event 사용

1.사용자 인터페이스의 스마트 계약 반환 값
2.데이터가있는 비동기식 트리거
3.더 싼 형태의 창고

사용자 인터페이스의 스마트 계약 반환 값

contract 호출에 대한 리턴값을 전달

```
contract ExampleContract {
  // some state variables ...
  function foo(int256 _value) returns (int256) {
    // manipulate state ...
    return _value;
  }
}
```

exampleContract 가 ExampleContract instance 라고 할 때,
다음의 형태로 반환값을 얻을 수 있다.

```
// eth_call
var returnValue = exampleContract.foo.call(2);
console.log(returnValue) // 2
```

web3js 정의, state 변화되지 않는다.

> web3.eth.call
> Executes a message call transaction, which is directly executed in the VM of the node, but never mined into the blockchain.

transaction 을 사용하는 경우

```
var returnValue = exampleContract.foo.sendTransaction(2, {from: web3.eth.coinbase})
console.log(returnValue) // transaction hash
```

반환값을 얻기 위해 권장된 솔루션은 이벤트 사용

```
contract ExampleContract {
  event ReturnValue(address indexed _from, int256 _value);
function foo(int256 _value) returns (int256) {
    ReturnValue(msg.sender, _value);
    return _value;
  }
}
A frontend can then obtain the return value:
var exampleEvent = exampleContract.ReturnValue({_from: web3.eth.coinbase});
exampleEvent.watch(function(err, result) {
  if (err) {
    console.log(err)
    return;
  }
  console.log(result.args._value)
  // check that result.args._from is web3.eth.coinbase then
  // display result.args._value in the UI and call
  // exampleEvent.stopWatching()
})
exampleContract.foo.sendTransaction(2, {from: web3.eth.coinbase})
```

데이터가 있는 비동기식 트리거

일반적으로 이벤트는 데이터가 포함된 비동기 트리거

```
// ICOP event
event UpdateState(uint8 ruleType, uint64 txId, uint8 state);

function refund(Context storage context, address _transferable, uint64[] _txIds) in
  ...
  emit UpdateState(context.ruleType, _txId, transaction.state);
  ...
}

function unlockInternal(Context storage context, uint64 _txId) {
  ...
  emit UpdateState(context.ruleType, _txId, transaction.state);
  ...
}
```

더 싼 형태의 창고

프론트 엔드에 의해 렌더링 될 수있는 히스토리 데이터를 저장하는 것입니다.
Logs basically cost 8 gas per byte, whereas contract storage costs 20,000 gas per 32 bytes.

```
contract CryptoExchange {
  event Deposit(uint256 indexed _market, address indexed _sender, uint256 _amount,
  function deposit(uint256 _amount, uint256 _market) returns (int256) {
    // perform deposit, update user's balance, etc
    Deposit(_market, msg.sender, _amount, now);
  }
}
```

사용자가 입금 할 때 UI를 업데이트한다

```
var depositEvent = cryptoExContract.Deposit ({_sender : userAddress});
depositEvent.watch (function (err, result) {
  if (err) {
    console.log (err)
    return
  }
  // UI에 result.args의 세부 사항을 추가
  // UI가 렌더링되면 depositEventAll.stopWatching ()이 호출되어야합니다
})
```

이전의 입금을 조회

```
var depositEventAll = cryptoExContract.Deposit({_sender: userAddress}, {fromBlock:
depositEventAll.watch(function(err, result) {
  if (err) {
    console.log(err)
    return;
  }
  // append details of result.args to UI
})
```

index 사용

최대 3개의 변수를 인덱싱해서 사용할 수 있다.

```
event Transfer(address indexed _from, address indexed _to, uint256 _value)
...
tokenContract.Transfer ({_ from : senderAddress})
...
```

```
tokenContract.Transfer ({_ to : receiverAddress})
...
tokenContract.Transfer ({_ from : senderAddress, _to : receiverAddress})
```

In solidity: The first topic is the hash of the signature of the event.

참고

Technical Introduction to Events and Logs in Ethereum