Solidity override vs. virtual functions

Kseniya Lifanova Follow

Jul 15, 2020 · 2 min read


```
/**
 * @title Smart Contract
 */
contract SmartContract is ERC20 {

    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }
```

In our most recent smart contract project, we used Solidity 0.6 where I learned about `override` vs `virtual`. As always, you can read the docs, but I wanted to write this post for those of us who prefer to read quick tutorials on Medium.

Solidity is an object-oriented programming language that supports multiple inheritances. You can inherit from a base contract and then override a function in that base contract. Before Solidity 0.6, there was no way of knowing what functions should be overridden. Now, you can explicitly label a function as `virtual` or `override`.

Function Overriding

A function that allows an inheriting contract to override its behavior will be marked at `virtual`. The function that overrides that base function should be marked as `override`. Let's take a look at an example from our favorite library, OpenZeppelin.

If you look at the ERC20.sol contract in the latest version of the `@openzeppelin/contracts` library you will see that the `transfer` function contains both the `virtual` and `override` keywords.

```
function transfer(address recipient, uint256 amount) public virtual
override returns (bool) {
```

```
        _transfer(_msgSender(), recipient, amount);
        return true;

    }
```

This means if you inherit the ERC20.sol contract into your project, you can write your own `transfer` function and mark it as `override` since it's overriding the base `transfer` function. If you forget to add the `override` the compiler will yell at you:

```
TypeError: Overriding function is missing "override" specifier.
```

The reason the OpenZeppelin `transfer` function includes the `override` keyword is because it's inheriting the IERC20 interface which also has a `transfer` function. *All functions in interface contracts are automatically considered* `virtual`.

If your contract is inheriting the *same* function from multiple base contracts (that are unrelated), you must explicitly state which contracts:

`override(Base1, Base2)`

```
pragma solidity >=0.5.0 <0.7.0;

contract Base1
{
    function foo() virtual public {}
}

contract Base2
{
    function foo() virtual public {}
}

contract Inherited is Base1, Base2
{
    // Derives from multiple bases defining foo(), so we must
explicitly
    // override it
    function foo() public override(Base1, Base2) {}
}
```

Conclusion

That's it, folks! This update makes it clear what functions are meant to be overridden vs what functions should be left as is. This is useful when working with libraries such as OpenZeppelin. This is also useful if you are building a contract that is meant to be used by other developers. You can be explicit in your intention for a function. If you do not want a function to be overridden, leave off the `virtual` marker.