# TrueChain Dapp Development Guides

The concepts section helps you learn about the parts of DApp(Decentralized App) and helps you obtain a deeper understand of how to develop.

**A DApp consists of two parts**:

* front which develop by web component
* backend which is consisting of the smart contract, that can be developed using a language called Solidity.

**The basic DApp development include**:

* develop the smart contract by Solidity language
* deploy the TrueChain private network
* Easy Dapp tool deployment contract
* The front end uses web3.js to call the contract

## Smart Contract develop

[Solidity](https://github.com/truechain/wiki/blob/master/developer/solidity.readthedocs.io) is a High-level programming language for implementing smart contracts. For those familiar with Javascript, I know you must be attention that Solidity language is very similar the type of Javascript. Which is supports static-type,inheritance, libraries, complex user-defined types, and other features. Originally designed to run on the Ethernet Workshop Virtual Machine (EVM).

TureChain intelligent contracts development supports the Solidity language. Refer to the example of TrueChain intelligent contracts. For more information, refer to the official documentation: [Understanding Solidity In Depth](https://github.com/truechain/wiki/blob/master/developer/solidity-cn.readthedocs.io/zh/develop/solidity-in-depth.html).

Development tools recommend using Solidity IDE [Remix](https://github.com/truechain/wiki/blob/master/developer/remix.ethereum.org), it is a Web browser-based IDE, which lets you write Solidity smart contracts and then deploy and run the smart contract. deployment of the TureChain's intelligent contracts , you can use the TrueChain team development of a tool, described later.

when the current file is changed or another file is selected each time,Remix recompiles the code , It also provides syntax highlighting of the Solidity keyword.

Deploy TrueChain Private Network

Generally DApp development phase, development and deployment of our smart contracts, by building their own private initial TrueChain node.

This specific operation can refer to [the test network building](https://github.com/truechain/wiki/blob/master/developer/github.com/truechain/wiki/blob/master/developer/1.4.truechain_testing_network_setup.md).

Easy Dapp deployment contract

There are two ways to deploy a contract through Easy Dapp Stellar: the source code of Solidity and the contract ABI. At the same time support the deployment of contracts to the TrueChain beta main network, local TrueChain network, Ethernet main network, and local Ethernet workshop network.

Firstly setting up the configure custom Provider. The endpoint format is IP:Port, for example: 127.0.0.1:8545

Second, adding to the account which is required for the deployment contract, can use the private key or importing the KeyStore file.

Finally choose the contract source code or ABI way, the deployment contract can be.

Font Call the Contract

Build client application interacts with smart contracts ,and write HTML and JavaScript just like normal front-end development.

The main difference is the part of interacts with the contract.

The front and rear end of the TrueChain DApp interacts, mainly through the web3.js.

It is a JavaScript library that allows developers to use HTTP or IPC to connect to local or remote TrueChain nodes, communicate with initial-chained nodes via JSON RPC calls.

To make DApp run on the TrueChain, use the web3 object provided by the web3.js library.

**Add web3:**

First you need to put the web3.js into the project. This can be done in the following ways:

install the web3.js dependence, Example:

```

> npm: npm install web3

> bower: bower install web3

> meteor: meteor add ethereum:web3

```

Then you need to create an instance of web3 and set up a Provider. To ensure that the set Provider, is not overwritten when using Stellar, first check that the web3 is available:

// Initialize web3 and set the provider to the test RPC.

if (typeof web3 !== 'undefined') {

web3Provider = web3.currentProvider;

web3 = new Web3(web3.currentProvider);

} else {

// set the provider you want from Web3.providers

web3Provider = new Web3.providers.HttpProvider('http://127.0.0.1:7545');

web3 = new Web3(web3Provider);

}

Then you can use the API of web3 object£¬ API reference: TrueChain API

**Callback**:

Because API is intended to be used with local RPC nodes, all its functions use synchronous HTTP requests by default.

If you want to make an asynchronous request, you can pass the optional callback as the last parameter to most functions.

All callbacks use the Error-First Callbacks style:

```

web3.eth.getBlock(48, function(error, result){

if(!error)

console.log(JSON.stringify(result));

else

console.error(error);

})

```