Swift进阶第一节课：类，对象，属性

Swift编译简介（了解）

SIL分析（掌握）

类结构探索

Swift属性

Swift编译简介（了解）

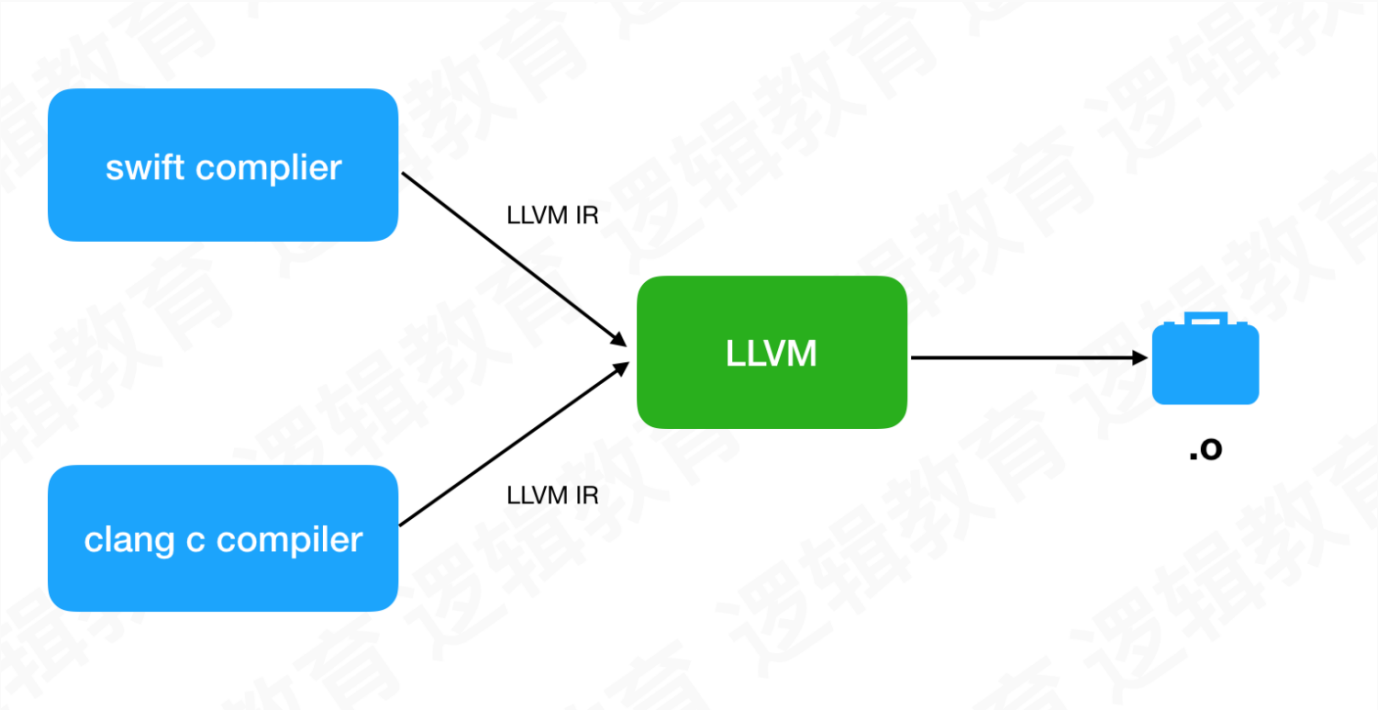
我们先来看一段简单的代码，下面这段代码中我们创建了一个`LGTeacher`的类，并通过默认的初始化器，创建了一个实例对象赋值给了`t`。

```
class LGTeacher{
    var age: Int = 18
    var name: String = "Kody"
}

let t = LGTeacher()
```

接下来我们想要研究的是，这个默认的初始化器到底做了一个什么样的操作？这里我们引入 `SIL (Swift intermediate language)`，再来阅读 `SIL` 的代码之前，我们先来了解一下什么是 `SIL`。

iOS开发的语言不管是`OC`还是`Swift`后端都是通过`LLVM`进行编译的，如下图所示：

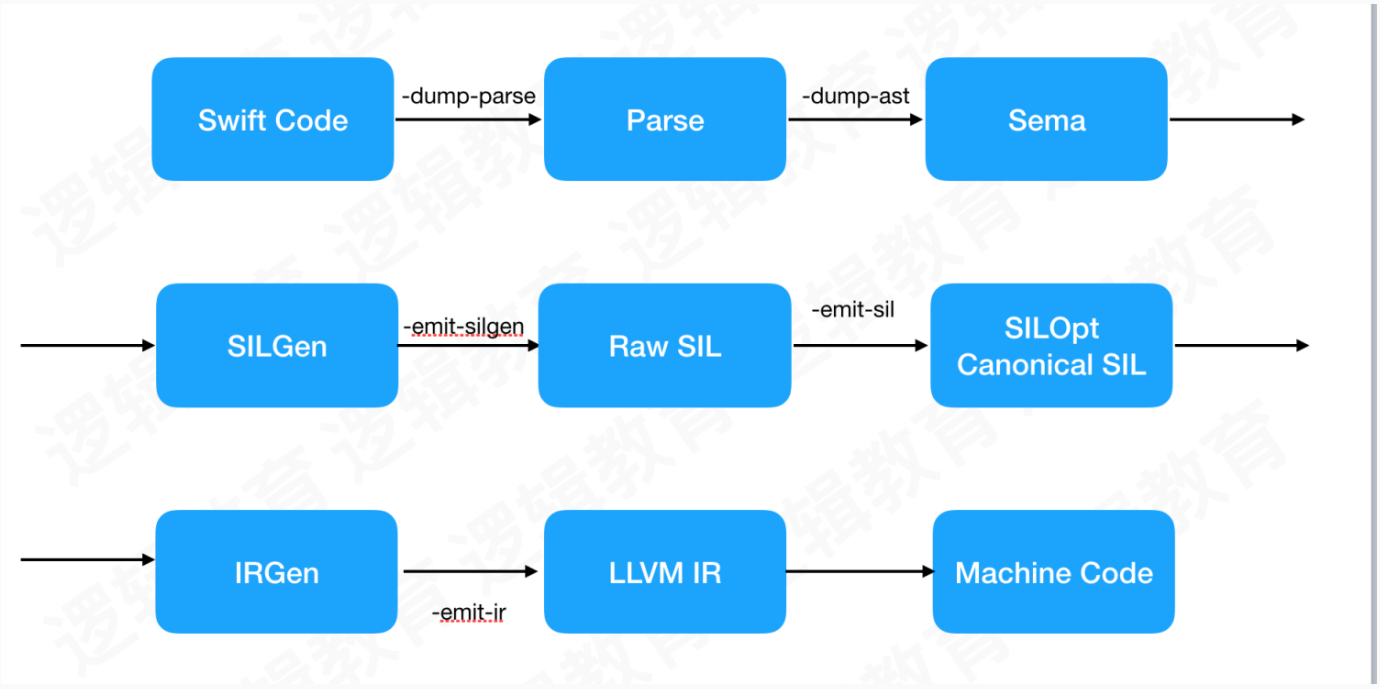


可以看到：

`OC`通过`clang`编译器，编译成IR，然后再生成可执行文件.o(这里也就是我们的机器码)

`Swift`则是通过`Swift`编译器编译成IR，然后在生成可执行文件。

我们再来看一下，一个 `swift` 文件的编译过程都经历了哪些步骤：



`swift`在编译过程中使用的前端编译器是`swiftc`，和我们之前在`OC`中使用的`Clang`是有所区别的。我们可以通过如下命令查看 `swiftc` 都能做什么样的事情：

`swiftc -h`

```
USAGE: swiftc

MODES:
  -dump-ast               Parse and type-check input file(s) and dump AST(s)
  -dump-parse             Parse input file(s) and dump AST(s)
  -dump-pcm               Dump debugging information about a precompiled Clang module
  -dump-scope-maps <expanded-or-list-of-line:column>
                          Parse and type-check input file(s) and dump the scope map(s)
  -dump-type-info         Output YAML dump of fixed-size types from all imported modules
  -dump-type-refinement-contexts
                          Type-check input file(s) and dump type refinement contexts(s)
  -emit-assembly          Emit assembly file(s) (-S)
  -emit-bc                Emit LLVM BC file(s)
  -emit-executable        Emit a linked executable
  -emit-imported-modules Emit a list of the imported modules
  -emit-ir                Emit LLVM IR file(s)
  -emit-library           Emit a linked library
  -emit-object            Emit object file(s) (-c)
  -emit-pcm               Emit a precompiled Clang module from a module map
  -emit-sibgen            Emit serialized AST + raw SIL file(s)
  -emit-sib               Emit serialized AST + canonical SIL file(s)
  -emit-silgen            Emit raw SIL file(s)
  -emit-sil               Emit canonical SIL file(s)
  -index-file             Produce index data for a source file
  -parse                  Parse input file(s)
  -print-ast              Parse and type-check input file(s) and pretty print AST(s)
  -resolve-imports        Parse and resolve imports in input file(s)
  -typecheck              Parse and type-check input file(s)
```

如果想要详细对 SIL 的内容进行探索，可以参考这个视频

SIL分析（掌握）

SIL参考文档

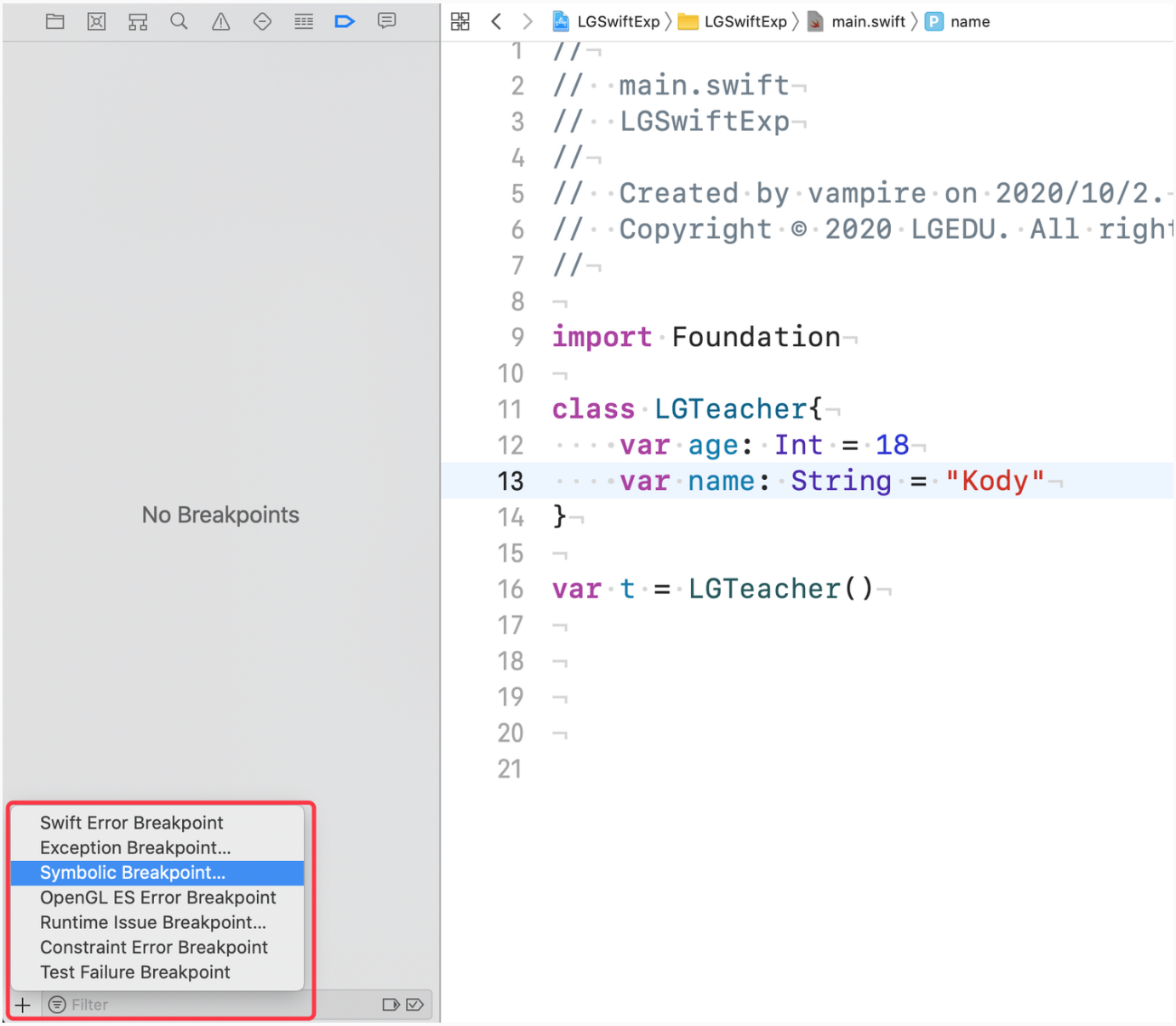
```
// main
sil @main : $@convention(c) (Int32, UnsafeMutablePointer<Optional<UnsafeMutablePointer<Int8>>>
bb0(%0 : $Int32, %1 : $UnsafeMutablePointer<Optional<UnsafeMutablePointer<Int8>>>):
  alloc_global @$s4main1tAA9LGTeacherCvp          // id: %2
  %3 = global_addr @$s4main1tAA9LGTeacherCvp : $*LGTeacher // user: %7
  %4 = metatype $@thick LGTeacher.Type            // user: %6
  // function_ref LGTeacher.__allocating_init()
  %5 = function_ref @$s4main9LGTeacherCACycfC : $@convention(method) (@thick LGTeacher.Type)
  %6 = apply %5(%4) : $@convention(method) (@thick LGTeacher.Type) -> @owned LGTeacher // use
  store %6 to %3 : $*LGTeacher                    // id: %7
  %8 = integer_literal $Builtin.Int32, 0          // user: %9
  %9 = struct $Int32 (%8 : $Builtin.Int32)        // user: %10
  return %9 : $Int32                              // id: %10
} // end sil function 'main'
```

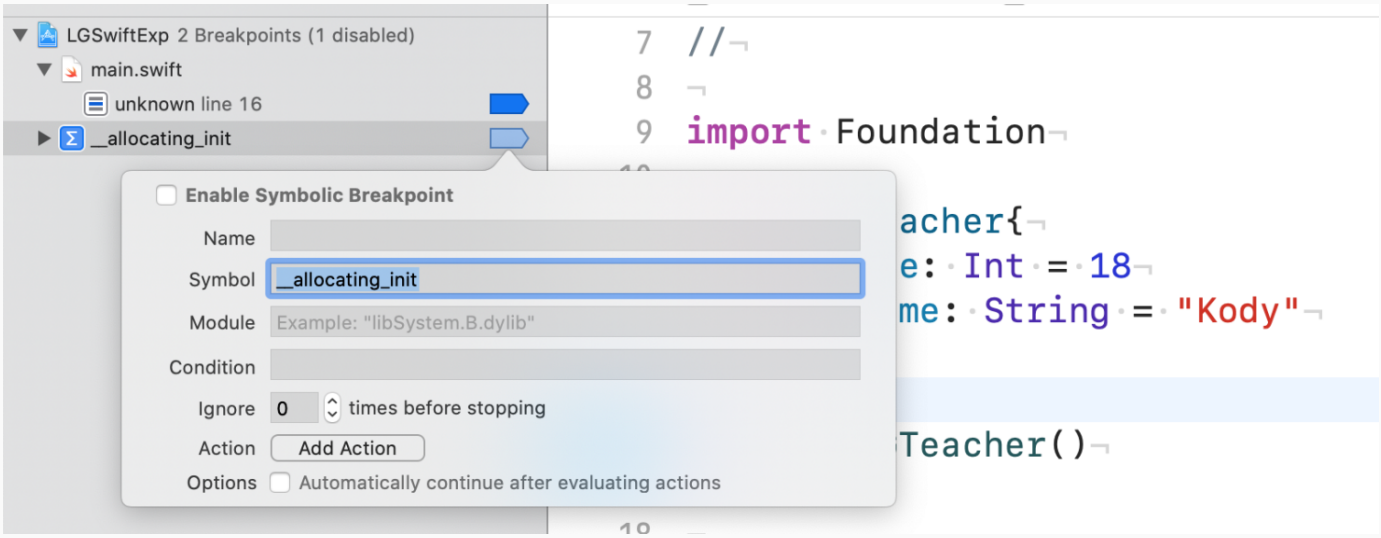
- @main 这里标识我们当前 main.swift 的入口函数，SIL 中的标识符名称以 @ 作为前缀
- %0， %1... 在 SIL 也叫做寄存器，这里我们可以理解为我们日常开发中的常量，一旦赋值之后就不可以再修改，如果 SIL 中还要继续使用，那么就不断的累加数字。同时这里所说的寄存器是虚拟的，最终运行到我们的机器上，会使用真的寄存器。
- `alloc_gobal` 创建一个全局变量
- `global_addr` 拿到全局变量的地址，赋值给 %3
- `metatype` 拿到 `LGTeacher` 的`Metadata` 赋值给 %4
- 将 `__allocating_init` 的函数地址赋值给 %5
- `apply` 调用 `__allocating_init`，并把返回值给 %6
- 将 %6 的值存储到 %3（也就是我们刚刚创建的全局变量的地址）
- 构建 `Int`，并 `return`

```
1 // LGTeacher.__allocating_init()
2 sil hidden [exact_self_class] @$s4main9LGTeacherCACycfC : $@conven
  tion(method) (@thick LGTeacher.Type) -> @owned LGTeacher {
3 bb0(%0 : $@thick LGTeacher.Type):
4   %1 = alloc_ref $LGTeacher                       // user: %3
5   // function_ref LGTeacher.init()
6   %2 = function_ref @$s4main9LGTeacherCACycfc : $@convention(metho
  d) (@owned LGTeacher) -> @owned LGTeacher // user: %3
7   %3 = apply %2(%1) : $@convention(method) (@owned LGTeacher) -> @
  owned LGTeacher // user: %4
8   return %3 : $LGTeacher                          // id: %4
9 } // end sil function '$s4main9LGTeacherCACycfC'
```

- `alloc_ref` :创建一个 `LGTeacher` 的实例对象，当前实例对象的默认引用计数为 1
- 调用 `init` 方法

这里我们其实可以通过符号断点来查看一下：





同样的，这里我们可以直接在 `VSCode` 中调试我们的源码，调试过程参考视频讲解（怎么调试）。

以上我们就可以得出一个很简单的结论：

- 这里我们简单过了一下Swift内存分配过程中发生的事情，`__allocating_init` -----> `swift_allocObject` -----> `_swift_allocObject_` -----> `swift_slowAlloc` -----> `Malloc`

- Swift对象的内存结构HeapObject，有两个属性： 一个是Metadata，一个是RefCount，默认占用16字节大小

- `init` 在这里扮演了初始化变量的职责，这和我们在 `OC` 中的认知是一致的

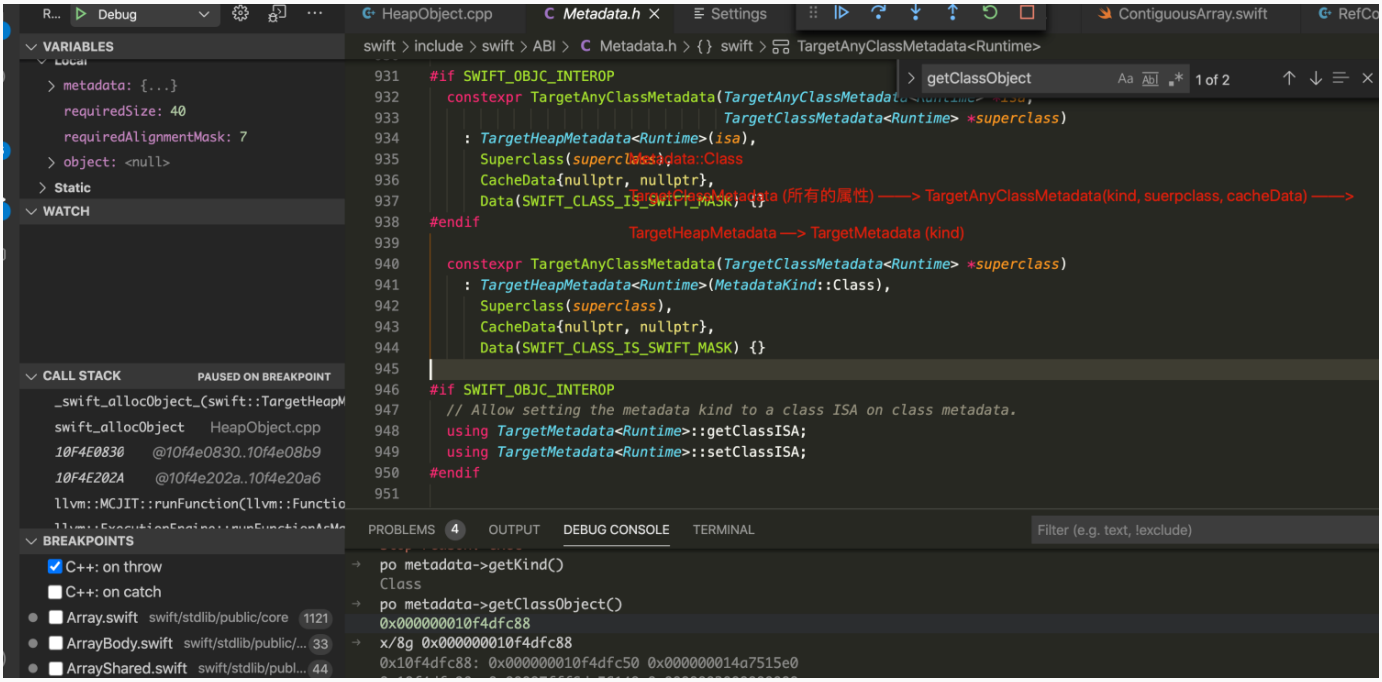
类结构探索

看了上面的内存分配之后，接下来我们应该注意到了一个`Metadata`，它的类型是`HeapMetadata`,我们来看一下它的具体内存结构是什么？

其中 `kind` 的种类:

name	Value
Class	0x0
Struct	0x200
Enum	0x201
Optional	0x202

ForeignClass	0x203
ForeignClass	0x203
Opaque	0x300
Tuple	0x301
Function	0x302
Existential	0x303
Metatype	0x304
ObjCClassWrapper	0x305
ExistentialMetatype	0x306
HeapLocalVariable	0x400
HeapGenericLocalVariable	0x500
ErrorObject	0x501
LastEnumerated	0x7FF



经过源码的阅读，我们应该能得出当前 `metadata` 的数据结构体了

```
struct swift_class_t: NSObject{
    void *kind; //isa, kind(unsigned long)
```

```
    void *superClass;
    void *cacheData
    void *data
    uint32_t flags;  //4
    uint32_t instanceAddressOffset; //4
    uint32_t instanceSize;//4
    uint16_t instanceAlignMask; //2
    uint16_t reserved; //2

    uint32_t classSize; //4
    uint32_t classAddressOffset; //4
    void *description;
    // ...
};
```

Swift属性

- 存储属性（要么是常量（let 修饰）存储属性，要么是变量（var 修饰）存储属性）

```
class LGTeacher{
    let age: Int = 18
    var name: String = "Kody"
}
let t = LGTeacher()
```

对于上面的 age, name 来说，都是我们的变量存储属性，这点我们在 `SIL` 中也可以看到：

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

- 计算属性（顾名思义计算属性是不占用存储空间的，本质get/set方法）

- 属性观察者：（willSet, didSet）
- 延迟存储属性
  - 延迟存储属性的初始值在其第一次使用时才进行计算。
  - 用关键字`lazy`来标识一个延迟存储属性。

- 类型属性
  - 类型属性属于这个类的本身，不管有多少个实例，类型属性只有一份，我们使用 `static` 来声明一个类型属性

- 如何正确的声明一个单利

```
class LGTeacher{
    var age: Int = 18
    var name: String = "Kody"

    static let sharedInstance = LGTeacher()

    private init(){}

}

var t = LGTeacher.sharedInstance
```

- 值类型 & 引用类型