



Smart Contract Programming

----beginner----

Victor Tran

ETHEREUM HANOI MEETUP

Smart contract



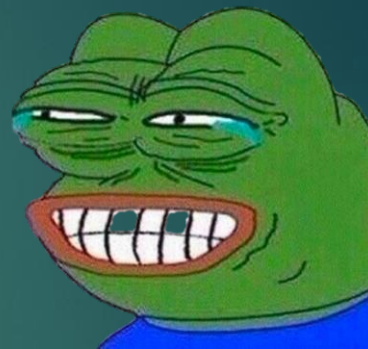
- ▶ Oh traditional contract is stupid?
 - ▶ Easy to read, hard to understand
 - ▶ Only lawyers can read it
 - ▶ Take courses to execute the contract
 - ▶ “Security holes” are features
- ▶ Smart contract is smart? No it is even more stupid!
 - ▶ Hard to read, even harder to understand
 - ▶ Only engineers can read it
 - ▶ Execute on Blockchain, money is publicly spent
 - ▶ Security holes are BUGS



Why is everyone crazy about it?



- ▶ It is new...very new and young
- ▶ Rules are written in code, so cool, I can build my own country
- ▶ It is transparent and deterministic, seriously
- ▶ It brings programming to a whole new level => A lot of excitements, a lot of considerations and discussions around it
- ▶ Ton of other reasons...





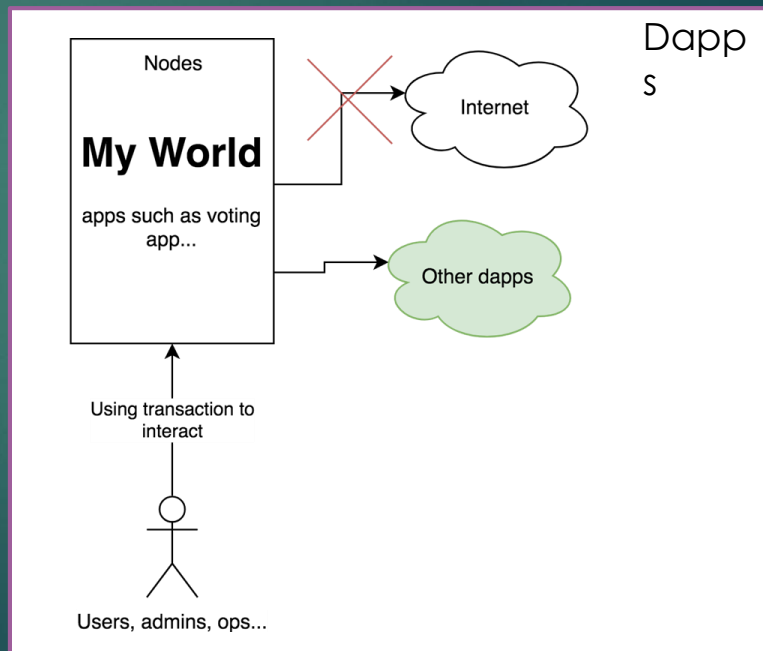
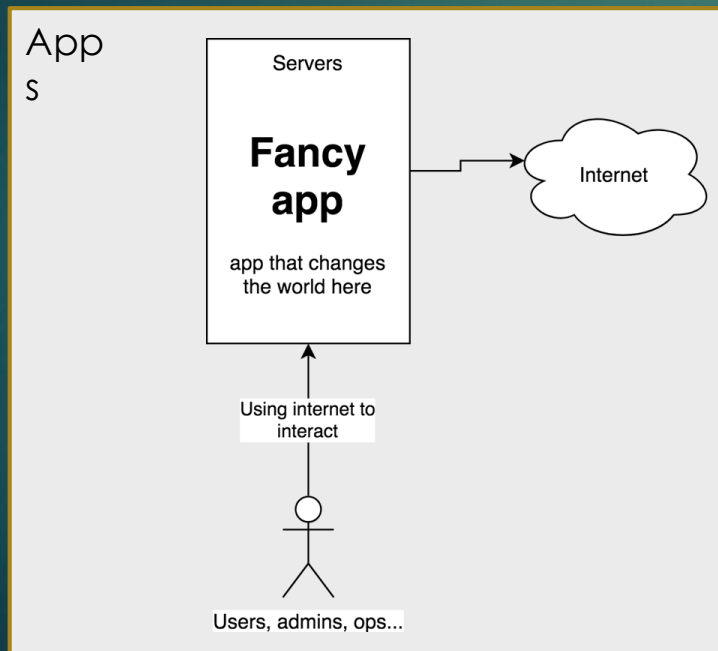
Lets get your head busy

- ▶ Solidity introduction
 - ▶ Basic concepts on smart contract
 - ▶ Ethereum virtual machine (EVM)
- ▶ Solidity Basic and Live Example Coding
- ▶ Common mistakes we learned with Kyber (advanced)

Solidity introduction



▶ Difference between solidity program and usual program





Basic concepts

- ▶ Ethereum smart contract is written in Solidity
- ▶ Solidity program is compiled to EVM byte code
- ▶ Compiled byte code is stored on Ethereum
- ▶ Smart contract is a “person”



Basic concepts - 2

- ▶ Everyone talks to that person using a transaction
- ▶ A transaction must be signed and it costs money (gas)
- ▶ Read functions
- ▶ Write function is executed in each block
- ▶ Blocks – blockchain concept – snapshot of a smart contract



Ethereum Virtual Machine

- ▶ Isolated environment to run byte code
- ▶ Accounts, Transactions, Gas
- ▶ Logs, memory and storage
- ▶ Many more, read more at <https://solidity.readthedocs.io/en/v0.4.24/introduction-to-smart-contracts.html#index-6>
- ▶ Why is a new Virtual machine needed? Why not Java nor Go?



Layout and Structure

- ▶ Simple and straight forward
- ▶ Very similar to JS with types
- ▶ Comment is not mandatory but very important

```
1 pragma solidity ^0.4.22;
2
3 import "../zeppelin/SafeMath.sol";
4
5 /// @title Voting with registration.
6 // This is only a structural contract, don't use it.
7 contract Vote {
8     // This is a type for a single proposal.
9     struct Proposal {
10         bytes32 name; // short name (up to 32 byte
11         uint voteCount; // number of accumulated vot
12     }
13
14     address public chairperson;
15     mapping(address => Voter) public voters;
16     Proposal[] public proposals;
17
18     event Voted(address voter, uint proposal);
19
20     /// Create a new ballot to choose one of `propos
21     constructor(bytes32[] proposalNames) public {
22         chairperson = msg.sender;
23         voters[chairperson].weight = 1;
24         // TODO:
25     }
26
27     function vote(uint p) public onlyRegistered {
28     }
29
30     function winningProposal() public view
31         returns (uint winningProposal_)
32     {
33     }
34 }
```



```
1 pragma solidity ^0.4.22;
2
3 import "../zeppelin/SafeMath.sol";
4
5 /// @title Voting with registration.
6 // This is only a structural contract, don't use it.
7 contract Vote {
8     // This is a type for a single proposal.
9     struct Proposal {
10         bytes32 name;    // short name (up to 32 byte
11         uint voteCount; // number of accumulated vot
12     }
13
14     address public chairperson;
15     mapping(address => Voter) public voters;
16     Proposal[] public proposals;
17
18     event Voted(address voter, uint proposal);
19
20     /// Create a new ballot to choose one of `propos
21     constructor(bytes32[] proposalNames) public {
22         chairperson = msg.sender;
23         voters[chairperson].weight = 1;
24         // TODO:
25     }
26
27     function vote(uint p) public onlyRegistered {
28     }
29
30     function winningProposal() public view
31         returns (uint winningProposal_)
32     {
33     }
34 }
```

Pragma: not mandatory, should declare
Current version: 0.4.24
Up coming version: 0.5 with many
improved features



```
1 pragma solidity ^0.4.22;
2
3 import "../zeppelin/SafeMath.sol";
4
5 /// @title Voting with registration.
6 // This is only a structural contract, don't use it.
7 contract Vote {
8     // This is a type for a single proposal.
9     struct Proposal {
10         bytes32 name; // short name (up to 32 bytes)
11         uint voteCount; // number of accumulated votes
12     }
13
14     address public chairperson;
15     mapping(address => Voter) public voters;
16     Proposal[] public proposals;
17
18     event Voted(address voter, uint proposal);
19
20     /// Create a new ballot to choose one of `proposals`
21     constructor(bytes32[] proposalNames) public {
22         chairperson = msg.sender;
23         voters[chairperson].weight = 1;
24         // TODO:
25     }
26
27     function vote(uint p) public onlyRegistered {
28     }
29
30     function winningProposal() public view
31         returns (uint winningProposal_)
32     {
33     }
34 }
```

Import: can import other sol files

- It can be relative to the current file or absolute
- Can name an import, make alias and more tricks like JS
- Zeppelin is a set of smart contracts that we should use to make smart contracts on top of



```
1 pragma solidity ^0.4.22;
2
3 import "../zeppelin/SafeMath.sol";
4
5 /// @title Voting with registration.
6 // This is only a structural contract, don't use it.
7 contract Vote {
8     // This is a type for a single proposal.
9     struct Proposal {
10         bytes32 name; // short name (up to 32 byte
11         uint voteCount; // number of accumulated vot
12     }
13
14     address public chairperson;
15     mapping(address => Voter) public voters;
16     Proposal[] public proposals;
17
18     event Voted(address voter, uint proposal);
19
20     /// Create a new ballot to choose one of `propos
21     constructor(bytes32[] proposalNames) public {
22         chairperson = msg.sender;
23         voters[chairperson].weight = 1;
24         // TODO:
25     }
26
27     function vote(uint p) public onlyRegistered {
28     }
29
30     function winningProposal() public view
31         returns (uint winningProposal_)
32     {
33     }
34 }
```

Contract declaration:

- Same to other OO languages
- Comment is needed here to explain what the contract does



```
1 pragma solidity ^0.4.22;
2
3 import "../zeppelin/SafeMath.sol";
4
5 /// @title Voting with registration.
6 // This is only a structural contract, don't use it.
7 contract Vote {
8     // This is a type for a single proposal.
9     struct Proposal {
10         bytes32 name; // short name (up to 32 byte
11         uint voteCount; // number of accumulated vot
12     }
13
14     address public chairperson;
15     mapping(address => Voter) public voters;
16     Proposal[] public proposals;
17
18     event Voted(address voter, uint proposal);
19
20     /// Create a new ballot to choose one of `propos
21     constructor(bytes32[] proposalNames) public {
22         chairperson = msg.sender;
23         voters[chairperson].weight = 1;
24         // TODO:
25     }
26
27     function vote(uint p) public onlyRegistered {
28     }
29
30     function winningProposal() public view
31         returns (uint winningProposal_)
32     {
33     }
34 }
```

Custom type declaration:

- Can contains many builtin types
- Will allocate continuous storage slots (*)
- Comment is needed here to explain what the type and each field represents



```
1 pragma solidity ^0.4.22;
2
3 import "../zeppelin/SafeMath.sol";
4
5 /// @title Voting with registration.
6 // This is only a structural contract, don't use it.
7 contract Vote {
8     // This is a type for a single proposal.
9     struct Proposal {
10         bytes32 name; // short name (up to 32 byte
11         uint voteCount; // number of accumulated vot
12     }
13
14     address public chairperson;
15     mapping(address => Voter) public voters;
16     Proposal[] public proposals;
17
18     event Voted(address voter, uint proposal);
19
20     /// Create a new ballot to choose one of `propos
21     constructor(bytes32[] proposalNames) public {
22         chairperson = msg.sender;
23         voters[chairperson].weight = 1;
24         // TODO:
25     }
26
27     function vote(uint p) public onlyRegistered {
28     }
29
30     function winningProposal() public view
31         returns (uint winningProposal_)
32     {
33     }
34 }
```

State variable declarations:

- Like field of a class, can be accessed in 2 ways
 - `this.vote()`
 - `vote()`
- Will be persisted to storage
- Common types: address, uint, int, bytes, map, []uint...
- Comment is needed here to explain what the variables represent and how they will be used in the contract



```
1 pragma solidity ^0.4.22;
2
3 import "../zeppelin/SafeMath.sol";
4
5 /// @title Voting with registration.
6 // This is only a structural contract, don't use it.
7 contract Vote {
8     // This is a type for a single proposal.
9     struct Proposal {
10         bytes32 name; // short name (up to 32 byte
11         uint voteCount; // number of accumulated vot
12     }
13
14     address public chairperson;
15     mapping(address => Voter) public voters;
16     Proposal[] public proposals;
17
18     event Voted(address voter, uint proposal);
19
20     /// Create a new ballot to choose one of `propos
21     constructor(bytes32[] proposalNames) public {
22         chairperson = msg.sender;
23         voters[chairperson].weight = 1;
24         // TODO:
25     }
26
27     function vote(uint p) public onlyRegistered {
28     }
29
30     function winningProposal() public view
31         returns (uint winningProposal_)
32     {
33     }
34 }
```

Event declaration:

- Event is used to store data on log facility
- Event is only persisted if the tx is successful
- Event is used to make sure something happened
- Only use event in write functions
- Comment is needed here to explain what the event is about and when it will be emit
- Event is important



```
1 pragma solidity ^0.4.22;
2
3 import "../zeppelin/SafeMath.sol";
4
5 /// @title Voting with registration.
6 // This is only a structural contract, don't use it.
7 contract Vote {
8     // This is a type for a single proposal.
9     struct Proposal {
10         bytes32 name; // short name (up to 32 byte
11         uint voteCount; // number of accumulated vot
12     }
13
14     address public chairperson;
15     mapping(address => Voter) public voters;
16     Proposal[] public proposals;
17
18     event Voted(address voter, uint proposal);
19
20     /// Create a new ballot to choose one of `propos
21     constructor(bytes32[] proposalNames) public {
22         chairperson = msg.sender;
23         voters[chairperson].weight = 1;
24         // TODO:
25     }
26
27     function vote(uint p) public onlyRegistered {
28     }
29
30     function winningProposal() public view
31         returns (uint winningProposal_)
32     {
33     }
34 }
```

Constructor:

- Constructor like other OO programming languages
- Constructor is usually public, it can be internal and can't be deployed. Other contracts can inherit though
- Can't use this in constructor
- Usually use to init all of the state variables
- Comment here should explain relationships between the params



```
1 pragma solidity ^0.4.22;
2
3 import "../zeppelin/SafeMath.sol";
4
5 /// @title Voting with registration.
6 // This is only a structural contract, don't use it.
7 contract Vote {
8     // This is a type for a single proposal.
9     struct Proposal {
10         bytes32 name; // short name (up to 32 byte
11         uint voteCount; // number of accumulated vot
12     }
13
14     address public chairperson;
15     mapping(address => Voter) public voters;
16     Proposal[] public proposals;
17
18     event Voted(address voter, uint proposal);
19
20     /// Create a new ballot to choose one of `propos
21     constructor(bytes32[] proposalNames) public {
22         chairperson = msg.sender;
23         voters[chairperson].weight = 1;
24         // TODO:
25     }
26
27     function vote(uint p) public onlyRegistered {
28     }
29
30     function winningProposal() public view
31         returns (uint winningProposal_)
32     {
33     }
34 }
```

Actual functions:

- Public, private, external, internal
- "Write", view, pure, payable (fallback)
- Fallback function is important to prevent mistaken fund transferring and other features
- Many more things...



What you can access with solidity

- ▶ Block and transaction information
 - ▶ Block diff, block hash...
 - ▶ Gas price, Gas left...
 - ▶ Msg sender, data, gas...
- ▶ Built-in functions (ABI encoding, crypto, mathematic, address related, contract calls)
- ▶ Currency units
- ▶ Time units

I'm bored



Bring on the code

Ethereum Virtual Machine – Why?



- ▶ Bytecode optimization on storage
- ▶ Feature optimizations and eliminate security concerns of untrusted code or arbitrary parties (think of Java)
- ▶ Perfect determinism
- ▶ Support DDOS resistance (infinite loop resistance)

Live coding



- ▶ Voting dapp
 - ▶ Only admin can change admin address
 - ▶ People can register with a bond of 0.1 eth
 - ▶ Only admin can create a campaign
 - ▶ A campaign comes with a life time (range)
 - ▶ Only added people can vote on the campaigns
- ▶ Framework: truffle
- ▶ Editor: vim
- ▶ Simple coding flow



Install the environment

▶ Truffle

npm install -g truffle

▶ Init the repository

mkdir voting && cd voting

truffle init

Voting app



- ▶ Write constructor
- ▶ Deployment script
- ▶ Deploy and test
- ▶ Continue writing the code
- ▶ Deploy and test until it is expected

Discussions around the voting app



- ▶ Privacy issue. how can we hide info of a person voting on something
- ▶ What if people don't vote (it is an issue if voting is a governance mechanism)
- ▶ What if a campaign is in draw situation?
- ▶ Is that the admin has too much of power as admin can decline to create a campaign if he doesn't like it

read more about DAO



Common mistakes we learned

- ▶ Many tokens look like ERC20, they are not (eg. OMG)
 - ▶ Interface only makes sure you will send the right msg, not the return
 - ▶ The compiler doesn't know if a smart contract implements a specific interface or not so depends on compiler version, it might not revert which hides the bug from you

```
/**
 * @dev transfer token for a specified address
 * @param _to The address to transfer to.
 * @param _value The amount to be transferred.
 */
function transfer(address _to, uint _value) onlyPayloadSize(2 * 32) {
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    Transfer(msg.sender, _to, _value);
}
```

Common mistakes we learned



- ▶ Permission is critical, modifiers can be forgotten easily
- ▶ Non internal, private functions are dangerous
 - ▶ Be very careful when you turn a function to public (for test for example)
 - ▶ Even experienced smart contract programmers miss this

Common mistakes we learned



- ▶ Compiler is tricky, truffle is no better
 - ▶ Should use interface instead of full source code (save gas, stable bytecode print)
 - ▶ Should delete build dir every time, otherwise tests will behave strangely
 - ▶ Compiler doesn't optimize the gas
 - ▶ Sort struct fields so it can fit the least byte words
 - ▶ Using `s1.x = s2.x; s1.y = s2.y` instead of `s1 = s2`



Security pitfalls

- ▶ Everything that interacts with your smart contract is public
- ▶ Be careful when calling a function to other contracts
 - ▶ Stack abuse (more than 1024 stack trace)
 - ▶ Gas consumption
 - ▶ Reentrancy
- ▶ Sending ETH to other addresses is even trickier
 - ▶ Trigger fallback function

Thanks



- ▶ All materials will be uploaded to event group
- ▶ Be curious, Ethereum is new, Solidity is new, don't limit yourself
- ▶ Check out the Solidity docs at
<https://solidity.readthedocs.io/en/v0.4.24>