第16节课内容总结

block的本质

block的本质是一个`Block_layout`类型的结构体。

```
struct Block_layout {
    void *isa;    指向block的三种类型
    volatile int32_t flags; // contains ref count  block的附加信息
    int32_t reserved; 保留的变量
    // libffi ->
    BlockInvokeFunction invoke;  block的实现的函数指针
    struct Block_descriptor_1 *descriptor;  存储copy，dispose函数，
    // imported variables                   block的大小，
};                                          block的签名等信息
```

copy和dispose函数是用来对block内部的对象进行内存管理的，block拷贝到堆上会调用copy函数，在block从堆上释放的时候会调用dispose函数。

`block的签名：@?`

解决block循环引用的三种方法

方法一（weak strong dance）：

```
self.name = @"lg";
__weak typeof(self) weakSelf = self;
self.block = ^ {
    __strong typeof(weakSelf) strongSelf = weakSelf;
    NSLog(@"%p",&strongSelf);
    dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(3 * NSEC_PER_SEC)),
        NSLog(@"%@",strongSelf.name);
    });
};
self.block();
```

方法二（临时变量）：

```
self.name = @"lg";
__block LGViewController *vc = self;
self.block = ^{
```

```
        dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(3 * NSEC_PER_SEC
        NSLog(@"%@",vc.name);
        vc = nil;
    });
};
self.block();
```

方法三（block参数）：

```
self.name = @"lg";
self.block = ^(LGViewController *vc){
    dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(3 * NSEC_PER_SEC)), 
        NSLog(@"%@",vc.name);
    });
};
self.block(self);
```

在block内部可以修改全局变量和静态变量的值，但是不允许修改局部变量的值。要箱子啊block内部修改局部变量的值需要用__block修饰。

__block的底层原理

用__block修饰的变量在编译过后会变成`__Block_byref__XXX`类型的结构体，在结构体内部有一个`__forwarding`的结构体指针，指向结构体本身。
block创建的时候是在栈上的，在将栈block拷贝到堆上的时候，同时也会将block中捕获的对象拷贝到堆上，然后就会将栈上的__block修饰对象的__forwarding指针指向堆上的拷贝之后的对象。这样我们在block内部修改的时候虽然是修改堆上的对象的值，但是因为栈上的对象的__forwarding指针将堆和栈的对象链接起来。因此就可以达到修改的目的。

__block不可以用于修饰静态变量和全局变量。

expression指令，简写为`e`，能够在调试时，动态的修改变量的值，同时打印出结果，还可以动态调用函数。它会实时的真正的执行后面的代码。