Topics in Solidity

Hari

ETHonline. September 2, 2022

Question to the audience

What is the most requested Solidity feature since the last 2 years?

Encoding a external call: abi.encodeWithSignature

```
interface MinimalERC20
{
    function transfer(address to, uint value) external;
}

function transferCalldata(address to, uint value)
    pure
    returns (bytes memory)
{
    return abi.encodeWithSignature(
        "transfer(address,uint256)",
        to,
        value
    );
}
```

abi.encodeWithSignature: Typo safe?

```
function transferCalldata(address to, uint value)
    pure
    returns (bytes memory)
{
    // BUGGY: typo
    return abi.encodeWithSignature(
        "transfer(address, uint)",
        to,
        value
    );
}
```

Encoding an external call: `abi.encodeWithSelector`?

```
interface MinimalERC20 {
    function transfer(address to, uint value) external;
}

function transferCalldata(address to, uint value)
    pure
    returns (bytes memory)
{
    return abi.encodeWithSelector(
        MinimalERC20.transfer.selector,
        to,
        value
    );
}
```

abi.encodeWithSelector: Type safe?

```
interface MinimalERC20 {
    function transfer(address to, uint value) external;
}

// BUGGY
function transferCalldata1(uint to, uint value)
    pure
    returns (bytes memory)
{
    return abi.encodeWithSelector(
        MinimalERC20.transfer.selector,
        to,
        value
    );
}
```

Encoding an external call: `abi.encodeCall`?

```
interface MinimalERC20 {
    function transfer(address to, uint value) external;
}

function transferCalldata(address to, uint value)
    pure
    returns (bytes memory)
{
    return abi.encodeCall(
        MinimalERC20.transfer,
        (to, value)
    );
}
```

Why `abi.encodeCall`?

- Type safe and typo safe!
- Avoid footguns.
- Available from `0.8.11`.
- Already in use: SafeDeployer.sol#L43.

abi.encodeError?

Dapphub / Foundry style testing:

```
error InvalidConduit(bytes32 conduitKey, address conduit);
vm.expectRevert(
    abi.encodeWithSignature(
        "InvalidConduit(bytes32,address)",
        conduitKeyAlice,
        mockConduit
    )
);
// perform the external call that will revert
// with the above data.
```

Source: Seaport.

abi.encodeError?

In the future:

```
error InvalidConduit(bytes32 conduitKey, address conduit);

vm.expectRevert(
    abi.encodeError(InvalidConduit,(key, conduit))
);
```

See issue: #13339.

What is the most requested solidity feature?

- `https://blog.soliditylang.org/2022/02/07/solidity-developer-survey-2021-results/`
- `https://blog.soliditylang.org/2021/01/26/solidity-developer-survey-2020-results/`

Fixed point arithmetic!

Fixed point arithmetic 101

- Pick a base B, say `B = 10**18`.
- Represent the number $\frac{p}{B}$ by storing it as p.
- For example, `1 wei = (1 / 10**18) ether`. Store it as 1.
  - **`1 ether = 10**18`** when stored.
- Addition and subtraction: no change, regular addition and subtraction!
- Multiplication: if p and q are the stored values, $p \times q \equiv \frac{p \times q}{B}$.
- Division: if p and q are stored values, $\frac{p}{q} \equiv \frac{p \times B}{q}$

History of Fixed point arithmetic:

- Solidity has the type `ufixed` and `fixed` for a long time, and in the ABI.
- But the types are unimplemented and therefore useless.
- You would get a compiler error if you use this.

How should fixed point arithmetic work?

- Decimal or Binary?
- How many decimal points?
- Shadow overflow: when $\frac{p \times q}{B}$ is in 256-bits, but $p \times q$ is not.
- Gas efficiency.

Wadmath

- Introduced by Maker
- Decimal
- Usually 18 decimals.
- Reverts in case of shadow overflow.
- Gas efficient.

A modern implementation: Solmate Wadmath

Uniswap V2

- UQ112x112.sol
- Binary
- Uses 112 bits
- Uses shifts (3 gas) instead of division (5 gas).
- Does not care about shadow overflow.
- Gas efficient.

Uniswap V3 Fixed point

- Uniswap V3 FullMath
- Binary fixed points
- Still need to produce the correct result in case of a shadow overflow.
- An implementation by Remco.
- More checks needed and complex arithmetic operations–more gas.

Muldiv

- Key computation: $\frac{p \times q}{B}$.
- Muldiv: `https://eips.ethereum.org/EIPS/eip-5000`
- `muldiv(p, q, B)` performs $\frac{p \times q}{B}$ in 512-bit arithmetic, truncated to 256 bit.

Future Plans

- User defined value types for fixed points.
- There will be no change the ABI. Better backwards compatability.

```
interface RealMinimalERC20 {
    function transfer(address a, uint value) external;
}

type Fixed18 is uint256;

interface MinimalERC20 {
    function transfer(address a, Fixed18 value) external;
}
interface BreakingERC20 {
    // `ufixed` is well defined type in ABI
    // Breaks the ERC20 interface
    function transfer(address a, ufixed value) external;
}
```

Future Plans

- `muldiv` allows a gas-efficient and generic implementation.
- Operators for fixed point types.
- Literal support: `Fixed18 x = 0.0002;`

```
uint constant BASE = 10**18;

type Fixed18 is uint256;
using {checked_mul as *} for Fixed18;

function checked_mul(Fixed18 a, Fixed18 b) returns (Fixed18 c) {
    uint _a = Fixed18.unwrap(a);
    uint _b = Fixed18.unwrap(b);
    uint _c;
    assembly {
        // 0 is an exceptional case in muldiv.
        // Detect truncation in a * b / BASE
        if iszero(lt(muldiv(a, b, 0), BASE)) {
            revert(0, 0)
        }
        _c := muldiv(a, b, BASE)
    }
    c = Fixed18.wrap(_c);
}
```

What can you do?

We want your feedback:

- `muldiv`: EIP-5000.
- The exceptional behaviour of `muldiv(a, b, 0)`.
- The syntax for user defined operators.
- Other use cases of `muldiv`.
- More notes in `https://notes.ethereum.org/@solidity/ryNbZ2xEq`

Slides

https://hrkrshnn.com/t/ethglobal2022.pdf