本课程文字讲义中涉及到知识点和概念的使用参见视频教程，您可通过以下方式联系并加入开发者引擎社区获取：

第三课 Cadence 语言深入学习（下）

本节课与第二课为深入学习 Cadence 语言，主要分为三个部分：

1、深入函数，结构体，资源 （相关讲解与演示在第二课）

2、Cadence 语言中的高级概念

3、引用、事件及其他

二、Cadence 语言中的高级概念

2.1、合约

合约的概念肯定都不陌生了，但真要单独拎出来讲，其实合约本身也没有太多难点的东西，主要强调以下 5 点内容：

1、使用 contract 关键字定义，位于顶层作用域；

2、合约定义不允许嵌套；

3、定义的资源不能在合约外创建，必需提供一个资源创建函数；

4、部署到某个账户下的合约中可以使用 self.account 可以获取到该账户，参见视频教程演示；

5、Flow 支持对已经部署的合约进行更新，涉及很多的条条框框，本文不再专门分析，参见官方文档。

2.2、接口

1、定义和声明

Cadence 中支持 struct, resource, contract 三种类别接口，由类别关键字+interface+自定义接口名定

义。其中合约接口在下节课在解析 ft 和 nft 标准的时候会遇到，而这节课我们主要以资源接口为例，结构体接口和资源接口区别差不多。

2、主要作用

一方面，接口中可以定义一组行为成员函数，也可以声明成员常(变)量，控制修饰符不可使用私有权限 priv/access(all)。实现者需要原样全部实现，即接口中定义的成员量和行为函数每个都要在实体中实现。

另一方面，接口更主要的作用在于权限的分离与控制。如下所示代码片段：

```
pub resource interface Count {
    pub fun getBookCount(): Int      // 获取书架上的书籍个数
}

pub resource interface Admin {
    pub fun addBook(book: @Book)     // 向书架上添加一本书籍
    pub fun delBook(index: Int)      // 从书架上删除一本书
}

pub resource BookShelf: Count, Admin {
    pub let books: @[Book]

    pub fun getBookCount(): Int {
        return self.books.length
    }

    pub fun addBook(book: @Book) {
        self.books.append(<- book)
    }

    pub fun delBook(index: Int) {
        let bk <- self.books.remove(at: index)
        destroy bk
    }
```

首先如上图，BookShelf 书架资源实现了 Count 和 Admin 接口，实现方式为定义的实体后加冒号，后接要实现的接口，多个接口之间用逗号隔开。

Count 接口可普遍化为只读接口，而 Admin 相当于具有写权限的接口。如果没有接口，就只能拿到完整的书架资源，这样谁都可以读和写，显然不能满足实际的业务开发。这就是 Cadence 中接口最主要的作用，稍后演示。

3、接口实现约定

1）前已述及，全部原样实现；

2）接口中定义常(变)量时：如果接口明确指定了常量 let/变量 var，则在实现时必须保持一致；如果接口中没有指定，则在实现时既可以是线程常量，也可实现为变量。

3）接口中定义的成员函数支持前置和后置条件块，在实体的实现函数中无需重复声明即适用，如下所示：

```
pub resource interface Admin {
    pub fun addBook(book: @Book)    // 向书架上添加一本书籍
    pub fun delBook(index: Int) {   // 从书架上删除一本书
        pre {
            index >= 0: "invalid index"
        }
        post {
            // ...
        }
    }
}
```

BookShelf 中实现的 delBook()方法中无需重复声明 pre{}条件块，在调用执行的时候，Cadence 同样会执行判断 pre{}，验证传入的 index 是否为非负数。参见视频教程的演示。

4、获取不同权限的资源演示

获取一个具有限制权限的资源，如下所示，参见视频教程的演示。核心语法形式为使用一对大括号指定权限接口，如{Count}，此时便不可再访问{Admin}权限的接口行为。

```
let bsf <- create BookShelf()

let bfCnt: @BookShelf{Count} <- bsf // 只读权限的资源
bfCnt.getBookCount() // 获取书籍个数

bfCnt.delBook(index: 2)   //报错，不允许

destroy bfCnt
```

2.3、账户

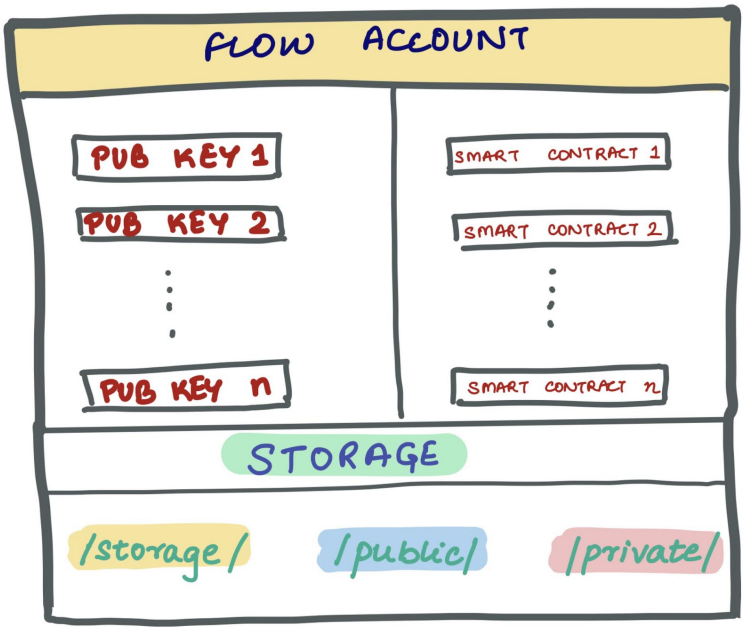
1、账户的基本概念

与以太坊相比，Flow 上的账户它不仅仅是一个地址，它除了地址以外，它还包含一块存储区域，存储着账户所拥有的资源，账户关联的公钥，以及在这个账户下部署的合约。这里有两篇文章，如果有兴趣的同学可以看一下这两者之间的比较。

Flow 账户与 Ethereum 账户有何不同？

Flow VS 以太坊， 深度对比两条公链以及合约开发语言

下图为文章当中所来画的一个 flow 账户的全貌图，能够清晰的看到存储的公钥以及在此账户下部署的合约。



最重要的一部分是 storage 存储区域，这是我们要重点关注的。在 Cadence 中，为了能访问 Storage 这片存储区域，引入了三个域，/storage、/public 和/private。简单来说，这三个域就相当于平时电脑上的文件目录。那么这三个域是怎么分配的呢？是不是把整个 Storage 空间平均分成了三份呢？比如前 1/3 属于/storage 域指向，中间的 1/3 是/public 域指向的，最后剩下的 1/3 是由/private 域指向的。其实不是，这三个域的含义如下：

/storage 域： 指向真正的存储空间，只有 Auth 账户访问；

/public 域： 协助实现基于 Capability 权限控制，Public 账户可访问；

/private 域：协助实现基于 Capability 权限控制，Public 账户不可访问；

其中，Auth 账户和 Public 账户是 Flow 中账户的两种不同情形下的角色，稍后再说。对于/private 和/public 再通俗写可以理解成 Linux 系统上的软链接，或者 Windows 系统上的快捷方式，它们指向账户下真正的存储区域/storage。

在真正使用的时候，这三个区域分别对应三种 Path 类型： StoragePath, PublicPath, PrivatePath。其中，PublicPath, PrivatePath 是 CapabilityPath 的子类型；而 StoragePath 和 CapabilityPath 是 Path 的子类型。所以，这三个并不是三分天下的并列关系。

此外，这三个 Path 可以与 String 类型相互转化：

```
fun toString(): String
```

```
fun PublicPath(identifier: string): PublicPath?
fun PrivatePath(identifier: string): PrivatePath?
fun StoragePath(identifier: string): StoragePath?
```

2、账户的角色

账户在我们真正的编程过程当中，存在两种角色，Public 公共账户和 Auth 授权账户。

1）授权账户 Auth

Auth 账户的权限是最大的，它可以访问整个账户下的存储空间。当然权限越大，那么限制也就越多，授权角色的账户只存在于两个地方：第一个就是我们之前接触过的就 transaction 中 prepare()函数的入参；

第二个地方就是我们之前提到过的合约当中使用 self.account 获取到的合约部署账户实际上是一种授权账户。

常用 API 如下：

```
fun save<T>(_ value: T, to: StoragePath)  // 将创建的资源保存到账户的Storage中
fun load<T>(from: StoragePath): T?   // 与save相反，取出资源

// 引用一个资源，与Capability的borrow()方法区分，Public账户没有
fun borrow<T: &Any>(from: StoragePath): T?

// 建立一个访问路径权限
fun link<T: &Any>(_ newCapabilityPath: CapabilityPath, target: Path): Capability<T>?

// 由CapabilityPath获取一个访问路径权限
fun getCapability<T>(_ path: CapabilityPath): Capability<T>
```

2）公共账户 Public

Public 账户可以访问账户中所有通过/public 域指向的资源。公共角色账户的获取几乎没有限制，在 transaction 和 script 中的任何地方都可以通过如下 getAccount()函数来获取。

```
fun getAccount(_ address: Address): PublicAccount
```

其常用 API 如下：

```
// 由CapabilityPath获取一个访问路径权限，注意与Auth账户的不同
fun getCapability<T>(_ path: PublicPath): Capability<T>
```

2.4、Capability 权限控制

这并不是 Cadence 独有引入的概念，简而言之，其概念如下：

Capability = 对象资源 + 一组授权操作 + 执行句柄

参见 https://en.wikipedia.org/wiki/Capability-based_security 中的 Example 示例：

/etc/passwd 不是一个 Capability

/etc/passw 和 O_RDWR 不是一个 Capability

int fd = open("/etc/passwd", O_RDR); 是一个 Capability

在 Cadence 语言中，这个文件资源就是指一个资源 resource；对这个文件资源的各种操作就对应着定义的各种权限的资源接口；最后这个执行句柄，允许他人访问和操作这个资源，对应着 Cadence 语言中有两个方式：第一个是之前提到的获取不同权限的资源；第二种就是接下来要演示的可以用 link()将/public 等路径链接到 资源存储的 storage 路径。实际编程中常见第二种。

2.5、交易 transaction

在之前的内容的演示中，都涉及到了交易 transaction 的使用，总结如下：

1、使用关键字 transaction 声明创建，可接收参数

2、用于改变链上数据存储状态

3、分为四个执行阶段：

1）prepare(): 入参为签名账户，处理签名账户的相关逻辑；
2）pre(): 交易开始前，相关数据的合法性检验；
3）execute: 真正的执行体；
4）post(): 验证交易执行是否复合预期。

2.6、脚本 Script

同样在之前的内容的演示中，都涉及到了脚本 Script 的使用，总结如下：
1、入口执行函数为 main()函数，可接收参数；
2、用于读取链上存储的数据；
3、脚本式的顺序执行。

2.7、实践演示

本节涉及到账户角色，基于 Capability 权限控制，以及资源的综合演示，参见视频教程，要点如下：
1、合约部署到账户 0x1 上，使用账户 0x1 签名。在 prepare 阶段获取公共和授权账户角色，并通过 Capability 访问资源。
2、合约部署到账户 0x1 上，使用账户 0x1 签名。在 execute 执行阶段获取公共账户角色，并通过 Capability 访问资源。
3、合约部署到账户 0x1 上，使用账户 0x2 签名。只能通过账户 0x1 的 Public 权限访问资源。

三、引用、事件及其他

3.1 引用

Cadence 中引用的含义与其他语言中引用的含义类似，相当于指针指向一个资源。引用不是一种资源，所以不会在账户下存储。本节参见视频教程。
1、基本的使用
&符号表明是一个引用，如下：

```
let bsf <- create BookShelf()

let bsfRef = &bsf as &BookShelf        // 获取资源的引用

let bsfCnt: &{Count} = &bsf as &{Count}      // 只读权限的引用

destroy bsf
```

关于以上注意两点：1）对资源的引用使用等号=，不是移动<-符号；2）&{Count}中的 Count 为资源实现的接口，用于限定引用的权限。
2、授权引用
Cadence 还支持一种授权的引用，使用关键字 auth，这种引用普通引用的不同在于：授权引用可以再转换成本体的引用，示例如下：

```
let bsf <- create BookShelf()

let bsfCnt: auth &{Count} = &bsf as auth &{Count} // 获取一个授权的只读引用
log(bsfCnt.getBookCount())
bsfCnt.delBook(index: 0)   // 不可以调用Admin接口的方法

let bsfr: &BookShelf? = bsfCnt as? &BookShelf    // 将授权引用再转换为本体引用
bsfr!.delBook(index: 0)  // 可以调用Admin接口的方法了

destroy bsf
```

3.2、事件

1、使用 event 关键字定义声明
2、使用 emit 关键字发送事件
3、Flow 包含一下默认的核心事件，参见官方文档 core events

3.3、枚举

严格来说，枚举类型也是一种数据类型，可划归到第一课的内容，但官方文档上是把它单独列出来了，本课程就放到这里了。

1、使用 enum 和 case 关键字
2、示例：

```
pub enum Color: UInt8 {
    pub case red
    pub case green
    pub case blue
}
let blue: Color = Color.blue
```

3.4、其他

这部分内容包括获取到当前的块，运行时环境信息，也包括一些内置的函数的介绍等等。主要先了解一下，有个印象，若真正用到时再去查阅官方文档。

1、环境信息：Environment Information
    getCurrentBlock() 等
2、运行时类型　　Run-Time Types
    获取变量的类型，getType()
3、内置函数 Built-in Function
    panic(), assert()
4、加密算法　　Crypto

本节课结束~