Desarrollo de software para blockchain 2024

Clase 04

Conceptos

- Web3
- web3 py

Web3

Web3

La Web3 es una nueva forma del desarrollo web que está emergiendo, caracterizada por tecnologías descentralizadas, principalmente basadas en blockchain. Es una "evolución" de la Internet que conocemos (Web2), con un enfoque en la propiedad, la descentralización, la resistencia a la censura y la participación directa del usuario.

Web3: características

1- Descentralizada: La Web3 utiliza blockchain y otras tecnologías descentralizadas para operar. Esto significa que no hay un punto central de control o fallo. Las aplicaciones en Web3 no son propiedad ni están controladas por una única entidad; en cambio, su operación y gobernanza a menudo están distribuidas entre los participantes de la red.

2- Propiedad de Datos: Los usuarios de la Web3 tienen control y propiedad sobre sus propios datos. En lugar de ceder datos a las plataformas para acceder a servicios, los usuarios de Web3 pueden controlar quién accede a sus datos y cómo se utilizan.

3- Modelo de Beneficios Distribuido: En lugar de que una sola empresa o entidad obtenga beneficios, los sistemas de Web3 permiten que los usuarios que contribuyen y participan en la plataforma también obtengan recompensas. Esto se realiza a menudo a través de sistemas de tokens que incentivan la participación activa.

Web3: características

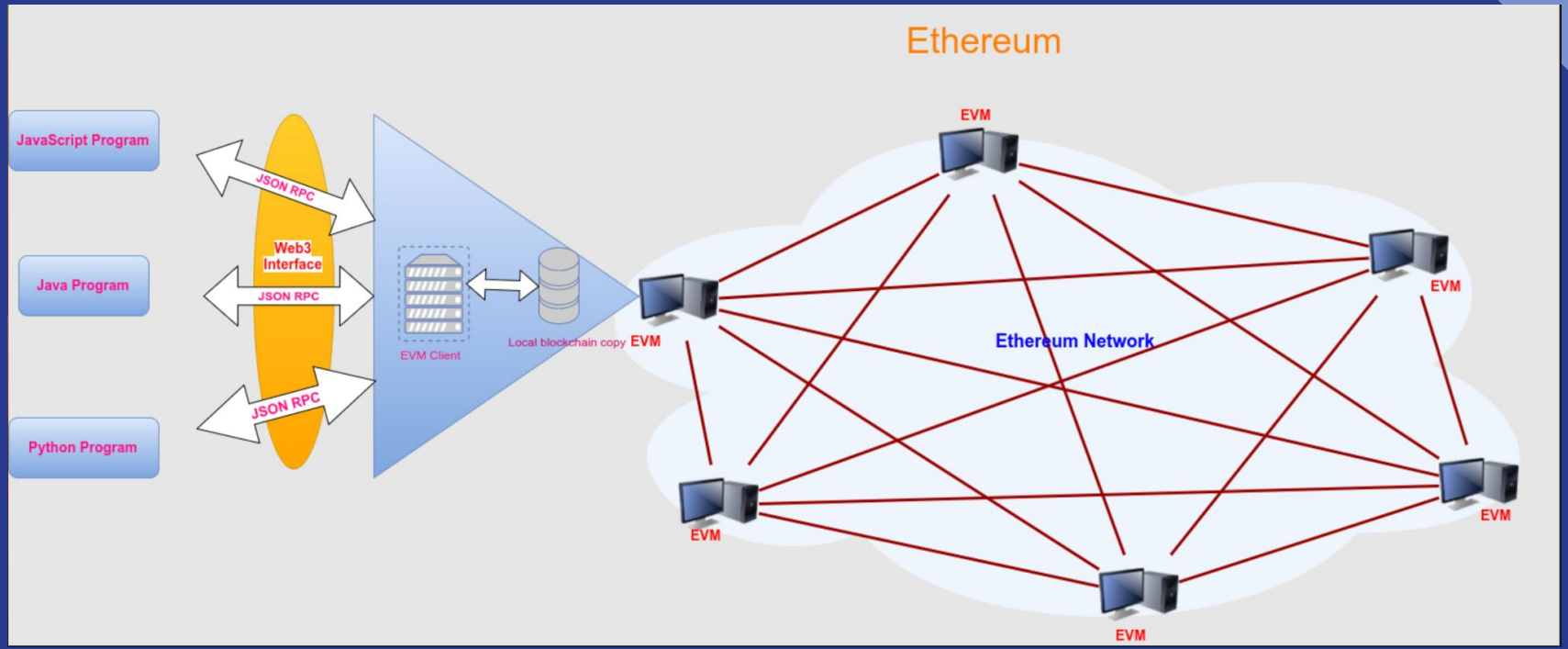
4- Interoperabilidad: Las plataformas y aplicaciones de Web3 están construidas con una mentalidad de composición. Se pueden combinar, integrar y construir conjuntamente, creando un ecosistema enriquecido y conectado.

5- Transparencia y Verificabilidad: Las transacciones y operaciones en Web3 son transparentes y pueden ser verificadas por cualquier usuario, lo que crea un ambiente de confianza. La confianza se basa en la verificabilidad del código y las operaciones, no solo en la reputación de una entidad.

6- Participación Directa: Los usuarios no son meros consumidores en Web3. Pueden ser participantes activos, involucrándose en la gobernanza, tomando decisiones, validando transacciones y aportando al desarrollo y dirección de las plataformas.

7- Resistencia a la Censura: Debido a su naturaleza descentralizada, es difícil para una sola entidad controlar, censurar o apagar una aplicación o plataforma de Web3.

Web3: clientes



Web3: clientes

un cliente web3 es una colección de bibliotecas que permiten interactuar con un nodo local o remoto de alguna blockchain, utilizando una conexión HTTP, RPC o WebSocket. Es esencialmente la "puerta de entrada" que permite a las aplicaciones en la web interactuar con la blockchain. Ls versiones más conocidas de estas bibliotecas son web3.js y web3.py, aunque hay implementaciones en otros lenguajes.

Web3: clientes - características

1. Conexión con la Blockchain: Permite conectarse a la red a través de un proveedor (por ejemplo, un nodo local como Geth o servicios como Infura).

2. Interacción con Contratos Inteligentes: Una vez obtenido el ABI (Application Binary Interface) de un contrato, se puede usar `web3` para interactuar con ese contrato: leer datos, enviar transacciones, etc.

3. Construcción y Envío de Transacciones: Se puede construir transacciones, firmarlas y luego enviarlas a la red.

4. Consulta de Datos: Se puede consultar datos de la blockchain, como el balance de una dirección, el estado de una transacción, o la información de un bloque.

5. Utils: también viene con una serie de utils para realizar tareas como convertir valores entre Ether y Wei, o calcular el hash de una serie de datos.

es esencial para cualquier dapp (aplicación descentralizada), ya que facilita la interacción entre la parte frontal (frontend) de la aplicación y la blockchain.

Web3py

necesitaremos:

1- Url de nodo para conectarnos para leer información o enviar una transacción, aquí podremos utilizar algún servicio, buscar algún nodo público o levantar un nodo propio.

2- una privada donde su pública tenga fondos para realizar transacciones.

Web3py

1- para simplificar podemos utilizar como proveedor: infura o alchemy

2- usaremos alguna pk de una billetera cargada previamente con fondos.

3- la doc oficial está en: https://web3py.readthedocs.io/en/stable/

Web3py: conectando a un nodo

```
#pip install web3
from web3 import Web3
import os


NODE = os.environ.get("NODE")
w3 = Web3(Web3.HTTPProvider(NODE))
w3.isConnected()
```

Web3py: obteniendo info del último bloque y la pública de una pk

```
print(w3.eth.get_block('latest'))
```

```
account = w3.eth.account.from_key(os.environ.get("pk"))
print(account.address)
```

Web3py: enviando fondos

```
signed_txn = w3.eth.account.sign_transaction({
    "nonce":w3.eth.get_transaction_count(account.address),
    "gas":21000,
    "gasPrice":w3.eth.gasPrice,
    "to":"0x27227775d3dd9a6b5A9F1DD02761470E7D7EA466",
    "value":1*10**17 ,#en wei esto seria 0.1 ETH
    "chainId":5,
    },
    os.environ.get("pk")
)

tx_hash = w3.eth.send_raw_transaction(signed_txn.rawTransaction)
```

Web3py: obteniendo data de una tx

```
txn_hash = "0x6aa5347995e7f468153f099d5a962e556203a000d643848b60f9bdd522e0910e"
data = w3.eth.get_transaction(txn_hash)
```

Web3py: leyendo data de un contrato

Para leer data de un contrato, aparte de la address del mismo necesitamos el ABI.

```
abi = """[{
            "inputs": [],
            "stateMutability": "nonpayable",
            "type": "constructor"
        },
        {
            "inputs": [],
            "name": "increment",
            "outputs": [],
            "stateMutability": "nonpayable",
            "type": "function"
        },
        {
            "inputs": [],
            "name": "retrieve",
            "outputs": [
                {
                    "internalType": "uint256",
                    "name": "",
                    "type": "uint256"
                }
            ],
            "stateMutability": "view",
            "type": "function"
        }
    ]"""
contract_address = w3.toChecksumAddress("0x2b5aadd271f1f1a1e7468d0e9abb5907ae6a4af2")
```

Web3py: leyendo data de un contrato

```
mi_contrato = w3.eth.contract(
    address=contract_address,
    abi=abi
)
mi_contrato.functions.retrieve().call()
```

Web3py: escribiendo en un contrato

```
tx = {
    "nonce":w3.eth.get_transaction_count (account.address),
    "gas":200000,
    "gasPrice":w3.eth.gasPrice,
    "value":0 ,
    "chainId":5,
}

contract_data  = mi_contrato.functions.increment().buildTransaction(  tx)
signed_txn  = w3.eth.account.signTransaction( contract_data , os.environ.get("pk"))
txn_hash  = w3.eth.sendRawTransaction(signed_txn.rawTransaction)
```

Web3py: escribiendo en un contrato, calculando el gas

```
#estimando el gas
tx = {
    "nonce":w3.eth.get_transaction_count(account.address),
    "gasPrice":w3.eth.gasPrice,
    "value":0 ,
    "chainId":5,
}
contract_data = mi_contrato.functions.increment().buildTransaction( tx)
estimated_gas = w3.eth.estimateGas(contract_data)
tx['gas'] = estimated_gas

contract_data = mi_contrato.functions.increment().buildTransaction( tx)
signed_txn = w3.eth.account.signTransaction(contract_data, os.environ.get("pk"))
txn_hash = w3.eth.sendRawTransaction(signed_txn.rawTransaction)
```

Web3py: obteniendo las txs de la mempool

```
# Obtener el bloque "pending", que representa el conjunto actual de transacciones pendientes
pending_block = w3.eth.getBlock('pending')

# Extraer las transacciones del bloque pendiente
pending_transactions = pending_block['transactions']

# Imprimir las transacciones pendientes
for tx in pending_transactions:
    print(tx)

# Si deseas obtener los detalles completos de cada transacción, puedes hacerlo con:
for tx_hash in pending_transactions:
    tx = w3.eth.getTransaction(tx_hash)
    print(tx)
```

Web3py: api leer data

API

- `web3.eth.get_balance()`
- `web3.eth.get_block()`
- `web3.eth.get_block_transaction_count()`
- `web3.eth.get_code()`
- `web3.eth.get_proof()`
- `web3.eth.get_storage_at()`
- `web3.eth.get_transaction()`
- `web3.eth.get_transaction_by_block()`
- `web3.eth.get_transaction_count()`
- `web3.eth.get_uncle_by_block()`
- `web3.eth.get_uncle_count()`

Web3py: api escribir data

API

- `web3.eth.send_transaction()`
- `web3.eth.sign_transaction()`
- `web3.eth.send_raw_transaction()`
- `web3.eth.replace_transaction()`
- `web3.eth.modify_transaction()`
- `web3.eth.wait_for_transaction_receipt()`
- `web3.eth.get_transaction_receipt()`
- `web3.eth.sign()`
- `web3.eth.sign_typed_data()`
- `web3.eth.estimate_gas()`
- `web3.eth.generate_gas_price()`
- `web3.eth.set_gas_price_strategy()`

Web3py: api contrato

API

- `web3.eth.contract()`
- `Contract.address`
- `Contract.abi`
- `Contract.bytecode`
- `Contract.bytecode_runtime`
- `Contract.functions`
- `Contract.events`
- `Contract.fallback`
- `Contract.constructor()`
- `Contract.encodeABI()`
- `web3.contract.ContractFunction`
- `web3.contract.ContractEvents`

Web3py: api eventos

API

- `web3.eth.filter()`
- `web3.eth.get_filter_changes()`
- `web3.eth.get_filter_logs()`
- `web3.eth.uninstall_filter()`
- `web3.eth.get_logs()`
- `Contract.events.your_event_name.create_filter()`
- `Contract.events.your_event_name.build_filter()`
- `Filter.get_new_entries()`
- `Filter.get_all_entries()`
- `Filter.format_entry()`
- `Filter.is_valid_entry()`