4 장. Web3.js 시작하기

1. Web3.js 소개
   - web3.js 는 geth 와 통신할 수 있는 자바스크립트 API 를 제공
   - 내부적으로는 JSON-RPC 이용
   - 위스퍼와 스웜과 관련된 API 도 제공
2. web3.js 불러오기
   - 프로젝트 디렉토리 내에서 npm install web3 를 실행하고, 소스코드에서 require("web3"); 를 사용해 불러오면 됨 => web3 객체 전역적 사용가능
3. 노드에 연결
   - web3.js 는 HTTP 또는 IPC 를 사용해 노드와 통신 가능, 다수의 노드 가능
   - HTTP 를 사용해 노드와 연결하는 예제 코드

```
if(typeof web3 !== 'undefined'){
    web3 = new Web3(web3.currentProvider);
} else {
Web3.providers
    Web3 = new Web3.providers.HttpProvider(http://localhost:8545));
}
```

4. API 구조
   - web3 는 이더리움 블록체인 상호작용을 위한 eth 객체와 위스퍼 상호작용을 위한 shh 객체를 포함하고 있고, 대부분의 API 는 이 두 객체 내부에 있음
   - 모든 API 는 동기식, 비동기식 요청을 만들고 싶으면 함수에 마지막 매개변수로 선택적인 콜백을 전달하면 됨
5. BigNumber.js
   - 큰 숫자를 다루거나 완벽한 계산이 필요한 애플리케이션은 BigNumber.js 라이브러리를 이용한다. => 자동으로 추가됨
   - web3.eth.getBalance("0x27E829fB34d14f33...c").toString();
     => 여기서 getBalance()메소드를 사용해 주소의 잔액을 얻고, 이 메소드는 BigNumber 객체를 반환한다. 숫자형 문자열로 변환하기

위해 toString()을 호출한 것임.

- 잔액을 wei 단위로 저장하고 보여줄 때 다른 단위로 변환하는 걸 권장

6. 단위 변환

- wei 잔액을 다른 단위로 (web3.fromWei("100000000000" , "ether");
- 다른 단위의 잔액을 wei 로 (web3.toWei("100000000000" , "ether");

7. 가스 가격, 잔액, 트랜잭션 상세 정보 검색

- web3.eth.gasPrice.toString() => 최신 블록 x 개의 가스가격 중앙값으로 가스 가격을 결정함
- web3.eth.getBalance("0x407dg...1", 45).toString() => 주어진 주소의 잔액을 리턴
- web3.eth.getTransactionReceipt("0x9f..8b") => 해시를 이용해 트랜잭션 상세 정보를 얻음. 트랜잭션을 찾으면 recipt 객체 리턴, 없으면 null 리턴

8. 이더 송금

- web3.ethsendTransaction() 메소드 사용 => 모든 종류의 트랜잭션 송신
- from, to, value, gas, gasPrice, data, nonce 포함.

```
var txnHash = web3.eth.sendTransaction({
        from: web3.eth.accounts[0],
        to: web3.eth.accounts[1],
        value: web3.toWei("1", "ether")
});  // 계좌번호 0 번으로부터 1 번으로 하나의 이더 전송하는 코드
```

9. 컨트랙트 작업

- web3.eth.contract() => 컨트랙트 객체 생성 => 인자는 컨트랙트 ABI
- 컨트랙트 객체의 new 메소드를 이용해 배포하거나 at 메소드를 사용해 ABI 와 매칭하는 이미 배포된 컨트랙트의 참조 값을 얻을 수 있다.
- 전달하는 객체는 from, 컨트랙트의 바이트코드, 사용할 최대 가스양의 정보를 갖고 있어야함. 그렇지 않으면 트랜잭션 생성 X

10. 컨트랙트의 메소드 호출을 위한 트랜잭션 전송하는 법

```
proof.set.sendTransaction("Owner Name", "e3b0c44298...55", {
        from: web3.eth.accounts[0],
        }, function(error, transactionHash){
```

```
    if (!err)
        console.log(transactionHash);
})
```

- sendTransaction 메소드에게 전달된 객체는 web3.eth.sendTransaction()과 같은 속성을 가진다.
- 트랜잭션을 생성하고 브로드캐스팅하는 대신 노드 자체에 있는 메소드를 호출하고 싶은 경우라면 sendTransaction 대신 call 사용가능

```
var returnValue = proof.get.call("e3b01582984 ... b855");
```

- 메소드를 호출하는 데 필요한 가스의 양을 알아내는 함수
  => web3.eth.estimateGas => 트랜잭션의 데이터를 생성해야 하므로 동일한 이름을 가진 객체의 estimateGas() 메소드를 사용 가능

```
var estimateGas = proof.get.estimateGas("e3b01582984 ... b855");
```

11. 모든 이벤트는 다음과 같은 속성을 포함하고 있는 객체로 표현된다.
- args : 이벤트 인자의 객체 - event : 이벤트 이름을 표현하는 문자열
- logIndex : 블록 내 로그 인덱스 위치를 표현하는 정수형
- transactionIndex : 인덱스 위치 로그가 생성된 트랜잭션을 표현하는 정수
- transactionHash : 이 로그가 생성된 트랜잭션의 해시를 표현하는 문자열
- address : 이 로그가 속해 있는 블록의 해시를 표현하는 문자열
- blockNumber : 이 로그가 속해 있는 블록 번호, 펜딩일 경우 null