第10章　Foundation框架的基本对象

- 在Objective-C中的程序编写主要用到2个框架Foundation和ApplicationKit。其中Foundation框架主要定义了一些基础类，供程序员来使用。Foundation框架中的所有类都继承自NSObject这个对象。本章主要讲解Foundation的基本对象。

10.1 数字对象

- Foundation框架中所提到的数字类型，可以通过使用NS数字类型将基本的数据类型封装到对象中，然后对相应的对象进行操作。本节主要讲解数字对象的定义形式和适应方法。

10.1.1 数字对象的介绍

- Foundation框架中提供了NSNumber对基本的数据类型进行封装。通过调用数字对象的类方法和对象方法，将基本的数据类型转化为数字对象。

10.1.1 数字对象的介绍

类型	说明
char	字符型对象
UnsignedChar	无符号字符型对象
Short	短整型对象
UnsignedShort	无符号短整型对象
Integer	整型对象
UnsignedInteger	无符号整型类型对象
int	整型对象
UnsignedInt	无符号整型对象
Long	长整型对象
UnsignedLong	无符号长整型对象
LongLong	长长整型对象
UnsignedLongLong	无符号长长整型对象
float	浮点型对象
double	双精度型对象
Bool	布尔型对象

10.1.2　数字对象的使用

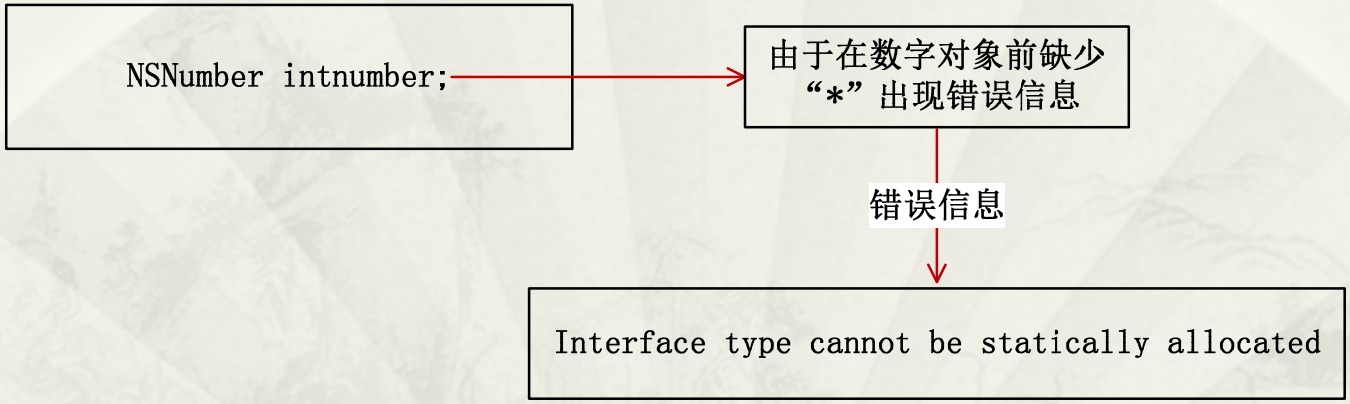
* 我们对数字对象有了一定的了解以后，我们来了解数字对象的在各方法中的使用。

1. 数字对象的声明

* 数字对象声明的形式。

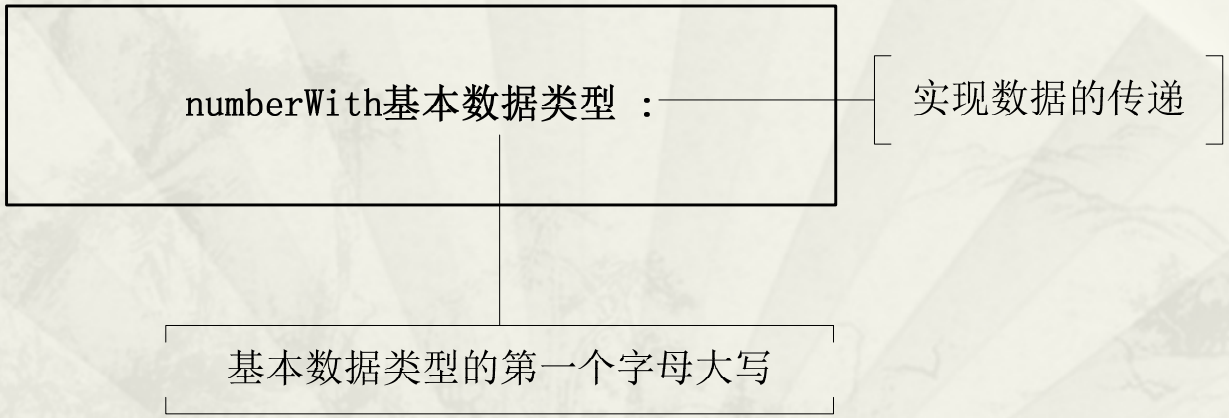
```
NSNumber *数字对象;
```

1. 数字对象的声明


NSNumber intnumber;
由于在数字对象前缺少“*”出现错误信息
错误信息
Interface type cannot be statically allocated

2. 数字对象的创建并初始化

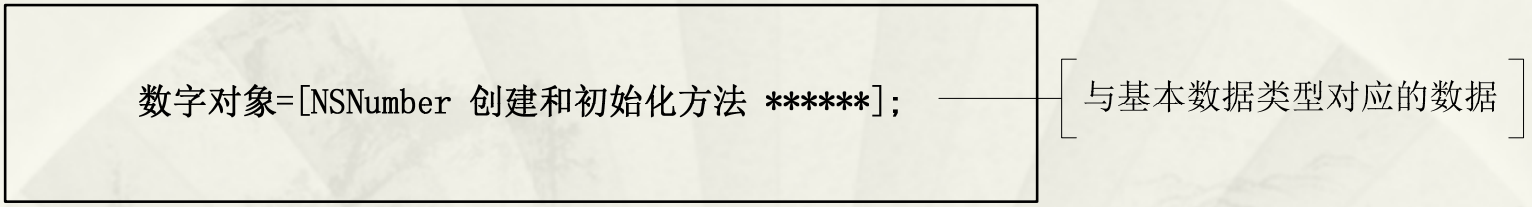
* 创建和初始化的形式。



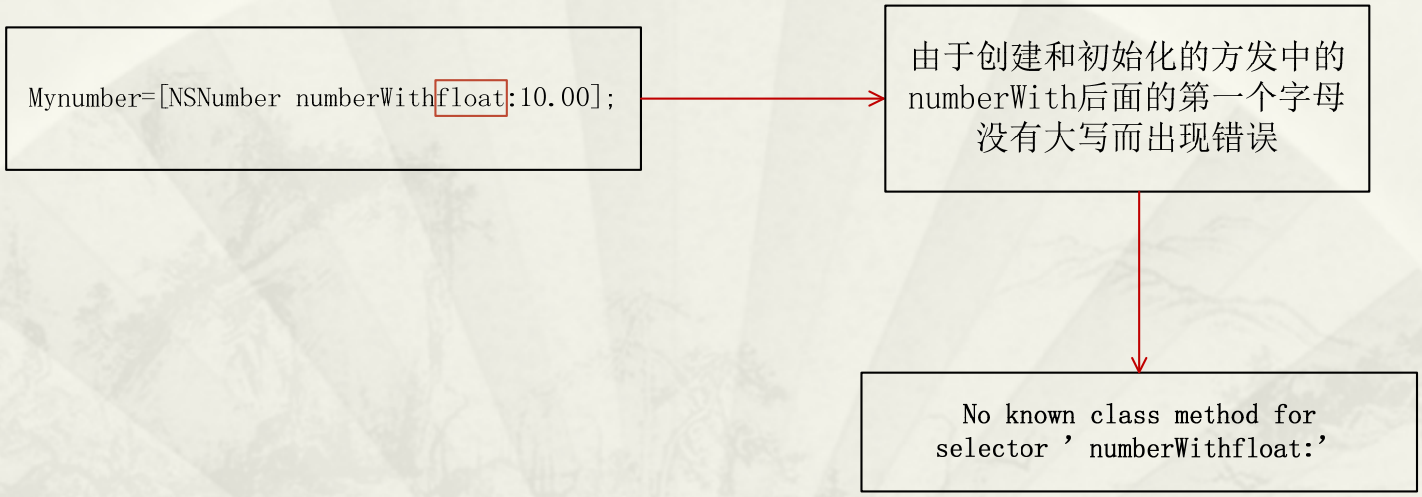
2. 数字对象的创建并初始化

名称	创建和初始化类方法
字符型对象的创建和初始化	numberWithChar:
无符号字符型对象的创建和初始化	numberWithUnsignedChar:
短整型对象的创建和初始化	numberWithShort:
无符号短整型对象的创建和初始化	numberWithUnsignedShort:
整型对象的创建和初始化	numberWithInteger:
无符号整型对象的创建和初始化	numberWithUnsignedInteger:
整型对象的创建和初始化	numberWithInt:initWithInt:
无符号整型对象的创建和初始化	numberWithunsignedInt:
长整型对象的创建和初始化	numberWithLong:
无符号长整型对象的创建和初始化	numberWithUnsignedLong:
长长整型对象的创建和初始化	numberWithLongLong:
无符号长长整型对象的创建和初始化	numberWithUnsignedLongLong:
浮点型对象的创建和初始化	numberwithFloat:
双精度浮点型对象的创建和初始化	numberWitnDouble:
布尔型对象的创建和初始化	numberWithBool:

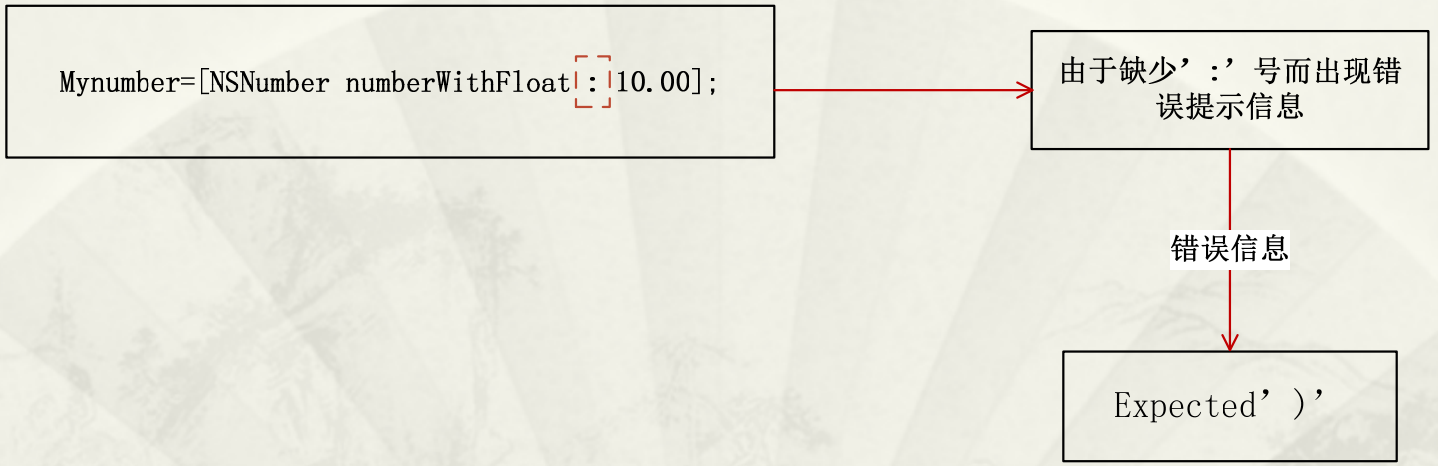
2. 数字对象的创建并初始化



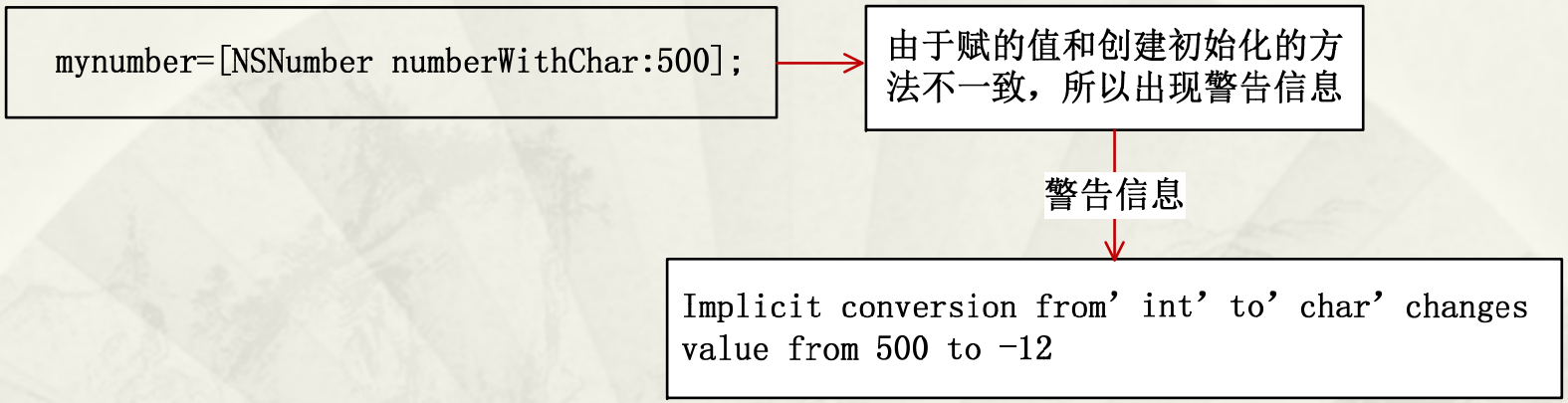
2. 数字对象的创建并初始化



2. 数字对象的创建并初始化

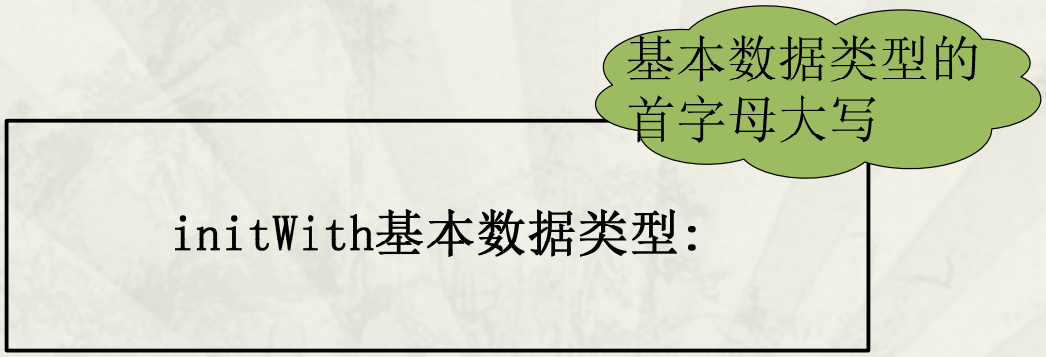


2. 数字对象的创建并初始化



3. 数字对象的初始化

* 初始化的方法。



3. 数字对象的初始化

名称	初始化示例方法
字符型对象的初始化	initWithChar
无符号字符型对象的初始化	initWithUnsignedChar
短整型对象的初始化	initWithShort
无符号短整型对象的初始化	initWithUnsignedShort
整型对象的初始化	initWithInteger
无符号整型对象的初始化	initWithUnsignedInteger
整型对象的初始化	initWithInt
无符号整型对象的初始化	initWithUnsignedInt
长整型对象的初始化	initWithLong
无符号长整型对象的初始化	initWithUnsignedLong
长长整型对象的初始化	initWithLongLong
无符号长长整型对象的初始化	initWithUnsignedLongLong
浮点型对象的初始化	initWithFloat
双精度浮点型对象的初始化	initWithDouble
布尔型对象的初始化	initWithBool

3. 数字对象的初始化

```
数字对象名=[[NSNumber alloc]]初始化方法 初始化值];
```

4.数字对象的取值

* 取值方法的表示形式。

4.数字对象的取值

名称	取值实例方法
字符型对象的取值	charValue
无符号字符型对象的取值	unsignedCharValue
短整型对象的取值	short Value
无符号短整型对象的取值	unsignedShortValue
整型对象的取值	integerValue
无符号整型对象的取值	unsignedIntegerValue
整型对象的取值	intValueunsigned
无符号整型对象的取值	unsignedIntValue
长整型对象的取值	longValue
无符号长整型对象的取值	UnsignedLongValue
长长整型对象的取值	longlongValue
无符号长长整型对象的取值	unsignedLongLongValue
浮点型对象的取值	floatValue
双精度浮点型对象的取值	doubleValue
布尔型对象的取值	boolValue

4.数字对象的取值

```
[对象名  取值方法];
```

10.2　字符串对象

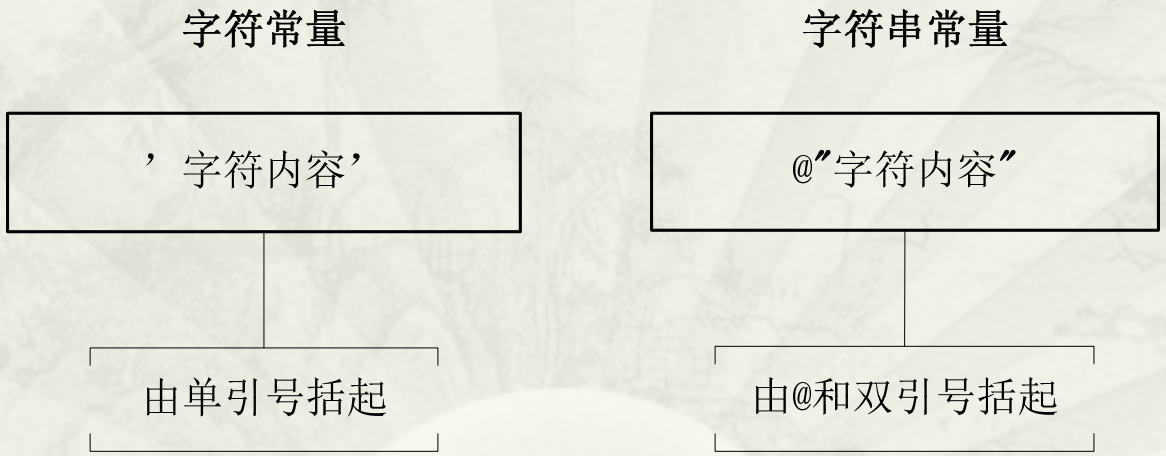
- Foundation框架中所提到的字符串类型，可以通过使用NS字符串类型将基本的字符串封装到对象中。然后对相应的对象进行操作。字符串对象可以分为可修改和不可修改两类。本节将主要讲解字符串对象的使用。

10.2.1　字符串和字符的区别

* 字符串常量和字符常量是不同的常量，它们之间的区别如下。

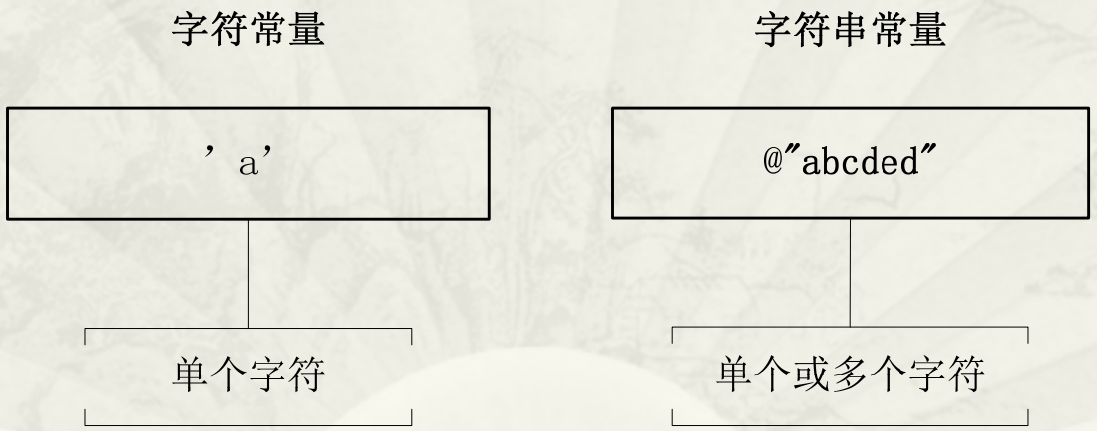
1组成格式

* 字符常量是由单引号括起来的，字符串常量使用@符号和双引号括起来。



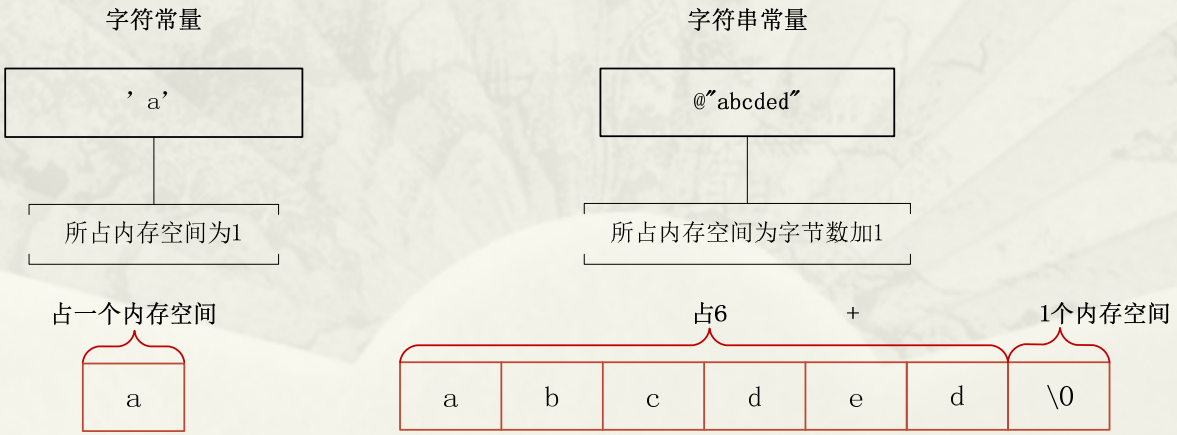
2. 字符的多少

* 字符常量只能是单个字符，字符串常量可以是单个或多个字符。



3. 占用的内存空间

- 字符常量占一个字节的内存空间。字符串常量占的内存字节数为字符串的字节数加1，其中加的一个字节中存放字符"\0"即结束标志。



10.2.2 不可修改的字符串

- 在Objective-C使用NSString类来操作字符串。在对字符串使用之前，我们要将字符串的头文件加入代码中。

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

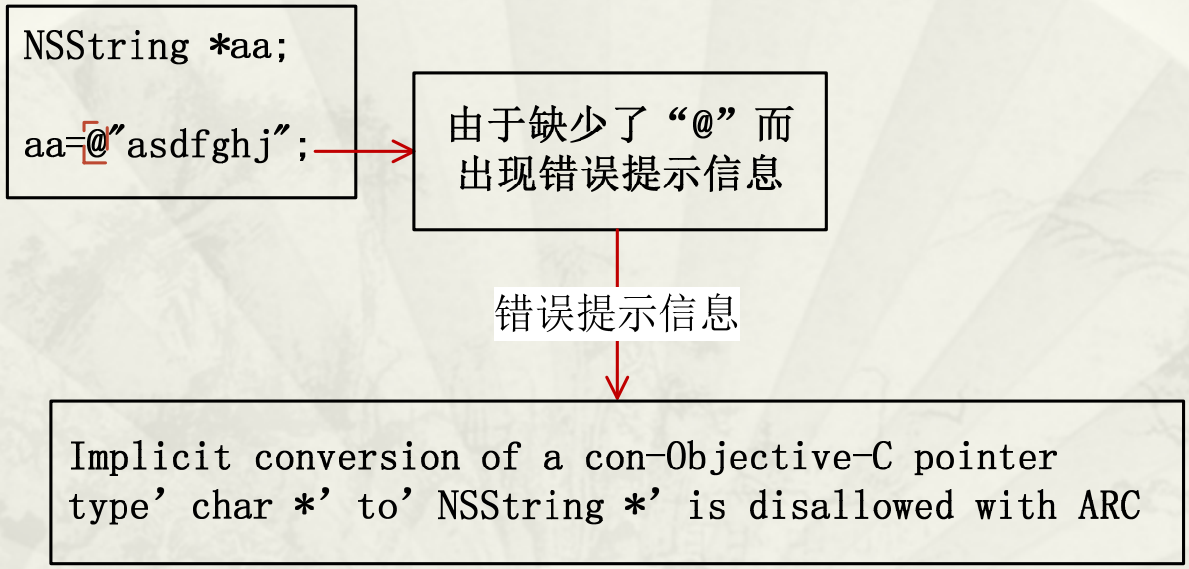
10.2.2　不可修改的字符串

```
NSString *标识符;
```

10.2.2　不可修改的字符串

```
标识符=@"需要赋的值";
```

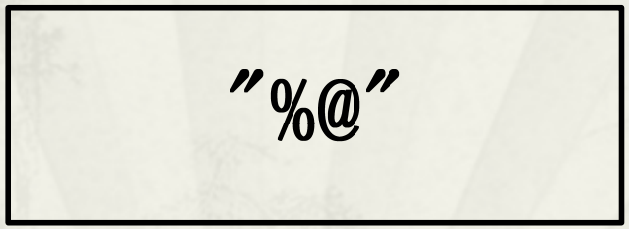
10.2.2 不可修改的字符串



10.2.2　不可修改的字符串

```
NSString *标识符=@"需要赋的值";
```

10.2.2　不可修改的字符串



1.stringWithString

- stringWithString的功能是用一个字符串对象来创建另一个字符串对象。

```
stringWithString:(NSString *)string
```

2. stringWithFormat

- stringWithFormat功能是用NSLog()格式创建字符串。

```
stringWithFormat:nsstring
```

3. stringByAppendingString

- stringByAppendingString的功能是在一个字符串后面增加一个新字符串。

```
stringByAppendingString:nsstring
```

4. 字符串的比较和判断

- isEqualToString的功能是比较两个字符串是否相等。

```
isEqualToString:nsstring
```

4. 字符串的比较和判断

```
hasPrefix:nsstring
```

5. 字符串的大小写转换

- uppercaseString的功能是将小写字母转为大写字母。

```
字符串变量2=[字符串变量1 uppercaseString];
```

5. 字符串的大小写转换

```
字符串变量2=[字符串变量1 lowercaseString];
```

6. substringToIndex

- substringToIndex主要的功能是截取字符串。

```
字符串变量2=[字符串变量1 substringToIndex:n];
```

需要截取的字符串的长度

6. substringToIndex

方法	说明
+(id)stringWithContentsOfFile:path encoding:enc error err	创建一个新字符串并将其设置为path指定的文件的内容，使用字符编码enc，在err上返回错误
+(id)stringWithContentsOfURL:url encoding:enc error:err	创建一个新字符串，并将其设置为url所指定的内容，使用字符编码enc，在err上返回错误
+(id)string	创建一个新的空字符串
-(id)initWithString:nsstring	创建一个新的空字符串，并将其内容设置设置为nsstring内容
(id)initWithContentsOfFile:path encoding:enc error:err	将字符串设置为path指定的文件的内容
-(id)initWithContentsOfURL:url encoding:enc error:err	将字符串设置为url所指定的内容，使用enc字符编码，在err上返回错误
-(UNSIgned int)length	返回字符串中的字符数目
-(unichar)characterAtIndex:i	返回索引i虽在的Unicode字符
-(NSString *)substringFromIndex:i	返回从i开始到结尾的子字符串
-(NSString *)substringWithRange:range	根据指定范围返回子字符串
-(NSString *)substribgToIndex:i	返回从字符串开始位置到i的子字符串
-(NSComparator *)caseInsensitiveCompare:nsstring	比较两个字符串（忽略大小写）
-(NSComparator *)Compare:nsstring	比较两个字符串的大小
-(BOOL)hasPrefix:nsstring	测试字符串是否以nsstring开始
-(BOOL)hasSuffix:nsstring	测试字符串是否以nsstring结尾
-(BOOL)isEqualToString:nsstring	测试两个字符串是否相等
-(NSString *)caoitalizedString	返回字符串，串中的每个单词的首字母大写，其余字母小写
-(NSString *)lowercaseString	返回转换为小写的字符串
-(NSString)uppercaseString	返回转换为大写的字符串
-(const char *)UTF8String	返回UTF8编码格式的字符串
-(double)doubleValue	返回转换为double类型的字符串
-(float)floatValue	返回转换为float类型的字符串
-(NSInteger)integerValue	返回转换为NSInteger类型的字符串
-(int)intValue	返回转换为int类型的字符串

10.2.3　可修改的字符串

- 使用NS字符串NSString的时候。字符串对象是不可以进行任何修改的。如果需要对字符串对象进行插入、删除或者其他的操作时，就需要使用NS可变的字符串NSMutableString。NSMutableString是NSString的子类。所以NSMutableString继承了NSString的所有类方法和对象方法。

10.2.3　可修改的字符串

```
NSMutableString *字符串变量;
```

10.2.3 可修改的字符串

stringWithString方法也是用与可修改的字符串对象

```
NSString *string1=@"ABCDEF";

NSMutableString *string2=[NSMutableString stringWithString:string1]
```

1. appendString

- appendString的功能是将一个字符串的末尾附加一个字符串。

```
[字符串变量名1 appendString:字符串或变量名];
```

2. appendFormat

- appendFormat的功能是附加一个格式化字符串。它的使用使用形式如图10.44所示。

```
[字符串变量1 appendFormat:@"%@",字符串变量2];
```

2. appendFormat

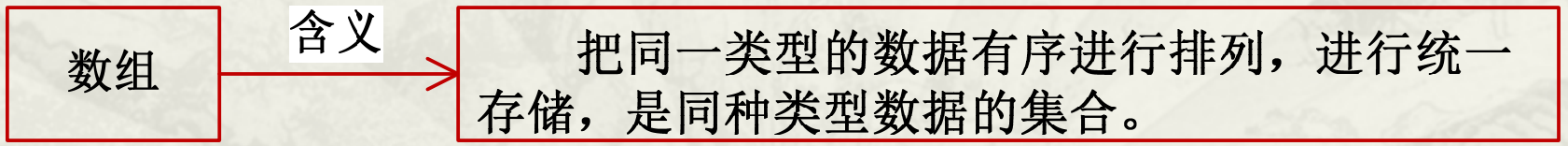
方法	功能
+(id)stringWithCapacity:size	创建一个字符串，size个字符容量
-(id)initWithCapacity:size	初始化一个字符串。size个字符容量
-(void)setString:nsstring	将字符串设置为nsstring
-(void)appendString:nsstring	在一个字符串末尾附加一个字符串nsstring
-(void)deleteCharactersInRange:range	删除指定rang中的字符
-(void)insertString:nsstring aTIndex:i	以i为起始位置插入nsstring
-(void)replaceCharactersInRange:rang withString:nsstring	使用nsstring代替range指定的字符串
-(void)replaceOccurrencesOfString:nsstring withString:nsstring2 Options:ops range:range	根据指定选项opts，使用指定range中的nsstring2替换所有的nsstring

10.3 数组对象

* 在程序设计中数组是使用非常频繁的。Foundation框架提供了NS数组类型，NS数组类型包括可修改的和不可修改的数组两类，只能存放相关的对象类型，提供了基本的数组操作方法，对数组的相关操作进行了封装。本节主要讲解数组对象的相关操作。

10.3.1 数组的定义

* 在程序设计的过程中，经常会处理一些数据类型相同的变量，为了方便，Objective-C提供了数组这一结构。



10.3.2　数组的声明

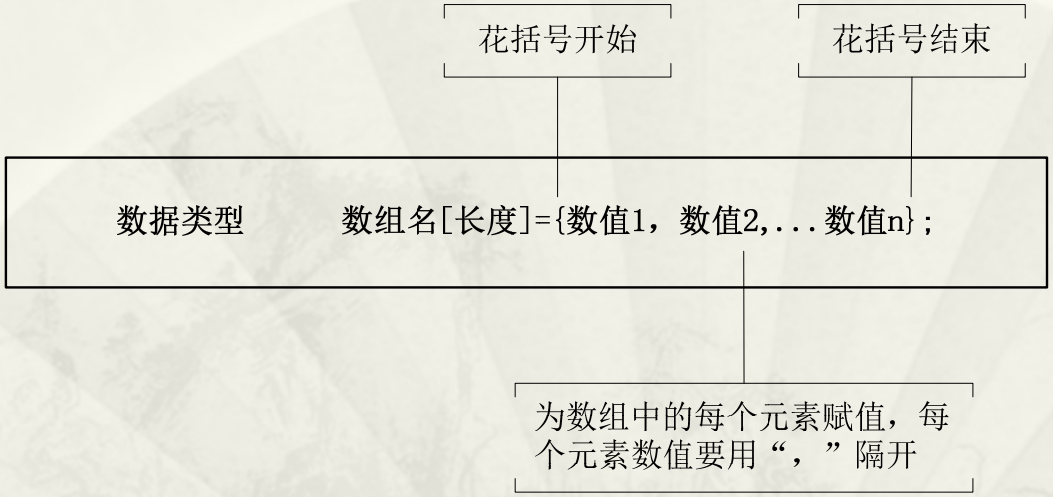
* 数组的声明和变量的声明类似。

```
数组类型 数组名[数组大小];
```

10.3.2　数组的声明

```
int score[5];
```

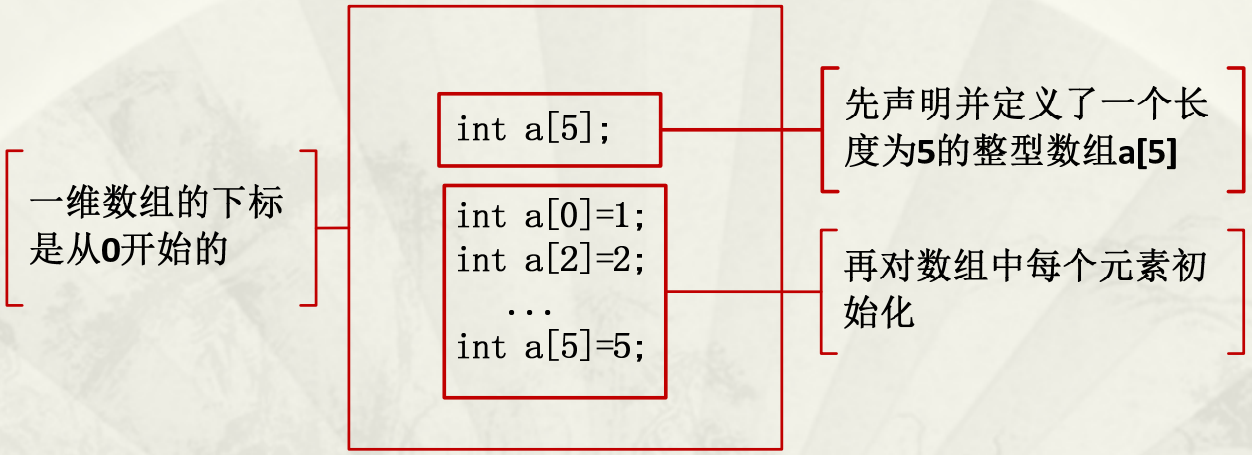
10.3.2 数组的声明



10.3.2　数组的声明

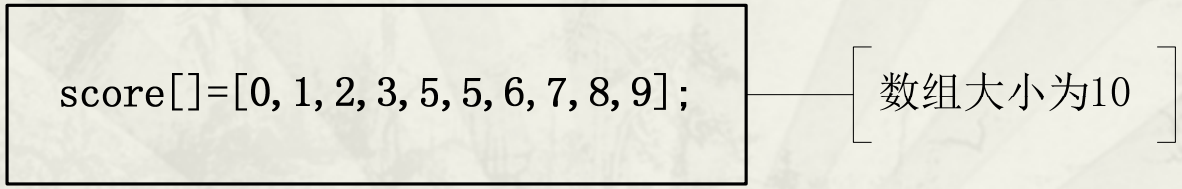
```
int score[5]={1,2,3,4,5};
```

10.3.2 数组的声明


int a[5];
int a[0]=1;
int a[2]=2;
...
int a[5]=5;
一维数组的下标是从0开始的
先声明并定义了一个长度为5的整型数组a[5]
再对数组中每个元素初始化

1. 数组大小

* 若对数组中的所有元素都赋予了初始值，可以不用指定数组的大小，系统将自动根据赋值的个数来确定数组的大小。



2. 部分元素初始化

* 若只对数组中的部分元素赋予初始值，则系统会自动为其他元素赋初始值0。

3. 不初始化

* 若只声明数组，而不为数组赋值，则数组中的元素值是不确定的。

4. 数组元素的大小

* 数组的大小只能是常量，而不能使用变量。

10.3.3 数组的引用

- 在Objective-C语言中，一维数组的引用其实就是对一维数组元素的使用。

数组名[下标]

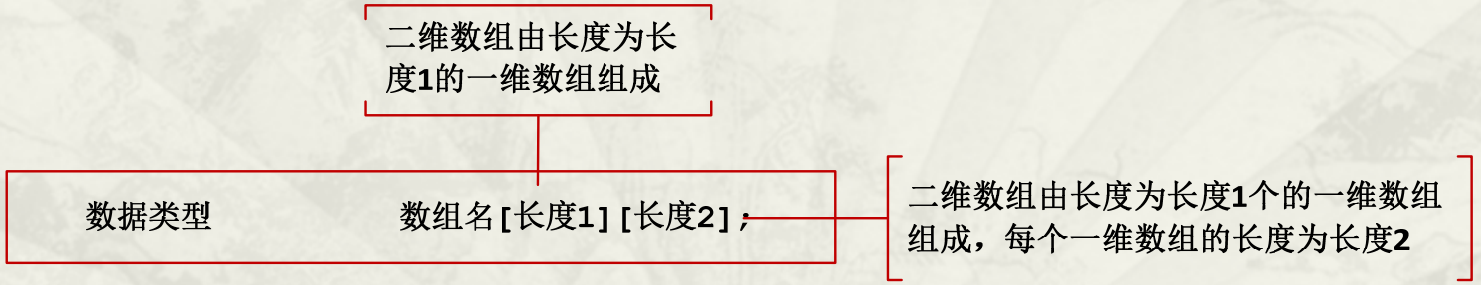
10.3.3 数组的引用

int a[i]

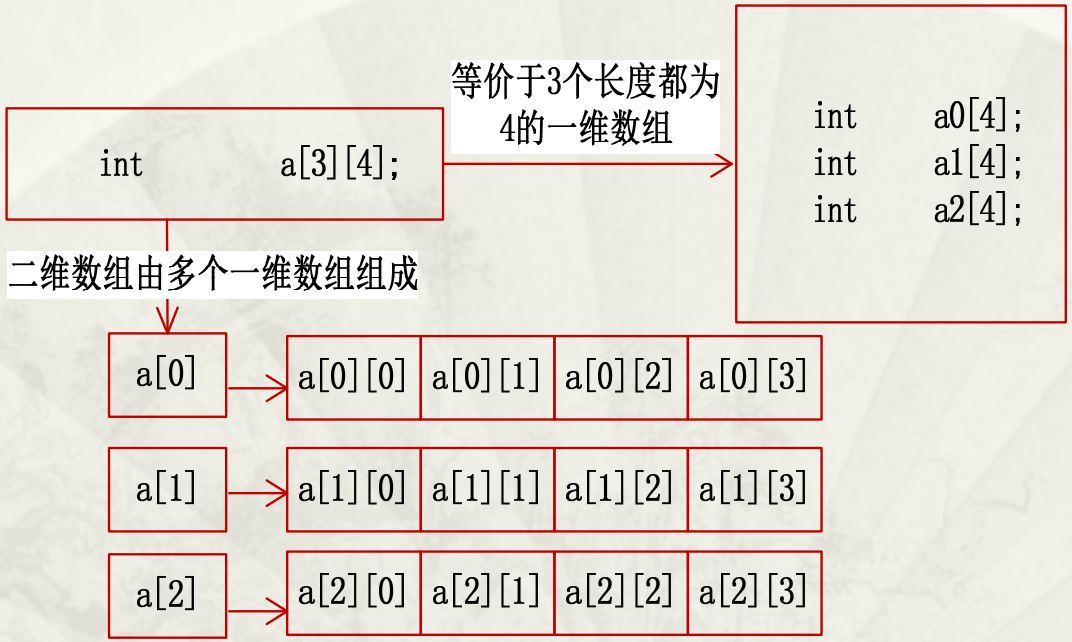
数组名为a，长度为n。数组的下标的下界是0，上界是n-1，0<=i<n

10.3.4 二维数组的声明和定义

- 二维数组有两个下标。二维数组中的元素和一维数组中的元素一样，具有同样的数据类型。



10.3.4 二维数组的声明和定义



10.3.5　二维数组初始化的方法

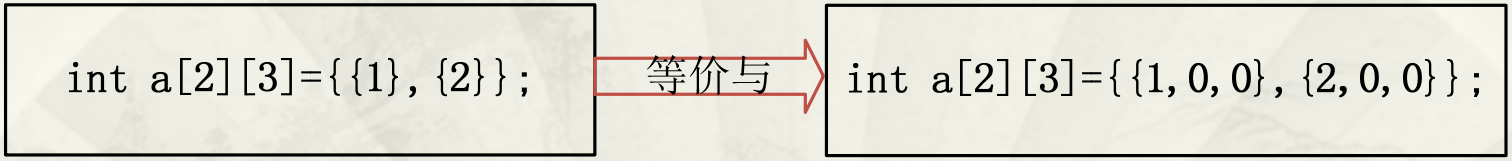
* 二维数组初始化的形式时多种多样，接下来就常用的几种形式给大家做一简单的讲解。

```
int a[2][3]={{0,1,2},{3,4,5}};
```

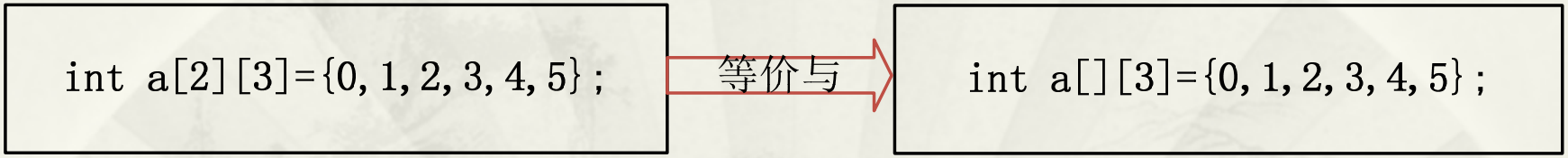
10.3.5　二维数组初始化的方法

```
int a[2][3]={0,1,2,3,4,5};
```

10.3.5 二维数组初始化的方法



10.3.5 二维数组初始化的方法



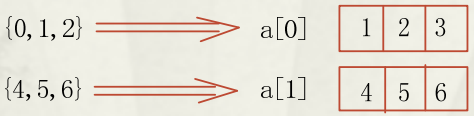
10.3.6　二维数组的引用

* 二维数组的引用形式。

```
数组名　　[下标1][下标2]
```

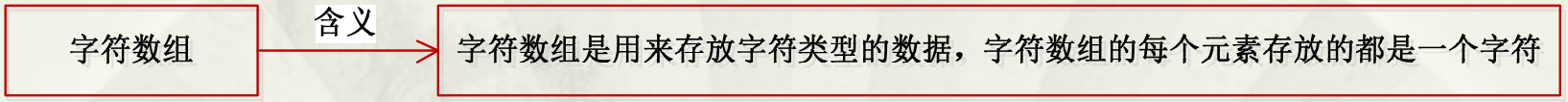
10.3.6　二维数组的引用

int a[2][3]={{0,1,2},{4,5,6}}



10.3.7　字符数组的含义

* 字符数组的含义。



10.3.7　字符数组的含义

一维字符数组

```
char        字符数组名[长度];
```

二维字符数组

```
char        字符数组名[下标1][下标2];
```

10.3.8　字符数组初始化

* 字符数组初始化分为两种，一种是逐个给字符数组赋值，一个种是用字符串直接给字符数组赋值。接下成我们主要讲解这两种赋值方式。

1. 逐个为字符数组赋值

* 每个字符会依次赋值给数组中的每个元素。

```
char c[6]={ 'H', 'e', 'l', 'l', 'o'};
```

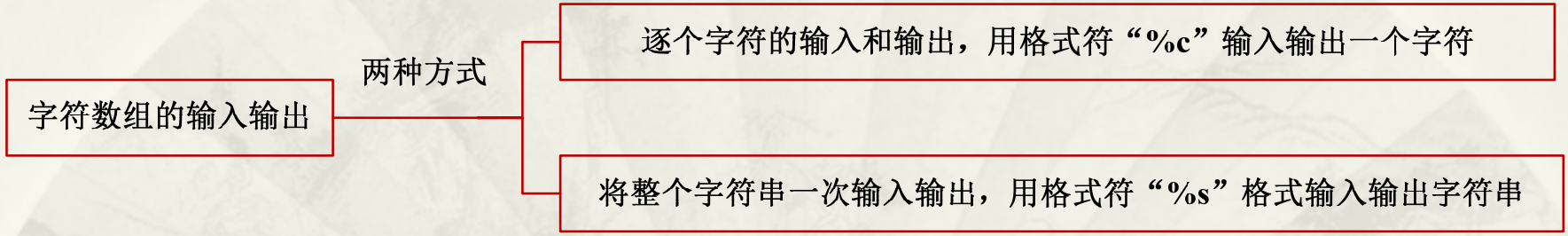
2. 用字符串直接给字符数组赋值

* 直接将字符串赋给字符数组来初始化。系统会依次读取字符串中的每个字符，赋值给数组每个元素，并在字符串的的结尾处加一个字符即结束符\0作为一个数组元素。

```
char c[6]={ "Hello"};
```

10.3.9 字符数组的引用

* 字符数组的输入输出有来那个中形式。



10.3.10 不可修改的数组

* 当我们要使用不可修改的数组时，就要使用NSArray来进行操作。在程序中要使用数组对象就要在开头插入数组对象的头文件。

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

1.nil和Nil

- nil在objc/objc.h中定义。nil表示一个Objective-C对象，这个对象所指的内容为空。

```
#define nil 0  /*id of Nil instance*/
```

1.nil和Nil

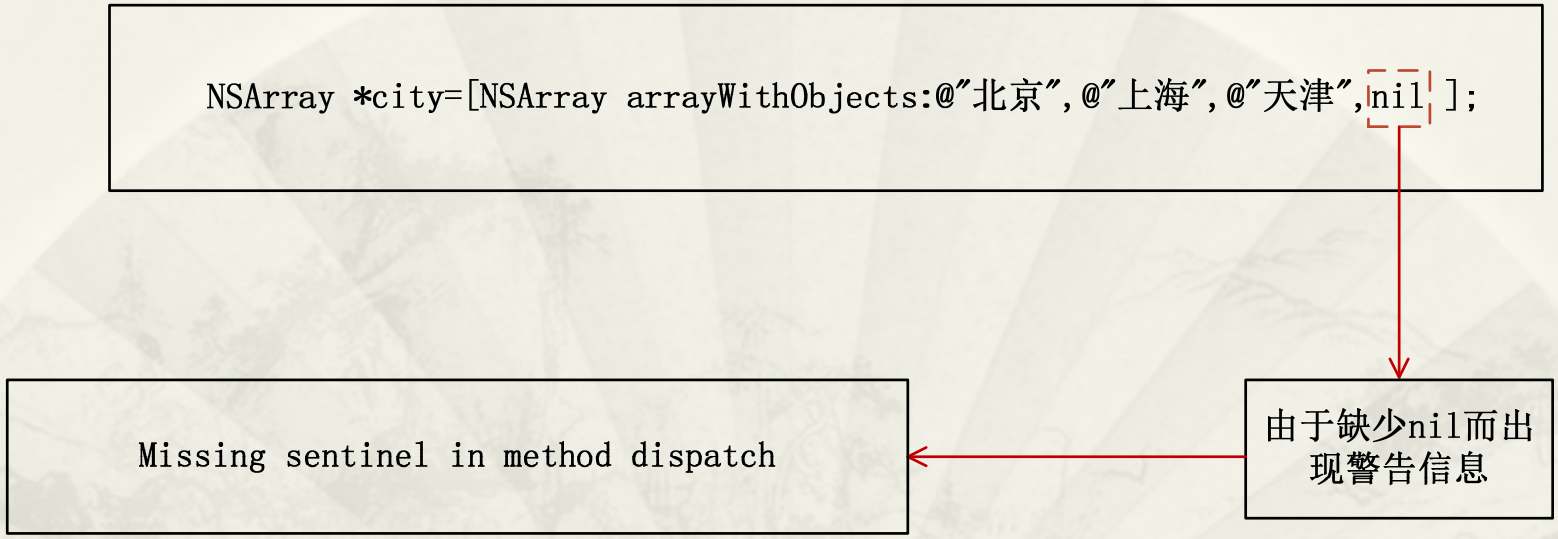
```
#define Nil 0   /*id of Nil class*/
```

2. 声明和创建数组

- 在不可修改的数组中，我们可用“arrayWithObjects”类方法对数组进行初始化。例如我们给一数组对象city进行创建和初始化。

```
NSArray *city=[NSArray arrayWithObjects:@"北京", @"上海", @"太原",nil];
```

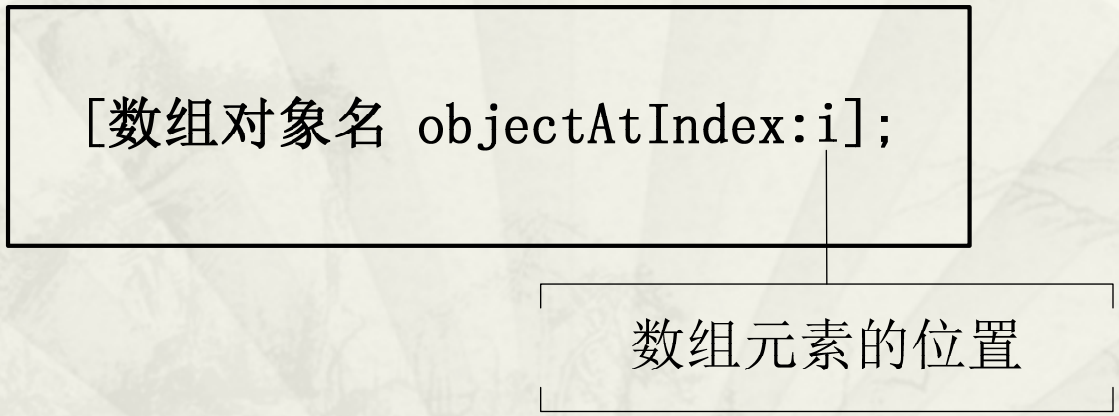
2. 声明和创建数组


NSArray *city=[NSArray arrayWithObjects:@"北京",@"上海",@"天津",nil];
由于缺少nil而出现警告信息
Missing sentinel in method dispatch

3.不可变的数组的使用

```
[数组对象名 count];
```

3.不可变的数组的使用



3.不可变的数组的使用

```
[数组对象名 indexOfObject:对象];
```

3.不可变的数组的使用

方法	说明
+(id)arryWithObjects:obj1,obj2,…nil	创建一个新的数组，obj1,obj2…是他的数组元素对象，以nil对象结尾
-(BOOL)containsObject:obj	确定数组中是否包含对象obj
-(NSUInteger)count	数组中元素的个数
-(NSUInteger)indexOfObject:obj	第一个包含数组元素的索引号
-(id)objectAtIndex:i	存储在位置i的对象
-(void)makeObjectsPerformSelector:(SEL)selector	将selector指示的消息发送给数组中的每个元素
-(NSArray*)sortedArrayUsingSelector:(SEL)selector	根据selector指定的比较方法对数组进行排序
-(BOOL)writeToFile:path atomically:(BOOL)flag	将数组写入指定的文件中，如果flag为YES,则需要先创建一个临时文件

10.3.11 可修该的数组

- NSArray是一个静态的使用，我们不能在数组中添加和删除元素。所以Objective-C提供了NSMutableArray来动态的管理这些数组。NSMutableArray是NSArray的子类，所以继承了NSArrayde中的所有方法，又有自己独特的方法。接下来我们主要讲解NSMutableArray中的这些独特的方法。

1. 创建和初始化可修改的数组

* 创建和初始化为可修改的数数组的形式。

```
NSMutableArray *数组对象名=[[NSMutableArray alloc]init];
```

2. 添加一个元素

* addObject是在数组中添加一个元素。

```
[数组对象名 addObject:添加的元素];
```

3. 删除指定的元素

- removeObject是从数组中删除指定的元素。

```
[数组对象名 removeObject:要删除的元素];
```

4. 删除所有元素

- removeAllObjects是将数组中的所有元素删除。

```
[数组对象名 removeAllObjects];
```

5. 在指定位置添加新元素

- inserObject是在指定位置为数组添加新元素。

```
[student insertObject:元素 atIndex:要插入的位置];
```

5. 在指定位置添加新元素

方法	说明
+(id)array	创建一个空数组
+(id)arrayWithCapacity:size	创建一个数组，指定容量为size
+(id)initCapacity:size	初始化一个新分配的数组，指定容量为size
-(void)addObject:obj	将对象obj添加到数组末尾
-(void)inserObject:obj atIndex:i	将对象obj插入数组的i元素
-(void)replaceObjectAtIndex:i withObject:obj	将数组中序号为i的对象用对象obj替换
-(void)removeObject:obj	从数组中删除所有是obj的对象
-(void)removeObjectAtIndex:i	从数组中删除索引为i的对象
-(void)sortUsingSelector:(SEL)selector	用selector指示的比较方法将数组排序

10.4 字典对象

* 我们可以用数组存放有序对象的集合，但是要存放具有关键字的对象的集合使用数组是不够的。所以Objective-C语言用提供了一个新的对象叫做字典。使用字典可以存放具有关键字的对象的集合。字典可分为不可修改的字典和可修改的字典两种。本节将主要讲解字典的相关知识。

10.4.1 不可修改的字典

* 字典中的每一项都有都有一个关键字和一个值，简称“键-值”。关键字在每个字典内是唯一的，但值可以是相同的，也可以是不同的。当我们要使用不可修改的字典时，就要使用NSDictionary来进行操作。在程序中要使用字典就要在开头插入字典的头文件。

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

1. 字典的创建和初始化

* 在不可修改的字典中，我们可用“dictionaryWithObjectsAndKeys”类方法进行初始化。例如我们employees字典进行创建和初始化。

```
[NSDictionary dictionaryWithObjectsAndKeys:@"Tom",@"1",@"lyy",@"2", nil];
```

2. 获得字典中“键-值”的个数

* count是获得字典中“键-值”的个数。

```
[字典对象名 count];
```

3.查找某一关键字对应的值

- objectForKey是在字典中查找某个关键字所对应的值。如果所对应关键字的值不存在，就返回nil。

```
[字典对象名 objectForKey:关键字];
```

3.查找某一关键字对应的值

方法	说明
+(id)dictionaryWithObjectsAndKeys: obj1,key1,obj2,key2,…nil	顺序添加对象和键值来创建字典，注意结尾是nil
-(id)initWithObjectsAndKeys: obj1,key1,obj2,key2,…nil	初始化一个新分配的字典，顺序添加关键字和值，结尾是nil
-(unsigned int)count	返回字典中“键-值”对数
-(NSEnumerator *)keyEnumerator	返回字典中所有的键到一个NSEnumerator对象
-(NSArray*)keysSortedByValueUsing Selector:(SEL)selector	将字典中所有的键按照selector指定的方法进行排序，并将排序的结果返回
-(NSEnumerator *)objectEnumerator	返回字典中所有的值到一个NSEnumerator类型对象
-(id)objectForKey:key	返回指定键的值

10.4.2. 可修改的字典

- 不可修改的字典和前面所说的不可修改的数组一样，不能动态的删除和添加，可修改的字典（NSDictionary）解决了这一难题。接下来我们主要讲解可修改字典的一些功能。

1. 动态字典的声明和初始化

* 动态字典声明和初始化的形式。

```
NSMutableDictionary *字典对象名=[[NSMutableDictionary alloc]init];
```

2. 设置值和键

- setObject是用来设置值和键的。

```
[字典对象名 setObject:值 forKey:键];
```

3. 删除键所指定的值

- removeObjectForkey是用来删除键所指定的值的。

```
[student removeObjectForKey:键];
```

4. 删除所有的元素

- removeAllObjects是删除字典对象中的所有的元素。

```
[字典对象 removeAllObjects];
```

4. 删除所有的元素

方法	说明
+(id)dictionaryWithCapacity:size	创建一个size大小的可修改字典
-(id)initWithCapacity:size	初始化一个size大小的可修改字典
-(void)removeAllObjects	删除字典中的所有元素对象
-(void)removeObjectForKey:key	删除字典中的key位置的值
-(void)setObject:obj forKey:key	添加（key，obj）到字典中；若key已存在，则替换值为Obj

10.5 集合对象

- NSArray类提供了一个可用于存储有序对象的集合的集合，NSSet提供了一个可用于无序对象的集合的集合。在集合中分为可修改的集合和不可修改的集合两种。本节将主要讲解关于集合对象操作的相关知识。

10.5.1 不可修改的集合

* 当我们要使用不可修改的集合时，就要使用NSSet来进行操作。在程序中要使用集合就要在开头插入集合的头文件。

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

1. 不可修改的集合对对象的创建和初始化

- 在不可修改的集合中，我们可用“setWithobjects”类方法进行初始化。例如，我们创建了一个set的集合，将该集合进行初始化。

```
NSSet *set=[NSSet setWithObjects:@"fool",@"bar",@"baz", nil];
```

2. 获得集合元素的个数

* count在集合中的功能功能是获得集合中元素的个数。

```
[集合对象 count];
```

3. 判断两个集合是否相等

- isEqualToSet是判断两个集合是否相等，如果BOOL就为“YES，不相等BOOL的值就为“NO”。

```
[集合对象1 isEqualToSet:集合对象2];
```

3. 判断两个集合是否相等

方法	说明
+(id)setWithObjects:obj1,obj2,…nil	使用一组元素对象创建新集合
-(id)initWithObjects:obj1,obj2,…nil	使用一组元素对象初始化新分配的集合
-(NSUInteger)count	返回集合的元素个数
-(BOOL)containsObject:obj	确定集合是否包含元素对象obj
-(BOOL)member:obj	确定集合是否包含元素对象obj
-(NSEnumerartor *)objectEnumerator	返回集合中所有元素对象到一个NSEnumerartor类型的对象
-(BOOL)isSubsetOfSet:nsset	判断是否是一个集合nsset的子集
-(BOOL)intersectsSet:nsset	判断两个集合的交集是否存在至少一个元素
-(BOOL)isEqualToSet:nsset	判断两个集合是否相等

10.5.2　可修改的集合

* 我们需要对集合中的元素对象进行添加删除等操作在NSSet中是不行的，所以我们来学习可修改的集合NSMutableSet的操作。

1.创建可修改的集合

* 我们在使用可修改的集合时，必须要创建集合，创建集合的形式有两种，一个是不带任何参数的创建，一个是带有固定长度的创建。

```
NSMutableSet *集合对象名=[NSMutableSet set];
```

1.创建可修改的集合

```
NSMutableSet *集合对象名=[NSMutableSet setWithCapacity:size];
```

集合的大小，即元素个数

2. 将元素对象添加到集合中

* addObject是在集合中添加元素对象。

```
[集合对象名 addObject:元素对象];
```

3. 删除集合中的元素对象

- removeObject是删除集合中的元素对象。

```
[集合对象名 removeObject:元素对象];
```

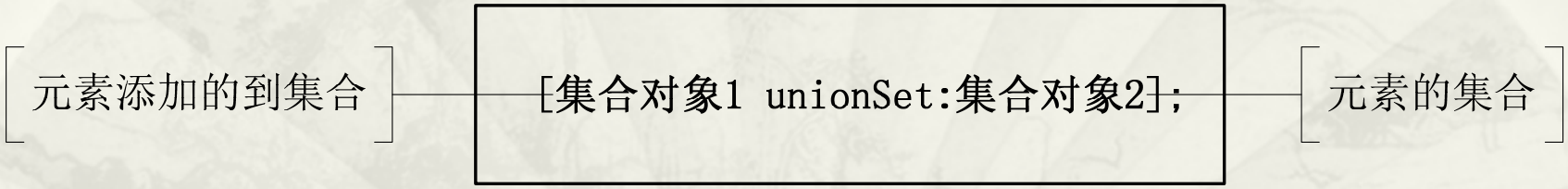
4. 删除集合中的所有元素

- removeAllObjects是将集合中的所有元素删除。

```
[集合对象名 removeAllObjects];
```

5. 在集合中元素对象添加到另一个集合中

- unionSet是将一个集合对象的元素添加到另一个集合对象中。



5. 在集合中元素对象添加到另一个集合中

方法	说明
-(id)setWithCapacity:size	创建一个有size大小的新集合
-(id)initWithCapacity:size	初始化一个新分配的集合，大小size
-(void)addObject:Obj	将元素对象添加到集合中
-(void)removeObject:obj	从集合删除指定的元素对象obj
-(void)removeAllObjects	删除集合中所有的元素对象
-(void)unionSet:nsset	将nsset（另一个集合）的所有元素对象添加到集合
-(void)minusSet:nsset	从集合中去掉所有nsset（另一个集合）中的元素
-(void)interectSet:nsset	集合和nsset另一个集合）做交集

10.6 小结

* 本章主要讲解了一些关于Foundation框架的基本数据类型，大家要了解数字对象的使用。本章的重点是通过可以对字符串对象、数组对象、字典对象以及集合的不可修改的类和可修改的类的使用中熟练掌握它们其中的方法。