如何选用 Solidity 数据结构

什么情况下应该使用 uint 或 int?

当需要存储整数，如数量或索引时，应使用 uint（无符号整数）或 int（有符号整数）。uint 适用于不允许负值的场景，如总供应量。

```
uint256 public totalSupply; // 代币供应量
int256 public balance;
```

如何选择存储以太坊地址使用的数据结构?

使用 address 数据类型来存储以太坊地址，适用于用户钱包地址或合约地址。

```
address public owner;
```

在何时使用 string 与 bytes?

当存储可变长度的文本数据时使用 string；当处理不需要字符编码的原始字节数据时使用 bytes。

```
string public name;
bytes32 public hash;
```

数组在 Solidity 中的应用场景是什么?

数组用于存储相同类型的元素列表，适用于需要存储多个值的情况，如数字列表或状态记录。

```
uint256[] public numbers;
```

为何以及如何使用 mapping?

mapping 用于创建键值对映射，常用于存储关联数据，如用户的余额。它在数据查找方面更高效。

```
mapping(address => uint256) public balances;
```

struct 的用途及实例?

struct 允许创建自定义的数据结构，包含多个不同类型的字段。适用于复杂数据组合的场景。

```
struct Person {
    string name;
    uint256 age;
}
```

何时使用 enum 以及其好处是什么?

enum 用于定义一组命名常量，限制变量的取值范围，适用于有限选项的情况。

```
enum Status { Pending, Approved, Rejected }
```

在设计合约时如何考虑存储和 Gas 成本?

应选择高效的数据结构以减少存储和执行成本。例如，mapping 通常比数组更节省 Gas 成本，特别是在大规模数据查找时。

优化 Solidity 合约的存储和 Gas 成本是开发中至关重要的一部分。通过减少存储写入、合理选择数据结构、优化逻辑和循环操作，可以显著降低合约的执行成本，从而提高合约的可用性和经济性。在开发过程中，始终保持对 Gas 成本的敏感性，并使用工具和测试来确保合约的高效执行。

如何根据数据访问模式选择数据结构?

根据合约的数据访问频率和类型选择数据结构。频繁变动的数据可能更适合使用 mapping，而静态数据或顺序访问的数据适合使用数组。

在复杂合约中选择数据结构的考虑因素有哪些?

需要评估合约的功能需求，选择可以支持这些功能的数据结构。复杂合约可能需要结合使用多种数据结构，如结合使用 struct 和 mapping。

如何决定使用固定长度的数组还是动态数组?

如果事先知道数组的最大长度，并且这个长度不会变化，使用固定长度数组可以节省 Gas 成本。如果数组长度会动态变化，应选择动态数组。

```
uint256[10] public fixedNumbers;
uint256[] public dynamicNumbers;
```

在 Solidity 中使用 mapping 和 array 的主要区别及使用场景是什么?

mapping 用于快速查找和更新键值对，适合用于账户余额等场景；而 array 适用于元素顺序重要或需要迭代处理的场景。

```
mapping(address => uint256) public userBalances;
address[] public userList;
```

如何利用 struct 在 Solidity 中模拟传统的数据库表?

可以使用 struct 来定义表的列，然后使用 mapping 或数组来存储 struct 实例，模拟行的概念。

```
struct Employee {
    uint256 id;
    string name;
    uint256 departmentId;
}
mapping(uint256 => Employee) public employees;
```

Solidity 中 enum 如何帮助降低错误的发生?

enum 限制变量的取值范围，减少非法值的输入，提高代码的可维护性和错误预防。

```
enum State { Active, Inactive, Suspended }
```

为何 bytes 类型有时比 string 更优?

当处理不需要字符处理功能的纯二进制数据时，bytes 类型更节省空间和 Gas 成本，因为它不涉及 UTF-8 编码处理。

```
bytes public rawData;
```

如何选择在 Solidity 中存储时间的最佳数据结构?

使用 uint256 来存储时间戳是最常见的方法，因为它可以直接与 Ethereum 虚拟机的时间函数兼容。

```
uint256 public lastUpdated;
```

在 Solidity 合约中，何时应考虑将数据封装在 struct 内部?

当数据项逻辑上属于同一实体或需要一起处理时，应将它们封装在一个 struct 内部以增加可读性和可维护性。

```
struct Order {
    uint256 orderId;
    uint256 quantity;
    uint256 price;
    address purchaser;
}
```

mapping 类型是否支持迭代？如果不支持，如何解决?

mapping 本身不支持迭代。如果需要迭代，可以维护一个单独的数组来存储所有键，然后通过这些键来访问 mapping。

```
mapping(address => uint256) public accounts;
address[] public accountList;
```

在设计一个包含多种资产类型的钱包合约时，应使用哪种数据结构?

可以使用 mapping 将资产类型（如 ERC20 代币地址）映射到另一个 mapping，后者将用户地址映射到余额。

```
mapping(address => mapping(address => uint256)) public balances;
```

使用 enum 定义状态时，应如何处理状态的转换逻辑?

定义状态转换的函数中应包含状态验证逻辑，确保合约状态按预定流程转换。

```
enum Stage { Init, Running, Ended }
Stage public stage = Stage.Init;
function nextStage() public {
    if (stage == Stage.Init) {
        stage = Stage.Running;
    } else if (stage == Stage.Running) {
        stage = Stage.Ended;
    }
}
```