**对Solidity 存储、内存和calldata的研究-王泽源**

当你调用一个智能合约时，EVM会运行并执行其字节码中的一组指令（=操作码）。其中一些操作码指示EVM从/向不同的位置读写数据。EVM需要这些多个数据位置来正确完成其工作。那么这些数据位置是怎么来确定并且位于哪里的呢？

EVM有五个主要的数据位置：

**存储（Storage）**

在以太坊中，每个特定地址的智能合约都有自己的 "存储"，由一个键值存储组成，将256位映射到256位。存储中的数据在函数调用和交易之间持续存在。

存储是所有合约状态变量所在的地方。每个合约都有自己的存储。存储中的变量在函数调用之间持续存在。然而，存储空间的使用是相当昂贵的。

由于存储指的是合约存储，它指的是永久存储在区块链上的数据。

**内存（Memory）**

EVM 内存是用来保存临时值的，在外部函数调用之间被擦除。然而，它的使用成本比较低。

在EVM中，内存是易失性的，是特定合约（环境）的上下文。这意味着，当执行环境从一个合约变为另一个合约时，“白板/写字板”被清除。在每一个新的消息调用中，都会获得一个新的被清除的内存实例。

因此，内存变量是暂时的。它们在对其他合约的外部函数调用之间被擦除。

**调用数据（Calldata）**

calldata相当于从船上或卡车上取出的一个集装箱。这些集装箱包含送到工厂进行加工的材料。Calldata是只读的。

calldata是交易的数据或外部函数调用的参数所在的位置。它是一个只读的数据位置。你不能写到它。

Calldata的行为主要类似于内存，是一个可由字节编址的空间。你必须为你想读取的字节数指定一个准确的字节偏移。

**堆栈（Stack）**

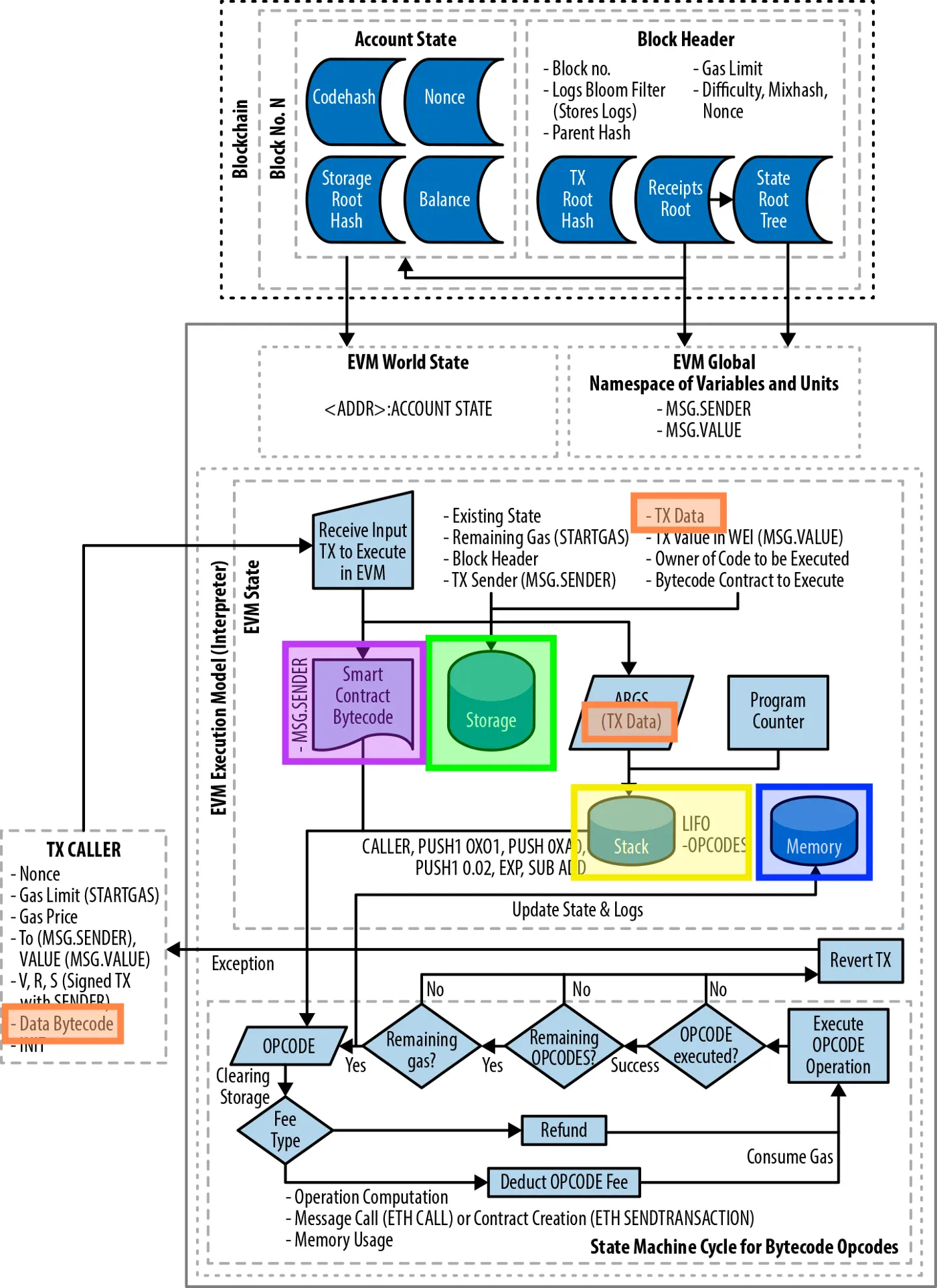
堆栈是用来存放小型局部变量的。它的使用几乎是免费的（用Gas很低），但大小有限，能容纳的项目数量也有限。

堆栈是大多数在函数内部创建的局部变量所在的地方。它是EVM的一个重要部分。

**代码（Code）**

代码指的是合约的字节码。你只能从合约字节码中读取，而不能写到它。通常是你在Solidity中定义为 constant的变量。大多数的EVM操作码从堆栈中消耗它们的参数。

字节码包含了很多关于合约的信息和逻辑，包括调度器，以及合约元数据。



(如图，图片源自网络，原图链接https://github.com/ethereumbook/ethereumbook/blob/develop/13evm.asciidoc)

**变量的存储位置**

* 定义为 constant的变量 = 合约代码(=bytecode)。

这些变量是不可改变的，一旦合约被部署就不能改变。它们是只读的，可以被内联使用。

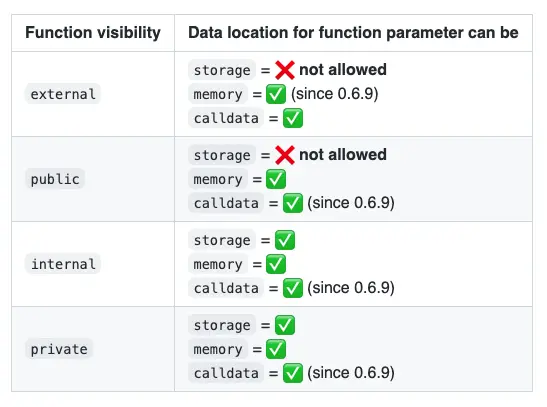
* 状态变量（在函数之外声明） = 默认情况下在存储（storage）中。

这些被称为状态变量，因为它们是合约状态的一部分，反过来也是区块链全局状态的一部分（=以太坊中所有智能合约的状态）。这些变量被永久地写入区块链。

* 本地变量（在函数体内声明）= 在堆栈中。值类型的变量（例如，uint256, bytes8 , address）驻留在堆栈中。
* 对于数组(固定或动态大小的数组, 如uint256[]), bytes, string, 结构和映射, 你必须明确提供存储值的数据区域. 这可以是storage，memory或calldata

需要注意的是，函数中只能有三个地方可以指定变量的数据位置

* 对于参数（=函数定义）
* 对于函数内部的局部变量（=函数主体）
* 返回值总是在内存中（=函数定义）。



（如图，为函数参数不同类型的可存储位置，图源网络，来自https://github.com/CJ42/All-About-Solidity/blob/data-locations/articles/Data-Locations.md#data-location-rules-for-function-parameters）

当storage被用作一个函数参数的引用时，它是一个指向合约存储的指针。

对于memory和calldata也是如此。这样的关键字指向EVM内存中的某个位置或从交易中进来的输入数据（=calldata）的指针。

**在函数体内的规则**

在函数内部，无论函数的可见性如何，都可以指定所有三个数据位置:

然而，引用类型之间的赋值是受特定规则约束的。(这里是变得复杂和 "略微扭曲舌头的地方！"）。

* storage引用：总是可以直接从合约存储中（=状态变量）或通过另一个 "存储" 引用 给一些变量赋值，但它们不能赋值一个 "内存 "或 "calldata "引用。
* memory引用：可以被分赋值任何东西（直接的状态变量，或storage、memory或calldata引用）。它总是创建一个副本。
* calldata引用：总是可以直接从calldata（= tx/message调用的输入），或通过另一个calldata引用赋值，但它们不能从storage或memory引用赋值。