Objective C类方法load和initialize的区别

过去两个星期里，为了完成一个工作，接触到了NSObject中非常特别的两个类方法(Class Method)。它们的特别之处，在于iOS会在运行期提前并且自动调用这两个方法，而且很多对于类方法的规则（比如继承，类别(Category)）都有不同的处理。

而因为这两个方法是在程序运行一开始就被调用的方法，我们可以利用他们在类被使用前，做一些预处理工作。比如我碰到的就是让类自动将自身类名保存到一个NSDictionary中。

先来看看NSObject Class Reference里对这两个方法说明：

+(void)initialize

> The runtime sends `initialize` to each class in a program exactly one time just before the class, or any class that inherits from it, is sent its first message from within the program. (Thus the method may never be invoked if the class is not used.) The runtime sends the initialize message to classes in a thread-safe manner. Superclasses receive this message before their subclasses.

+(void)load

> The `load` message is sent to classes and categories that are both dynamically loaded and statically linked, but only if the newly loaded class or category implements a method that can respond.
> The order of initialization is as follows:
>
> 1. All initializers in any framework you link to.
> 2. All `+load` methods in your image.
> 3. All C++ static initializers and C/C++ `__attribute__(constructor)` functions in your image.
> 4. All initializers in frameworks that link to you.
>
> In addition:
>
> - A class's `+load` method is called after all of its superclasses' `+load` methods.
> - A category `+load` method is called after the class's own `+load` method.
>
> In a custom implementation of `load` you can therefore safely message other unrelated classes from the same image, but any `load` methods implemented by those classes may not have run yet.

Apple的文档很清楚地说明了initialize和load的区别在于：load是只要类所在文件被引用就会被调用，而initialize是在类或者其子类的第一个方法被调用前调用。所以如果类没有被引用进项目，就不会有load调用；但即使类文件被引用进来，但是没有使用，那么initialize也不会被调用。

它们的相同点在于：方法只会被调用一次。（其实这是相对runtime来说的，后边会做进一步解释）。

文档也明确阐述了方法调用的顺序：父类(Superclass)的方法优先于子类(Subclass)的方法，类中的方法优先于类别(Category)中的方法。

不过还有很多地方是文章中没有解释详细的。所以再来看一些示例代码来明确其中应该注意的细节。

+(void)load与+(void)initialize初探

```
1 +(void)load会引发+(void)initialize
2 /******* Interface *******/
```

```objc
@interface SuperClass : NSObject
@end

@interface ChildClass : SuperClass
@end

@interface Insideinitialize : NSObject
- (void)objectMethod;
@end

/******* Implementation *******/

@implementation SuperClass

+ (void) initialize {
    NSLog(@"%@ %s", [self class], __FUNCTION__);
}

+ (void) load {
    NSLog(@"%@ %s", [self class], __FUNCTION__);
}

@end

@implementation ChildClass

+ (void) initialize {
    NSLog(@"%@ %s", [self class], __FUNCTION__);
    Insideinitialize * obj = [[Insideinitialize alloc] init];
 [obj objectMethod];
 [obj release];
}

@end

@implementation Insideinitialize

- (void)objectMethod {
    NSLog(@"%@ %s", [self class], __FUNCTION__);
}

+ (void) initialize {
    NSLog(@"%@ %s", [self class], __FUNCTION__);
}

+ (void) load {
    NSLog(@"%s", __FUNCTION__);
}

@end
```

这个示例代码中，一个SuperClass实现了`+(void)load`和`+(void)initialize`方法（实际上应该算是重写覆盖了NSObject的这两个方法）；ChildClass继承于SuperClass，但是只重写`+(void)initialize`没有`+(void)load`；Insideinitialize类也有`+(void)load`和`+(void)initialize`方法，它在ChildClass的i`+(void)initialize`方法中被构建出一个对象。类中的每个函数的实现都非常简单，只是输出类名和方法名。除了Insideinitialize的`+(void)load`方法只输出了类名，没有使用[self class]。

首先我们在Xcode的项目中只简单import这些类，而不去使用他们的，然后运行项目就会得到下边的结果：

```
SuperClass +[SuperClass initialize]
SuperClass +[SuperClass load]
Insideinitialize +[Insideinitialize load]
```

就像Apple的文档中说的一下，只要有引用runtime就会自动去调用类的`+(void)load`方法。不过从输出中，我们还发现SuperClass的`+(void)initialize`也被调用了，而且是在`+(void)load`之前被执行；而Insideinitialize的`+(void)initialize`并没有执行。这是因为在SuperClass的`+(void)load`方法中，我们调用了类的class方法（[self class]），这就符合文档中对`+(void)initialize`的说明：在类的第一个方法被调用前调用。同时也说明run time对`+(void)load`的调用并不视为类的第一个方法。而ChildClass因为没有用到，所以`+(void)initialize`的方法被没有被执行，而且它也没有去执行父类的`+(void)load`方法（虽然它有继承下该方法）。

+(void)load和+(void)initialize可当做普通类方法(Class Method)被调用

接着， 在程序中让ChildClass直接调用load:

```
[ChildClass load];
```

程序正常运行，并输出了结果：

```
SuperClass +[SuperClass initialize]
SuperClass +[SuperClass load]
+[Insideinitialize load]
ChildClass +[ChildClass initialize]
Insideinitialize +[Insideinitialize initialize]
Insideinitialize -[Insideinitialize objectMethod]
ChildClass +[SuperClass load]
```

前面三个结果跟之前一样，不过之后ChildClass的`+(void)initialize`也被自动执行调用，并且我们可以在其中安全创建出Insideinitialize类并使用它，而Insideinitialize因为调用`alloc`方法是第一次使用类方法， 所以激发了Insideinitialize的`+(void)initialize`。

另一个方面，ChildClass继承下了`+(void)load`而且可以被安全地当做普通类方法(Class Method)被使用。这也就是我之前所说的load和initialize被调用一次是相对runtime而言（比如SuperClass的initialize不会因为自身load方法调用一次，又因为子类调用了load又执行一次），我们依然可以直接去反复调用这些方法。

子类会调用父类的+(void)initialize

接下来，我们再修改一下SuperClass和ChildClass：去掉SuperClass中的`+(void)load`方法；让ChildClass来重写`+(void)load`，但是去掉`+(void)initialize`。

```
1 /******* Interface *******/

2 @interface SuperClass : NSObject
3 @end

4

5 @interface ChildClass : SuperClass
6 @end

7

8 @interface Insideinitialize : NSObject
9 - (void)objectMethod;
10 @end
```

```
11
12 /******* Implementation *******/
13 @implementation SuperClass
14
15 + (void) initialize {
16     NSLog(@"%@ %s", [self class], __FUNCTION__);
17 }
18
19 @end
20
21 @implementation ChildClass
22
23 + (void) load {
24     NSLog(@"%@ %s", [self class], __FUNCTION__);
25 }
26
27 @end
```

依然还是简单的引入这些类，并不去使用它们。运行之后，我们会得到这样的结果：

```
SuperClass +[SuperClass initialize]
ChildClass +[SuperClass initialize]
ChildClass +[ChildClass load]
```

和之前一样，`+(void)load`会引起`+(void)initialize`。也很Apple文档中讲得那样，子类方法的调用会激起父类的`+(void)initialize`被执行。不过我们也看到虽然ChildClass没有定义`+(void)initialize`，但是它会使用父类的`+(void)initialize`。而之前的示例，我们看到子类并不会在runtime时去使用父类的`+(void)load`，也就是说只有新定义的`+(void)load`才会被runtime去调用执行。

类别(Category)中的+(void)load的+(void)initialize

我们再来看看类实现(@implementation)和类的类别(Category)中`+(void)load`和`+(void)initialize`的区别。

```
1 /******* Interface *******/
2 @interface MainClass : NSObject
3 @end
4
5 /******* Category Implementation *******/
6 @implementation MainClass(Category)
7
8 + (void) load {
9     NSLog(@"%@ %s", [self class], __FUNCTION__);
10 }
11
12 + (void) initialize {
13     NSLog(@"%@ %s", [self class], __FUNCTION__);
14 }
15
16 @end
17
18 @implementation MainClass(OtherCategory)
19
```

```
20 + (void) load {
21     NSLog(@"%@ %s", [self class], __FUNCTION__);
22 }
23 
24 + (void) initialize {
25     NSLog(@"%@ %s", [self class], __FUNCTION__);
26 }
27 
28 @end
29 
30 /******* Implementation *******/
31 @implementation MainClass
32 
33 + (void) load {
34     NSLog(@"%@ %s", [self class], __FUNCTION__);
35 }
36 
37 + (void) initialize {
38     NSLog(@"%@ %s", [self class], __FUNCTION__);
39 }
40 
41 @end
```

简单import，运行，我们看到的结果是：

```
MainClass +[MainClass(OtherCategory) initialize]
MainClass +[MainClass load]
MainClass +[MainClass(Category) load]
MainClass +[MainClass(OtherCategory) load]
```

同样的`+(void)initialize`优先于`+(void)load`先执行。但是很明显的不同在于，只有最后一个类别(Category)的`+(void)initialize`执行，其他两个都被隐藏。而对于`+(void)load`，三个都执行，并且如果Apple的文档中介绍顺序一样：先执行类自身的实现，再执行类别(Category)中的实现。

Runtime调用

`+(void)load`

时没有autorelease pool

最后再来看一个示例

```
1 @interface MainClass : NSObject
2 @end
3 
4 @implementation MainClass
5 
6 + (void) load {
7     NSArray *array = [NSArray array];
8     NSLog(@"%@ %s", array, __FUNCTION__);
9 }
10 
11 @end
```

运行这段代码，Xcode给出如下的信息：

```
objc[84934]: Object 0x10a512930 of class __NSArrayI autoreleased with no pool in place - just leaki
2012-09-28 18:07:39.042 ClassMethod[84934:403] (
) +[MainClass load]
```

其原因是runtime调用`+(void)load`的时候，程序还没有建立其autorelease pool，所以那些会需要使用到autorelease pool的代码，都会出现异常。这一点是非常需要注意的，也就是说放在`+(void)load`中的对象都应该是alloc出来并且不能使用autorelease来释放。

不需要显示使用super调用父类中的方法

当我们定义–(id)init和–(void)dealloc方法时，我们总是需要使用super关键字来调用父类的方法，让父类也完成相同的操作。这是因为对对象的初始化和销毁过程，Objective–C不像C++,C#那样会自动调用父类默认构造函数。因此我们总是需要将这两个函数写成这样：

```
1 - (id)init {
2     if ((self = [super init])) {
3         //do initialization

4  }
5

6     return self;
7 }
8

9 - (void)dealloc {
10     //do release

11

12  [super dealloc];
13 }
```

但是`+(void)initialize`和`+(void)load`不同，我们并不需要在这两个方法的实现中使用super调用父类的方法：

```
1 + (void)initialize {
2     //do initialization thing

3  [super initialize];
4 }
5

6 + (void) load {
7     //do some loading things

8  [super load];
9 }
```

super的方法会成功调用，但是这是多余的，因为runtime对自动对父类的`+(void)load`方法进行调用，而`+(void)initialize`则会随子类自动激发父类的方法（如Apple文档中所言）不需要显示调用。另一方面，如果父类中的方法用到的self（像示例中的方法），其指代的依然是类自身，而不是父类。

总结：

	+(void)load	+(void)initialize
执行时机	在程序运行后立即执行	在类的方法第一次被调时执行
若自身未定义，是否沿用父类的方法?	否	是
类别中的定义	全都执行，但后于类中的方法	覆盖类中的方法，只执行一个

References:

1. Objective–C Class Loading and Initialization
2. Cocoa Application Startup

3. +initialize Can Be Executed Multiple Times (+load not so much)
4. NSObject Class Reference
5. Should +initialize/+load always start with an: if (self == [MyClass class]) guard? — stackoverflow
6. [super initialize] — Apple Mailing Lists
7. Objective-C类初始化:load与initialize