iOS中Block的用法，举例，解析与底层原理

摘要

这篇文章，首先在第1节中介绍Block的定义，以及与C里面函数的对比。然后，第2节介绍实际开发中经常会用到的Block语法形式，以供读者日后查阅。只知道怎么用却不知什么时候用？所以随后的第3节将介绍Block的应用场景。然而，用Block不当导致了Crash？所以，第4节有必要了解Block捕获变量的特性，以及循环引用的解决。另外，千万不要懒，一碰到Block就weak，要区分哪些不会引起循环引用。然而，如果对Block的内存机制不熟悉，也会导致Crash，所以第5节会介绍Block的内存机制。学到这里已经够用了。然而，你却想进一步了解Block的实现机制？第6节将简单介绍下clang的编译与Block的实现及其原理。

Block的定义

Block：带有自动变量（局部变量）的匿名函数。它是C语言的扩充功能。之所以是拓展，是因为C语言不允许存在这样匿名函数

1.1 匿名函数

匿名函数是指不带函数名称函数。C语言中，函数是怎样的呢？类似这样：

```
int func(int count);
```

调用的时候：

```
int result = func(10);
```

func就是它的函数名。也可以通过指针调用函数，看起来没用到函数名：

```
int result = (*funcptr)(10);
```

实际，在赋值给函数指针时，必须通过函数的名称才能获得该函数的地址。完整的

步骤应该是：

```
int (*funcptr)(int) = &func;
int result = (*funcptr)(10);
```

而通过Block，就能够使用匿名函数，即不带函数名称的函数。

1.2 带有自动变量

关于“带有自动变量（局部变量）”的含义，这是因为Block拥有捕获外部变量的功能。在Block中访问一个外部的局部变量，Block会持用它的临时状态，自动捕获变量值，外部局部变量的变化不会影响它的的状态。

捕获外部变量，看一个经典block面试题：

```
int val = 10;
void (^blk)(void) = ^{
    printf("val=%d\n",val);
};
val = 2;
blk();
```

上面这段代码，输出值是：val = 10，而不是2。 block 在实现时就会对它引用到的它所在方法中定义的栈变量进行一次只读拷贝，然后在 block 块内使用该只读拷贝；换句话说block截获自动变量的瞬时值；或者block捕获的是自动变量的副本。由于block捕获了自动变量的瞬时值，所以在执行block语法后，即使改写block中使用的自动变量的值也不会影响block执行时自动变量的值。

所以，上面的面试题的结果是2不是10。

解决block不能修改自动变量的值，这一问题的另外一个办法是使用__block修饰符。

```
__block int val = 10;
void (^blk)(void) = ^{printf("val=%d\n",val);};
val = 2;
blk();
```

上面的代码，跟第一个代码段相比只是多了一个__block修饰符。但是输出结果确是

2。

2. Block语法大全

约定：用法中的符号含义列举如下：

- return_type 表示返回的对象/关键字等(可以是void，并省略)
- blockName 表示block的名称
- var_type 表示参数的类型(可以是void，并省略)
- varName 表示参数名称

2.1 Block声明及定义语法，及其变形

1. 标准声明与定义

```
return_type (^blockName)(var_type) = ^return_type (var_type
varName) {
    // ...
};
blockName(var);
```

2. 当返回类型为void

```
void (^blockName)(var_type) = ^void (var_type varName) {
    // ...
};
blockName(var);
```

可以省略写成

```
void (^blockName)(var_type) = ^(var_type varName) {
    // ...
};
blockName(var);
```

3. 当参数类型为void

```
return_type (^blockName)(void) = ^return_type (void) {
    // ...
};
blockName();
```

可以省略写成：

```
return_type (^blockName)(void) = ^return_type {
    // ...
};
blockName();
```

4. 当返回类型和参数类型都为void

```
void (^blockName)(void) = ^void (void) {
    // ...
};
blockName();
```

可以省略写成：

```
void (^blockName)(void) = ^{
    // ...
};
blockName();
```

5. 匿名Block

Block实现时，等号右边就是一个匿名Block，它没有blockName，称之为匿名Block：

```
^return_type (var_type varName)
{
    //...
};
```