北京工业大学

Objective-C语法基础

北京工业大学 信息学部
刘轩

Objective-C

- Objective-C语言是一种通用的，面向对象的编程语言，Smalltalk风格消息传送到C编程语言。它是使用苹果OS X和iOS操作系统及彼等各自的API， Cocoa 和Cocoa Touch主要的编程语言。
- Objective-C是C语言的严格超集－－任何C语言程序不经修改就可以直接通过Objective-C编译器，在Objective-C中使用C语言代码也是完全合法的。Objective-C被描述为盖在C语言上的薄薄一层，因为Objective-C的原意就是在C语言主体上加入面向对象的特性。

Objective-C代码的文件扩展名

.h 头文件。头文件包含类，类型，函数和常数的声明。

.m 源代码文件。这是典型的源代码文件扩展名，可以包含 Objective-C 和 C 代码。

.mm 源代码文件。带有这种扩展名的源代码文件，除了可以包含Objective-C和C代码以外还可以包含C++代码。

Test-1

```
#import <Foundation/Foundation.h>


int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSLog(@"Hello, World!");
    }
    return 0;
}
```

Test-2

```
#import <Foundation/Foundation.h>

@interface SampleClass:NSObject
-(void)sampleMethod;
@end

@implementation SampleClass
-(void)sampleMethod{
    NSLog(@"SampleClass Say Hello");
}
@end

int main(int argc, const char * argv[]) {
    SampleClass *sampleClass=[[SampleClass alloc] init];
    [sampleClass sampleMethod];
    return 0;
}
```

关键字

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double	protocol	interface	implementation
NSObject	NSInteger	NSNumber	CGFloat
property	nonatomic;	retain	strong
weak	unsafe_unretained;	readwrite	readonly

数据类型-整型

类型	存储长度	值范围
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Test-3

```
int main(int argc, const char * argv[]) {

    char c='a';
    NSLog(@"%d",c);

    int i=100;
    NSLog(@"%d",i);

    short s=32767;
    NSLog(@"%d",s);

    long l=2147483647;
    NSLog(@"%lu",l);

    return 0;
}
```

NSLog 格式化占位符

%@ 对象
%d, %i 整数
%u　无符整形
%f 浮点/双字
%x, %X 二进制整数
%o 八进制整数
%zu size_t
%p 指针
%e　浮点/双字　(科学计算)
%g　浮点/双字
%s C 字符串

%.*s Pascal字符串
%c 字符
%C unichar
%lld 64位长整数（long long）
%llu　无符64位长整数
%Lf 64位双字
%i Bool类型值默认为False，即0，为True时输出为1。
%f %0.2f 只保留两位小数

数据类型-浮点类型

类型	存储大小	取值范围	精确
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

```
float f=0.1f;  NSLog(@"%f",f);
double d=0.375;  NSLog(@"%f",d);
long double ld=0.333333333;  NSLog(@"%Lf",ld);
```

运算符

- 算术运算符
- 关系运算符
- 逻辑运算符
- 位运算符
- 赋值运算符
- 其他运算符

运算符	描述	示例
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator increases integer value by one	A++ will give 11
--	Decrement operator decreases integer value by one	A-- will give 9

关系运算符

运算符	描述	示例
==	Checks if the values of two operands are equal or not; if yes, then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not; if values are not equal, then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand; if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand; if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then condition becomes true.	(A <= B) is true.

逻辑运算符

运算符	描述	示例
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

位运算符

运算符	描述	例子
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61, which is 1100 0011 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 0000 1111

赋值运算符

运算符	描述	示例
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assigns the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assigns the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assigns the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assigns the result to left operand	C %= A is equivalent to C = C % A

流程控制语句-循环

Loop Type	描述
while 循环	一个给定的条件为真时，重复执行一个语句或语句组。执行循环体测试前的状态.
for 循环	执行的语句序列多次缩写的代码管理循环变量.
do...while 循环	while语句一样，只不过它在循环体结束测试条件.
内嵌循环	可以使用任何其他一个或多个循环 while, for or do..while 循环.

控制语句	描述
break 语句	终止循环或开关语句将执行的语句紧随循环或开关。
continue 语句	导致循环，跳过它的主体的其余部分，并立即重新测试前的重申状况。

Test-4

```
#import <Foundation/Foundation.h>
int main(int argc, const char * argv[]) {
    int a=50;
    while (a<=100) {
        NSLog(@"%d",a);
        a++;
    }

    int b=0;
    do{
        NSLog(@"%d",b);
        b=b+1;
    }while (b<=10);

    for(int i=0;i<=5;i++){
        NSLog(@"i=%d",i);
    }
    return 0;
}
```

流程控制语句-决策

语句	描述
if 语句	if语句由一个布尔表达式后跟一个或多个语句。
if...else 语句	if语句后面可以通过一个可选的else语句，布尔表达式为假时执行。
内嵌if 语句	可以使用一个或else if语句if或else if语句在另一个（次）。
switch 语句	switch语句允许值的列表相等对变量进行测试。
内嵌switch 语句	在另一个switch语句（次），可以使用一个switch语句。

Test-5

```
int main(int argc, const char * argv[]) {
    int a=10;
    if(a>=5){
        NSLog(@"a>5");
    }
    if(a>3){
        NSLog(@"a>3");
    }else if(a>7){
        NSLog(@"a>7");
    }else{
        NSLog(@"a!>3 && a!>7");
    }

    return 0;
}
```

Test-6

```
#import <Foundation/Foundation.h>
int main(int argc, const char * argv[]) {

    char c='B';
    switch (c) {
        case 'A':
            NSLog(@"A");
            break;
        case 'B':
            NSLog(@"B");
            break;
        case 'C':
            NSLog(@"C");
            break;
        default:
            NSLog(@"其他");
            break;
    }

    return 0;
}
```

Test-7

```
/*
    1. 输出从1到100之间所有的整数;(简单的单层for循环)**for循环遍历整数**
    2. 输出从1到100之间所有的奇数;(通过与if-else结合，有条件的输出语句)
    3. 输出从1到100之间所有不能被3整除的数;并输出这些整数的和;(通过增加条件，更合理的练习for循环与if语句结合)
    */

    for(int i=1;i<=100;i++){
        NSLog(@"%d",i);
    }

    for(int j=1;j<=100;j++){
        if(j%2!=0){
            NSLog(@"%d",j);
        }
    }

    for(int k=1;k<=100;k++){
        if(k%3!=0){
            NSLog(@"%d",k);
        }
    }
```

Test-8

```
#import <Foundation/Foundation.h>
int main(int argc, const char * argv[]) {
    /*
    问题：鸡兔同笼，鸡兔一共35只。笼子里脚一共94只，请问分别有多少只鸡和兔?
    思路：首先明确思路，鸡的数量*2加上兔子的数量*4等于脚的总数94
    */
    int sum=35;
    int foot=94;
    for(int c=1;c<=foot/2;c++){
        int r=sum-c;
        if(r*4+c*2==foot){
            NSLog(@"鸡:%d, 兔:%d",c,r);
            break;
        }
    }

}
```

字符串 NSString (Test-9)

```
NSString *str=@"abc";
//转为大写
NSLog(@"%@",[str uppercaseString]);
//全部转为小写
NSLog(@"%@",[str lowercaseString]);
//首字母大写
NSLog(@"%@",[str capitalizedString]);
//比较两个字符串内容是否相同
BOOL b =[str isEqualToString:@"abb"];
NSLog(@"%i",b);
//两个字符串内容比较
//NSOrderedAscending    右边 > 左边
//NSOrderedSame         内容相同
//NSOrderedDescending   左边 > 右边
NSString *str1=@"a";
NSString *str2=@"b";
NSComparisonResult result = [str1 compare:str2];
if (result == NSOrderedAscending) {
    NSLog(@"右边 > 左边");
}else if(result == NSOrderedSame){
    NSLog(@"内容相同");
}else if (result == NSOrderedDescending){
    NSLog(@"左边 > 右边");
}
```

```
//忽略大小写进行比较，返回值与compare一样
 result = [str1 caseInsensitiveCompare:str2];
 if (result == NSOrderedAscending) {
    NSLog(@"右边 > 左边");
 }else if(result == NSOrderedSame){
    NSLog(@"内容相同");
 }else if (result == NSOrderedDescending){
    NSLog(@"左边 > 右边");
 }
    //判断字符串是否以指定字符串开头
    [str1 hasPrefix:@"aaa"];
    //判断字符串是否以指定字符串结尾
    [str1 hasSuffix:@"aaa"];
    //判断字符串是否包含指定字符串，返回位置和长度
    NSRange range = [@"123456" rangeOfString:@"45"];
    NSLog(@"%@", NSStringFromRange(range));
    //反向搜索
    range = [@"123456456qweasasd456" rangeOfString:@“456"
options:NSBackwardsSearch];
    NSLog(@"%@",NSStringFromRange(range));
    //指定范围进行搜索
    range = NSMakeRange(0, 9);
    range = [@"123456456qweasasd456" rangeOfString:@“456"
options:NSBackwardsSearch range:range];
    NSLog(@"%@",NSStringFromRange(range));
```

数字 NSNumber (Test-10)

```
//NSNumber：专门用来装基础类型的对象，把整型、单精度、双精度、字符型等基础类型存储为对象
NSNumber *intNumber = [[NSNumber alloc] initWithInt:5];
NSNumber *floatNumber = [[NSNumber alloc] initWithFloat:3.14f];
NSNumber *doubleNumber = [[NSNumber alloc] initWithDouble:6.7];
NSNumber *charNumber = [[NSNumber alloc] initWithChar:'A'];

//初始化
NSNumber * intNumber2 = [NSNumber numberWithInt:6];
//这种比较也是可以跨不同对象的，比如：比较intNumber和floatNumber
BOOL ret = [intNumber isEqualToNumber:intNumber2];
NSLog(@"%i",ret);

//NSNumber比较
if ([intNumber compare:intNumber2] == NSOrderedAscending){
    NSLog(@"<");
}else if([intNumber compare:intNumber2] == NSOrderedSame){
    NSLog(@"=");
}else if([intNumber compare:intNumber2] == NSOrderedDescending){
    NSLog(@">");
}

//转型
NSInteger *i=[intNumber integerValue];
NSString *s=[intNumber stringValue];
```

数据结构数组

type arrayName [arraySize];

type name[size1][size2]...[sizeN];

```
double arr[10];
int arr2[5]={1,2,3,4,5};

for(int i=0;i<5;i++){
    NSLog(@"%d",arr2[i]);
}

int arr3[3][2]={{1,2},{3,4},{5,6}};
for(int i=0;i<3;i++){
    for(int j=0;j<2;j++){
        NSLog(@"arr3:%d",arr3[i][j]);
    }
}
```

数组对象 NSArray & NSMutableArray

NSArray是用来装不可变的对象数组，NSMutableArray用于容纳一个可变数组对象。
Mutablility有助于改变数组中运行一个预分配的数组，但如果我们使用NSArray 只能更换现有数组，并不能改变现有数组的内容。
NSArray 的重要方法如下:

- alloc/initWithObjects: 用来初始化一个数组对象。
- objectAtIndex: 在特定索引|allReturns对象。
- count: 返回的对象的数量

NSMutableArray继承自NSArray，因此NSMutableArray有NSArray 所有实例方法
重要的 NSMutableArray 方法如下：

- removeAllObjects: 清空数组。
- addObject: 数组末尾插入一个给定的对象。
- removeObjectAtIndex: 这是用来删除objectAt 指定索引的对象
- exchangeObjectAtIndex:withObjectAtIndex: 改变阵列中的对象在给定的索引。
- replaceObjectAtIndex:withObject: 替换的对象与对象在索引。

Test-11

```
//创建数组
NSArray *array1 = [NSArray arrayWithObject:@"1"];
NSArray *array2 = [NSArray arrayWithObjects:@"1",@"2",@"3", nil];
NSArray *array3 = [NSArray arrayWithArray:array2];
NSLog(@"array1 = %@",array1);
NSLog(@"array2 = %@",array2);
NSLog(@"array3 = %@",array3);

//获取数组内数据个数
unsigned long count = [array2 count];
NSLog(@"array2 size is %lu",count);

//访问数组内的数据
NSString *str1 = [array2 objectAtIndex:0];
NSLog(@"array2 first content is %@",str1);
```

```
//数组中插入数据  返回新的数组
NSArray *array4 = [array3 arrayByAddingObject:@"4"];
NSLog(@"array4 = %@",array4);

//数组内的数据以制定字符连接
NSString *str2 = [array4 componentsJoinedByString:@","];
NSLog(@"str2 = %@",str2);

//判断数组中是否包含某对象
BOOL b1 = [array4 containsObject:@"4"];
BOOL b2 = [array4 containsObject:@"5"];
NSLog(@"b1 = %d,b2 = %d",b1,b2);

//取数组内制定对象的索引
int index = [array4 indexOfObject:@"4"];
NSLog(@"index = %d",index);

NSString *str3 = [array4 lastObject];
NSLog(@"array4 last object is %@",str3);
```

Test-12

```
//初始化数组 指定数组长度  但可变
NSMutableArray *mArray1 = [NSMutableArray arrayWithCapacity:5];
//向数组中添加元素
[mArray1 addObject:@"aaaa"];
[mArray1 addObject:@"cccc"];
NSLog(@"mArray1 = %@",mArray1);

//向指定位置插入元素
[mArray1 insertObject:@"bbbb" atIndex:1];
[mArray1 insertObject:@"dddd" atIndex:[mArray1 count]];
[mArray1 insertObject:@"eeee" atIndex:[mArray1 count]];
NSLog(@"mArray1 = %@",mArray1);

//移除元素
[mArray1 removeObject:@"eeee"];
NSLog(@"mArray1 = %@",mArray1);
[mArray1 removeObjectAtIndex:[mArray1 count]-1];
NSLog(@"mArray1 = %@",mArray1);
NSArray *array5 = [NSArray arrayWithObjects:@"bbbb",@"cccc", nil];
[mArray1 removeObjectsInArray:array5];
NSLog(@"mArray1 = %@",mArray1);

//向数组内插入数组
NSMutableArray *mArray2 = [NSMutableArray arrayWithObjects:@"aaaa",@"aaaa", nil];
[mArray2 addObject:@"bbbb"];
NSLog(@"mArray2 = %@",mArray2);
[mArray2 addObjectsFromArray:[NSArray arrayWithObjects:@"a",@"b", nil]];
NSLog(@"mArray2 add = %@",mArray2);
```

Test-13

```
//替换元素
[mArray2 replaceObjectAtIndex:[mArray2 count]-1 withObject:@"5"];
NSLog(@"mArray2 = %@",mArray2);

//遍历数组  常规方法:性能较低
NSArray *array6 = [NSArray arrayWithObjects:@"a",@"b",@"c",@"d",@"e", nil];
int len = [array6 count];
for (int i=0; i<len; i++) {
    NSString *value = [array6 objectAtIndex:i];
    NSLog(@"array6 %d content is %@",i,value);
}

//枚举遍历  相当于java中的增强for循环
for (NSString *string in array6) {
    NSLog(@"array6 content is %@",string);
}

NSLog(@"----------------------");

//当不确定数组元素类型时
for(id string in array6){
    NSLog(@"array6 content is %@",string);
}
```

字典对象 NSDictionary & NSMutableDictionary

NSDictionary 用于容纳一个不可变的对象，字典 NSMutableDictionary 用于容纳一个可变的对象字典。

重要的NSDictionary方法如下

- alloc/initWithObjectsAndKeys: 初始化一个新分配的字典带构建从指定的集合值和键的条目。
- valueForKey: 返回与给定键关联的值。
- count: 返回在字典中的条目的数量。

NSMutableDictionary 继承自 NSDictionary ， 因此NSMutableDictionary 实例拥有 NSDictionary 的所有方法

重要的NSMutableDictionary方法如下：

- removeAllObjects:清空字典条目。
- removeObjectForKey: 从字典删除给定键及其关联值。
- setValue:forKey: 添加一个给定的键 - 值对到字典中。

Test-14

```
///创建字典
NSDictionary *dic1 = [NSDictionary dictionaryWithObject:@"value" forKey:@"key"];
NSLog(@"dic1 :%@", dic1);

//创建多个字典
NSDictionary *dic2 = [NSDictionary dictionaryWithObjectsAndKeys:
                      @"value1", @"key1",
                      @"value2", @"key2",
                      @"value3", @"key3",
                      @"value4", @"key4",
                      nil];
NSLog(@"dic2 :%@", dic2);

//根据现有的字典创建字典
NSDictionary *dic3 = [NSDictionary dictionaryWithDictionary:dic2];
NSLog(@"dic3 :%@", dic3);

//根据key获取value
NSLog(@"key3 value :%@", [dic3 objectForKey:@"key3"]);

//获取字典数量
NSLog(@"dic count :%d", dic3.count);

//所有的键集合
NSArray *keys = [dic3 allKeys];
NSLog(@"keys :%@", keys);

//所有值集合
NSArray *values = [dic3 allValues];
NSLog(@"values :%@", values);
```

Test-15

```
//可变数组对象
NSMutableDictionary *mutableDic = [[NSMutableDictionary alloc] initWithObjectsAndKeys:
                                   @"mvalue1", @"mkey1",
                                   @"mvalue2", @"mkey2", nil];
//添加现有的字典数据
[mutableDic addEntriesFromDictionary:dic3];
NSLog(@"mutableDic :%@",mutableDic);

//添加新的键值对象
[mutableDic setValue:@"set1" forKey:@"setKey1"];
NSLog(@"set value for key :%@",mutableDic);

//以新的字典数据覆盖旧的字典数据
[mutableDic setDictionary:dic2];
NSLog(@" set dictionary :%@",mutableDic);

//根据key删除value
[mutableDic removeObjectForKey:@"key1"];
NSLog(@"removeForkey :%@",mutableDic);

//快速遍历
for(id key in mutableDic) {
    NSLog(@"key :%@  value :%@", key, [mutableDic objectForKey:key]);
}

//枚举遍历
NSEnumerator *enumerator = [mutableDic keyEnumerator];
id key = [enumerator nextObject];
while (key) {
    NSLog(@"enumerator :%@", [mutableDic objectForKey:key]);
    key = [enumerator nextObject];
}

//根据key数组删除元素
[mutableDic removeObjectsForKeys:keys];
NSLog(@"removeObjectsForKeys :%@",mutableDic);

[mutableDic removeAllObjects];
//删除所有元素
NSLog(@"remove all :%@", mutableDic);
```

集合对象 NSSet & NSMutableSet

NSSet 是用来保持一个不变集的不同对象，NSMutableDictionary 用于容纳一个可变设置的不同对象。

重要的NSSet方法如下：

表。• alloc/initWithObjects: 初始化一个新分配的成员采取从指定的对象列

- allObjects - 返回一个数组，包含集合的成员或一个空数组（如果该组没有成员）。
- count: 返回集合中的成员数量。

NSMutableSet 继承自NSSet，因此所有NSSet方法的在NSMutableSet 的实例是可用的。

重要的 NSMutableSet方法如下：

- removeAllObjects: 清空其所有成员的集合。
- addObject: 添加一个给定的对象的集合（如果它还不是成员）。
- removeObject: 从集合中删除给定的对象。

Test-16

```
NSSet *set1 = [NSSet setWithObjects:@"a", @"b", @"c", @"d", nil];
NSSet *set2 = [[NSSet alloc] initWithObjects:@"1", @"2", @"3", nil];
NSArray *array = [NSArray arrayWithObjects:@"a", @"b", @"c", nil];
NSSet *set3 = [NSSet setWithArray:array];

NSLog(@"set1 :%@", set1);
NSLog(@"set2 :%@", set2);
NSLog(@"set3 :%@", set3);

//获取集合个数
NSLog(@"set1 count :%d", set1.count);

//以数组的形式获取集合中的所有对象
NSArray *allObj = [set2 allObjects];
NSLog(@"allObj :%@", allObj);

//获取任意一对象
NSLog(@"anyObj :%@", [set3 anyObject]);

//是否包含某个对象
NSLog(@"contains :%d", [set3 containsObject:@"obj2"]);

//是否包含指定set中的对象
NSLog(@"intersect obj :%d", [set1 intersectsSet:set3]);

//是否完全匹配
NSLog(@"isEqual :%d", [set2 isEqualToSet:set3]);

//是否是子集合
NSLog(@"isSubSet :%d", [set3 isSubsetOfSet:set1]);

NSSet *set4 = [NSSet setWithObjects:@"a", @"b", nil];
NSArray *ary = [NSArray arrayWithObjects:@"1", @"2", @"3", @"4", nil];
NSSet *set5 = [set4 setByAddingObjectsFromArray:ary];
NSLog(@"addFromArray :%@", set5);
```

Test-17

```
//动态集合
NSMutableSet *mutableSet1 = [NSMutableSet setWithObjects:@"1", @"2", @"3", nil];
NSMutableSet *mutableSet2 = [NSMutableSet setWithObjects:@"a", @"2", @"b", nil];
NSMutableSet *mutableSet3 = [NSMutableSet setWithObjects:@"1", @"c", @"b", nil];

//集合元素相减
[mutableSet1 minusSet:mutableSet2];
NSLog(@"minus :%@", mutableSet1);

//只留下相等元素
[mutableSet1 intersectSet:mutableSet3];
NSLog(@"intersect :%@", mutableSet1);

//合并集合
[mutableSet2 unionSet:mutableSet3];
NSLog(@"union :%@", mutableSet2);

//删除指定元素
[mutableSet2 removeObject:@"a"];
NSLog(@"removeObj :%@", mutableSet2);

//删除所有数据
[mutableSet2 removeAllObjects];
NSLog(@"removeAll :%@", mutableSet2);
```

日期对象 NSDate (Test-18)

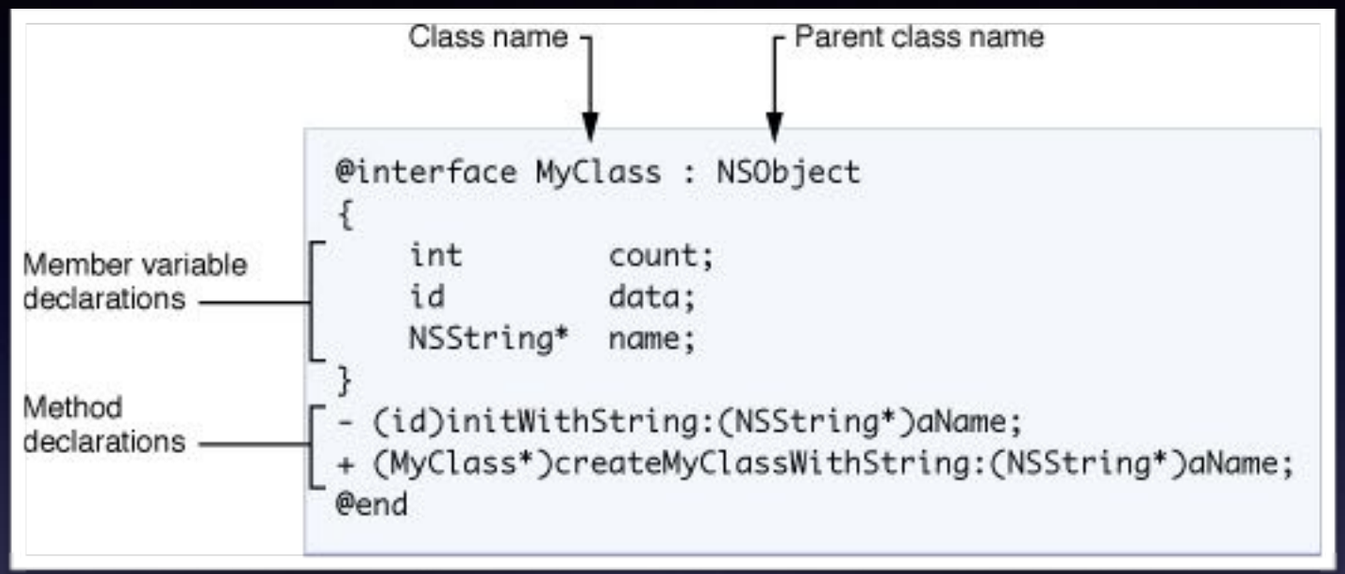
```
//显示当前时间
NSDate *date = [NSDate date];//获取当前的时间
NSDateFormatter *formatter = [[NSDateFormatter alloc]init];
[formatter setDateFormat:@"yyyy-MM-dd a HH:mm:ss EEEE"];
NSLog(@“date = %@",[formatter stringFromDate:date]);

//得到昨天此时的时间
NSDate *dateOfYesterday = [NSDate dateWithTimeInterval:-24*3600
sinceDate:date];
NSLog(@"yesterday = %@",[formatter stringFromDate:dateOfYesterday]);

//将秒数常用时间格式显示
NSDate *dateS = [NSDate dateWithTimeIntervalSince1970:100];
NSDateFormatter *formatterS = [[NSDateFormatter alloc]init];
[formatterS setDateFormat:@"mm:ss"];
NSLog(@"100s = %@",[formatterS stringFromDate:dateS]);

//自定义区域语言
formatter.locale = [NSLocale localeWithLocaleIdentifier:@"zh_CN"];
NSLog(@"data = %@",[formatter stringFromDate:date]);
```

日期对象 NSDate (Test-18)

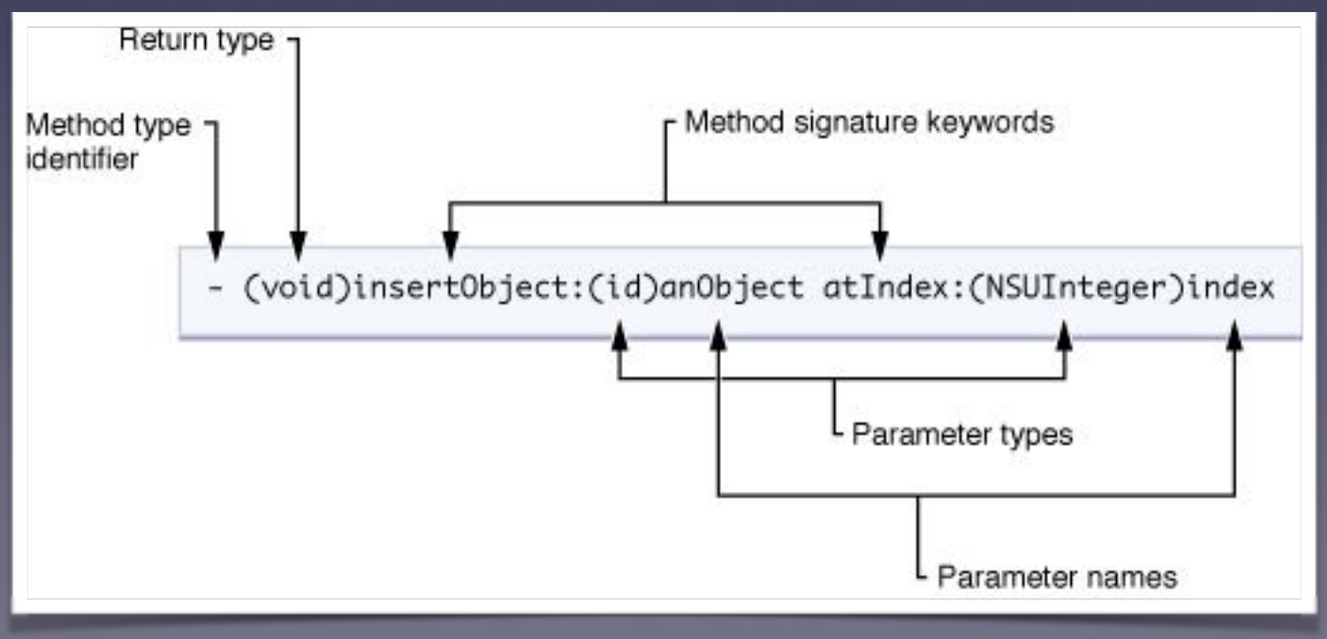
```
//将字符串转化位NSDate类型
NSString *dateString = @"2017-11-15 上午 11:50:40";
[formatter setTimeZone:[NSTimeZone systemTimeZone]];
[formatter setDateFormat:@"yyyy-M-dd a HH:mm:ss"];
NSDate *dateFromString =[formatter dateFromString:dateString];
NSLog(@"date = %@",[formatter stringFromDate:dateFromString]);


//时间戳转换
    NSDate *date2= [NSDate dateWithTimeIntervalSince1970:12123123123];
    NSDateFormatter *formatter2 = [[NSDateFormatter alloc]init];
    [formatter2 setTimeZone:[NSTimeZone defaultTimeZone]];
    [formatter2 setDateFormat:@"MM月d日 HH:mm"];
    NSString *str2 = [formatter2 stringFromDate:date2];
    NSLog(@“%@",str2);
    //当前系统时间戳
    NSLog(@"%@",[NSString stringWithFormat:@"%.f",[NSDate
date].timeIntervalSince1970*1000]);
```

类/属性/方法

类的声明



方法的声明



异常处理

```
int main(int argc, const char * argv[]) {
    NSMutableArray *array = [[NSMutableArray alloc]init];
    @try{
        NSString *string = [array objectAtIndex:10];
    }
    @catch (NSException *exception){
        NSLog(@"%@ ",exception.name);
        NSLog(@"Reason: %@ ",exception.reason);
    }
    @finally{
        NSLog(@"@@finaly Always Executes");
    }
}
```

刘 轩

13910230876

软件学院楼404