Swift进阶第二节课：值类型与引用类型

延迟存储属性

```
class LGTeacher{
    lazy var age: Int = 10
}
```

- 用 lazy 修饰的存储属性
- 延迟存储属性必须有一个默认的初始值
- 延迟存储在第一次访问的时候才被赋值
- 延迟存储属性并不能保证线程安全
- 延迟存储属性对实例对象大小的影响

类型属性

```
class LGTeacher{
    static var age: Int = 10
}
```

- 使用关键字 static 修饰
- 类型属性必须有一个默认的初始值
- 类型属性只会被初始化一次

单例的正确写法

```objc
#import "LGThread.h"

@implementation LGThread

+ (instancetype)sharedInstance {
    static LGThread *sharedInstance = nil;
    static dispatch_once_t onceToken;

    dispatch_once(&onceToken, ^{
        sharedInstance = [[LGThread alloc] init];
    });
    return sharedInstance;
}

@end
```

```swift
class LGTeacher{
    static let sharedInstance: LGTeacher = LGTeacher()
  private init(){}
}
```

- 使用 static let 创建声明一个实例对象
- 给当前 init 添加访问控制权限 private(什么意思)

结构体的初始化

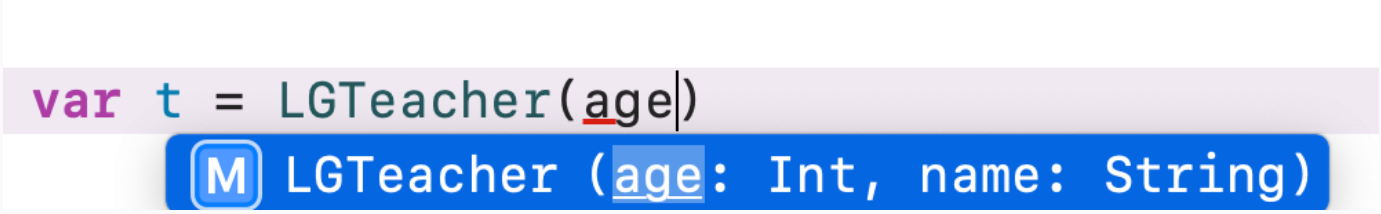
- 结构体不需要自定义初始化方法，对比下面两段代码

```swift
//1
struct LGTeacher{
    var age: Int
    var name: String
}

//2
class LGStudent{
```

```
 9      var age: Int
10      var name: String
11  }
```

其中 2 在代码编译过程中会报错："Class 'LGStudent' has no initializers"

这是因为编译器在结构体中自动帮组我们合成了初始化方法，也就意味着我们可以这样调用：



这里我们通过 `SIL` 来查看：

```
struct LGTeacher {
  @_hasStorage var age: Int { get set }
  @_hasStorage var name: String { get set }
  init(age: Int, name: String)
}
```

- 如果我们的属性有默认初始值，系统会提供不同的默认初始化方法

```
// default argument 0 of LGTeacher.init(age:name:)
sil hidden @default argument 0 of main.LGTeacher.init(age: Swift.Int, name: Swift.String) -> main.LGTeacher : $@convention(thin) () -> Int {
bb0:
  %0 = integer_literal $Builtin.Int64, 18         // user: %1
  %1 = struct $Int (%0 : $Builtin.Int64)          // user: %2
  return %1 : $Int                                // id: %2
} // end sil function 'default argument 0 of main.LGTeacher.init(age: Swift.Int, name: Swift.String) -> main.LGTeacher'

// LGTeacher.init(age:name:)
sil hidden @main.LGTeacher.init(age: Swift.Int, name: Swift.String) -> main.LGTeacher : $@convention(method) (Int, @owned String, @thin LGTeacher.Type) -> @owned LGTeacher {
// %0                                             // user: %3
// %1                                             // user: %3
bb0(%0 : $Int, %1 : $String, %2 : $@thin LGTeacher.Type):
  %3 = struct $LGTeacher (%0 : $Int, %1 : $String) // user: %4
  return %3 : $LGTeacher                          // id: %4
} // end sil function 'main.LGTeacher.init(age: Swift.Int, name: Swift.String) -> main.LGTeacher'
```

- 如果我们自定义初始化方法，系统就不会帮我们生成初始化方法

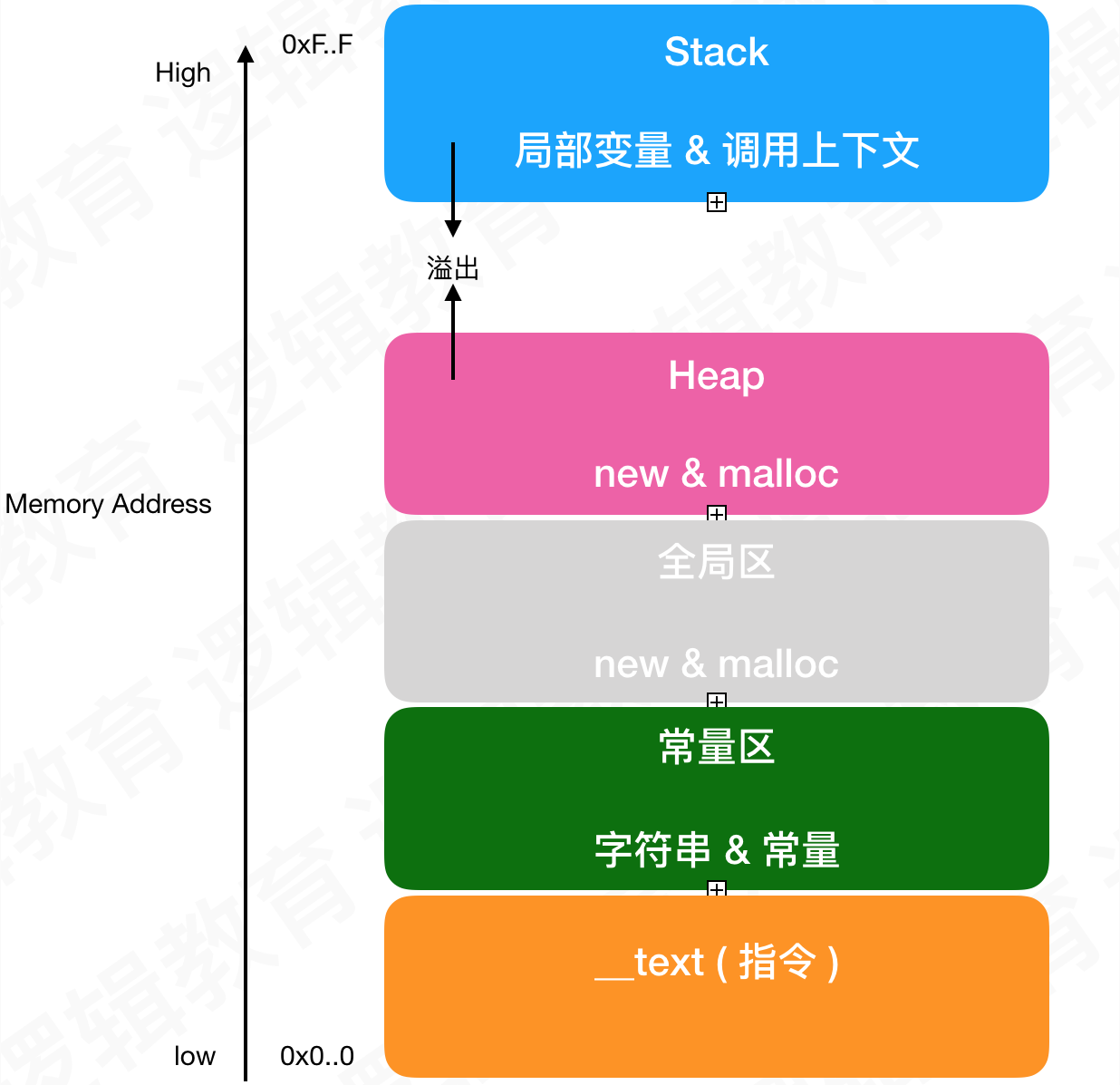
结构体是值类型

- 什么是值类型

我们首先来看一个例子：

```
func test(){
    var age = 18

    var age2 = age

    age = 30

    age2 = 45

    print("age=\(age),age2=\(age2)")
}
```

- 内存分区模型：



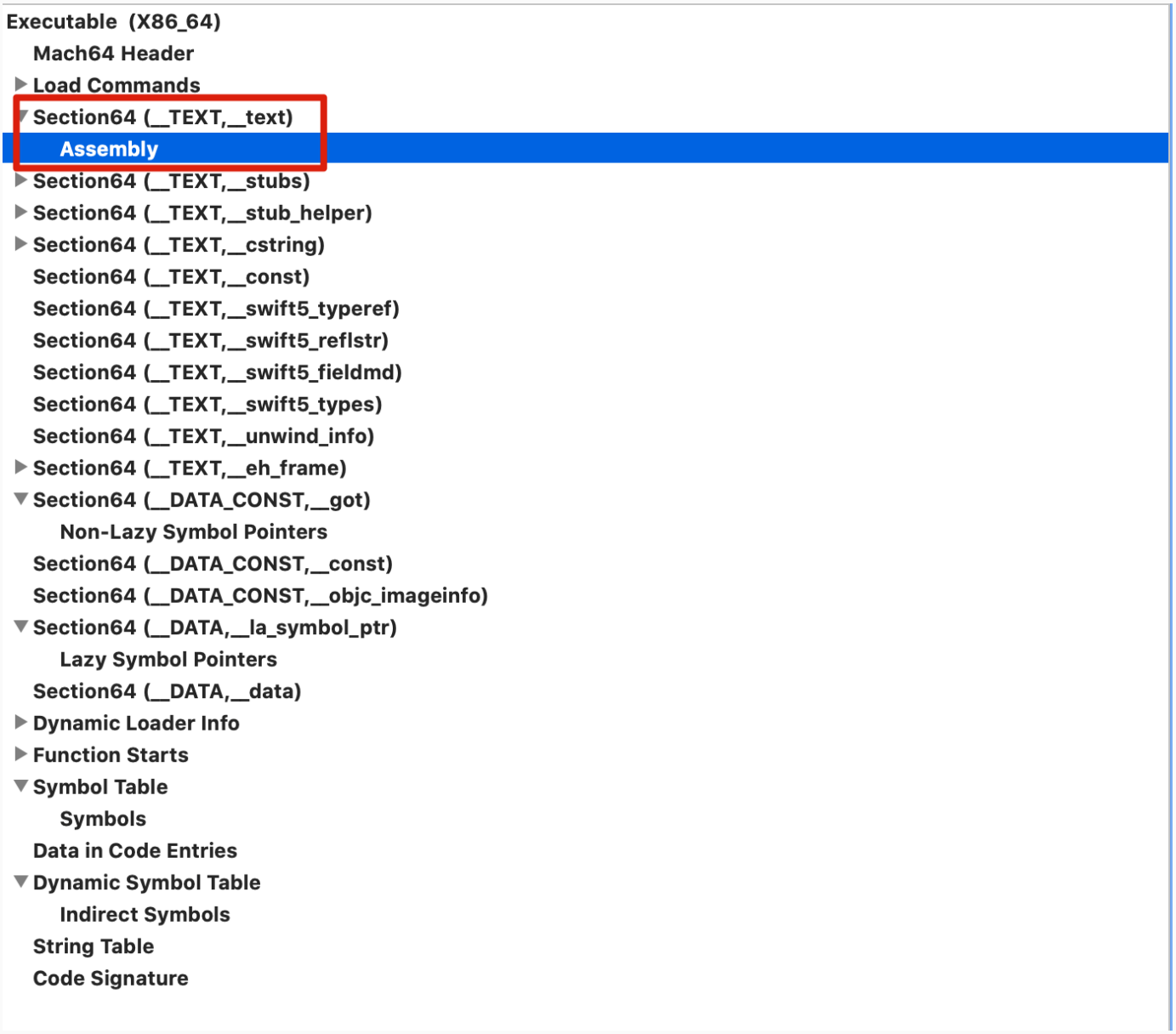
- 通过 LLDB 查看结构体的内存模型

其实值类型和引用类型就像什么，一个是在线表格，一个是本地的excel,当我和你共享编辑一个在线表格的时候其实就是一个引用类型，而当我们要通过QQ传给你一个excel的时候，这个时候就相当于一个值类型，你修改的内容我是不知道的。

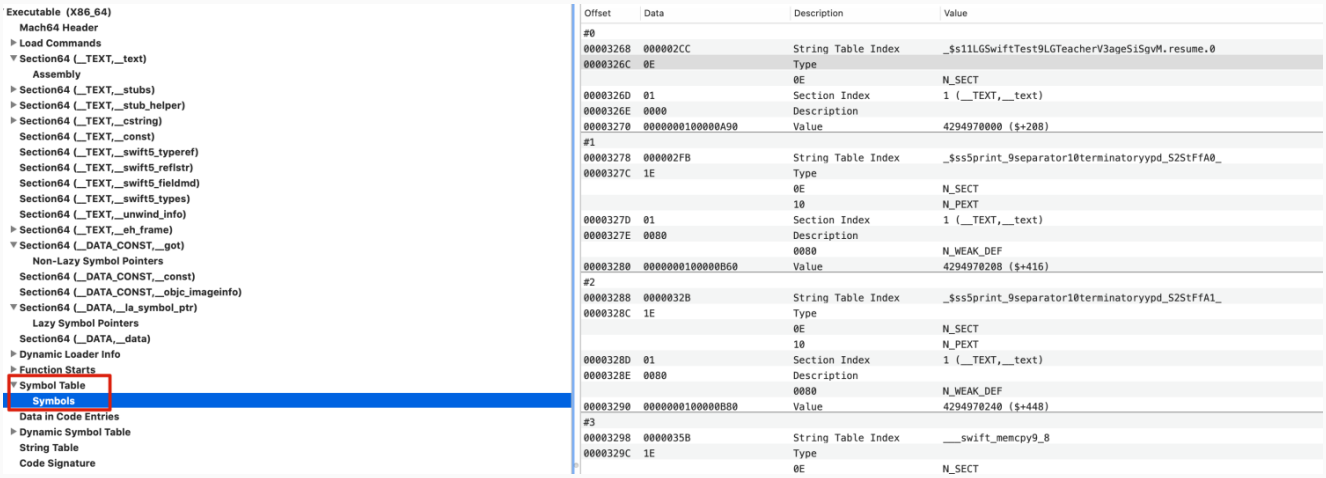
- mutating

值类型本身创建之后是不允许修改的，如果要修改，需要使用 mutating 关键字

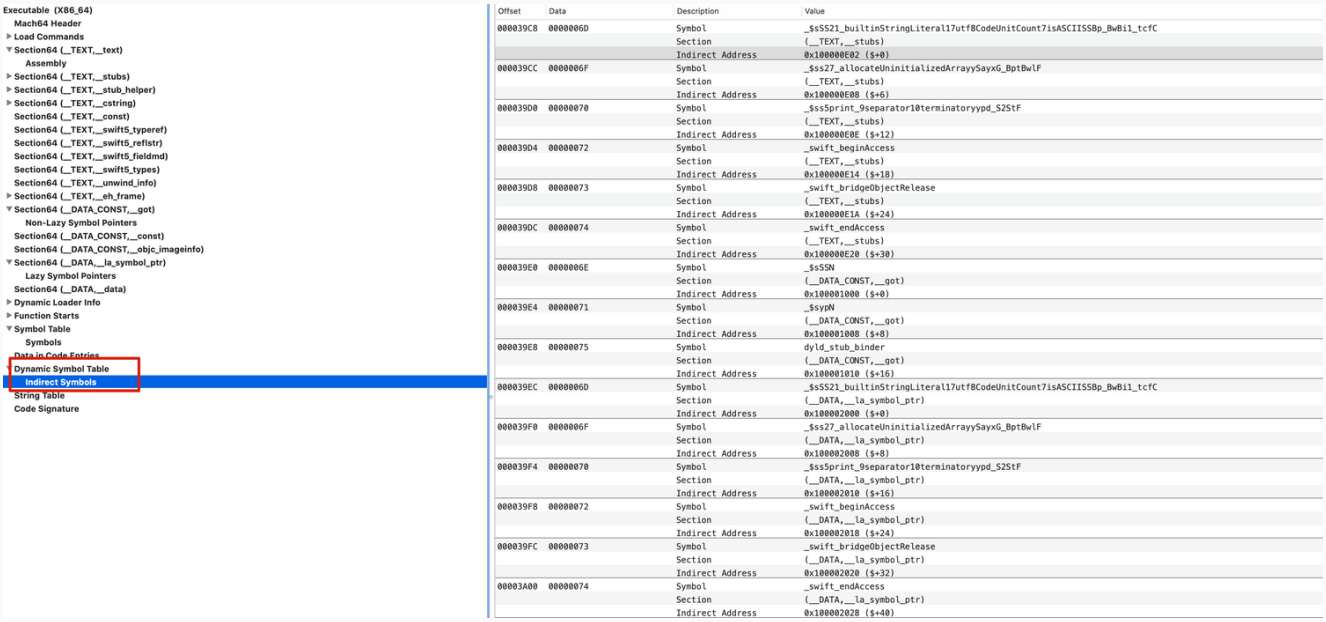
- 结构体中的方法调度是静态调度（编译，链接完成之后当前的函数地址就已经确定存放在了代码段）



  - Symbol Table ： 存储符号位于字符串表的位置



- Dynamic Symbol Table：动态库函数位于符号表的偏移信息



- 类中的方法调度是一般函数表调度，同时和函数声明的位置与关键字有关。
  - extesion声明方法
  - final关键字
  - static关键字
  - @objc关键字
  - dynamic关键字