SwiftCon China 2016

www.swiftconchina.com

SwiftCon
www.swiftconchina.com
x
Think
ThinkInLAMP.com

SWIFTCON CHINA 2016

Swift语言的设计取舍及跨语言调用

董一凡

- 十年程序员生涯，八年移动客户端开发，也略懂 web 前端后端的开发
- 拇指阅读，SketchBook App 的主要开发者
- 刚刚结束了 Autodesk 数字艺术部门的工作，目前是一名快乐的自由职业者

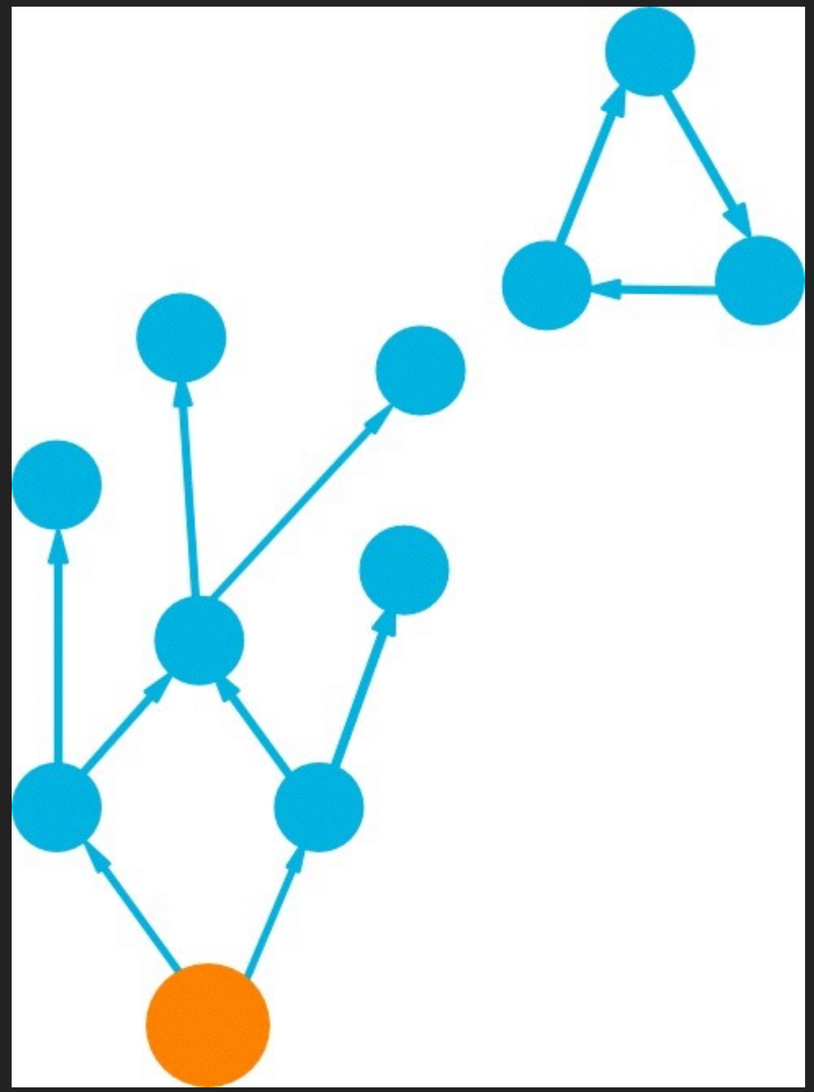
七镜花园

主要内容

- GC 与 ARC
- 值语义与引用语义
- 面向对象中的继承
- 跨语言调用

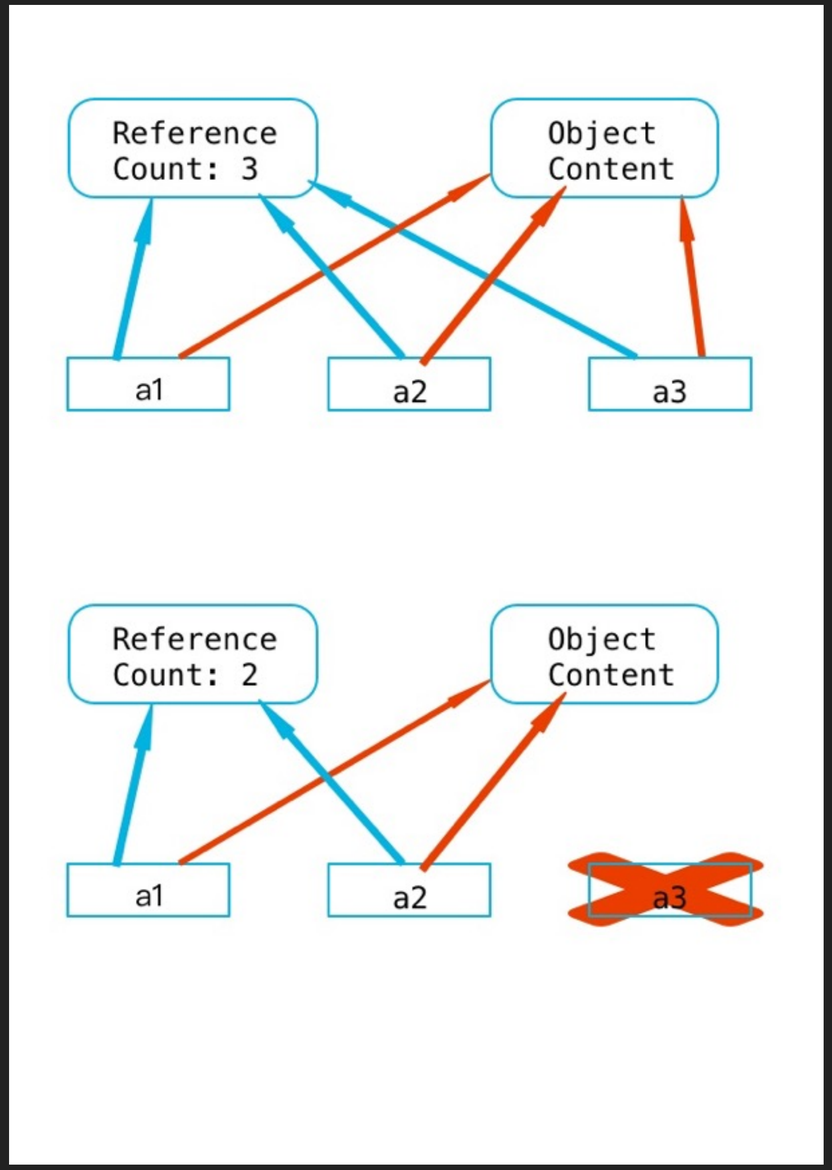
GC 是指什么

- 以 Java 为代表
- 标记删除型的 GC



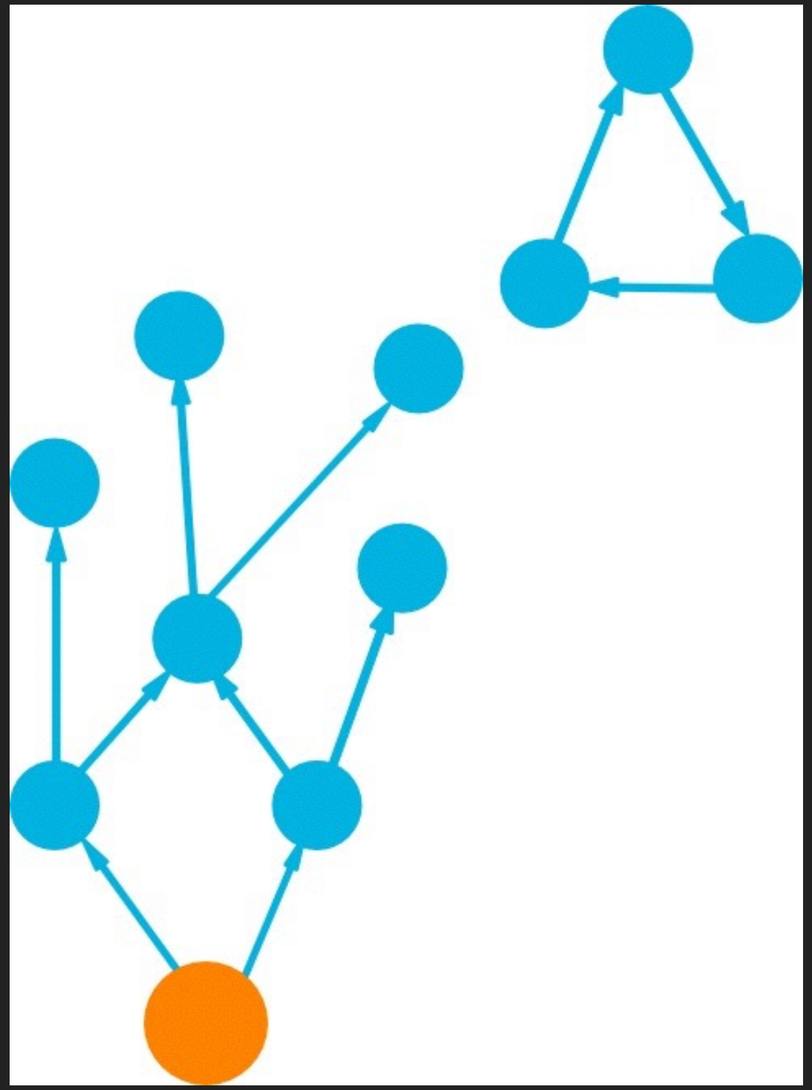
ARC的本质

- 引用计数


Reference
Count: 3
Object
Content
a1
a2
a3
Reference
Count: 2
Object
Content
a1
a2
a3

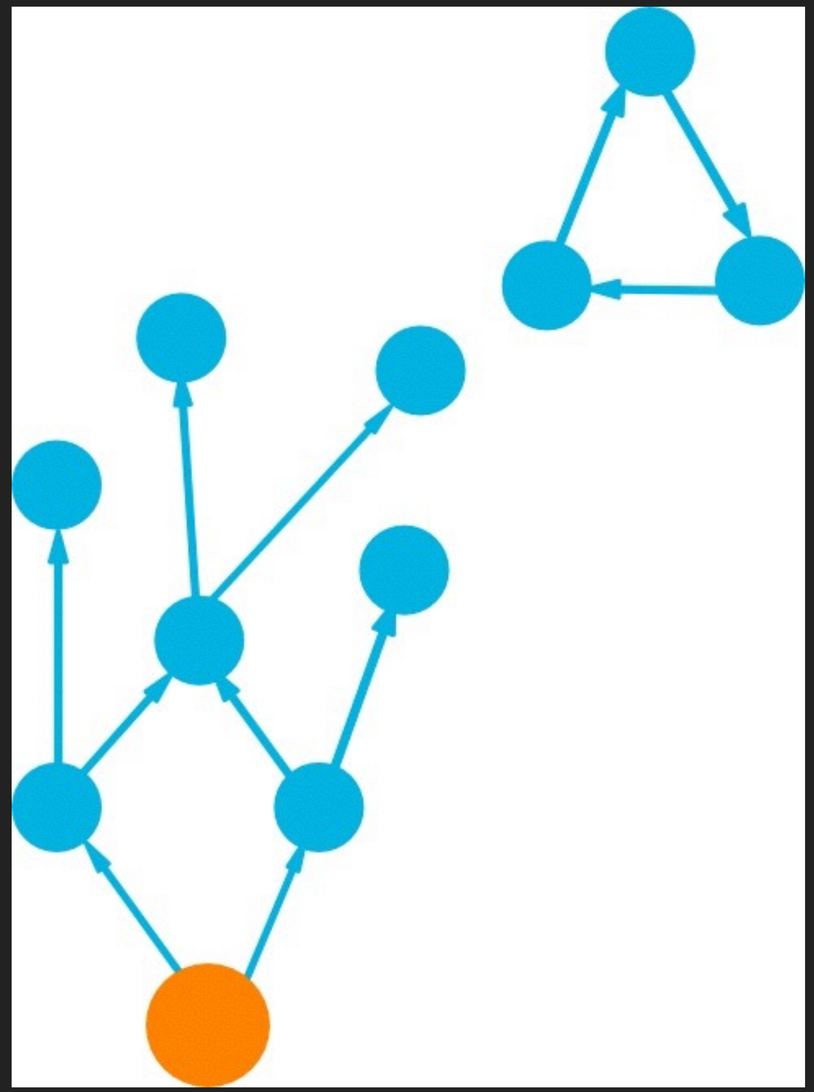
GC

- 优势
  - 内存清理彻底
- 劣势
  - 世界停止问题
  - 非内存资源的管理



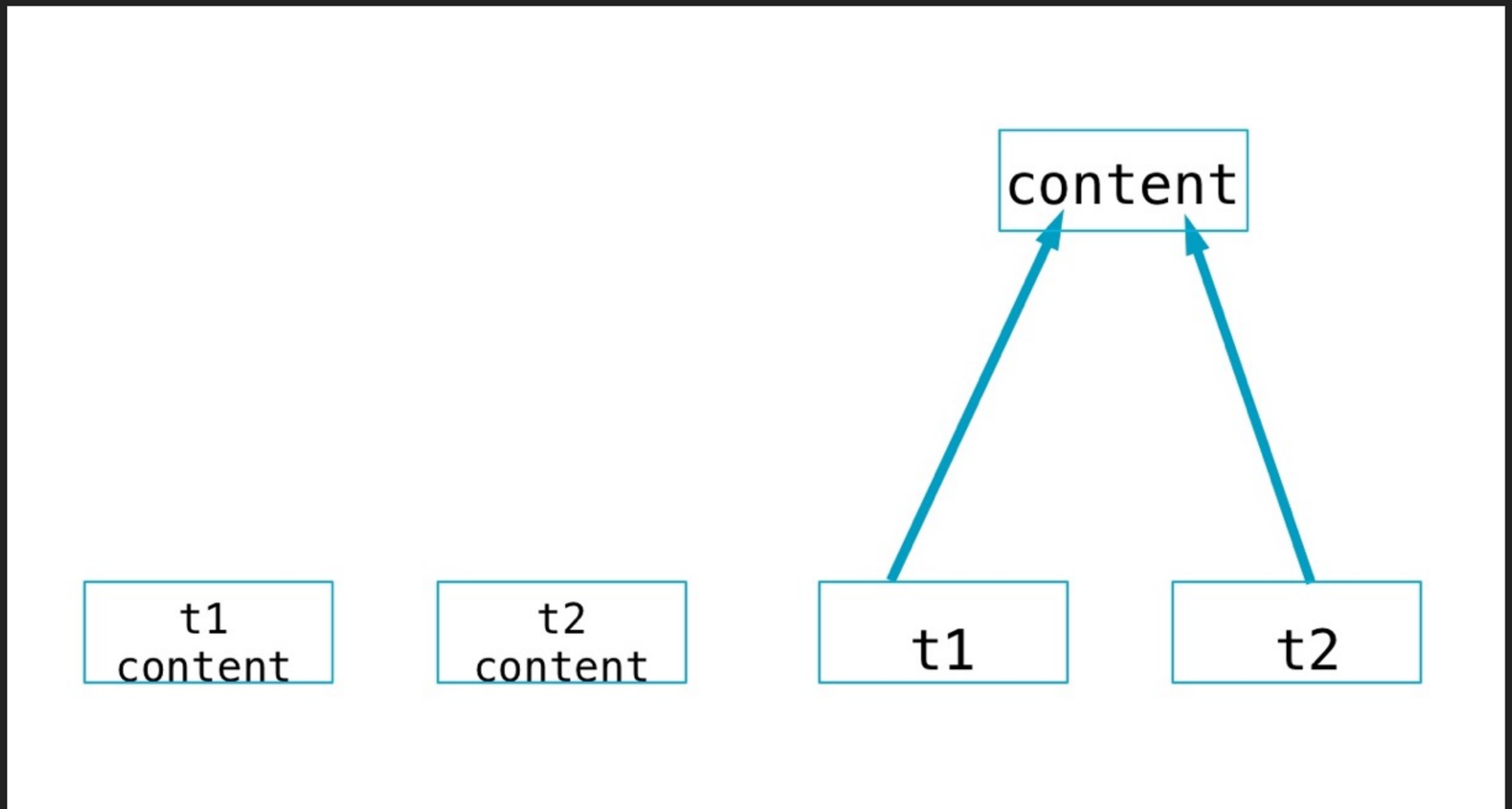
ARC

- 优势
  - 可以明确知道何时回收
  - 可管理非内存资源
- 劣势
  - 循环引用的内存无法清理



值语义与引用语义

```
let t1 = Tree()
t2 = t1
```



值语义的特点

- 降低心智负担

```
let tree1 = Tree()
let tree2 = tree1

let i1 = 1
let i2 = i1
```

值语义的特点

- 多线程友好

```
func rename(tree: Tree) -> Tree {
    var t = tree
    t.name = "pine"
    return t
}
```

值语义的特点

- 通过协议实现异构数组

```
protocol RenameProtocol {
    func rename(name:String)
}

struct Tree : RenameProtocol{
    func rename(name:String) {
    }
}

struct Room : RenameProtocol {
    func rename(name: String) {
    }
}

let a : [RenameProtocol] = [Tree(), Room()]
```

值语义的特点

- 没有 deinit 方法
- 可用 defer 进行操作

预先假设

```
class Parent{

}
class Child : Parent {

}
```

Subtypes must be substitutable for their base types

Liskov Substitution Principle(李氏替换原则)

协变与逆变

```
class Base {
    func doSomethind(child: Child) -> Parent {
        return Parent()
    }
}
class Derived : Base {
    override func doSomethind(child: Child) -> Parent {
        return Parent()
    }
}
```

协变与逆变

```
class Base {
    func doSomethind(child: Child) -> Parent {
        return Parent()
    }
}
class Derived : Base {
    override func doSomethind(parent: Parent) -> Child {
        return Child()
    }
}
```

```
let p : Parent = b.doSomethind(Child())
```

Java 中的一个奇怪设计

```
Derived[] deriveds = new Derived[5];

Base[] bases = deriveds;

base[0] = new Base();

Derived d = deriveds[0];
```

Java 中的一个奇怪设计

```
void doSomething(Base[] bases) {
    base[0] = new Base();
}

Derived[] deriveds = new Derived[5];
doSomething(deriveds)
```

Swift & C

XXX.c file:

```c
int getTheAnswer() {
    return 42;
}
```

XXX-Bridging-Header.h

```c
int getTheAnswer();
```

swift file:

```swift
let i = getTheAnswer()
```

Swift & C++

```
class Document {
public:
    void setTitle(const std::string& title) {
        title_ = title;
    }

    std::string getTitle() {
        return title_;
    }

private:
    std::string title_;
};
```

Swift & C++

DocumentWrapper.mm

```
@interface DocumentWrapper () {
    Document doc;
}

@end

@implementation DocumentWrapper

- (NSString *)getTitle {
    return [NSString stringWithUTF8String:doc.getTitle().c_str()];
}
- (void) setTitle:(NSString*)title {
    doc.setTitle(std::string([title cStringUsingEncoding:
        NSUTF8StringEncoding]));
}
```

Swift & C++

XXX-Bridging-Header.h

```
#import "DocumentWrapper.h"
```

Swift

```
let doc = DocumentWrapper()
doc.setTitle("this is a title")
print(doc.getTitle())
```

谢谢