巨匠線上真人

iOS行動程式基礎開發上架

swift：初始化

本堂教學重點

1. 為儲存屬設定初始值
   - 初始者
   - 預設的屬性值
2. 自訂初始化
   - 初始化參數
   - 參數名稱和引數標籤
   - 沒有引數標籤的初始化參數
   - 可nil的屬性類型
   - 初始化期間，指定常數屬性
3. 預設的初始化
   - 結構的智慧型初始化
4. 值類型的初始化委派
5. 類別繼承和初始化
   - 主要的初始化者和便利初始化者的語法
   - 類別的初始化委派
   - 二階段的初始化
   - 初始化的繼承和覆寫
   - 自動化的初始化繼承
   - 實作主要的初始化者和便利初始化者
   - 可失敗的初始化
6. Required 初始化者
7. 使用閉鎖和函式設定屬性值

1.為儲存屬設定初始值

覆寫方法

```
init() {
    // perform some initialization here
}
```

```
struct Fahrenheit {
    var temperature: Double
    init() {
        temperature = 32.0
    }
}
var f = Fahrenheit()
print("The default temperature is \(f.temperature)° Fahrenheit")
// Prints "The default temperature is 32.0° Fahrenheit"
```

1.為儲存屬設定初始值

預設的屬性值

```
struct Fahrenheit {
    var temperature = 32.0
}
```

2.自訂初始化

初始化參數

```
struct Celsius {
    var temperatureInCelsius: Double
    init(fromFahrenheit fahrenheit: Double) {
        temperatureInCelsius = (fahrenheit - 32.0) / 1.8
    }
    init(fromKelvin kelvin: Double) {
        temperatureInCelsius = kelvin - 273.15
    }
}
let boilingPointOfWater = Celsius(fromFahrenheit: 212.0)
// boilingPointOfWater.temperatureInCelsius is 100.0
let freezingPointOfWater = Celsius(fromKelvin: 273.15)
// freezingPointOfWater.temperatureInCelsius is 0.0

```

2.自訂初始化

參數名稱和引數標籤

```
struct Color {
    let red, green, blue: Double
    init(red: Double, green: Double, blue: Double) {
        self.red   = red
        self.green = green
        self.blue  = blue
    }
    init(white: Double) {
        red   = white
        green = white
        blue  = white
    }
}

let magenta = Color(red: 1.0, green: 0.0, blue: 1.0)
let halfGray = Color(white: 0.5)

let veryGreen = Color(0.0, 1.0, 0.0)
// this reports a compile-time error - argument labels are required
```

2.自訂初始化

沒有引數標籤的初始化參數

```
struct Celsius {
    var temperatureInCelsius: Double
    init(fromFahrenheit fahrenheit: Double) {
        temperatureInCelsius = (fahrenheit - 32.0) / 1.8
    }
    init(fromKelvin kelvin: Double) {
        temperatureInCelsius = kelvin - 273.15
    }
    init(_ celsius: Double) {
        temperatureInCelsius = celsius
    }
}
let bodyTemperature = Celsius(37.0)
// bodyTemperature.temperatureInCelsius is 37.0

```

2.自訂初始化

可nil的屬性類型

```
class SurveyQuestion {
    var text: String
    var response: String?
    init(text: String) {
        self.text = text
    }
    func ask() {
        print(text)
    }
}
let cheeseQuestion = SurveyQuestion(text: "Do you like cheese?")
cheeseQuestion.ask()
// Prints "Do you like cheese?"
cheeseQuestion.response = "Yes, I do like cheese."
```

2.自訂初始化

初始化期間，指定常數屬性

```
class SurveyQuestion {
    let text: String
    var response: String?
    init(text: String) {
        self.text = text
    }
    func ask() {
        print(text)
    }
}
let beetsQuestion = SurveyQuestion(text: "How about beets?")
beetsQuestion.ask()
// Prints "How about beets?"
beetsQuestion.response = "I also like beets. (But not with cheese.)"
```

3.預設的初始化

```
class ShoppingListItem {
    var name: String?
    var quantity = 1
    var purchased = false
}
var item = ShoppingListItem()
```

4.值類型的初始化委派

```
struct Size {
    var width = 0.0, height = 0.0
}
struct Point {
    var x = 0.0, y = 0.0
}

struct Rect {
    var origin = Point()
    var size = Size()
    init() {}
    init(origin: Point, size: Size) {
        self.origin = origin
        self.size = size
    }
    init(center: Point, size: Size) {
        let originX = center.x - (size.width / 2)
        let originY = center.y - (size.height / 2)
        self.init(origin: Point(x: originX, y: originY), size: size)
    }
}
```

4.值類型的初始化委派

```
let basicRect = Rect()
// basicRect's origin is (0.0, 0.0) and its size is (0.0, 0.0)


let originRect = Rect(origin: Point(x: 2.0, y: 2.0),
                      size: Size(width: 5.0, height: 5.0))
// originRect's origin is (2.0, 2.0) and its size is (5.0, 5.0)


let centerRect = Rect(center: Point(x: 4.0, y: 4.0),
                      size: Size(width: 3.0, height: 3.0))
// centerRect's origin is (2.5, 2.5) and its size is (3.0, 3.0)
```

5.類別繼承和初始化

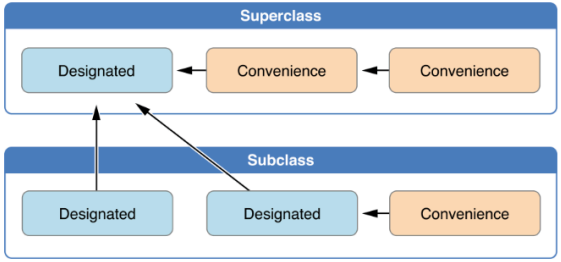
主要的初始化者和便利初始化者的語法

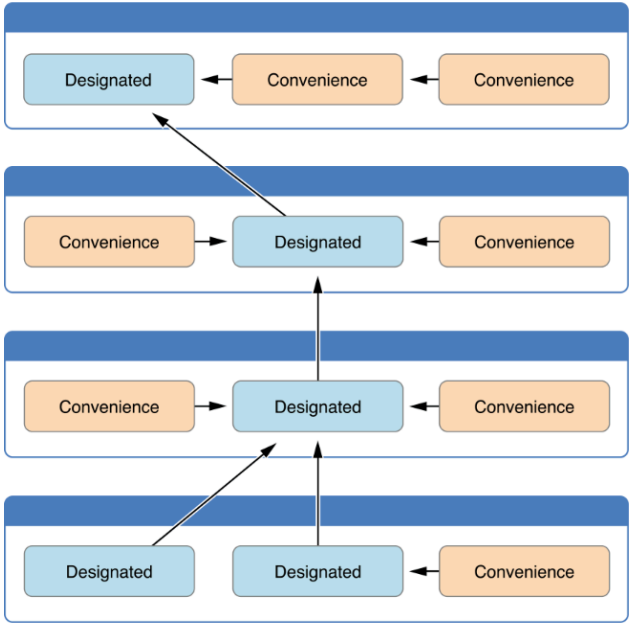
```
init(parameters) {
    statements
}

convenience init(parameters) {
    statements
}
```

5.類別繼承和初始化

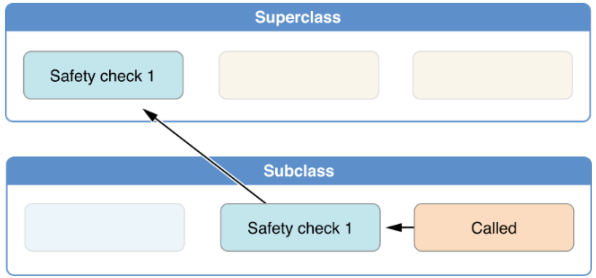
類別的初始化委派

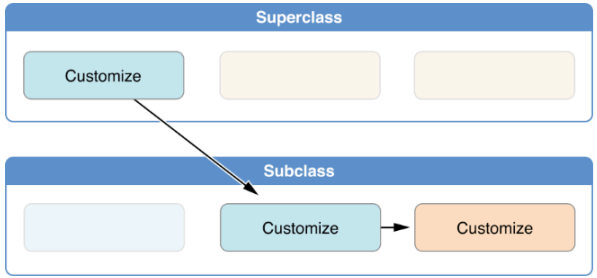




5.類別繼承和初始化

二階段的初始化





5.類別繼承和初始化

初始化的繼承和覆寫1

```
class Vehicle {
    var numberOfWheels = 0
    var description: String {
        return "\(numberOfWheels) wheel(s)"
    }
}

let vehicle = Vehicle()
print("Vehicle: \(vehicle.description)")
// Vehicle: 0 wheel(s)


class Bicycle: Vehicle {
    override init() {
        super.init()
        numberOfWheels = 2
    }
}
```

5.類別繼承和初始化

初始化的繼承和覆寫2

```
let bicycle = Bicycle()
print("Bicycle: \(bicycle.description)")
// Bicycle: 2 wheel(s)

class Hoverboard: Vehicle {
    var color: String
    init(color: String) {
        self.color = color
        // super.init() implicitly called here
    }
    override var description: String {
        return "\(super.description) in a beautiful \(color)"
    }
}

let hoverboard = Hoverboard(color: "silver")
print("Hoverboard: \(hoverboard.description)")
// Hoverboard: 0 wheel(s) in a beautiful silver
```

5.類別繼承和初始化

自動化的初始化繼承

Rule 1

如果子類別沒有定義任何的主要初始化者，自動繼承父類別的所有主要初始化者

Rule 2

如果子類別實做所有父類別的主要初始化者，不管是不是由父類別繼承下來的主要初始化者。則會繼承所有父類別的便利初始化者

5.類別繼承和初始化

實作主要的初始化者和便利初始化者1

```
class Food {
    var name: String
    init(name: String) {
        self.name = name
    }
    convenience init() {
        self.init(name: "[Unnamed]")
    }
}

let namedMeat = Food(name: "Bacon")
// namedMeat's name is "Bacon"

let mysteryMeat = Food()
// mysteryMeat's name is "[Unnamed]"
```

5.類別繼承和初始化

實作主要的初始化者和便利初始化者2

```
class RecipeIngredient: Food {
    var quantity: Int
    init(name: String, quantity: Int) {
        self.quantity = quantity
        super.init(name: name)
    }
    override convenience init(name: String) {
        self.init(name: name, quantity: 1)
    }
}

let oneMysteryItem = RecipeIngredient()
let oneBacon = RecipeIngredient(name: "Bacon")
let sixEggs = RecipeIngredient(name: "Eggs", quantity: 6)
```

5.類別繼承和初始化

實作主要的初始化者和便利初始化者3

```
class ShoppingListItem: RecipeIngredient {
    var purchased = false
    var description: String {
        var output = "\(quantity) x \(name)"
        output += purchased ? " ✔" : " ✘"
        return output
    }
```

```
var breakfastList = [
    ShoppingListItem(),
    ShoppingListItem(name: "Bacon"),
    ShoppingListItem(name: "Eggs", quantity: 6),
]
breakfastList[0].name = "Orange juice"
breakfastList[0].purchased = true
for item in breakfastList {
    print(item.description)
}
// 1 x Orange juice ✔
// 1 x Bacon ✘
// 6 x Eggs ✘
```

5.類別繼承和初始化

可失敗的初始化1

```
struct Animal {
    let species: String
    init?(species: String) {
        if species.isEmpty { return nil }
        self.species = species
    }
}

let someCreature = Animal(species: "Giraffe")
// someCreature is of type Animal?, not Animal

if let giraffe = someCreature {
    print("An animal was initialized with a species of \(giraffe.species)")
}
// Prints "An animal was initialized with a species of Giraffe"

```

5.類別繼承和初始化

可失敗的初始化2

```
let anonymousCreature = Animal(species: "")
// anonymousCreature is of type Animal?, not Animal

if anonymousCreature == nil {
    print("The anonymous creature could not be initialized")
}
// Prints "The anonymous creature could not be initialized"
```

5.類別繼承和初始化

Required 初始化者

```
class SomeClass {
    required init() {
        // initializer implementation goes here
    }
}

class SomeSubclass: SomeClass {
    required init() {
        // subclass implementation of the required initializer goes here
    }
}
```

6.Required 初始化者

```
class SomeClass {
    required init() {
        // initializer implementation goes here
    }
}
```

```
class SomeSubclass: SomeClass {
    required init() {
        // subclass implementation of the required initializer goes here
    }
}
```

7.使用閉鎖和函式設定屬性值

```
class SomeClass {
    let someProperty: SomeType = {
        // create a default value for someProperty inside this closure
        // someValue must be of the same type as SomeType
        return someValue
    }()
}
```

7.使用閉鎖和函式設定屬性值

```
struct Chessboard {
    let boardColors: [Bool] = {
        var temporaryBoard = [Bool]()
        var isBlack = false
        for i in 1...8 {
            for j in 1...8 {
                temporaryBoard.append(isBlack)
                isBlack = !isBlack
            }
            isBlack = !isBlack
        }
        return temporaryBoard
    }()
    func squareIsBlackAt(row: Int, column: Int) -> Bool {
        return boardColors[(row * 8) + column]
    }
}

```

```
let board = Chessboard()
print(board.squareIsBlackAt(row: 0, column: 1))
// Prints "true"
print(board.squareIsBlackAt(row: 7, column: 7))
// Prints "false"
```