1、Solidity 语言结构

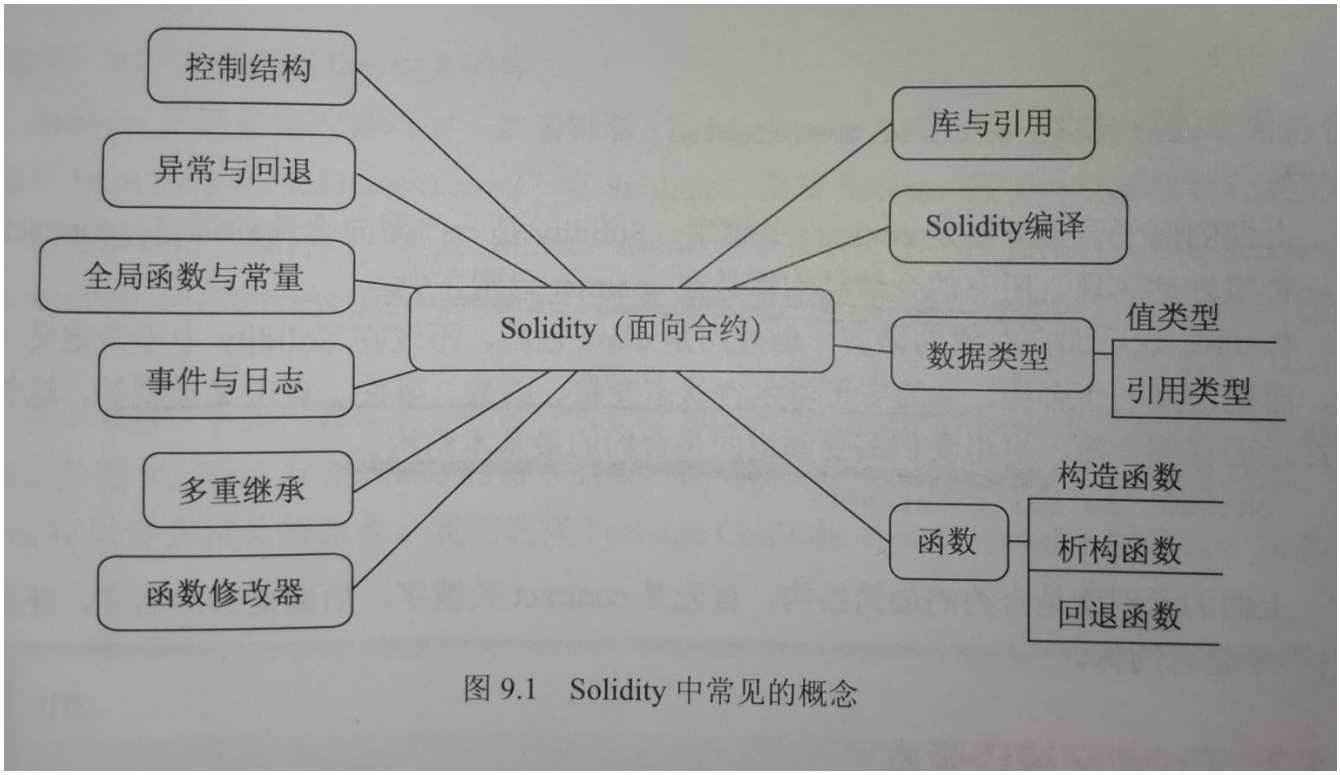


图 9.1　Solidity 中常见的概念

值类型：当用在函数参数或者赋值的时候，始终执行的都是复制操作。

引用类型：当用在函数参数或者赋值的时候，不一定都是赋引用。根据数据位置的不同有可能执行的是复制操作，也可能是赋引用。

2、值类型

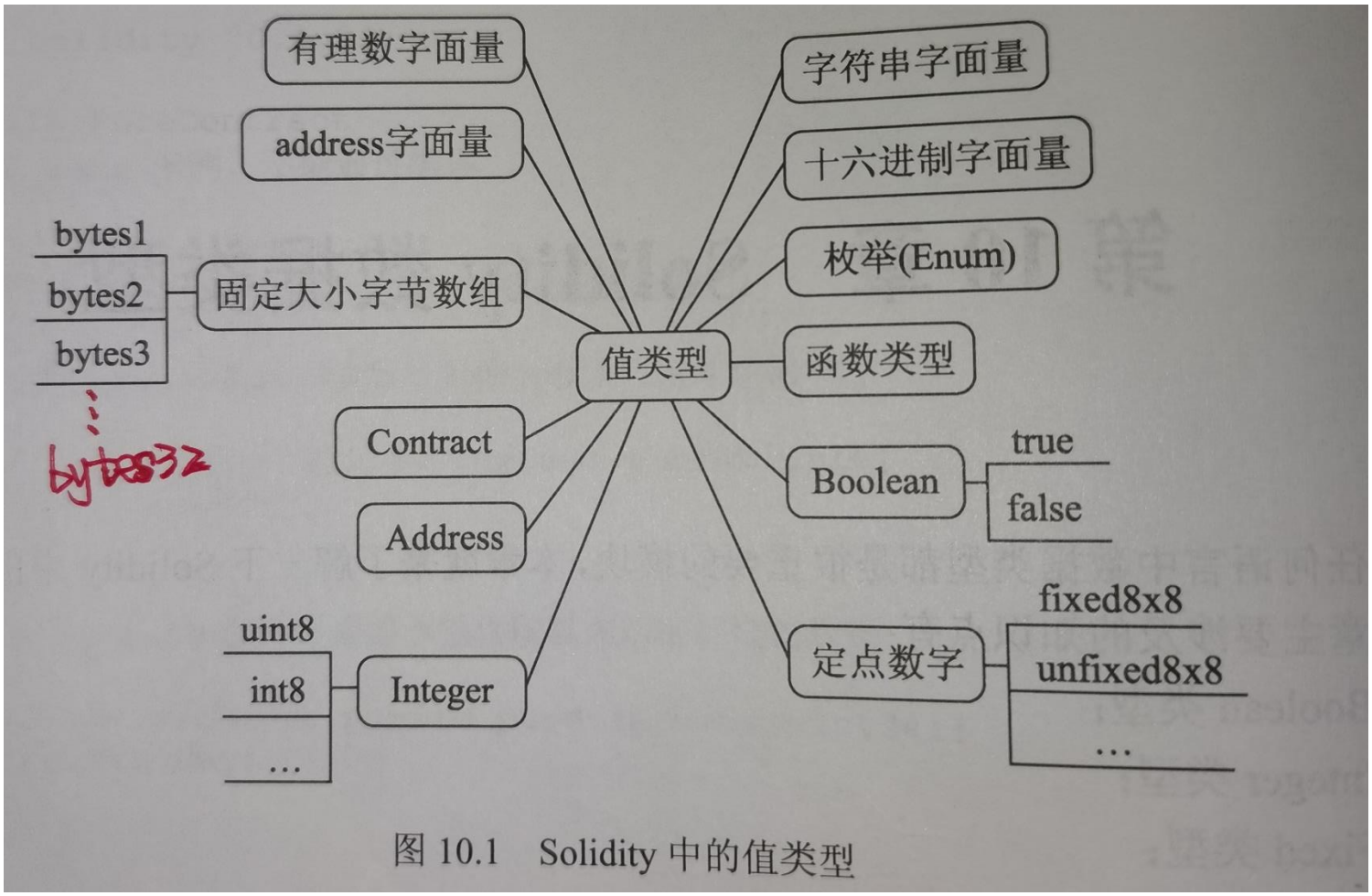


图 10.1　Solidity 中的值类型

3、引用类型

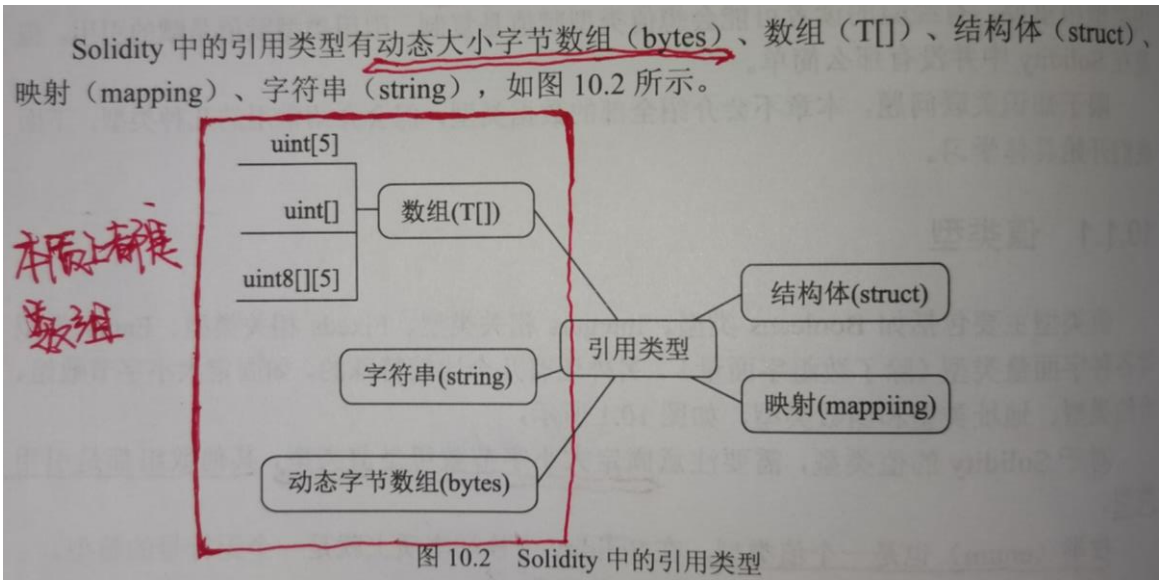


图 10.2　Solidity 中的引用类型

4、固定大小字节数组

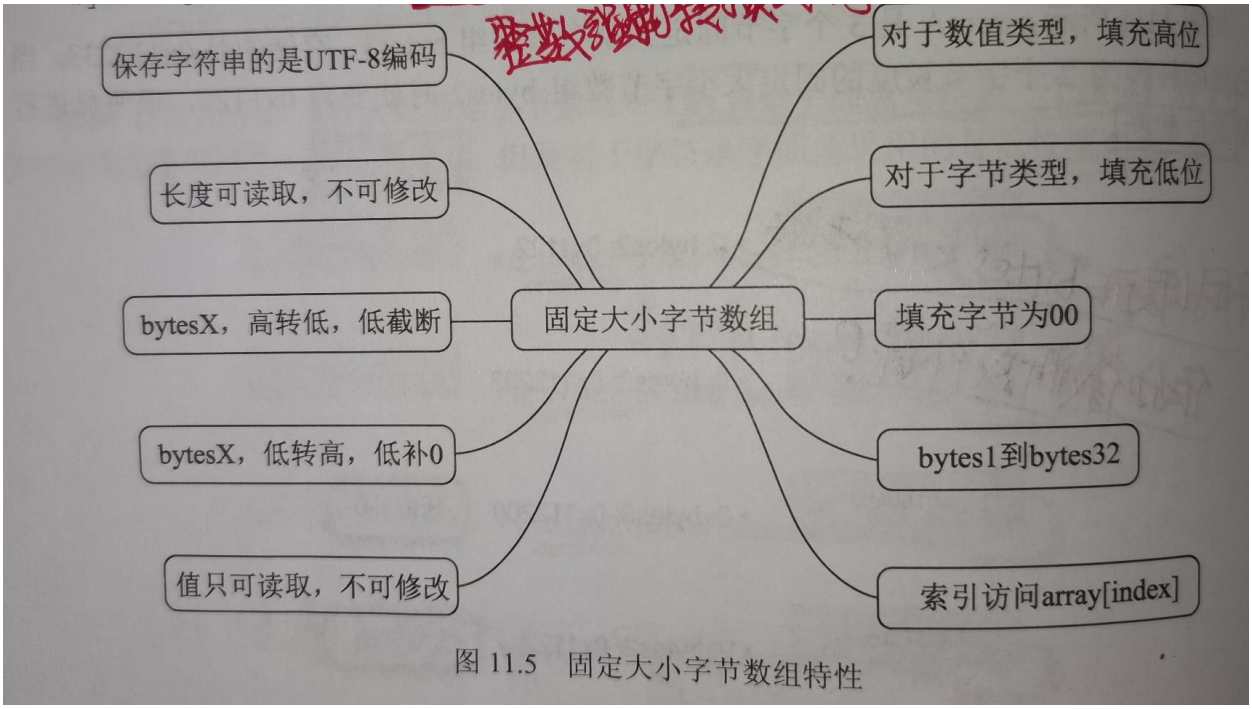


图 11.5　固定大小字节数组特性

固定大小字节数组中存放**字符串字面量时**，其中存放 UTF-8 编码，不同码位所用占字节数也不同，utf-8 编码兼容 ascii 码，所以汉子占 3 个字节。**但此时采用低位填充 0**，

如：bytes2 public h = "h"; 实际存储效果是：0x6800

存放有理数字面量时，有理数字面量作为一个整体，只使用需要的字节，**高位填充 0**。

如： bytes3 public numC = 256; 实际存储效果是：0x000100

存放十六进制字面量时，每两位一个字节，0x1122 是十六进制数据，hex“1122”是十六进制字面量前者高位填充，后者在低位填充。

如：bytes5 public dataC = 0x1122;**实际存储效果：bytes5: 0x0000001122**

bytes5 public dataD = hex"1122";**实际存储效果：bytes5: 0x1122000000**

5、动态大小字节数组

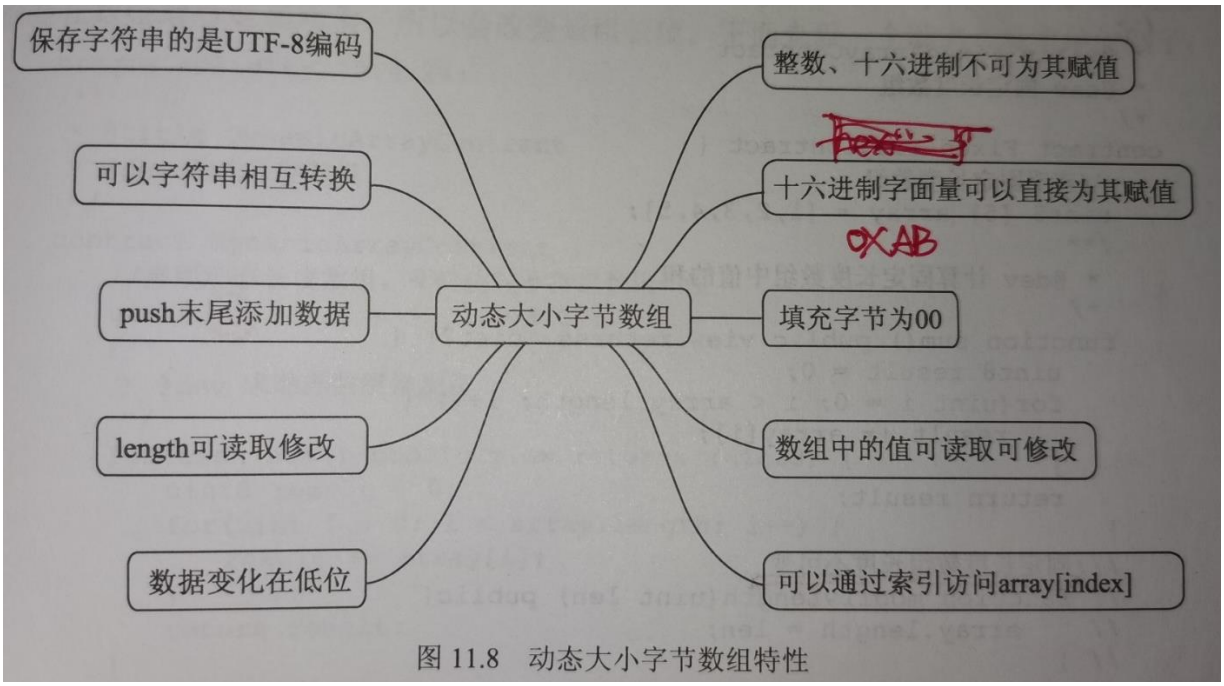


图 11.8　动态大小字节数组特性

不确定数据字节大小，可以使用 string 或 bytes,如果能将数据控制在 bytes32 以内，考虑使用固定大小字节数组。Memory 数据位置的动态大小字节数组不能使用 push 成员。

```
contract DynamicBytesContract {
    // bytes public bytesA = 255;
    // bytes public bytesA = bytes(byte(255));
    // bytes public hexNum = 0x112233;
    bytes public hello = "hello";                    //字符串赋值给动态大小字节数组
    bytes public hexLiteral = hex"112233";//十六进制字面量赋值给动态大小字节数组
    bytes public bytesB = new bytes(2);              //使用 new 关键字创建动态大小字节数组
}
```

怎么创建？可以使用 new 关键字，和创建数组类似，如上例所示，参数 2 是指定动态大小字节数组大小的初始值，就是指定 length 的值。当然也可以直接赋值给动态大小字节数组，但只能是字符串或者十六进制字面量可以直接赋值给动态大小的字节数组，整数、十六进制值和固定大小数组都不可以直接赋值给动态大小字节数组。

如图 11.6 所示为创建之后的初始值，不同的情况初始值有所不同，使用 new 关键字创建的动态大小字节数组，数组的初始值为 0，对于字符串赋值给动态大小字节数组的情况，动态大小字节数组中的初始值就是字符串对应的 UTF-8 编码，长度也和字符串一致，对于十六进制字面量，2 个十六进制字符占用一个字节。

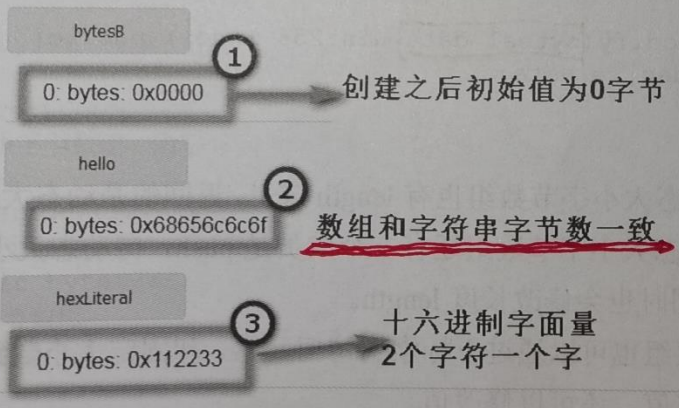


图 11.6　动态大小数组初始值

6、数组

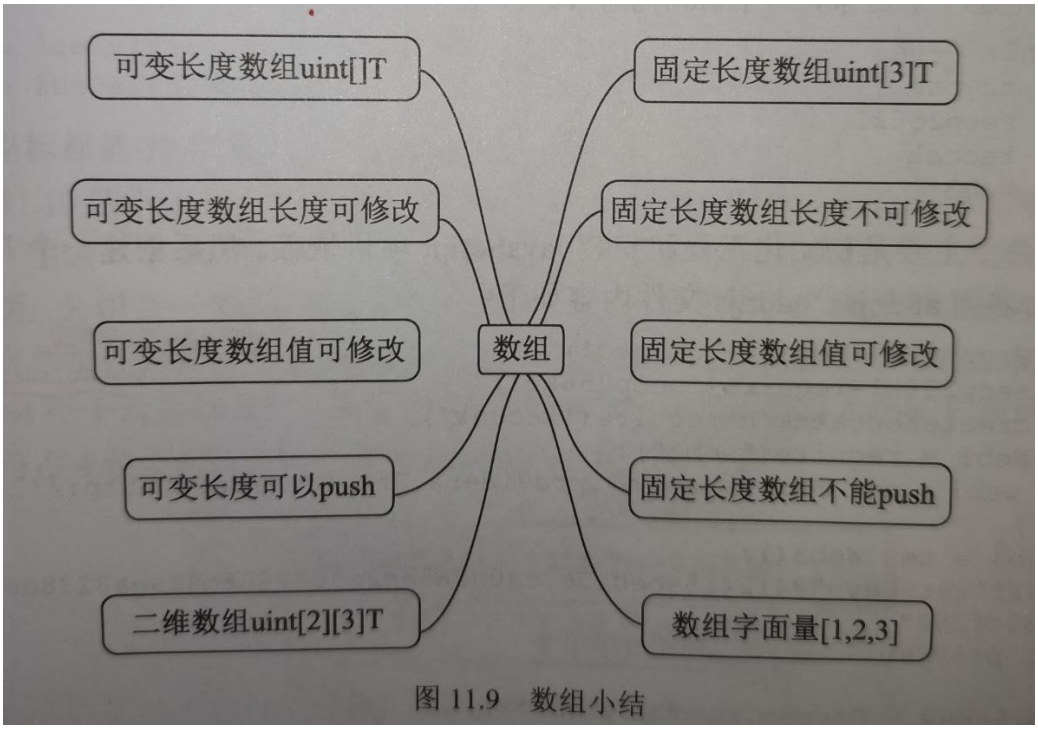
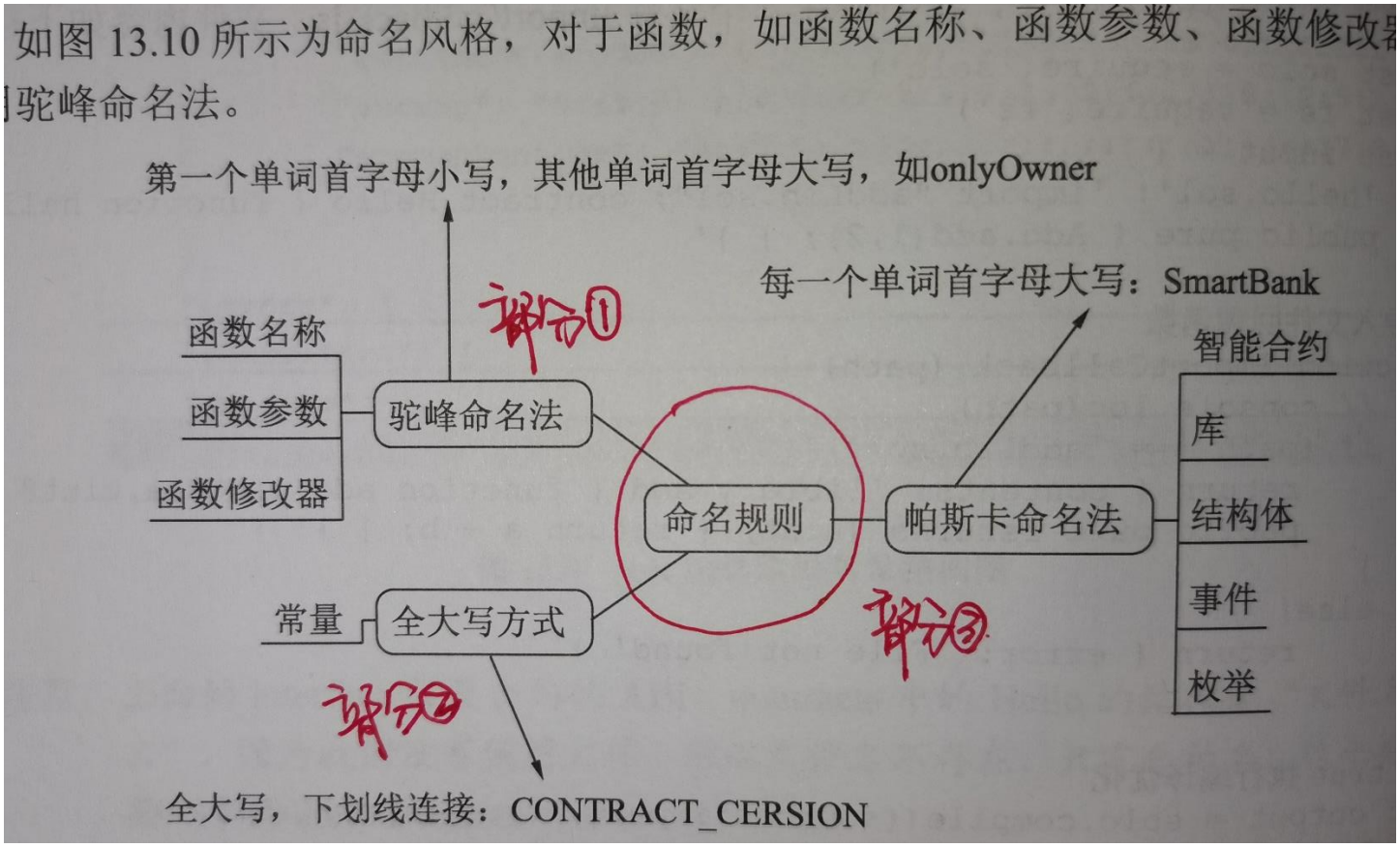
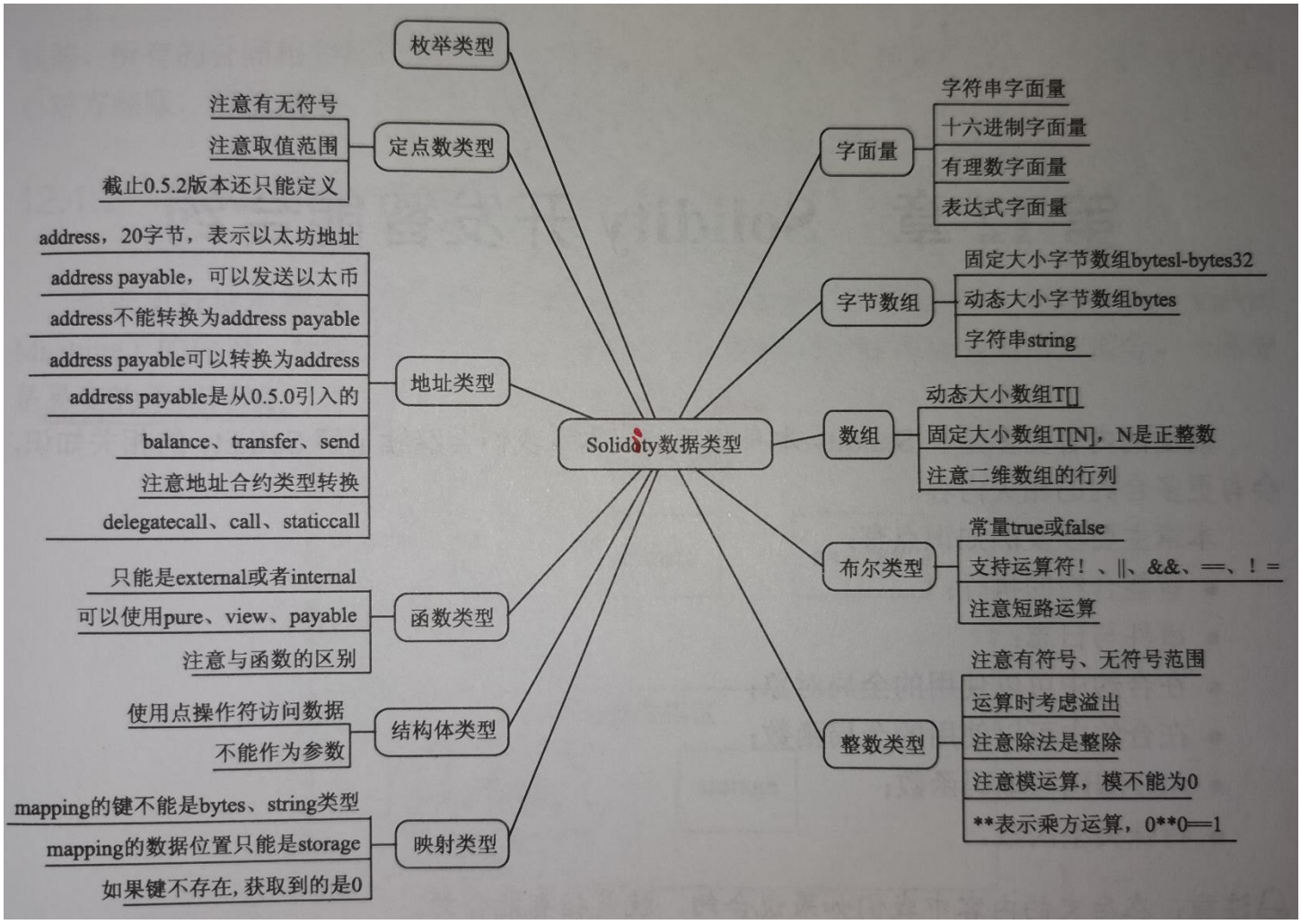


图 11.9　数组小结

7、solidity 变量命名风格



8、solidity 数据类型总结



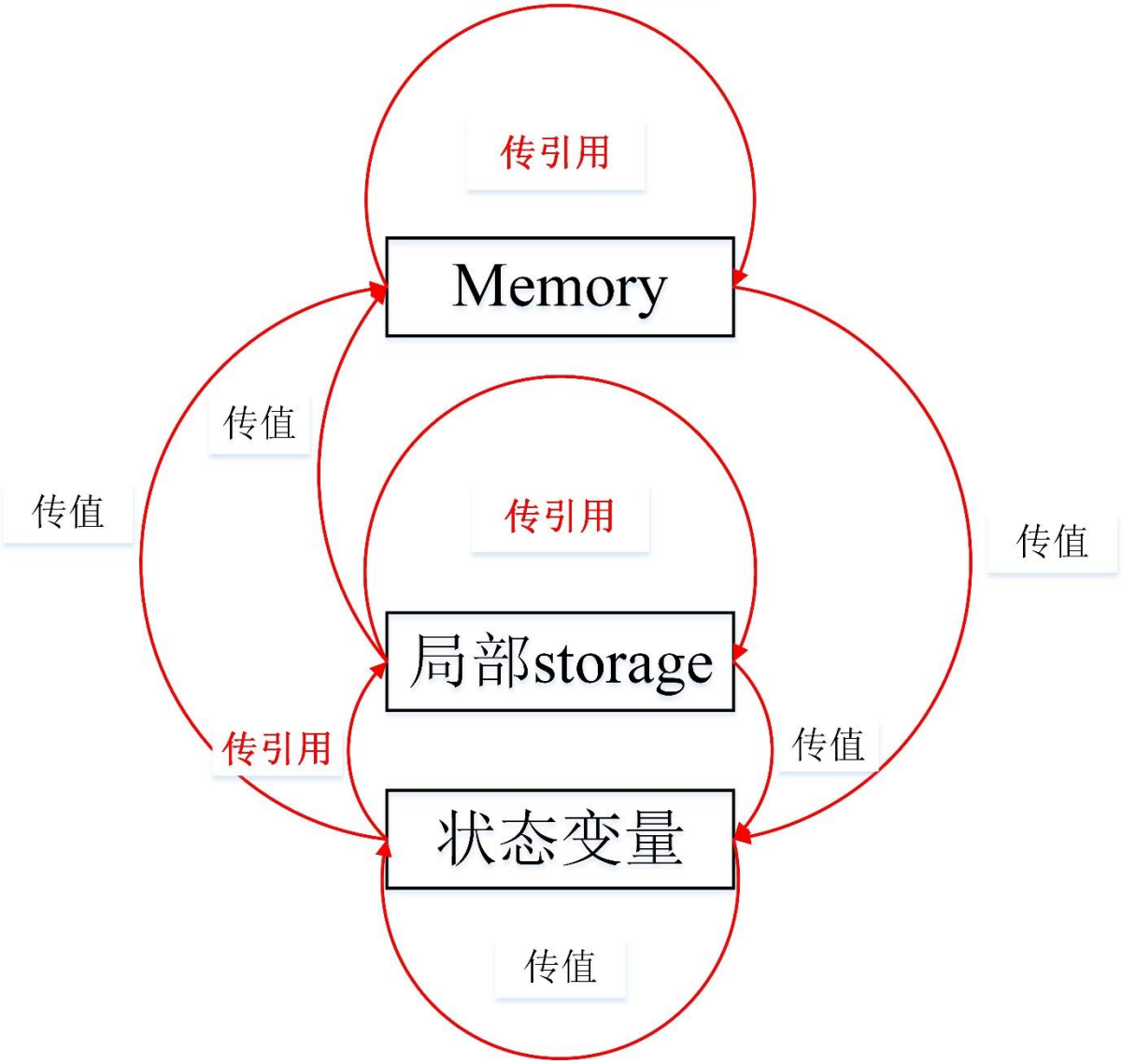
9、Solidity 不同位置间数据赋值关系表

表 13.9　类型转换与操作

转换类型	执行操作
状态变量→状态变量	拷贝
状态变量→memory局部变量	拷贝
状态变量→storage局部变量	指针
局部memory变量→局部memory变量	指针
局部memory变量→状态变量	拷贝
局部memory变量→局部storage变量	X（不能直接转换）
局部storage变量→局部storage变量	指针
局部storage变量→局部memory变量	拷贝
局部storage变量→状态变量	拷贝

10、关系图如下所示

Memory、状态变量与局部storage之间的赋值关系图，箭头代表赋值方向

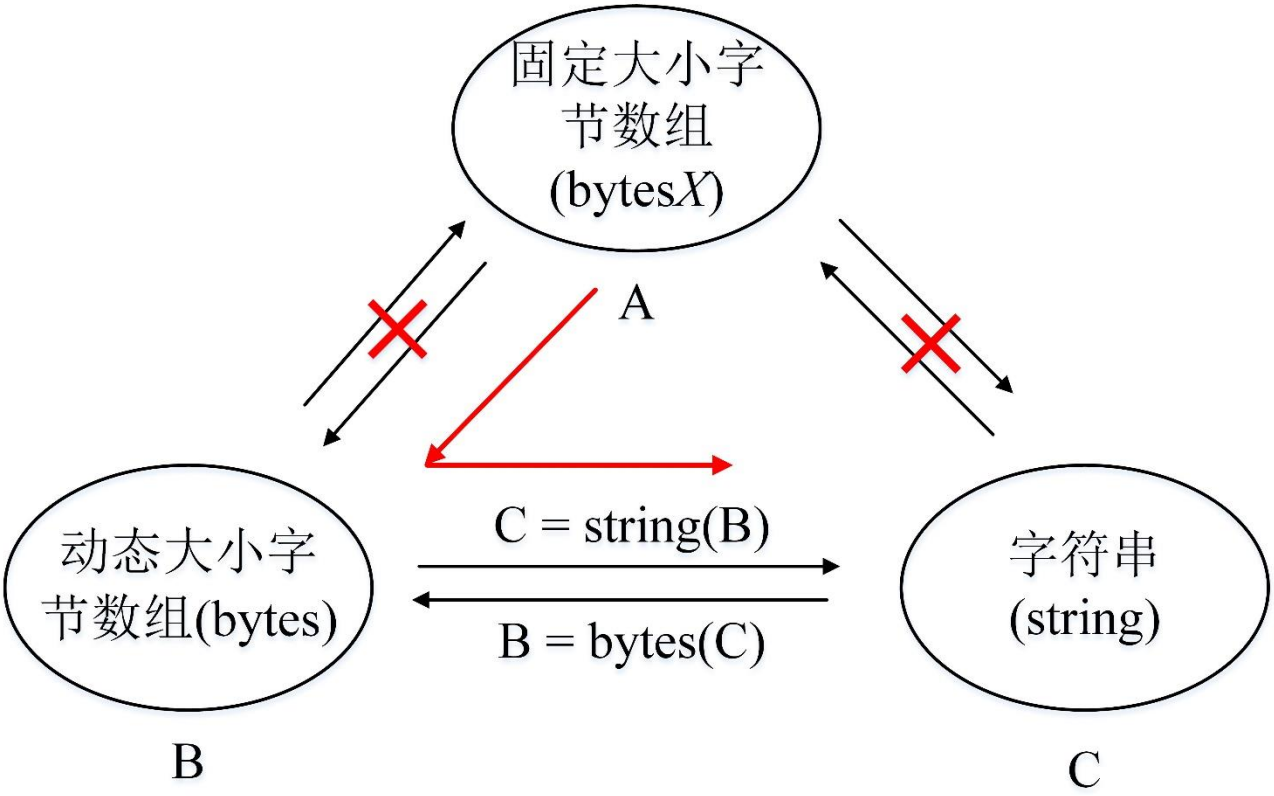


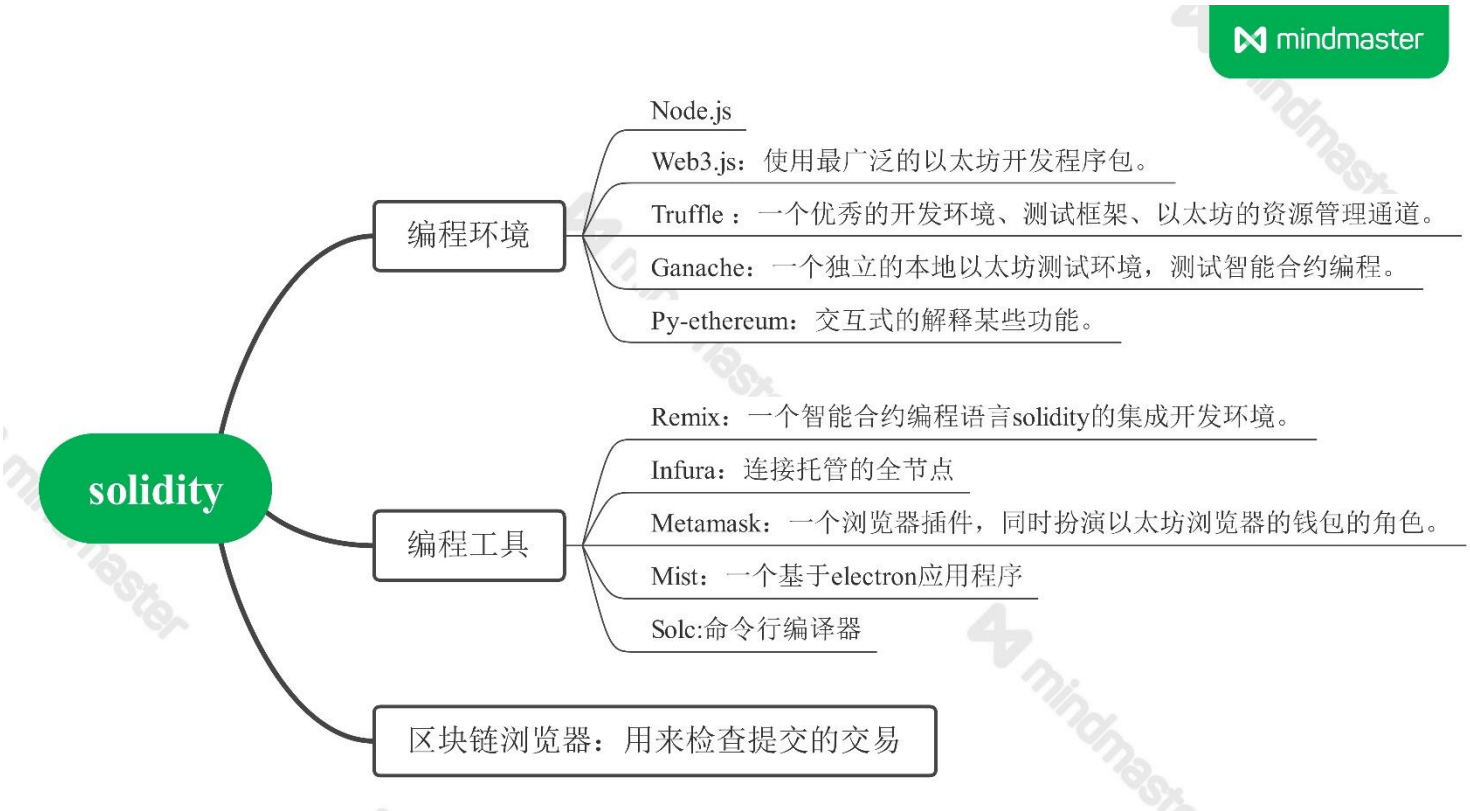
11、字节数组与字符串之间的转换表

表 11.3　字节数组与字符串间转换

转换方式	强制转换	间接转换
固定大小字节数组→动态大小字节数组	不能	可以
固定大小字节数组→字符串	不能	可以
动态大小字节数组→固定大小字节数组	不能	不能
动态大小字节数组→字符串	可以	可以
字符串→固定大小字节数组	不能	不能（可以直接赋值）
字符串→动态大小字节数组	可以	可以

表示不能直接转换 红色箭头表示三者间的间接转换方向





Web3.js是一个JavaScript库

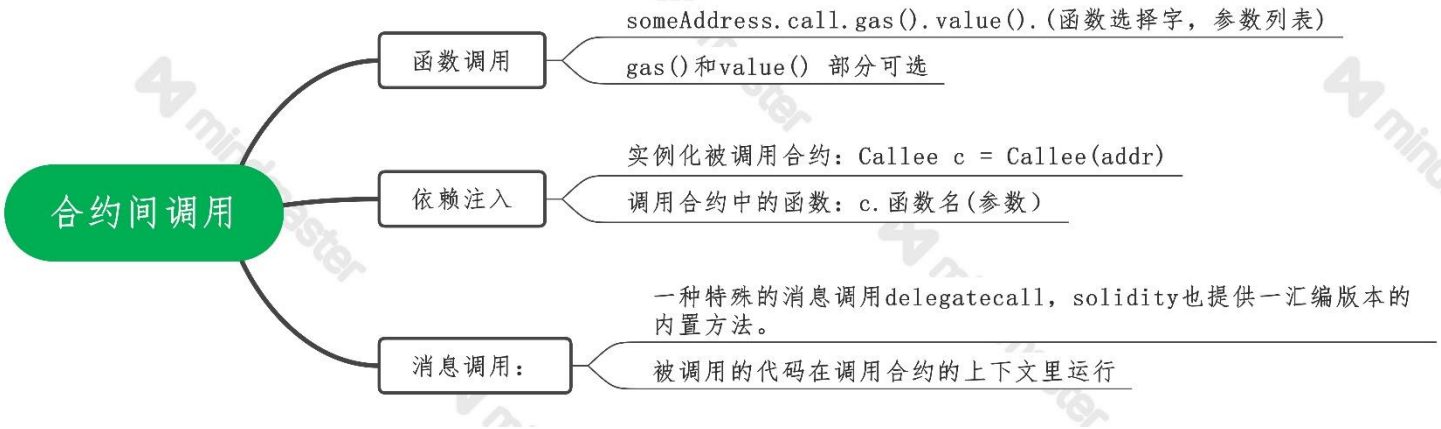
- **Web3.eth** 模块包用来与以太坊区块链和以太坊智能合约进行交互。
  - web3.eth.subscribe 方法让你可以订阅区块链中的指定事件
  - web3.eth.Contract 对象让你可以轻松地与以太坊区块链上的智能合约进行交互
  - web3.eth.accounts 包中包含用于生成以太坊账户和用来签名交易与数据的一系列函数。
  - web3-eth-personal 包让你可以同以太坊节点上的账户进行交互。
  - web3.eth.ens 相关函数让你可以与 ENS 进行交互
  - web3.eth.Iban 相关函数让我们可以将以太坊地址和 IBAN/BBAN 地址之间相互转换。
  - web3.eth.abi 函数用来解码及编码为 ABI (Application Binary Interface应用程序二进制接口) 以用于 EVM（以太坊虚拟机）进行函数调用。
- **Web3.bzz**模块与去中心化的文件存储**swarm**交互
- **Web3.net**模块让你可以与以太坊节点交互来获取网络属性
- **Web3.ssh**模块允许您交互群集去中心化的文件存储
- **Web3.utils**模块为以太坊 **DApp** 和其它的 **web3.js** 包提供了工具性函数

web3.js 连接以太坊的三种方式

- **HTTP**方式：最常用的方式，使用流程是，创建目录-->初始化一个工程-->安装依赖-->创建文件-->输入代码-->执行代码
- **IPC**方式：通过进程间通信的方式
- **WebSocket**方式：和**HTTP**方式相同

可升级的合约设计

- 代理合约：通过**delegatecall**指令来调用目标合约里的函数，而目标合约是可以升级的。
- 分离逻辑和数据：将合约的数据和相关的函数放在数据合约中，将商业逻辑的实现代码放在逻辑合约里。
- 通过键值对来分离数据和逻辑：所有数据都经过了抽象化，从而可以通过键值对来访问。
- 部分升级：设计部分可升级的合约，通行做法。


合约间调用
函数调用
someAddress.call.gas().value().(函数选择字，参数列表)
gas()和value() 部分可选
依赖注入
实例化被调用合约：Callee c = Callee(addr)
调用合约中的函数：c.函数名(参数)
消息调用：
一种特殊的消息调用delegatecall，solidity也提供一汇编版本的内置方法。
被调用的代码在调用合约的上下文里运行