Objective-C

Курс лекций и семинаров для студентов,
желающих научиться программировать под iPhone

Осень-Зима 2013

Лекция №2

Objective-C: История языка. Особенности. Использование.

Автор: Дмитрий Волков, iPhone Developer, Sibers

Sibers®

Что вы узнаете сегодня?

1. История создания языка Objective-C
2. Современное состояние языка
3. Особенности языка
4. Синтаксис
5. Управление памятью

История создания

- Основан на SmallTalk

  Smalltalk был создан в кузне IT-технологий — Xerox PARC
- Создатели – Brad Cox и Tom Love. Создан в начале 80- годов
- Является надмножеством языка C — любая программа на C так же скомпилируется и выполнится в Objective-C

  Это позволяло программистам, разрабатывающим на C, быстрее освоить новый язык.

  Кардинальные отличные синтаксиса Smalltalk-80 затрудняли его интеграцию.

История создания

- Первая версия — 1981 год. Object Oriented Pre-Processor (Был основан на стандартных компонентах UNIX — awk, sed, компиляторы C)
- В 1986 году Brad Cox выпускает полное описание языка, а так же книгу «*Object-Oriented Programming, An Evolutionary Approach*»
- Сердце языка — Objective C Runtime Library.
- Основной идеей было создание языка, позволяющего переиспользовать уже имеющиеся компоненты, и на их основе строить большие, расширяемые системы (в преддверии набирающего обороты Объектно-Ориентированного Подхода к созданию программ).

История создания

- В 1988 году NeXT лицензирует язык Objective C и добавляет его поддержку в компилятор GCC.
- В 1993 году создается спецификация OpenStep API — совместная работа NeXT и Sun Microsystems по созданию портируемой версии ОС NextStep.
- Созданы основные переиспользуемые компоненты системы, написанные на Objective C — Foundation Kit и Application Kit.
- В 1996 году Apple покупает компанию NeXT и на основе ее наработок создает Mac OS X.

Современное состояние языка

- На данный момент существует две основные ветки развития:

  GNUStep — open-source реализация runtime библиотеки, а так же собственный стек компонентов.

  Apple — open-source реализация библиотеки, распространяющаяся под лицензией APSL. На данный момент — самая развитая ветка.
- Objective-C — 4й по популярности язык в индексе TIOBE.
- Основной язык разработки для Mac OS X / iOS. Основные фреймворки Apple так же написаны на Objective-C.

Современное состояние языка

На данный момент существует спецификация языка “Objective-C 2.0”, выпущенная компанией Apple в 2006 году:

- Поддержка сборки мусора
- Поддержка C++
- Fast enumeration (for … in)
- Улучшения runtime библиотеки

Так же, Apple вносит нестандартизированные расширения в язык C, что так же позволяет их использовать в Objective-C:

- Блоки (они же лямбы, они же замыкания)
- Модули компиляции

Особенности языка

- «Динамический» компилируемый язык
- Широкие возможности для получения информации об объектах во время исполнения программы
- Все, кроме нативных типов(void*, int, float, …) считается объектом
- Стандартный набор Объектно-Ориентированных возможностей — наследование, инкапсуляция, полиморфизм
- Отсутствие раннего связывания — какой именно метод выполнится у объекта определяется во время исполнения
- Объекты — слабо типизированы

Особенности языка

- Другая семантика вызова методов у объектов:

Объектам посылаются сообщения, на которые они могут ответить, а могут и не ответить. Все решается во время исполнения программы.

- Единый обобщенный тип, описывающий объект – **id**
- Сообщения, посылаемые объектам называются ***селектор*** и имеют свой тип — **SEL**
- Так как, грубо говоря, селектор является строкой, в языке *отсутствует* возможность *перегрузки* методов.
- В языке отсутствуют пространства имен (namespace в C++)
- Отсутствие множественного наследования (реализуется посредством протоколов)

Синтаксис

Smalltalk:

```
|object|
object doStuff
object doStuffWithNumber: 5
```

Objective C:

```
id object;
[object doStuff];
[object doStuffWithNumber: 5];
```

C++:

```
Object object;
object.doStuff();
object.doStuffWithNumber(5);
```

Синтаксис

Objective C:

```
id object;
[object doStuff];
[object doStuffWithNumber: 5];
```

Object – объект-получатель сообщения

doStuff – сообщение, посылаемое объекту (метод, вызываемый у объекта)

doStuffWithNumber: 5 – параметр, передаваемый в метод.

Синтаксис: Соглашение о наименованиях

При написании кода на Objective-C, Apple использует следующие соглашения о наименованиях:

- Имена *классов* начинаются с большой буквы. Каждое слово в имени класса так же начинается с большой буквы.

  Object. ***MutableObject***.

- Рекомендуется использовать префиксы для классов, которые создаете вы.

  AUCAObject. ***AUCAMuableObject***.

- Такой стиль именования называется ***CamelCase***.

Синтаксис: Соглашение о наименованиях

При написании кода на Objective-C, Apple использует следующие соглашения о наименованиях:

- Имена *внутренних переменных* классов начинаются с символа _ (нижнее подчеркивание) и маленькой буквы. Все последующие слова — с большой.

  _variable

  _instanceVariable

- Имена *методов* начинаются с маленькой буквы, все последующие слова — с большой.

  (void) method

  (void) doStuff

Синтаксис: Соглашение о наименованиях

При написании кода на Objective C, Apple использует следующие соглашения о наименованиях:

Методы, в которые передаются аргументы, так же начинаются с маленькой буквы и каждое слово с большой.

Но слово, описывающее каждый следующий аргумент, так же начинается с маленькой буквы:

(void) addNumer:(NSInteger)

firstNumber

toNumber:(NSInteger)

secondNumber

Синтаксис

Объявление класса: (Tool.h)

```
@interface Tool: NSObject //базовый наследуемый класс
{
    id _innerObject;//внутренние переменные класса
    NSInteger innerInteger;
}
@property(nonatomic, retain) id objectProperty;//объявление свойства
@property(nonatomic) NSInteger integerProperty;
+ (void) classMethod; //метод класса
- (void) instanceMethod; //метод объекта
@end
```

Синтаксис

Реализация класса: (Tool.m)

```
@implementation Tool
+ (void) classMethod
{
    //do class stuff
}
- (void) instanceMethod
{
    //do instance stuff
}
@end
```

Синтаксис

Создание объекта, вызов методов:

#import “Object.h”

//Объекты никогда не создаются на стеке. Для любого объекта выделяется память в куче

```
Tool* object = [[Tool alloc] init];
[Tool classMethod];
[object instanceMethod];
object.objectProperty = @"String";
object.integerProperty = 15;
```

Синтаксис: Протоколы

Протокол – аналог *интерфейсов* Java / *абстрактных классов* C++.

Объявление протокола (ObjectProtocol.h)

```
@protocol ToolProtocol <NSObject> //протоколы могут наследовать
любое количество других протоколов
@optional
- (void) optionalProtocolMethod;

@required
- (NSInteger) countOfObjects;
@end
```

Синтаксис: Протоколы

Реализация протокола:

```
@interface Tool: NSObject<ToolProtocol>//объекты могут реализовывать любое количество протоколов
{
    //…
}
@end

@implementation Object
- (void) optionalProtocolMethod {};
- (NSInteger) countOfObjects { return 10;};

@end
```

Синтаксис: Протоколы

Вызов метода, наследованного из протокола

#import “Object.h”

#import “ObjectProtocol.h”

```
id<ObjectProtocol> object = [[Tool alloc] init];
NSInteger count = [object countOfObjects];
if([object respondsToSelector:@selector(optionalProtocolMethod)])
{
    [object optionalProtocolMethod];
}
```

Синтаксис: Категории

- В Objective-C существует механизм, позволяющий расширять функциональность класса, не используя наследование — **Категории.**

- Категории позволяют добавлять к классу новые методы, либо переопределять старые.

- В методах, описанных в категории есть полный доступ к внутренним переменным класса и его методам.

- В категориях нельзя добавлять новые переменные к классу.

Синтаксис: Категории

Объявление категории (UIFont+ArialFont.h):

```
@interface UIFont (ArialFont)

+ (UIFont*) arialFontWithSize:(CGFloat) size;

@end
```

Синтаксис: Категории

Реализация категории (UIFont+ArialFont.m):

```
@implementation UIFont (ArialFont)

+ (UIFont*) arialFontWithSize:(CGFloat)size
{
    return [UIFont fontWithName:@"Arial" size:size];
}
//не делайте так
+ (UIFont *)systemFontOfSize:(CGFloat)fontSize
{
    return [self arialFontWithSize:fontSize];
}

@end
```

Синтаксис: Свойства

Свойства = геттер / сеттер для внутренней переменной класса или изменения его внутреннего состояния

То же, что и в C++:

```
class Tool
{
private:
    int m_count;
public:
    int getCount() {return m_count;};
    void setCount(int count) {m_count = count;};
};
```

Синтаксис: Свойства

Объявление:

```
@interface Tool

@property(retain) id retainedProperty; //aka strong
@property(copy) id copiedProperty;
@property(assign) id assignedProperty; //aka weak

@end
```

Синтаксис: Свойства

Объявление:

```
@interface Tool

@property(nonatomic, retain) id nonatomicProperty;
@property(nonatomic) NSInteger nonatomicIntegerProperty;
@property(readonly) id readOnlyProperty;
@property(getter=isEnabled, setter=setEnabled:) BOOL enabled;

@end
```

Синтаксис: Свойства, Модификаторы

Модификаторы свойств:

retain / strong — увеличивает счетчик ссылок присваиваемого объекта на 1

copy — копирует присваиваемый объект

assign / weak — присвоить объект, не изменяя счетчик ссылок.

nonatomic / atomic — использовать ли внутренний lock при доступе / изменении свойства. Все свойства atomic поумолчанию.

readonly — можно только получать значение свойства, но не задавать его.

Синтаксис: Свойства, Внутреннее устройство

Для свойств компилятор сгенерирует 2 метода:

- (void) set(***#PropertyName***):(***#PropertyClass***) propertyName;
- (***#PropertyClass***) ***#propertyName***;

А так же сгенерирует внутренние переменные класса:

- ***#PropertyClass _#propertyName***;

Можно указать компилятору, что свойство привязано к уже объявленной переменной:

- @synthesize #propertyName = _instanceVariable;
- Так же, можно указать, что для свойства не требуется создавать переменную / методы и программист определит их сам:
- @dynamic #propertyName;

Синтаксис: Свойства, Внутренне устройство. Пример.

@property(nonatomic, retain) id retainedProperty;

Методы и переменные, генерируемые компилятором:

```
- (void) setRetainedProperty:(id) retainedProperty;

- (id) retainedProperty;
```

Переменная класса id _retainedProperty

Синтаксис: Свойства, Внутренне устройство. Пример.

@property(nonatomic, retain) id retainedProperty;

Реализация метода по умолчанию зависит от модификаторов свойства:

```
- (void) setRetainedProperty:(id) retainedProperty
{
    if(_retainedProperty!=retainedProperty)
   {
        [_retainedProperty release];
        _retainedProperty = [retainedProperty retain];//либо copy
    }
}
```

Синтаксис: Свойства, Внутренне устройство. Пример.

@property(nonatomic, assign) id assignedProperty;

Реализация метода по умолчанию зависит от модификаторов свойства:

```
- (void) setAssignedProperty:(id) assignedProperty
{
    _assignedProperty = assignedProperty;
}
```

Синтаксис: Свойства, Внутренне устройство. Пример.

@property(assign) id assignedAtomicProperty;

Реализация метода по умолчанию зависит от модификаторов свойства:

```
- (void) setAssignedAtomicProperty:(id) assignedAtomicProperty
{
    @synchronized(self)
    {
        _assignedAtomicProperty = assignedAtomicProperty;
    }
}
```

Управление памятью

Objective-C на iOS не использует сборщик мусора.

> Сборщик мусора был доступен в Mac OS X, но был объявлен устаревшим в 10.8

В Objective-C используется модель управления памятью, основанная на ***подсчете ссылок*** на объект:

- При создании или копировании, счетчик ссылок на объект = 1
- Посылка сообщения **retain** увеличивает счетчик на 1
- Посылка сообщения **release** уменьшает счетчик на 1
- Посылка сообщения **autorelease** уменьшит счетчик ссылок на 1 в ближайшее время
- Если количество ссылок на объект = 0, он уничтожается

Управление памятью. Пример.

Код	Счетчик ссылок
`id object = [[Tool alloc] init];`	Retain count = 1.
`[object retain];`	+1. Retain count = 2.
`[object release];`	-1. Retain count = 1.
`[object autorelease];`	-1. Retain count = 1. Уменьшится в ближайшее время до 0.
`id copied = [object copy];`	Retain count = 1.

Управление памятью.
Пример: Утечка памяти.

Код (неправильный)

```
id object = [[Tool alloc] init];

object = [[Hammer alloc] init];
```

Счетчик ссылок

- Retain count = 1.
- Retain count объекта, на который указывает object все так же = 1. Но мы потеряли указатель на этот объект и присвоили другой.
- У нас нет возможности послать сообщение release первому объекту, а значит память, выделенная для него, никогда не освободится.

Управление памятью.
Пример: Исправляем утечку памяти.

Код (правильный)

```
id object = [[Tool alloc] init];

[object release];

object = [[Hammer alloc] init];
```

Счетчик ссылок

- Retain count = 1.
- -1. Retain count = 0. Память, выделенная под объект, на который указывает object, освобождена.
- Теперь мы можем присвоить переменной object указатель на новый объект.

Управление памятью: ARC (Automatic Reference Counting)

- ARC — технология Apple уровня компилятора. Освобождает от необходимости вручную управлять памятью через посылку retain / release сообщений.
- Во время компиляции сам компилятор вставляет необходимые вызовы retain / release.
- Заметно сокращает количество написанного кода.
- Запрещает вызовы методов retain / release.
- В 99% случаев работает корректно.
- По данным от Apple, приложения на ARC работают быстрее. Стабильнее точно.
- По данным коммьюнити, работает медленнее, чем ручное управление.

Управление памятью: ARC. Пример.

Код

```
id object = [[Tool alloc] init];
```

Счетчик ссылок

- Retain count = 1. После того, как объект покинет текущую область видимости(функция, блок кода). Компилятор вставит вызов метода release для этого объекта.

```
object = [[Hammer alloc] init];
```

- При присваивании указателю object нового значения у предыдущего объекта будет вызван метод release. Утечки нет

Заключение

1. Objective-C — Объектно-ориентированный, предоставляющий богатые возможности для разработки приложений.

2. Objective-C — язык из 80-х, вдохновленный SmallTalk.

3. Run-time библиотека Objective-C — сердце языка, предоставляет огромнейшие возможности для изменения программы во время исполнения.

4. Apple постоянно совершенствует язык и фреймворки (fast enumeration, блоки, модули компиляции), что упрощает жизнь разработчиков и позволяет быстрее и качественнее создавать библиотеки.

В следующей лекции:

Инструменты:

IDE. Фреймворки. Технологии

Спасибо за внимание!

Жду ваших вопросов.