以太坊改进提案EIP1820（2019.03.04，需要EIP165和EIP214）修复了由Solidity0.5更新引入的ERC165逻辑中存在不兼容的问题，而除了对ERC165进行修复外，ERC1820在功能上与ERC820是等同的（现在必须用ERC1820代替ERC820）。

提案内容

EIP1820标准定义了一个通用的注册表智能合约，任何地址（包括合约或普通账户）都可以注册它所支持的接口以及实际负责该实现的智能合约。

该标准定义了一个注册表，智能合约和普通账户可以直接或通过代理合约发布它们实现的功能。任何人都可以在注册表上查询某个特定的地址是否实现了一个给定的接口，以及知道哪个智能合约负责具体实现该接口。这个注册表可以被部署在任何链上，并在所有链上共享相同的地址。

最后28个字节为0的接口被认为是ERC-165接口（ERC165标准中接口标识符为4字节）。

规格和标准化

下面是一个ERC-1820注册表智能合约的标准实现。

```
/* ERC1820 Pseudo-introspection Registry Contract
 * This standard defines a universal registry smart contract where any address (contract or regular account) can
 * register which interface it supports and which smart contract is responsible for its implementation.
 *
 * Written in 2019 by Jordi Baylina and Jacques Dafflon
 *
 * To the extent possible under law, the author(s) have dedicated all copyright and related and neighboring rights to
 * this software to the public domain worldwide. This software is distributed without any warranty.
 *
 * You should have received a copy of the CC0 Public Domain Dedication along with this software. If not, see
 * <http://creativecommons.org/publicdomain/zero/1.0/>.
 *
 *    ███████╗██████╗  ██████╗ ██╗ █████╗ ██████╗  ██████╗
 *    ██╔════╝██╔══██╗██╔════╝███║██╔══██╗╚════██╗██╔═████╗
 *    █████╗  ██████╔╝██║     ╚██║╚█████╔╝ █████╔╝██║██╔██║
 *    ██╔══╝  ██╔══██╗██║      ██║██╔══██╗██╔═══╝ ████╔╝██║
 *    ███████╗██║  ██║╚██████╗ ██║╚█████╔╝███████╗╚██████╔╝
 *    ╚══════╝╚═╝  ╚═╝ ╚═════╝ ╚═╝ ╚════╝ ╚══════╝ ╚═════╝
 *
 *    ██████╗ ███████╗ ██████╗ ██╗███████╗████████╗██████╗ ██╗   ██╗
 *    ██╔══██╗██╔════╝██╔════╝ ██║██╔════╝╚══██╔══╝██╔══██╗╚██╗ ██╔╝
 *    ██████╔╝█████╗  ██║  ███╗██║███████╗   ██║   ██████╔╝ ╚████╔╝
 *    ██╔══██╗██╔══╝  ██║   ██║██║╚════██║   ██║   ██╔══██╗  ╚██╔╝
 *    ██║  ██║███████╗╚██████╔╝██║███████║   ██║   ██║  ██║   ██║
 *    ╚═╝  ╚═╝╚══════╝ ╚═════╝ ╚═╝╚══════╝   ╚═╝   ╚═╝  ╚═╝   ╚═╝
 *
 */
pragma solidity 0.5.3;
// IV is value needed to have a vanity address starting with '0x1820'.
// IV: 53759

/// @dev The interface a contract MUST implement if it is the implementer of
/// some (other) interface for any address other than itself.
// 如果一个合约为除了它自己以外的其它合约实现其他接口，则这个合约必须实现下面这个接口
interface ERC1820ImplementerInterface {
    /// @notice Indicates whether the contract implements the interface 'interfaceHash' for the address 'addr' or not.
    /// @param interfaceHash keccak256 hash of the name of the interface
    /// @param addr Address for which the contract will implement the interface
    /// @return ERC1820_ACCEPT_MAGIC only if the contract implements 'interfaceHash' for the address 'addr'.
    // 判断实现了ERC1820接口的合约是否为addr地址实现了interfaceHash接口
    // 如果该合约为addr实现了interfaceHash接口，则应该返回ERC1820_ACCEPT_MAGIC
    function canImplementInterfaceForAddress(bytes32 interfaceHash, address addr) external view returns(bytes32);
}


/// @title ERC1820 Pseudo-introspection Registry Contract
/// @author Jordi Baylina and Jacques Dafflon
/// @notice This contract is the official implementation of the ERC1820 Registry.
/// @notice For more details, see https://eips.ethereum.org/EIPS/eip-1820
// ERC1820注册合约
contract ERC1820Registry {
    /// @notice ERC165 Invalid ID.
    // ERC165中非法的接口标识符
    bytes4 constant internal INVALID_ID = 0xffffffff;
    /// @notice Method ID for the ERC165 supportsInterface method (= `bytes4(keccak256('supportsInterface(bytes4)'))`).
    // ERC165接口标识符
    bytes4 constant internal ERC165ID = 0x01ffc9a7;
    /// @notice Magic value which is returned if a contract implements an interface on behalf of some other address.
    // 如果ERC1820注册表合约为某个地址实现了某个接口，则在查询时需要返回ERC1820_ACCEPT_MAGIC
    bytes32 constant internal ERC1820_ACCEPT_MAGIC = keccak256(abi.encodePacked("ERC1820_ACCEPT_MAGIC"));

    /// @notice mapping from addresses and interface hashes to their implementers.
    // 根据地址和接口哈希找到实现为该地址实现该接口的实现者地址
    mapping(address => mapping(bytes32 => address)) internal interfaces;
    /// @notice mapping from addresses to their manager.
    // 根据地址找到其管理者地址
    mapping(address => address) internal managers;
    /// @notice flag for each address and erc165 interface to indicate if it is cached.
    // 查询某个地址是否实现了以ERC165为标准的接口（缓存，需要更新）
    mapping(address => mapping(bytes4 => bool)) internal erc165Cached;

    /// @notice Indicates a contract is the 'implementer' of 'interfaceHash' for 'addr'.
    // 广播implementer是地址addr中interfaceHash接口的实现者
    event InterfaceImplementerSet(address indexed addr, bytes32 indexed interfaceHash, address indexed implementer);
    /// @notice Indicates 'newManager' is the address of the new manager for 'addr'.
    // 广播newManager是addr的新管理者
    event ManagerChanged(address indexed addr, address indexed newManager);

    /// @notice Query if an address implements an interface and through which contract.
    /// @param _addr Address being queried for the implementer of an interface.
    /// (If '_addr' is the zero address then 'msg.sender' is assumed.)
    /// @param _interfaceHash Keccak256 hash of the name of the interface as a string.
    /// E.g., 'web3.utils.keccak256("ERC777TokensRecipient")' for the 'ERC777TokensRecipient' interface.
    /// @return The address of the contract which implements the interface '_interfaceHash' for '_addr'
    /// or '0' if '_addr' did not register an implementer for this interface.
    // 查询一个地址是否实现了特定接口，如果实现了具体是通过哪个合约实现的
    function getInterfaceImplementer(address _addr, bytes32 _interfaceHash) external view returns (address) {
        // 如果传入的地址是0地址则代表查询msg.sender是否实现了该接口
        address addr = _addr == address(0) ? msg.sender : _addr;
        // 查看是否是ERC165的接口
        if (isERC165Interface(_interfaceHash)) {
            bytes4 erc165InterfaceHash = bytes4(_interfaceHash);
            return implementsERC165Interface(addr, erc165InterfaceHash) ? addr : address(0);
        }
        // 非ERC165接口
        return interfaces[addr][_interfaceHash];
    }

    /// @notice Sets the contract which implements a specific interface for an address.
    /// Only the manager defined for that address can set it.
    /// (Each address is the manager for itself until it sets a new manager.)
    /// @param _addr Address for which to set the interface.
    /// (If '_addr' is the zero address then 'msg.sender' is assumed.)
    /// @param _interfaceHash Keccak256 hash of the name of the interface as a string.
    /// E.g., 'web3.utils.keccak256("ERC777TokensRecipient")' for the 'ERC777TokensRecipient' interface.
    /// @param _implementer Contract address implementing '_interfaceHash' for '_addr'.
    // 为某个地址设置为其实现某个接口的合约地址
    function setInterfaceImplementer(address _addr, bytes32 _interfaceHash, address _implementer) external {
        //如果_addr是0地址则是为msg.sender设置
        address addr = _addr == address(0) ? msg.sender : _addr;
        //msg.sender需要是_addr的管理者地址
        require(getManager(addr) == msg.sender, "Not the manager");
        //不能是ERC165接口
        require(!isERC165Interface(_interfaceHash), "Must not be an ERC165 hash");
        //如果接口实现者不是0地址或者是msg.sender
        if (_implementer != address(0) && _implementer != msg.sender) {
            //查询_implementer是不是_addr对应接口的实际实现者
            require(
                ERC1820ImplementerInterface(_implementer)
                    .canImplementInterfaceForAddress(_interfaceHash, addr) == ERC1820_ACCEPT_MAGIC,
                "Does not implement the interface"
            );
        }
        //设置_implementer
        interfaces[addr][_interfaceHash] = _implementer;
        emit InterfaceImplementerSet(addr, _interfaceHash, _implementer);
    }

    /// @notice Sets '_newManager' as manager for '_addr'.
    /// The new manager will be able to call 'setInterfaceImplementer' for '_addr'.
    /// @param _addr Address for which to set the new manager.
    /// @param _newManager Address of the new manager for 'addr'. (Pass '0x0' to reset the manager to '_addr'.)
    // 为_addr设置管理者，调用该函数的地址必须为_addr的地址
    // 如果_newManager原本就是_addr的话则将管理者设置为_addr自身，表现为全0地址
    function setManager(address _addr, address _newManager) external {
        require(getManager(_addr) == msg.sender, "Not the manager");
        managers[_addr] = _newManager == _addr ? address(0) : _newManager;
        emit ManagerChanged(_addr, _newManager);
    }

    /// @notice Get the manager of an address.
    /// @param _addr Address for which to return the manager.
    /// @return Address of the manager for a given address.
    // 获得一个地址的管理者地址，默认管理者地址为自身
    function getManager(address _addr) public view returns(address) {
        // By default the manager of an address is the same address
        if (managers[_addr] == address(0)) {
            return _addr;
        } else {
            return managers[_addr];
        }
    }

    /// @notice Compute the keccak256 hash of an interface given its name.
    /// @param _interfaceName Name of the interface.
    /// @return The keccak256 hash of an interface name.
    // 计算某个接口名称的哈希值
    function interfaceHash(string calldata _interfaceName) external pure returns(bytes32) {
        return keccak256(abi.encodePacked(_interfaceName));
    }

    // --- ERC165 Related Functions ---  ERC165相关功能
    // --- Developed in collaboration with William Entriken. ---

    /// @notice Updates the cache with whether the contract implements an ERC165 interface or not.
    /// @param _contract Address of the contract for which to update the cache.
    /// @param _interfaceId ERC165 interface for which to update the cache.
    // 更新ERC165接口缓存，记录某个地址已实现ERC165接口，同时将自身地址放在interfaces中
    function updateERC165Cache(address _contract, bytes4 _interfaceId) external {
        interfaces[_contract][_interfaceId] = implementsERC165InterfaceNoCache(
            _contract, _interfaceId) ? _contract : address(0);
        erc165Cached[_contract][_interfaceId] = true;
    }

    /// @notice Checks whether a contract implements an ERC165 interface or not.
    //  If the result is not cached a direct lookup on the contract address is performed.
    //  If the result is not cached or the cached value is out-of-date, the cache MUST be updated manually by calling
    //  'updateERC165Cache' with the contract address.
    /// @param _contract Address of the contract to check.
    /// @param _interfaceId ERC165 interface to check.
    /// @return True if '_contract' implements '_interfaceId', false otherwise.
    // 通过cache检查某个合约是否实现了ERC165接口
    function implementsERC165Interface(address _contract, bytes4 _interfaceId) public view returns (bool) {
        if (!erc165Cached[_contract][_interfaceId]) {
            return implementsERC165InterfaceNoCache(_contract, _interfaceId);
        }
        return interfaces[_contract][_interfaceId] == _contract;
    }

    /// @notice Checks whether a contract implements an ERC165 interface or not without using nor updating the cache.
    /// @param _contract Address of the contract to check.
    /// @param _interfaceId ERC165 interface to check.
    /// @return True if '_contract' implements '_interfaceId', false otherwise.
    // 不通过cache检查某个合约是否实现了ERC165接口
    function implementsERC165InterfaceNoCache(address _contract, bytes4 _interfaceId) public view returns (bool) {
        uint256 success;
        uint256 result;

        (success, result) = noThrowCall(_contract, ERC165ID);
        if (success == 0 || result == 0) {
            return false;
        }

        (success, result) = noThrowCall(_contract, INVALID_ID);
        if (success == 0 || result != 0) {
            return false;
        }

        (success, result) = noThrowCall(_contract, _interfaceId);
        if (success == 1 && result == 1) {
            return true;
        }
        return false;
    }

    /// @notice Checks whether the hash is a ERC165 interface (ending with 28 zeroes) or not.
    /// @param _interfaceHash The hash to check.
    /// @return True if '_interfaceHash' is an ERC165 interface (ending with 28 zeroes), false otherwise.
    // 检查某个接口是否是ERC165接口，即标识符为4个字节的接口
    function isERC165Interface(bytes32 _interfaceHash) internal pure returns (bool) {
        return _interfaceHash & 0x00000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF == 0;
    }

    /// @dev Make a call on a contract without throwing if the function does not exist.
    // 判断某个合约是否实现了ERC165接口，如果没有也不抛出异常
    function noThrowCall(address _contract, bytes4 _interfaceId)
        internal view returns (uint256 success, uint256 result)
    {
        bytes4 erc165ID = ERC165ID;

        assembly {
            let x := mload(0x40)               // Find empty storage location using "free memory pointer"
            mstore(x, erc165ID)                // Place signature at beginning of empty storage
            mstore(add(x, 0x04), _interfaceId) // Place first argument directly next to signature

            success := staticcall(
                30000,                         // 30k gas
                _contract,                     // To addr
                x,                             // Inputs are stored at location x
                0x24,                          // Inputs are 36 (4 + 32) bytes long
                x,                             // Store output over input (saves space)
                0x20                           // Outputs are 32 bytes long
            )

            result := mload(x)                 // Load the result
        }
    }
}
```

合约的功能在注释中已给出，下面是用于部署该合约的原始交易：（可以部署在任何链）

```
0xf90a388085174876e800830c35008080b909e56080604052348015610010576000 80fd5b506109c
5806100206000396000f3fe6080604052348015610010576000 80fd5b50600436106100a557600035
7c0100000000000000000000000000000000000000000000000000000000900480 63a41e7d5111610
078578063a41e7d51146101d4578063aabbb8ca1461020a578063b705676514610236578063f712f3
e814610280576100a5565b806329965a1d146100aa5780633d584063146100e25780635df8122f146
1012457806365ba36c114610152575b600080fd5b6100e0600480360360608110156100c057600080
fd5b50600160a060020a0381358116916020810135916040909101351661 02b6565b005b610108600
480360360208110156100f857600080fd5b5035600160a060020a031661057 0565b60408051600160
a060020a039092168252519081900360200190f35b6100e0600480360360408110156101 3a5760008
0fd5b50600160a060020a038135811691602001351661 05bc565b6101c26004803603602081101561
016857600080fd5b810190602081018135640100000000811115610183576000 80fd5b82018360208
201111561019557600080fd5b803590602001918460018302840111640100000000831117156101b7
57600080fd5b509092509050610 6b3565b60408051918252519081900360200190f35b6100e060048
036036040811015610 1ea57600080fd5b508035600160a060020a03169060200135600160e060020a
0319166106ee565b610108600480360360408110156102205760 0080fd5b50600160a060020a03813
5169060200135610778565b61026c6004803603604081101561024c57600080fd5b508035600160a0
60020a03169060200135600160e060020a0319166107ef565b604080519115158252519081900360 2
00190f35b61026c6004803603604081101561029657600080fd5b508035600160a060020a03169060
200135600160e060020a0319166108aa565b6000600160a060020a038416156102cd57836102cf565
b335b9050336102db82610570565b600160a060020a0316146103395760408051 60e560020a62461b
cd02815260206004820152600f60248201527f4e6f7420746865206d616e616765720000000000000
00000000000000000000006044820152905190819003606401 90fd5b6103428361092a565b15610397
576040805160e560020a62461bcd02815260206004820152601a60248201527f4d757374206e6f742
0626520616e2045524331363520686173680000000000006044820152905190819003606401 90fd5b
600160a060020a0382161580159061 03b85750600160a060020a0382163314155b156104ff5760405
160200180807f455243313832305f4143434550545f4d414749430000000000000000000000008152
506014019050604051602081830303815290604052805190602001208260 0160a060020a031663249
cb3fa85846040518363ffffffff167c01000000000000000000000000000000000000000000000000
000000000281526004018083815260200182600160a060020a0316600160a060020a0316815260200
19250505060206040518083038186803b158015610 47e57600080fd5b505afa158015610492573d60
00803e3d6000fd5b505050506040513d602081101561 04a857600080fd5b5051146104ff576040805
160e560020a62461bcd028152602060048201819052602482015 27f446f6573206e6f7420696d706c
656d656e742074686520696e746572666163656044820152905190819003606401 90fd5b600160a06
0020a03818116600081815260208181526040808320888452909152808220805473ffffffffffffff
ffffffffffffffffffffffffff19169487169485179055518692917f93baa6efbd2244243bfee6ce4
cfdd1d04fc4c0e9a786abd3a41313bd352db15391a450505050565b600160a060020a038181166000
90815260016020526040812054909116151561059a5750806105b7565b50600160a060020a0380821
66000908152600160205260409020541 65b919050565b336105c683610570565b600160a060020a03
161461062457604080516 0e560020a62461bcd02815260206004820152600f60248201527f4e6f742
0746865206d616e6167657200000000000000000000000000000000006044820152905190819003 60
640190fd5b81600160a060020a031681600160a060020a031614610643578061064656 5b60005b600
160a060020a03838116600081815260016020526040808220805473ffffffffffffffffffffffffff
ffffffffffffff19169585169590951790945592519184169290917f605c2dbf762e5f7d60a546d42
e7205dcb1b011ebc62a61736a57c9089d3a43509190a35050565b600082826040516020018083838 0
828437808301925050509250505060405160208183030381529060405280519060200120905 05b929
15050565b6106f882826107ef565b61070357600061070556 5b815b600160a060020a039283166000
818152602081815260408083206001 60e060020a0319969096168084529582528083208054 73fffff
ffffffffffffffffffffffffffffffffff1916959097169490941790955590815260028452818120
928152919092522080546 0ff1916600117905556 5b6000806 00160a060020a038416156107905783 6
10792565b335b905061079d8361092a565b156107c357826107ad82826108aa565b6107b857600061
07ba565b815b925050506106e8565b600160a060020a03908116600090815260208181526040808 32
0868452909152902054169050929150505 65b600080806108 1d857f01ffc9a7000000000000000000
```

```
00000000000000000000000000000000000000061094c565b909250905081158061082d575080155b1
561083d576000925050506106e8565b61084f85600160e060020a031961094c565b90925090508115
806108605750801515 5b156108705760009250505061 06e8565b61087a858561094c565b909250905
060018214801561088f5750806001145b1561089f576001925050506106e8565b506000949350505050
50565b600160a060020a0382166000908152600260209081526040808320600160e060020a0319851
6845290915281205460ff161515610 8f2576108eb83836107ef565b90506106e8565b50600160a060
020a03808316600081815260208181526040808320600160e060020a031987168452909152902054909
0911614929150505 65b7bffffffffffffffffffffffffffffffffffffffffffffffffffffffff1615
90565b6040517f01ffc9a700000000000000000000000000000000000000000000000000000000808
2526004820183905260009182919060208160248189617530fa90519096909550935050505056fea1
65627a7a72305820377f4a2d4301ede9949f163f319021a6e9c687c292a5e2b2c4734c126b524e6c0
0291ba01820182018201820182018201820182018201820182018201820182018201820a01820182 0
182018201820182018201820182018201820182018201820182018201820
```

上面交易签名最后重复的1820是用某个私钥对交易进行ECDSA签名后输出的V和S组件，很容易就能看出这样的组件是人为赋予的，因此任何人都无法反推出部署该合约的私钥。

注册表合约部署

上述的注册表合约使用的是无密钥部署方法（也被称为Nick方法）进行部署，该方法的工作原理如下：

1.生成一个合约部署交易，合约中不能实现EIP-155（因为需要在任意链上部署），该交易必须有一个相对较高的gas price才能在任何链上部署，在这种需求下gas price为100Gwei。

2.签名的R和S组件由重复的1820数组组成。

3.恢复用于广播这笔交易的发送地址（实际上可以理解成通过ecrecover函数用R、S组件和部署交易的消息生成一个公钥地址，因为不关心私钥，也不需要对私钥进行验证，所以R、S组件可以随便填，因为是EIP1820提案就用1820填充）。

4.向第3步生成的公钥地址发送0.08个ETH用于部署注册表合约。

5.广播合约部署交易，完成合约部署。

相同的合约部署方法可以用于任何链，并且由于没人知道私钥因为所部署的合约是完全去中心化的。

通过上述方法部署的注册表合约地址为0x1820a4B7618BdE71Dce8cdc73aAB6C95905faD24，其部署者的地址为0xa990077c3205cbDf861e17Fa532eeB069cE9fF96

接口名称

任何接口名称都会使用keccak256进行哈希，并发送给getInterfaceImplementer()，为了方便，注册表合约提供了一个函数来计算链上的哈希值。

```
function interfaceHash(string _interfaceName) public pure returns(bytes32)
```

1 ERC提案

如果某个接口是已经批准的ERC提案的一部分，那么它必须被命名为**ERC###XXX**，其中##是ERC的编号，XXX是CamelCase中的接口名称，这个接口含义应该在制定的ERC中定义。

例子：

- `keccak256("ERC20Token")`
- `keccak256("ERC777Token")`
- `keccak256("ERC777TokensSender")`
- `keccak256("ERC777TokensRecipient")`

2 ERC-165兼容接口

EIP1820和EIP165提案兼容，任何最后28个字节为0的接口都应该被视为ERC-165接口，任何人都可以通过调用注册表合约中的下列函数明确地检查一个合约是否使用注册表实现了ERC165接口：

```
function implementsERC165Interface(address _contract, bytes4 _interfaceId) public view returns (bool)

function implementsERC165InterfaceNoCache(address _contract, bytes4 _interfaceId) public view returns (bool)
```

3 ERC-165缓存

可以通过手动缓存ERC-165接口查询地结果以节省接口查询的gas消耗。

如果一个合约改变了它的接口并依赖于ERC1820注册表和ERC165 cache缓存，则缓存必须被手动更新，缓存更新必须使用updateERC165Cache函数来完成：

```
function updateERC165Cache(address _contract, bytes4 _interfaceId) external
```

最后

提案的原文最后主要是关于EIP1820合约实现的一些解释，由于合约实现都相对比较简单，上面合约代码中的注释已经给出，因此这里也不过多赘述。

资料来源

ERC-1820: Pseudo-introspection Registry Contract (ethereum.org)

How to send Ether to 11,440 people | by Nick Johnson | Medium