What are the differences among calldata, memory and storage in Solidity?

一張含有 文字, 螢幕擷取畫面, 字型, 筆跡 的圖片

AI 產生的內容可能不正確。

What are the differences between POW and POS in solidity?

一張含有 文字, 螢幕擷取畫面 的圖片

AI 產生的內容可能不正確。

calculate hash in blockchain

一張含有 文字, 字型, 螢幕擷取畫面, 文件 的圖片

AI 產生的內容可能不正確。

Merkel

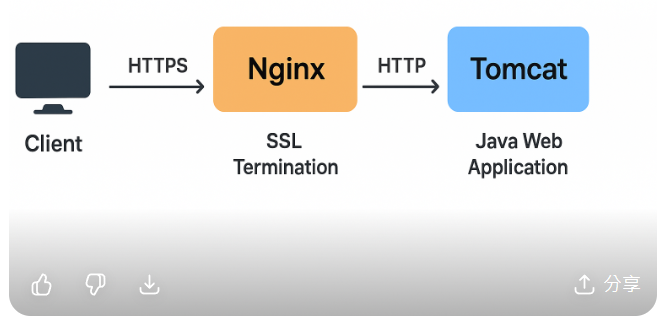
一張含有 文字, 圖表, 方案, 圖解 的圖片

AI 產生的內容可能不正確。

一張含有 文字, 字型, 白色, 收據 的圖片

AI 產生的內容可能不正確。

應用程式 ---> Docker (打包成容器) ---> Kubernetes (管理容器群)



**以太坊 Solidity 的存儲佈局 (storage layout)** 裡，EVM 的 storage 是以 **slot** 為單位的，每個 slot 固定 **32 bytes (256 bits)**

 如果它們的大小總和 **不超過 32 bytes**，編譯器會嘗試把它們 **打包 (pack)** 到同一個 slot 裡。

 例如：

contract C {

uint128 a; // 16 bytes

uint128 b; // 16 bytes

// a 和 b 會放在同一個 32-byte slot 裡

一張含有 文字, 圖表, 行, 字型 的圖片

AI 產生的內容可能不正確。

一張含有 文字, 圖表, 方案, 螢幕擷取畫面 的圖片

AI 產生的內容可能不正確。

Go channel VS (RabbitMQ or Kafka)

**Summary:**

* **Go Channels** are great for **concurrent** programming within a single application. They provide simple **message passing** between goroutines and do not persist messages.
* **Message Queues** like **Kafka** and **RabbitMQ** are designed for **distributed systems** and provide **reliable delivery**, **persistence**, **scalability**, and **fault tolerance** across multiple machines.
* Go channels and MQs have similar goals (communication between producers and consumers), but **Go channels are more lightweight** and fit well within **single-process** concurrency, while **MQs** are used for **distributed message passing**.

Do not communicate by sharing memory. Instead, share memory by communicating.

一張含有 文字, 螢幕擷取畫面, 字型, 文件 的圖片

AI 產生的內容可能不正確。

**Assembly Block:**  
The assembly block is a low-level way to interact directly with the Ethereum Virtual Machine (EVM) and is used here for efficiency and to avoid the overhead of Solidity's higher-level constructs.

Go 语言作为一个强类型语言，不同类型之间必须要显式的转换（而且必须有相同的基础类型）。但是，Go 语言中 interface 是一个例外：非接口类型到接口类型，或者是接口类型之间的转换都是隐式的。这是为了支持鸭子类型，当然会牺牲一定的安全性。

In Go, "鸭子类型" (duck typing) refers to the ability of a type to implicitly satisfy an interface if it has the required methods, without the need for explicit declarations. This allows Go to have a flexible and powerful type system while maintaining static typing.

func panic(interface{})

func recover() interface{}

**Conclusion:**

* **Transparent Proxy** is good for those who prefer explicit control over the proxy and implementation separation.
* **UUPS Proxy** is more gas-efficient and self-contained, with a focus on reducing complexity and improving upgrade efficiency, making it a better option for modern upgradable smart contracts.

If you're working with OpenZeppelin's contracts, both proxy types are available, but **UUPS** is gaining popularity due to its gas efficiency.