;



@end
```

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods
@property (nonatomic,strong) Progress *curProgress
@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;
@synthesize curProgress = _curProgress;
-(void)setCredit:(double)credit
{
      if(credit>0) _credit = credit;
}
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
      // put the code here



}
@end
```

开始写样例代码，
体会一下OC语法

Objective-C messages

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;



@end
```

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods
@property (nonatomic,strong) Progress *curProgress
@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;
@synthesize curProgress = _curProgress;
-(void)setCredit:(double)credit
{
      if(credit>0) _credit = credit;
}
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
      // put the code here
      double credit = [self credit];



}
@end
```

中括号[ ]语法，用来发消息。
这里是调用自己实例的credit属性的getter函数。

Objective-C messages

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;



@end
```

中括号[ ]可嵌套。
包含2个参数的函数updateLang:toUnit:的调用消息将发送给Progress类的一个实例。

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods
@property (nonatomic,strong) Progress *curProgress
@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;
@synthesize curProgress = _curProgress;
-(void)setCredit:(double)credit
{
    if(credit>0) _credit = credit;
}
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
    // put the code here
    double credit = [self credit];
    [[self curProgress] updateLang:aLang
                            toUnit:unit];
}
@end
```

Objective-C dot notation

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;


@end
```

调用getter和setter太重要，还有一种语法：
dot notation 点表示法。
与中括号表示法等价.

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods
@property (nonatomic,strong) Progress *curProgress
@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;
@synthesize curProgress = _curProgress;
-(void)setCredit:(double)credit
{
    if(credit>0) _credit = credit;
}
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
    // put the code here
    double credit = self.credit;
    [[self curProgress] updateLang:aLang
                           toUnit:unit];
}
@end
```

Objective-C dot notation

Student.h

```
#import "Person.h"
#import "Language.h"

@interface Student : Person
// declaration of public methods


@property(nonatomic) double credit;


-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang;

@end
```

Student.m

```
#import "Student.h"

@interface Student()
// declaration of private methods
@property (nonatomic,strong) Progress *curProgress
@end

@implementation Student

// implementation of public and
// private methods
@synthesize credit = _credit;
@synthesize curProgress = _curProgress;
-(void)setCredit:(double)credit
{
    if(credit>0) _credit = credit;
}
-(void)learnUnit:(int)unit
          ofLang:(Language *)aLang
{
    // put the code here
    double credit = self.credit;
    [self.curProgress updateLang:aLang
                          toUnit:unit];
}
@end
```

这里也可以用dot notation.

Hands On Lab

- Try yourself to define 2 classes: Person, Student
- With some @property's and testing getter and setter.
- Use NSLog to print out the results.

Have a rest

- You've learned how to

  Define a class's public @interface and private @implementation in a .h and .m file respectively

  Add a private @interface to .m file

  Create a @property, both for a primitive type (like double) and a pointer

  Use nonatomic in @property declarations

  Use strong or weak in @property declarations of pointers to objects

  Use @synthesize to create a @property's setter and getter and backing instance variable

  Use "= _propertyName" to choose the name @synthesize uses for its backing instance variable

  …

More on Properties

- Why properties?

  provides safety and subclassability for instance variables.

  lazy instantiation, UI updating, consistency checking (e.g. credit>0), etc.

- Instance Variables

  It is not required to have an instance variable backing up a @property (just skip @synthesize).

  Some @propertys might be "calculated" (usually readonly) rather than stored.

  It is possible to have instance variables without a @property.

- Why dot notation?

  Pretty.

  Makes access to @propertys stand out from normal method calls.

  Synergy with the syntax for C structs (i.e., the contents of C structs are accessed with dots too).

Dot Notation

```
// @property access looks just like C
struct member access
typedef struct {
    float x;
    float y;
}CGPoint;
```

C struct命名很像class，
只是不能发消息给它
(没有方法可被调用)

Dot Notation

```
// @property access looks just like C
struct member access
typedef struct {
    float x;
    float y;
}CGPoint;
……
@property CGPoint position;
@property CGPoint center;
……
```

```
……
@synthesize position, center;
……
if(self.position.x > self.center.x)
    // on the left side
else
    // on the right side
```

Dot notation访问
对象实例的@property

Dot notation访问
标准C结构体的成员变量

strong vs. weak

- strong "keep this in the heap until I don't point to it anymore"

  I won't point to it anymore if I set my pointer to it to nil.

  Or if I myself am removed from the heap because no one strongly points to me!

- weak "keep this as long as someone else points to it strongly"

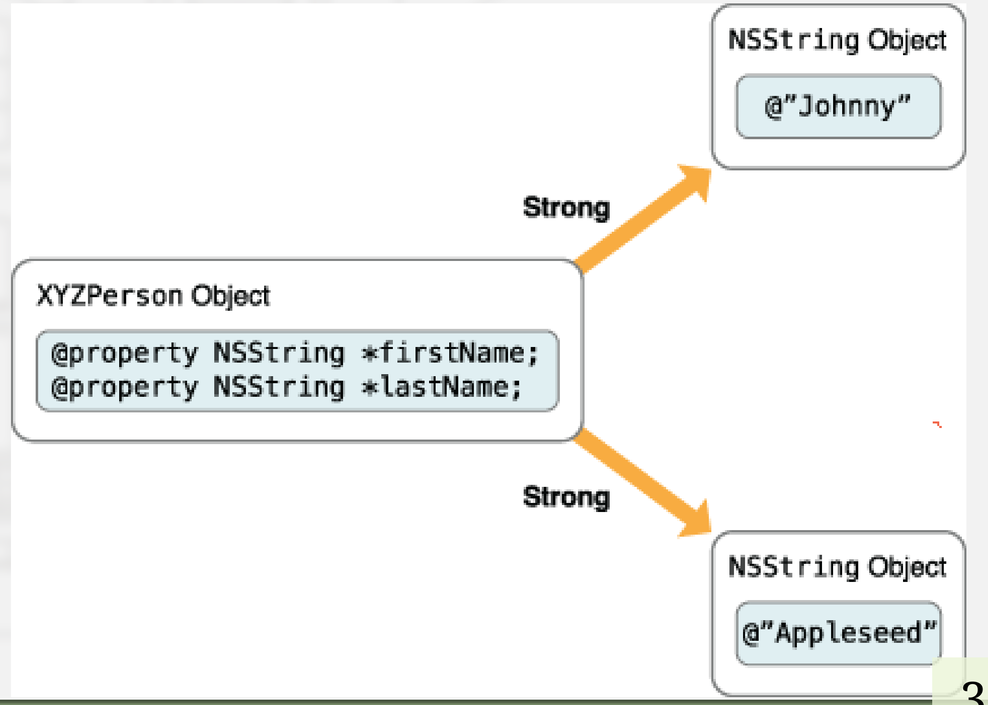
  If it gets thrown out of the heap, set my pointer to it to nil automatically.

- This is not garbage collection but better!

  It's reference counting done automatically for you.

- References

  阅读： Manage the Object Graph through Ownership and Responsibility



nil

- The value of an object pointer that does not point to anything

  NSString *hello = nil;

- Like "zero" for a primitive type (int, double, etc.)

  Actually, it's not "like" zero: it is zero.

- All instance variables start out set to zero

  Thus, instance variables that are pointers to objects start out with the value of nil.

- Can be implicitly tested in an if statement

  if (obj) { } // curly braces will execute if obj points to an object

- Sending messages to nil is (mostly) okay. No code gets executed.

  If the method returns a value, it will return zero.

  int i = [obj methodWhichReturnsAnInt]; // i will be zero if obj is nil

  Be careful if the method returns a C struct. Return value is undefined.

  CGPoint p = [obj getLocation]; // p will have an undefined value if obj is nil

id

- id is a type
- id means "pointer to an object of any class"
- id is already a pointer

  So "id *" would be a pointer to a pointer.
- id does not mean "object of any class".
- Example

```
id obj = nil; //obj是一个可指向任意对象的指针，初始指向nil
```

BOOL

- OC's Boolean "type" (just a typedef)

  Can be tested implicitly

  if(flag) {}

  if(!flag) {}

  YES means "true", NO means "false"

  NO == 0, YES is anything else

Instance vs. Class Methods

Instance method

```
// starts with a dash
-(BOOL) dropBomb:(Bomb *)bomb
              at:(CGPoint)position
            from:(double)altitude;
```

Bomb类对象参数，
传递指针（堆）

C struct参数，
传递值（栈）

Class method

```
// starts with a plus sign
+(Language*)motherLanguage;
+(NSString*)stringWithFormat:…
```

Instance vs. Class Methods

Instance method

```
// starts with a dash
-(BOOL) dropBomb:(Bomb *)bomb
              at:(CGPoint)position
            from:(double)altitude;
```

顾名思义，就是实例的方法

```
// Calling syntax
[<pointer to instance> method]
Ship *ship = … //ship是类Ship的一个实例
ret = [ship dropBomb:…]
```

Class method

```
// starts with a plus sign
+(Language*)motherLanguage;
+(NSString*)stringWithFormat:…
```

类方法则是创建对象或公共工具的方法

```
// Calling syntax
[Class method]
NSString *str=
  [NSString stringWithFormat:@"%g", ret];
[[ship class] doSomething];
```

ship是实例.
"class"是实例方法，返回一个类.
doSomething则是类方法.

Instantiation

- Asking other objects to create objects for you

  NSString类： -(NSString *)stringByAppendingString:(NSString *)otherString;

  NSString类、NSArray类： -(id) mutableCopy;

- Not all objects handed out by other objects are newly created

  NSArray类: -(id)lastObject;

  -(id)objectAtIndex:(int)index;

  Unless the method has the word "copy" in it, if the object already exists, you get a pointer to it.

- Using class methods to create objects

  NSString类： + (id)stringWithFormat:(NSString *)format, ...

  UIButton类： + (id)buttonWithType:(UIButtonType)buttonType;

  NSMutableArray类： + (id)arrayWithCapacity:(int)count;

  NSArray类： + (id)arrayWithObject:(id)anObject;

Instantiation

- Allocating and initializing an object from scratch

  a two step process: allocation, then initialization.

  ```
  NSMutableArray *stack = [[NSMutableArray alloc] init];
  CalculatorBrain *brain = [[CalculatorBrain alloc] init];
  ```

- Allocating

  Heap allocation for a new object is done by the NSObject class method +(id)alloc

  It allocates enough space for all the instance variables (e.g., the ones created by @synthesize).

- Initializing

  Classes can have multiple, different initializers (with arguments) in addition to plain init.

  If a class can't be fully initialized by plain init, it is supposed to raise an exception in init.

  NSObject's only initializer is init.

Instantiation

- If an initialization method has arguments, it should still start with the four letters init.
- Examples of multiple initializers with different arguments from NSString:

  -(id)initWithCharacters:(const unichar *)characters length:(int)length;

  -(id)initWithFormat:(NSString *)format, ...;

  -(id)initWithData:(NSData *)data encoding:(NSStringEncoding)encoding;
- Classes must designate an initializer for subclassers

  This is the initializer that subclasses must use to initialize themselves in their designated initializer.
- Static typing of initializers

  For subclassing reasons, init methods should be typed to return id (not statically typed)

  Callers should statically type though, e.g., MyObject *obj = [[MyObject alloc] init];

Instantiation

- Creating your own initialization method

  We use a sort of odd-looking construct to ensure that our superclass inited properly.

  Our superclass's designated initializer can return nil if it failed to initialize. In that case, our subclass should return nil as well.

  This looks weird because it assigns a value to self, but it's the proper form.

- Here's an example of what it would look like if init (plain) were our designated initializer:

```
@implementation MyObject
-(id)init{
   self = [super init]; //调用父类指定的初始化构造函数
   if(self) {
      //初始化构造子类
   }
   return self;
}
@end
```

Summary

- 顺利完成第2~~劫~~节！
- 学习重点是OC与C/C++不一样的地方，建议完整阅读官方文档，系统地学习一遍，其实很多内容似曾相识。
- 下周继续讨论更难的部分
- 留意课程网站，完成第一次（简单）必做作业，最迟在第5周实验课完成验收。