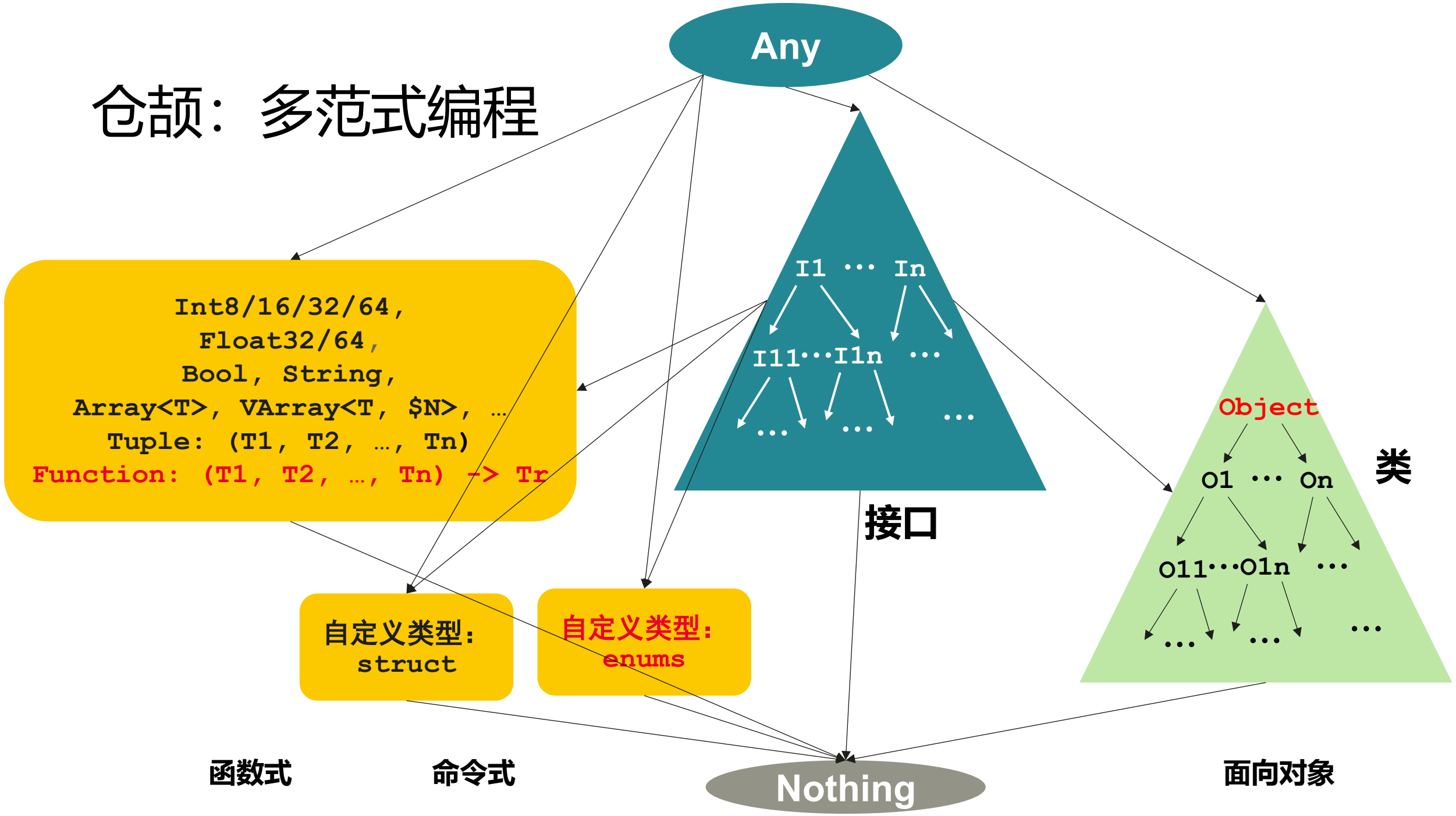
仓颉语言面向对象编程

仓颉：多范式编程


Any
Int8/16/32/64,
Float32/64,
Bool, String,
Array<T>, VArray<T, $N>, …
Tuple: (T1, T2, …, Tn)
Function: (T1, T2, …, Tn) -> Tr
I1 … In
I11…I1n …
接口
Object
O1 … On
O11…O1n …
类
自定义类型:
struct
自定义类型:
enums
函数式
命令式
Nothing
面向对象

面向对象特性

- 类和接口
  - 类似Java
  - 类采用单继承
  - 接口支持多继承
- 所有类型都是Any的子类型
- 所有的class都是Object的子类

```
open class Shape <: Equitable, ToString {
    open func move(x: Int, y: Int): Unit {
        print("Shape moved to ($x, $y)")
    }
    // Other common methods for shapes
}

class Circle <: Shape {
    func move(x: Int, y: Int): Unit {
        print("Circle moved to ($x, $y)")
    }
    // Other common methods for circles
}

class Rectangle <: Shape {
    func move(x: Int, y: Int): Unit {
        print("Rectangle moved to ($x, $y)")
    }
    // Other common methods for rectangles
}

let s1: Shape = Circle()
let s2: Shape = Rectangle()
s1.move(10, 20) // Should print "Circle moved to (10, 20)"
s2.move(30, 40) // Should print "Rectangle moved to (30, 40)"
```

面向对象特性

- 继承
  - 类缺省不能被继承
    - 类似Java中的 “final”
  - 用“open”关键字修饰的类才可以继承
    - 所有interface确实open
  - 只有“open” method 才能被override
    - 类似C++的虚函数
    - override关键字可以省略
    - Interface中的方法都是open的
  - 动态派遣机制
    - open method的调用采用动态派遣

```
open class Shape <: Equitable, ToString {
    open func move(x: Int, y: Int): Unit {
        print("Shape moved to ($x, $y)")
    }
    // Other common methods for shapes
}

class Circle <: Shape {
    override func move(x: Int, y: Int): Unit {
        print("Circle moved to ($x, $y)")
    }
    // Other common methods for circles
}

class Rectangle <: Shape {
    override func move(x: Int, y: Int): Unit {
        print("Rectangle moved to ($x, $y)")
    }
    // Other common methods for rectangles
}

let s1: Shape = Circle()
let s2: Shape = Rectangle()
s1.move(10, 20) // Should print "Circle moved to (10, 20)"
s2.move(30, 40) // Should print "Rectangle moved to (30, 40)"
```

面向对象特性

- Interface
  - 仅提供方法签名
  - 但可以提供default实现
  - 多继承问题
    - 多个interface中对于同名函数，有多个缺省实现 —— 编译报错
- 其他
  - Constructor
    - init函数，可以重载
    - 调用顺序规定与Java类似
  - Finalizer
    - ~init
    - 不允许this逃逸
  - 封装

```
interface Comparable<T> {
    func lt(other: T): Bool
    func ge(other: T): Bool {
        return !lt(other)
    }
    // Other comparison methods can be defined here
}


class MyClass <: Comparable<MyClass> {
    override func lt(other: MyClass): Bool {
        // Implement less-than logic here
        return true // Placeholder implementation
    }
}
```

```
interface I {
    func m(): Int
}

class A {
    let x: Int = 10
    open func foo(): Int {
        return x + 1
    }
}

class B <: A, I {
    let y: Int = 20

    open func baz(): Int {
        return x + 3
    }

    override func m(): Int {
        return y + 1
    }
}
```

```
class C <: B {
    let z: Int = 30

    override func foo(): Int {
        return x + 2
    }

    open func bar(): Int {
        return foo() + 3
    }
}
```