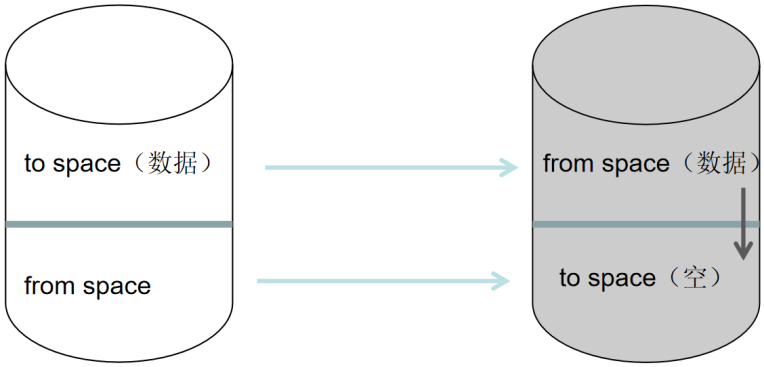
IC 存储

Canister

- 8G & 4G & 2G & GC：
  - 8G：
    - Canister的总体内存包含wasm run time 和 stable memory，对一个Canister而言， 两个都为4G
      - So a canister can under normal conditions store 8GiB of storage. However, when a canister is upgraded [1], its wasm heap is wiped so for all practical purposes, it only really has access to 4GiB of storage in the stable memory. In the past we demonstrated a proof of concept of BigMap which is a solution to enable an application to scale its storage by sharding its data across multiple canisters.
    - 由于wasmtime是32bit的指针地址域，因此是4G，而stable memory需要和wasm time适配， 所以也是4G，现在提高的是stable内存， 可以简单的理解为此次升级是在Canister和stable memory内存之间加了一个中间件， 让32bit的wasmtime可以使用8G的stable内存（实际上是扩容到了16T， 即64bit， 但是官方此次仅提升到8G）
      - wasmtime [8] does not support the wasm64 specification and hence has 32 bit addressing.
      - A stable memory [7] which is also currently constrained to 4 GiB because it too only has 32 bit addressing.
    - 4G 的stable内存（非WebAssembly）
      - Canisters have the ability to store and retrieve data from a secondary memory. The purpose of this *stable memory* is to provide space to store data beyond upgrades. The interface mirrors roughly the memory-related instructions of WebAssembly, and tries to be forward compatible with exposing this feature as an additional memory.
    - 4G 的堆内存（GC限制）
  - 4G （这里说的是Canister堆内存）： IC在进行技术选型时， WASM尚未开发出64 bit 的虚拟机， 因此当时采用了32 bit的WASM虚拟机， 造成现在单体Canister的内存为4G.
    - 2G： wasm 单体的内存空间为4G， 但是由于Motoko所选用的GC（Garbage Collector 垃圾回收）算法为Copying 算法[1]， 造成由Motoko写的Canister只能使用2G内存空间

      Note that Motoko currently uses a 2-space copying collector so the heap will be at most half the memory, and the heap is thus bounded by 2GB. We have another, compacting, GC in the pipeline that will allow access to much more of the 4GB for the heap.

    - [1] Coping Collector Algorithm （也称为Minor GC）：
      - 简述该算法在WASM Canister中的应用：
        - 4G 的Canister被划分为两个区域： from space & to space， 分别占用2G；数据写入时， 所有的数据都在to space中。
        - 在进行dfx canister install xxx(default : --all) -- mode upgrade的时候， 会进行GC，即：将活动对象（正在使用的对象，可以是引用也可以是元数据）所在空间换为from space， 原from space换为to space， 然后对from space（数据所在的内存空间）进行遍历， 将所有的活动对象迁移到新的to space中， 剩下的就是非活动对象， 也就是内存垃圾， Canister会将这部分数据清理掉。
        -

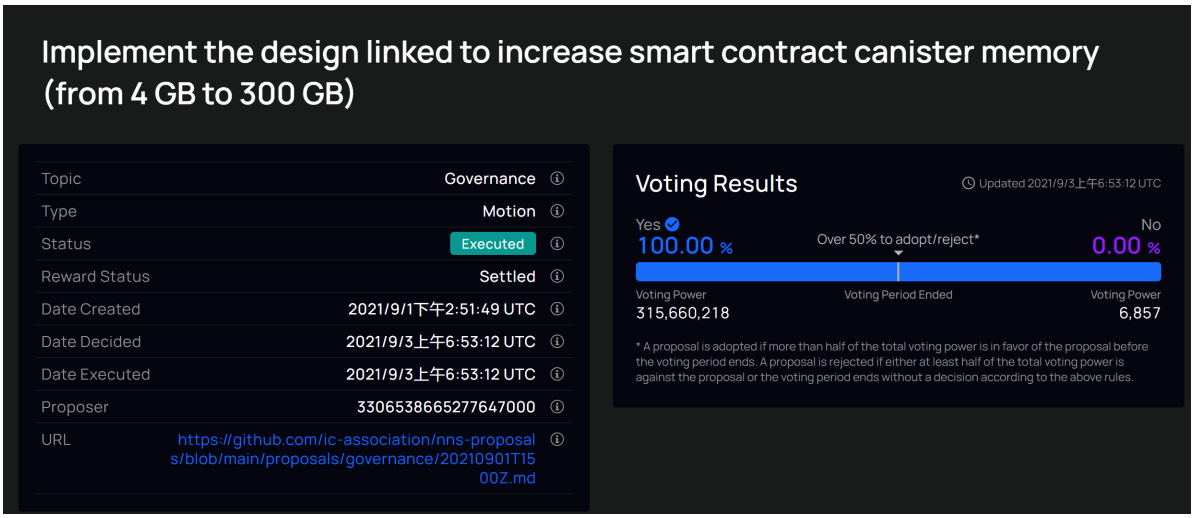


- 优势
  - 内存使用率很高，写数据时是对堆进行连续写入， 所以利用率更高 。适用于快速增删数据的场景。 （现在Java的JVM等新一代VM都一定程度上采用了Minor GC）
- 缺陷：
  - 浪费了一半的堆内存空间，可能会使用Compacting GC算法， 这个算法不会占用一半的内存来进行GC，可以使用全部的WASM内存
- 对4G stable内存进行读写仅发生在upgrade过程。
- Before an upgrade, the stable variables are serialized from wasm memory to stable memory. After an upgrade, the stable variables are deserialized from stable memory into wasm memory. Motoko doesn't read or write to stable memory during execution outside of upgrades.
  - 劣势： 当堆内存在升级时有大量要写入stable内存的数据时， 可能会将cycle消耗完， 导致无法升级。

- 其他GC的算法： https://blog.csdn.net/stinge/article/details/84022369?ops_request_misc=&request_id=&biz_id=102&utm_term=2-space%20copying%20collector&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-84022369.nonecase&spm=1018.2226.3001.4187

存储提案

Forum讨论： https://forum.dfinity.org/t/increased-canister-storage/6148/67



提案内容： 提供一个系统API， 使每个Canister都可以使用子网所有的内存空间， 所有的副本都有自己的数据状态，即并非所有副本共用一个数据状态（这很好理解， 如果这样， 那就会出现并发问题，需要锁或者通道机制来避免冲突， 但事实上子网内使用的是共识来保证数据一致性）。

新增的System API （面向Canister与）：

github： https://gist.github.com/ulan/8cc37022c72fe20dc1d57fdfd0aaf1fd

- `ic0.stable64_write: (offset: i64, src: i64, size: i64) -> ()`
  - 这个API接口是为了WASM 64进行提前准备的
- `ic0.stable64_read: (dst: i64, offset: i64, size: i64) -> ()`
- `ic0.stable64_size: () -> (page_count: i64)`
- `ic0.stable64_grow: (additional_pages: i64) -> (old_page_count: i64)`

底层原理：在执行层加中间件，将wasm指针地址赋值空间从32位赋值空间增长到64位赋值空间。

提案现状：

- 提案于9.3日通过NNS审核， 接下来就是更新代码的事情。提案只是一个征求社区看法的提案。（我猜是让社区能感受到自己在参与IC的治理。。）
- 9.6日将提交更改IC代码的提案

带来影响：

- 对Canister而言， Canister可以使用的内存从4G -> 8G（stable内存而非堆内存）
  - 8G是当前提案的结果， 以后根据社区反馈会提升这个数字
  - DFINITY engineer here. In the initial roll out we will limit the stable memory size to 8GB. We will increase it in future release after gathering feedback.
  - **如果stable内存高于4G， 那么使用了升级内存API的Canister和没有使用此API的Canister交互将会出现问题，当前的方案是发生上述情况时， 会发生trap（应该是编译时）**

社区现状：

未来解决方案：

- 沿用现在的解决方案： 子网内存共享， 3T内存分出300G给stable内存存储（尚不清楚*300G*这个数字是怎么来的， *8G*是怎么来的）
  - 缺陷： 这种情况下，存储内存最大限制就是一个子网内的stable内存的限制， 这显然是不能长久的。
- BigMap： 通过多个Canister分片存储数据，当前状况是没有达到生产级别的要求， 还没有放出。
- WASM 64： 单个Canister升级为64bit的地址空间， 最大内存为2^64 Byte（16 T）
  - 数据迁移所牵扯的数据量会很大，会出现新的问题。

建议：

- 现阶段还是一个实验性的API， 建议不要使用。等待成熟以后再使用。
- based on the feedback we get, it turns out that the API needs some adjustments. And if developers have deployed canisters using the API, then making adjustments will be difficult. So we are planning on
- canisters yet. And if based feedback, some adjustments to the API are needed, then the existing canisters using it will be impacted. We will of course discuss and announce impending changes so that

Motoko， Rust 获取Canister存储状态

Motoko

RTS : Run Time System 运行时系统， 包含GC， 序列化（通信传输用）， low-level 库（底层分配内存等）的调用

rts_memory_size: () -> Nat： 当前WASM的内存， 不是使用了多少的内存， 而是已经分配了的堆内存

rts_heap_size: () -> Nat： 当前实际堆内存大小

rts_max_live_size : () -> Nat： 从上次GC到现在堆最大的大小

rts_total_allocation: () -> Nat;

rts_version : **()** -> Text

rts_reclaimed : **()** -> Nat;

rts_max_live_size : **()** -> Nat;

Rust

使用IC "aaaaa-aa" Actor可以访问IC.status， 也可以返回上面说到的内存数据

关于存储的其他Tips：

1. 作用于存储的数组最好使用Blob而非[Nat8]， 因为初始化数组时， 由于泛型的需要， 所有通过[X]或者Array.init初始化的数组内存布局是相同的。
2. 当前比较成熟的解决方案：存储的数据直接放入stable内存中， 堆内存用来放置检索数据。
    1. Note that even already now, the Internet Identity canister does precisely that: Keep all relevant data in stable memory, and use the heap only as scratch space. Makes upgrades simpler.