Филология программирования

Инициализация объектов в современных языках программирования

Алексей Недоря, февраль 2024

3D: Памяти Никлауса Вирта, Филология программирования, Инициализация

ФИЛОЛОГИЯ (от фило… и …логия), содружество гуманитарных дисциплин – языкознания, литературоведения, текстологии, источниковедения, палеографии и др., – изучающих духовную культуру человека через языковой и стилистич. анализ письменных текстов.

План и Постановка задачи

План

- Постановка задачи
- Go way
- Swift way
- Kotlin way
- Тривиль
- Заключение

Какие задачи должна решать “инициализация объекта”:

1. Задать определенные значения полям объекта (нет UB)
2. Перевести объект в “корректное” состояние (инвариант класса) или прервать выполнение, если это невозможно (паника, исключение)

Как должна записываться и выполняться инициализация:

- безопасно (нет возможности ошибиться)
- просто, насколько возможно (но, обязательно безопасно)
- должна работать для всех “нормальных” случаев (но, просто)

Важно: Две фазы инициализации!

Go Way - Инициализация для языка с опасными ссылками (not null safe)

Инициализация:

- композитные литералы
- значения по умолчанию (для всех типов)
- принцип “сделай сам”

```
type Node struct {
  Count      int
  Next       *Node
}

func main() {
  var root = &Node{}
  fmt.Println(root)
}
```

```
&{0 <nil>}
```

```
type Node struct {
    count      int   // not exported
    next       *Node
}

var counter = 0

func Create() *Node {
  counter++
  return &Node{count: counter}
}
```

```
func main() {
  var root = nodes.Create{}
  fmt.Println(root)
}
```

```
&{1 <nil>}
```

Swift Way - тщательно, надежно, сложно (24 страницы в спецификации)

Инициализация:

- начальные значения полей
- инициализаторы
- строгое описание порядка и правил инициализации

Two-Phase Initialization

Class initialization in Swift is a two-phase process. In the **first phase**, each stored property is assigned an **initial value** by the class that introduced it. Once the initial state for every stored property has been determined, the **second phase** begins, and each class is given the opportunity to **customize** its stored properties further before the new instance is considered ready for use.

```
var counter = 0

class C {
    var count: Int
//!    var next: C

    init() {
      // фаза 1
      counter += 1
      self.count = counter
      // фаза 2
    }
}
var root = C()
print(root.count)
```

https://docs.swift.org/swift-book/documentation/the-swift-programming-language/initialization/#Two-Phase-Initialization

Swift Way - Safety Checks

```
var counter = 0

class C {
    var count: Int

    init() {
      // фаза 1
      counter += 1
      print(self.count) // CTE - used before initialized
      self.count = counter
      // фаза 2
    }
}
var root = C()
print(root.count)
```

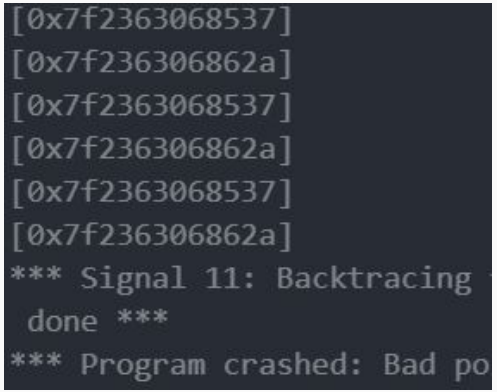
```
var counter = 0

class C {
    var count: Int
    var next: C // CTE - not initialized

    init() {
      counter += 1
      self.count = counter
    }
}
var root = C()
print(root.count)
```

```
class C {
    var next: C

    init() {
      self.next = C()
    }
}
var root = C()
print(root.count)
```



Swift Way - Optionals

```
var counter = 0

class C {
    var count: Int
    var next: C? = Optional.none // or nil

    init() {
      counter += 1
      self.count = counter
    }
}
var root = C()
print(root.next ?? "none")

root.next = C()
print(root.next!.count)
```

```
none
2
```

Kotlin Way - сложно, как в Swift, но... (см. дальше)

```
var counter = 0

class C {
    var count: Int

    init {
      counter += 1
      count = counter
    }
}

fun main() {
  val root = C()
  print(root.count)
}
```

```
var counter = 0

class C {
    var count: Int

    init {
      println(count) // CTE - not initialized
      counter += 1
      count = counter
    }
}

fun main() {
  val root = C()
  print(root.count)
}
```

```
class C {
    var next: C = C()
}

fun main() {
  val root = C()
}
```

```
java.lang.StackOverflowError
; (File.kt:1)
; (File.kt:2)
; (File.kt:2)
```

Kotlin Way - Увы! (доказано: Александр Когтенков)

```
class A { val f = 1 }

class C {
    private val a: A
    init {
        fn()
        a = A()
    }
    public fun fn() {
        println(a)
        println(a.f)
    }
}

fun main() {
    val c = C()
    c.fn()
}
```

```
null
Exception in thread "main" java.lang.NullPointerEx
Cannot invoke "A.getF()" because "this.a" is null
    at C.fn (File.kt:15)
    at C.&amp;lt;init&amp;gt; (File.kt:9)
    at FileKt.main (File.kt:20)
```

```
class A { var f = 1 }

class C {
    var a: A
    init() {
        fn() // CTE - phase 2
        a = A()
    }
    public fun fn() { }
}
```

Swift молодец!

Тривиль

```
тип К = класс {
  номер: Цел64 = позже
}

пусть счетчик := 0

вход{
  счетчик++
  пусть к = К{номер: счетчик}
  вывод.ф("$;", к.номер)
}
```

```
тип К* = класс {
  номер: Цел64 = позже
}

пусть счетчик := 0

фн создать*(): К {
  счетчик++
  вернуть К{номер: счетчик}
}
```

```
type Node struct {
    count      int
}

var counter = 0

func Create() *Node {
  counter++
  return &Node{count: counter}
}
```

Тривиль	**Go**
• обязательная инициализация полей • нет значений по умолчанию • есть поздняя инициализация • есть неизменяемые поля	• нет инициализации полей • используется значение по умолчанию

Тривиль - пробуем сломать

Отсутствие инициализации поля:

```
тип К = класс {
    номер: Цел64 = позже
}
пусть к = К{} // CTE - нет инициализации
```

Swift, Kotlin: бесконечная рекурсия компилятора. Тривиль: ошибка компиляции.

```
тип К* = класс {
    номер: Цел64 = позже
    след: К = К{} // CTE - циклическое определение
}
```

Вызов метода перед завершением инициализации:

- Kotlin: “дыра” в безопасности
- Swift: компилятор показывает ошибку
- Тривиль: нельзя написать (нельзя обратиться к объекту, инициализация которого не завершена)

```
class A { var f = 1 }

class C {
    var a: A
    init() {
        fn() // CTE - phase 2
        a = A()
    }
    public fun fn() { }
}
```

```
тип А = класс { ф := 1 }

тип К = класс {
    а: А = позже
}
фн (к: К) Ф() {}

пусть к = К{а: А{}}
к.Ф()
```

Мораль сей басни

- Есть существенная разница: Go, Тривиль (влияние Вирта) и Swift, Kotlin (другая школа)

- Сделать просто - не просто и, как правило, очень трудно.
- Выигрыш от простого (but not simpler) решения, многократно оправдывает затраты - по крайней мере, в дизайне языков программирования.

Вопросы и ответы