Truffle Pet Shop Tutorial
Introductory Contracts
Inter-Contract Execution gist

CALL vs CALLCODE vs DELEGATECALL in Solidity from ~2:20

Difference between call, callcode and delegatecall - stackexchange

Interactions between contracts

Inheritance in Solidity - Jules Dourlens
Inheritance - Solidity Docs

****

Solidity development has some drawbacks.

First of all, just by the nature of the Ethereum system, every node has to execute and verify every transaction. This can be expensive and users of the system pay for this with gas.

Also, there are few standard libraries for Solidity, managing and manipulating arrays and strings can be painful.

Blockchains are isolated environments. It is relatively difficult to get verified data into the blockchain, there are not APIs like we are used to in traditional web development. You have to use services called oracles to get external data into the blockchain.

Also, the immutability of deployed contracts makes upgrading contracts more difficult than upgrades in other systems. We can design systems in such a way to enable upgradability, we just have to pay extra attention to this.

How can we address some of these drawbacks? Libraries can help.

What are libraries?

Libraries are contracts that do not have storage, they cannot hold ether and they cannot inherit or be inherited by other contracts. Libraries can be seen as implicit base contracts of the contracts that use them.

They exist for the purpose of code reuse. So contracts can call library functions without having to implement or deploy the functions for itself. This enables developers to use code that has already been audited and battle-tested in the wild.

To create a library contract we simply use the library keyword in place of the contract keyword. You will notice in this example that no storage variables are defined. Remember that libraries do not have a state, although it is possible for libraries to modify the calling contracts' state. We will get to that in a moment.

Once you have defined a library, you can import it with an import statement at the top of the solidity file. Now you can call the functions defined in the library as methods on the library name. You can also use the 'using for' syntax shown, which allows you to call the library function as a method on the first parameter defined in the library's function.

Libraries use the DELEGATECALL opcode of the EVM which preserves the context of the contract from which the library method is being called. This allows libraries to modify the state of calling contracts. The library can define the struct data type, but it will not be saved in storage until it is actually implemented in a contract. This implemented struct can be passed to the library and modified, and the modifications will persist in the contracts' storage. This is a seldom-used feature of libraries in solidity, but it is good to be aware of.

Libraries offer some other benefits in terms of contract system design.

In the contract factory design pattern a contract might be deploying several additional contracts. Reducing the size of the factory output contracts can save a lot of gas in deployment costs over the lifetime of the contract.

Moving functions that are referenced by the factory output contract to a library can reduce the size of the factory output contract.

Calling library functions from the factory output contracts is a bit more expensive than calling internal functions, so there is a tradeoff to consider.

If the factory output contract functions are frequently called, it may be better to pay the higher deployment cost to get cheaper function calls. You will have to determine which is best for your use case.

To connect to a library, you need the library contract as well as the address of the deployed instance.

Oftentimes you will deploy the library yourself, so the address management will be handled by truffle.

Otherwise, the address of the deployed library has to be added to the bytecode by a linker.

You can do this manually in your truffle project by adding the library contract to your contracts directory and compiling to get the artifact.

You can modify the truffle artifact to specify the correct network and address that the library is deployed at.

Library and contract package management can be simplified by using EthPM. EthPM is essentially npm for Ethereum contracts. Truffle has support for EthPM so you can install a package with truffle install and the package name. You can go to ethpm.com to browse the available packages. You should add an ethpm.json file to your project to track which packages and versions you are using.

Importing an ethpm package into a solidity file simply requires the file path reference at the top of your solidity file.

Truffle comes with an option for migrating contracts that enables you to only deploy contracts if there isn't already a deployed instance. You should reference the truffle docs for specific implementation details.

Useful links:

Library Driven Development in Solidity - Jorge Izquierdo

Libraries - Solidity Docs

What are the steps to compile and deploy a library in solidity? - stackexchange

Package Management with EthPM in Truffle

One Reason to Start Using Solidity Libraries - Ethereum Alarm Clock

Towards better Ethereum voting protocols

Token-weighted Voting Implementation Part 1

Token-weighted Voting Implementation Part 2

Other uses of Commit Reveal

Please refer to the following document for the Simple Bank exercise.

1. Simple Bank Exercise

2. Simple Bank Exercise Directory

Please refer to the following document for the Supply Chain exercise.

1. Supply Chain Exercise

2. Supply Chain Exercise Directory