LOG
O

Управление на паметта в Objective C

Станимира Железова

Съдържание

Създаване на обект

- Създаването на обект става на две фази:
  - Заделяне на памет

    NSString * bla = [NSString alloc]; – връща указател към заделена памет за обект от тип NSString
  - Инициализиране на променливите на обекта

    [bla initWithString: @”BLA :)”]; – инициализира стойността на стринга с дадения аргумент
  - Най-често се използват заедно:

```
NSString * bla = [[NSString alloc] initWithString: @”BLA :)”] ;
```

Създаване на обект

- + (id)new – метод който комбинира alloc и init

  NSString * bla = [NSString new]; - заделя памет, инициализира обекта и връща указател към него

- Създаване чрез копиране - (id)copy :
  - NSString *otherString = [bla copy];
  - - създава функционално копие на обекта. В общия случай заделя нова памет и копира цялото състояние на обекта

Броене на референциите

- Всеки обект знае броя на референциите към себе си

  [obj retainCount]; - връща броя на референциите
- При създаване на обект (заделяне на памет с alloc/copy), retainCount започва да се брои от 1

  ```
  MyClass * obj = [[MyClass alloc] init];
  ```
- Добавяме 1 за всеки retain извикан на обекта
- Вадим 1 за всеки release

  ```
  NSLog(@"obj refs : %d", [obj retainCount]);
  ```

Броене на референциите

Разрушаване на обект

- Методът dealloc се извиква щом броят на референциите стане 0.
  - Всяка памет, която се реферира от обекта, трябва да бъде освободена (както и трябва да е била задържана чрез alloc/retain)

```
-(void) dealloc {
[var release];
…
[super dealloc];
}
```

Управление на паметта

```
@implementation StringWrapper
-(id) init {
self= [super init];
if (self) {
name = [NSString new];
}
return self;
}

-(NSString *) name {
return name;
}

-(void) setName:(NSString *)input {
[input retain];
[name release];
name = input;
}
-(void) dealloc {
[name release];

[super dealloc];
}
@end
```

Управление на паметта

```
int main(int argc, const char *argv[]) {
NSString *arg = [[NSString alloc] init];
StringWrapper wrapper = [StringWrapper alloc] init];
[wrapper setName: arg];
    NSLog(@”arg refs : %d”, [arg retainCount]);
[arg release];
...
[wrapper relsease];
return 0;
}
```

Управление на паметта

- Важно – за всеки retain/alloc трябва да има съответен release:
  - Ако заделяте памет в init, трябва да я освободите в dealloc
  - Ако пишете своя set фунция:
    - retain-вайте новата стойност
    - release-вайте старата стойност
    - проверявайте дали се получавате същата стойност
- Внимавайте за циклични референции
  - Ако държите на parent връзка, референцията не трябва да се retain-ва

Управление на паметта

- Можете да пращате съобщения на nil, но не и на деалокирани обекти:
  - [obj someMessage] – ако обектът е бил деалокиран, предизвиква EXC_BAD_ACCESS
  - Това може да се случи:
    - ако пазите референция към обект без да сте му извикали retain
    - ако някъде в програмата ползвате същия обект и сте извикали два пъти release за един и същ retain
  - За debug цели можете да пращате съобщения към деалокирани обекти – трябва да сетнете YES на NSZombieEnabled

Управление на паметта

- Броене на референции vs автоматичен garbage collection
  - Паметта се освобождава възможно най-скоро
  - Не са нужни сложни алгоритми за проследяване на референциите към обект
  - Не се справя с циклични референции – когато един обект, пряко или не, сочи себе си
  - Повече възможности за грешка
    - Memory leaks – изгубване на указател към заделена памет
    - Освобождаване на памет, която все още се ползва

Автоматично освобождаване

- autorelease
  - освобождава паметта автоматично
  - не се отразява на retainCount-а при извикването си
- Използването на autorelease не се препоръчва, но е неизбежно когато връщате обект като резултат от метод
  - не можете да release-нете обекта преди да е retain-нат от получателя

    `NSString* string1 = [NSString string];` - връщаната стойност е autorelease-ната

Автоматично освобождаване

- @”BLA” – обект от тип NSString, който е автоматично освободен при създаването си
  - Ако ще ползваме така създаден низ извън scope-а, в който е създаден, трябва да го retain-нем

```
-(void) generateNewName {
[name autorelease];
name = [@”random name” retain];
}
- (NSString *) description {
return @“Some description“;
}
```

Автоматично освобождаване

- Стандартно обектите се създават:
  - директно, с извикване на alloc и init (с или без параметри)

    [[NSString alloc] initWithString:@”Bla”]; - връща указател към заделена памет, която трябва да бъде освободена ръчно с release
  - чрез статични методи на класовете

    [NSString stringWithString:@”Bla”]; - връща autorelease-ната референция към обект от класа. Може да бъде задържана с retain

NSAutoreleasePool

- Обект, който „държи“ всички autorelease-нати обекти. Създава се стандартно чрез alloc/init, но не може да бъде retain-ван.

```
NSAutoreleasePool *a = [[NSAutoreleasePool alloc] init];
NSString * aStr = @”Autoreleased string”;
```

- Когато IPool се изпразни, всички обекти, които „държи“ ще получват съобщение release
  - [loopPool drain]; / [loopPool release];

    drain има ефект на release - намаля броя на референциите до 0 и деалокира обекта.

NSAutoreleasePool

- Когато има няколко вложени обекта от NSAutoreleasePool, autorelease-натите обектите се добавят в „най-близкия“ (най-скоро създадения) pool
- Един NSAutoreleasePool се създава и разрушава в същия контекст - тяло на метод, цикъл и тн.
- Ако създавате отделна нишка в приложението си, задължително трябва да съзададете NSAutoreleasePool в началото и да го разрушите в края.
- Създаването на NSAutoreleasePool обекти като цяло се изпозлва за да се постигне по-добро управление на паметта и освобождаване на autorelease-натите обекти

NSAutoreleasePool

```
- (void) main {
NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
        NSLog (@"The date and time is: %@", [NSDate date];);
StringWrapper wrapper = [[StringWrapper new] autorelease];
NSString *bla = @”Some autoreleased string”;
[bla retain];
[wrapper setName: bla];
    NSLog(@”Bla refs : %d”, [bla retainCount]);
[bla release];
[pool release];
}
```

Пропъртита

- Представят виртуални променливи от класа:

```
@interface Bla : NSObject {
@private int iVar;
}
@property (readwrite,assign) int iVar;
@end

@implementation Bla
@synthesize iVar; // Tells the compiler to instantiate it: “Generate a getter and a setter // messages, named iVar and setIVar.
// -(void)setIVar:(int)I { iVar = i; }
// -(int)iVar { return self->iVar; }
// Note that those methods can be redefined by the programer.
@end
```

Пропъртита

- За присвояване:
  - assign – присвоява на променливата подадената стойност
  - retain - присвоява на променливата подаден указател към обект и увеличва броя на референциите към него с 1
  - copy – присвоява копие на подадения обект
- Atomic / Nonatomic
  - It’s a good practice to use nonatomic attribute, while providing own thread-safety.
- Readonly
- @property (getter=getI, setter=setI:) int iVar;

Пропъртита

- self.name = извиква метода, генериран за пропъртито
- name = директно присвоява стойност на променливата

```
@interface StringWrapper: NSObject {
@private NSString *name;
}
@property (readwrite, retain) NSString * name;
@end

@implementation StringWrapper {
@synthesize name;
-(id) initWithName: (NSString *) name_ {
name = [name_ retain]; // self.name = name_
}
```

Пропъртита

```
@implementation StringWrapper {

@synthesize name; //generates getter and setter

-(id) initWithName: (NSString *) name_ {
self.name = name_ ;
// Alternatively: name = [name_ retain];
}


....
-(void) dealloc {
    [name release]; //it was retained by the property
}
```

NSArray

- Списък, който retain-ва добавените в него обекти:

```
NSString * obj = [NSString new];
NSArray *arrayA = [NSArray arrayWithObject: obj];
NSLog(@”Obj refs : %d”, [obj retainCount]);
[obj release]; //it is already retained by the array
```

- Полезни методи:
  - - (NSUInteger)count;
  - - (id)objectAtIndex:(NSUInteger)index;
  - - (BOOL)containsObject:(id)anObject;

NSArray

- Можете да изпратите съобщение към всички обекти от един списък:

```
[array makeObjectsPerformSelector: @selector(bla)];
```

- Можете да сортирате списъка по критерий:

```
NSArray *sorted = [array sortedArrayUsingSelector: @selector
(someComparator:)];
```

- NSMutableArray – NSArray, който можете да променяте
  - addObject:
  - insertObject:atIndex:
  - removeObjectAtIndex:

NSDictionarry

- Мап, който retain-ва добавените обекти и копира ключовете:

```
NSString * key = @”Key”;
NSArray *keys = [NSArray arrayWithObjects: key, nil];
NSDictionarry * dict = [NSDictionarry alloc] initWithObjects: array forKeys: keys];
NSLog(@”Key refs : %d”, [key retainCount]);
```

- NSDictionarry може да съдържа само обекти
- Обектите, които се ползват за ключ, трябва да имплементират протокола NSCopying

NSDictionary

- Полезни методи:
  - - (NSArray *)allKeys
  - - (NSArray *)allKeysForObject:(id)anObject
  - - (NSArray *)allValues
  - - (id)objectForKey:(id)aKey

- (NSSet *)keysOfEntriesPassingTest:(BOOL (^)(id key, id obj, BOOL *stop))predicate
  - Връща ключовете на записите, които удовлетворяват предиката

LOG
O

Благодаря за вниманието!

Въпроси?