以太坊改进提案EIP777（ERC777，2017-11-20，需要EIP-1820）是ERC20标准的升级版，该提案定义了代币合约的标准接口和行为。

ERC777标准定义了一种与代币合约交互的新方式，同时与ERC20保持向后兼容。在该标准中，操作者（operators）可以代表另一个地址（合约或常规账户）发送代币，同时因为在send/receive 加入了钩子函数（hook），代币持有者对代币有更多的控制权。

该标准利用了ERC1820的优势，能够判断接收代币地址的地址类型（合约或常规账户），并且判断合约是否兼容ERC777标准。。

提案内容

该提案对应的ERC777标准试图改进广泛使用的ERC20代币标准，该标准主要有以下优点：

1.使用和以太相同的发送方式来发送代笔，即使用**send(dest，value，data)**发送代币。

2.合约和常规账户都可以通过注册**tokensToSend**（hook）函数来控制和拒绝发送哪些token（通过**tokensToSend**里的**revert**实现拒绝token发送的操作）。

3.合约和常规账户都可以通过注册**tokensReceived**（hook）函数来控制和拒绝发送哪些token（通过**tokensReceived**里的**revert**实现拒绝接收token的操作）。

4.**tokensReceived**（hook）可以在一个交易里完成对代币的发送和授权，而ERC20标准中同样的操作需要发两次交易（approve/transferFrom）。

5.交易所等金融机构（operators）可以替用户操作他的代币，用于可以授权或者撤销这些机构对其代币的操作权。

6.每一个代币交易都包含**data**和**operatorData**字段，可以分别发送来自代币持有者或者操作员（operators）的数据。

7.可以通过为不包含**tokensReceived**函数的钱包部署一个包含**tokensReceived**实现的代理合约，使得该钱包兼容ERC777标准。

ERC777标准接口如下：

```
interface ERC777Token {
    function name() external view returns (string memory);
    function symbol() external view returns (string memory);
    function totalSupply() external view returns (uint256);
    function balanceOf(address holder) external view returns (uint256);
    function granularity() external view returns (uint256);

    function defaultOperators() external view returns (address[] memory);
    function isOperatorFor(
        address operator,
        address holder
    ) external view returns (bool);
    function authorizeOperator(address operator) external;
    function revokeOperator(address operator) external;

    function send(address to, uint256 amount, bytes calldata data) external;
```

```
    function operatorSend(
        address from,
        address to,
        uint256 amount,
        bytes calldata data,
        bytes calldata operatorData
    ) external;

    function burn(uint256 amount, bytes calldata data) external;
    function operatorBurn(
        address from,
        uint256 amount,
        bytes calldata data,
        bytes calldata operatorData
    ) external;

    event Sent(
        address indexed operator,
        address indexed from,
        address indexed to,
        uint256 amount,
        bytes data,
        bytes operatorData
    );
    event Minted(
        address indexed operator,
        address indexed to,
        uint256 amount,
        bytes data,
        bytes operatorData
    );
    event Burned(
        address indexed operator,
        address indexed from,
        uint256 amount,
        bytes data,
        bytes operatorData
    );
    event AuthorizedOperator(
        address indexed operator,
        address indexed holder
    );
    event RevokedOperator(address indexed operator, address indexed holder);
}
```

一个ERC777代币合约必须实现上述的接口，接口的实现必须符合下面提出的规范，并且代币合约必须通过ERC1820注册表合约注册**ERC777Toekn**接口。

如果代币合约有针对ERC777功能的开关，则每一次操作开关时都必须通过ERC1820合约对应地进行ERC777Toekn接口的注册或注销（将implementer置为0地址）。

当与代币合约交互式，所有代币数量和余额都必须被设置为无符号整数（uint256），这些value总是以10的18次方存储（decimals为18），并且当这些value展示给用户时则需要除以10的18次方。换句话说，假设合约中返回一个代币的余额为5x10的18次方，则用户看到的将是5个代币，如果用户希望发送3个代币，则代币合约收到的数量将会是3x10的18次方。

view 类型函数

下面的view类型的函数必须被实现

name函数

```
function name() external view returns (string memory)
```

返回代币的名称，比如“MyToken”。

symbol函数

```
function symbol() external view returns (string memory)
```

返回代币的符号，比如“MYT”

totalSupply函数

```
function totalSupply() external view returns (uint256)
```

返回代币的总供应量，代币的总供应量必须等于所有地址拥有的该代币的总和，必须等于所有铸造的代币数量减去所有销毁的代币数量的差。

balanceOf函数

```
function balanceOf(address holder) external view returns (uint256)
```

根据传入的地址返回该地址的代币余额，余额必须大于等于0。

granularity函数

```
function granularity() external view returns (uint256)
```

获得该代币的最小粒度，换句话说，粒度代表该代币所能发送，铸造或销毁的最小数量。粒度的设定必须遵循以下规则：

- 粒度的值必须在合约创建的时候设定
- 粒度的值不得变更

- 粒度的的值必须大于等于1
- 所有的代币余额，铸造、发送、销毁的数量必须是粒度的倍数
- 任何导致代币余额不是粒度的倍数的操作都必须被回滚

大部分代币应该是可以被划分的，也就是说granularity函数应该返回1，除非有充分的理由不允许对代币进行划分。

除了上述得view函数外，**operators**下定义的**defaultOperators**和**isOperatorFor**也是view函数。

为了兼容ERC20标准，代币的decimals必须始终是18位。对于纯ERC777代币，ERC20的decimals是可选的，与合约交互时不应该于依赖这个值（18是隐含值）。为了兼容ERC20代币，decimals函数要求必须返回18。(ERC20中decimals是可选的并且没有默认值，如果没有定义则会被认为是0)

操作者Operators

operators是一个允许代表代币持有者进行代币发送和销毁的地址，当一个地址变为某个代币持有者地址的operators时，必须触发**AuthorizedOperator**事件，同样的当一个operators被撤销时，必须触发**RevokedOperator**事件。一个代币持有者可以同时拥有多个operators。

一个代币可能定义了默认的operators，默认的operators拥有所有代币持有者的隐含权限，定义默认operators时不允许触发**AuthorizedOperator**事件，默认operators需要遵循以下规则：

- 代币合约在创建的时候必须定义默认operators
- 默认operators必须是不变量，设置后不允许修改
- 定义默认operators时不允许触发**AuthorizedOperator**事件
- 代币持有者必须拥有撤销默认operators的权限，除非默认operators本身就是代币持有者
- 代币持有者必须有重新授权之前被撤销的默认operators的权限
- 当默认operators被代币持有者明确地授权或撤销时，必须触发**AuthorizedOperator**或**RevokedOperator**事件
- 所有地operators需要遵循以下规则（包括默认operators）：
- 一个地址是他自己的operators，且这个身份不能撤销
- 如果一个地址是某个代币持有者的operators，则调用**isOperatorFor**函数必须返回true，反之返回false
- **AuthorizedOperator**事件和**RevokedOperator**事件中的value必须正确

一个operators可能被再次授权，必须触发**AuthorizedOperator**事件，撤销操作也一样。

AuthorizedOperator事件

```
event AuthorizedOperator(address indexed operator, address indexed holder)
```

表示holder授权了他的operator，该事件不能在operator授权操作之外被触发。

RevokedOperator事件

```
event RevokedOperator(address indexed operator, address indexed holder)
```

表示holder撤销了他的operator，该事件不能在operator撤销操作之外被触发。

必须实现**defaultOperators**、**authorizeOperator**、**revokeOperator**和**isOperatorFor**函数来管理operators，当然代币合约可以实现更多的函数用于管理operators。

defaultOperators函数

```
function defaultOperators() external view returns (address[] memory)
```

返回代币合约定义的默认operators地址列表，如果没有定义任何默认operators，则返回空列表。

authorizeOperator函数

```
function authorizeOperator(address operator) external
```

为msg.sender设置一个第三方地址为operator，替msg.sender管理代币。msg.sender始终是自己的operators且不可被撤销，当msg.sender调用上述函数为自己授权时，必须**revert**函数。

revokeOperator函数

```
function revokeOperator(address operator) external
```

为msg.sender撤销某个地址operator的身份。msg.sender始终是自己的operators且不可被撤销，当msg.sender调用上述函数撤销自己的operator身份时，必须**revert**函数。

isOperatorFor函数

```
function isOperatorFor(
    address operator,
    address holder
) external view returns (bool)
```

表明某个地址是否是某个代币持有者的operator。

当一个操作者替代币持有者向接收者发送**amount**数量的代币且携带**data**和**opertorData**时，代币合约需要遵循以下规则：

- 任何操作者可以向任何地址发送代币（0地址除外）
- 代币持有者的余额需要减去amount，接收者的余额需要增加amount
- 代币持有者的余额需要大于等于amount
- 代币合约需要触发Sent事件
- 操作者可以在opertorData字段中附带信息

- 如果代币持有者在ERC1820注册表合约注册了**ERC777TokensSender**，则代币合约必须调用代币持有者的hook函数tokensToSend
- 如果代币持有者在ERC1820注册表合约注册了**ERC777TokensRecipient**，则代币合约必须调用代币持有者的hook函数tokensRecipient
- data和opertorData在整个代币发送过程中不可变，因为其中的数据必须被用于调用hook函数和触发Sent事件

在发送代币过程中如果遭遇下列情况，则代币合约必须revert此次交易：

- 操作者地址为未被代币持有者授权的地址
- 完成代币发送后代币持有者和接收者的余额数量不为代币粒度的整数倍
- 接收者是个合约，但是该合约没有在ERC1820注册表合约注册并实现**ERC777TokensRecipient**接口
- 代币持有者或者接收者的地址为0地址
- 代币发送后持有者和接收者的余额变为负数
- 代币持有者的tokensToSend函数中的hook执行了revert
- 接收者的tokensRecipient函数中的hook执行了revert

当代币合约将代币从多个持有者发送到多个接收者时，需要遵循以下规则：

- 所有的持有者和接收者需要遵循前面提到的所有规则
- 所有持有者余额减少的数量和接收者余额增加的数量要等于代币的总发送量
- 每个从持有者到接收者的代币发送行为都必须触发一个Sent事件，并且所有事件的代币发送数量总和要等于代币实际发送量。

收取手续费按照发给多个接收者的标准来收取，代币发送行为可能是链式的，后续行为均需要遵循上面提到的规则。发送0个代币的交易也必须被视为正常交易。

实现函数功能时需要遵循以下规则：

- 代币合约在更新状态前必须调用**tokensToSend**
- 代币合约必须要更新状态之后调用**tokesReceoved**

data中包含了由代币持有者提供的信息，与发送以太交易中的data字段类似，tokensToSend()和tokensReceived()函数可能利用这部分信息来决定是否拒绝该交易。**operatorData**与**data**类似，但是这部分信息由操作者提供，operatorData主要用于日记记录和一些特殊情况（例如记录付款证明，支票号码、签名等），在大多数情况下接收者会忽略operatorData，或者仅对operatorData做记录。

Sent事件

```
event Sent(
    address indexed operator,   //  操作者地址
    address indexed from,   //代币持有者地址
    address indexed to, //接收者地址
    uint256 amount, //发送数量（wei）
    bytes data, //持有者数据
    bytes operatorData  //操作者数据
)
```

指明amount数量的代币被从from地址发送往to地址，该交易的操作者地址为operator。

Sent函数

```
function send(address to, uint256 amount, bytes calldata data) external
```

发送amount数量的代币到to地址，operator和持有者必须都是msg.sender。

operatorSend事件

```
function operatorSend(
    address from,   //持有者地址
    address to, //接收者地址
    uint256 amount, //代币数量
    bytes calldata data,
    bytes calldata operatorData
) external
```

从from地址发送amount数量的代表到to地址，若操作者地址没有被from地址授权，则该交易必须被revert。持有者和操作者可以为同一个地址，在这种情况下此次调用则被视为附带了持有者指定的operatorData的send函数（send函数无法附带operatorData数据）。

send和operatorSend函数必须实现，代币合约可以实现其它发送代币的函数。

铸造代币

ERC777没有明确指定铸造代币函数的接口，因为对于不同类型的代币来说可能需要不同的铸造方式，因此若限定铸造代币的函数接口则有可能限制ERC777的应用范围。

尽管ERC777没有明确铸造代笔函数接口，但是规定了以下规则：

- 铸造代币的接收地址不能为0地址
- 铸造代币时代币的总供应量必须对应增加
- 铸造代币时0地址的代币余额不能被减少（相当于不能从0地址拿取代币）
- 铸造代币时代币接收者的余额数量必须对应增加
- 铸造代币时代币合约必须触发Minted事件，且铸造数量必须对应。
- 如果接收者通过ERC1820注册并实现了ERC777TokensRecipient接口，则铸造代币时代币合约必须调用tokensReceived函数
- 铸造代币的过程中data和operatorData的值不允许改变，因为这些值会被用于调用tokensReceived函数和触发Minted事件

铸造代币过程中如果发生下列情况则函数必须revert：

- 铸造后接收者的余额不为代币粒度的整数倍
- 接收者是合约地址，但是该地址没有在ERC1820合约注册实现ERC777TokensRecipient接口

- 接收者为0地址
- 触发了tokensReceived函数中的revert判定条件

虽然ERC777中铸造代币时不允许触发Sent事件，但是如果代币合约兼容了ERC20标准，则应该按照ERC20标准触发from参数为0地址的Transfer事件。

代币合约可能在一次铸造代币的行为内将所铸造的代币发送给多个接收者，这种情况下代币合约要遵循以下规则：

- 需要遵循前面提到的规则
- 所有接收者的余额增加的总量要等与总铸造数量
- 需要为每一个接收者触发一次对应数量的Minted事件
- 所有Minted事件中铸造数量的总和必须等于总铸造数量

铸造0个代币的行为是被允许的，且应被视为常规铸造行为。铸造代币的过程中data字段由代币合约或操作者提供，而operatorData的值由操作者提供，且tokensReceived()会根据operatorData的内容决定是否拒绝该交易。

Minted事件

```
event Minted(
    address indexed operator,    //铸币操作者地址
    address indexed to, //接收者地址
    uint256 amount, //铸币数量
    bytes data, //由接收者提供的数据
    bytes operatorData  //由操作者提供的数据
)
```

指明amount数量的代币由operator铸造且发送往to地址，该事件不能被非铸币行为触发。

销毁代币

ERC777标准为销毁代币行为定义了两个函数接口（**burn**和**operatorBurn**），这些函数便于在钱包和dapp中集成销毁过程。代币合约可以出于任何原因阻止持有者销毁代币，也可以定义其它函数来销毁代币。

销毁代币时需要遵循以下规则：

- 任何持有者地址都可以销毁代币（0地址除外）
- 销毁代币时代币总供应量必须对应减少
- 销毁代币时0地址的余额不能增加（相当于不能通过向0地址发送代币来进行销毁）
- 销毁代币时持有者余额必须对应减少
- 必须触发指定value的Burned事件
- 如果持有者通过ERC1820合约注册并实现了ERC777TokensSender接口，则销毁代币时代币合约必须调用tokensToSend函数
- 在销毁代币过程中operatorData的值不允许被改变，因为这部分的值会被用于调用tokensToSend函数和触发Burned事件

销毁代币过程中若发生以下情况则必须revert：

- 操作者地址没有经过持有者授权
- 销毁代币后持有者的余额不为代币粒度的整数倍
- 持有者余额小于代币销毁数量
- 持有者地址为0地址
- 触发了tokensToSend函数中的revert条件

销毁代币过程中不能触发Sent事件，若ERC777代币合约兼容ERC20标准，则应该触发to为0地址的Transfer事件（ERC20中没有定义销毁代币的概念，但是上述行为普遍被ERC20代币项目接受）

代币合约可能一次销毁多个持有者的代币，这种情况下需要遵循以下标准：

- 需要遵循前面提到的销毁代币的规则
- 所有持有者代币的减少量必须等于总销毁量
- 需要为每一个持有者触发对应销毁数量的Burned事件
- 所有Burned事件的销毁数量需要等于总销毁数量

销毁0个代币的行为是被允许的，并且应该被视为常规行为，data字段由持有者提供，该字段会被tokensToSend()或tokensReceived()函数用于判断是否拒绝该销毁交易，operatorData由操作者提供，与data类似。

Burned事件

```
event Burned(
    address indexed operator,   //操作者地址
    address indexed from,   //持有者地址
    uint256 amount, //销毁数量
    bytes data, //由持有者提供的数据
    bytes operatorData  //由操作者提供的数据
)
```

指明operator销毁了from地址的amount数量的代币，该事件不能被非销毁代币行为触发。

Burn函数

```
function burn(uint256 amount, bytes calldata data) external
```

销毁msg.sender地址中amount数量的代币，操作者和代币持有者必须都为msg.sender。

operatorBurn函数

```
function operatorBurn(
    address from,   //持有者地址
    uint256 amount, //销毁数量
    bytes calldata data,    //由持有者提供的数据
    bytes calldata operatorData //由操作者提供的数据
) e
```

销毁from地址中amount数量的代币，若操作者地址没有得到from地址的认证，则revert该函数。

操作者可以通过operatorData传递任何信息，操作者和持有者可以是同一个地址，这样就相当于持有者调用了一个可以携带指定operatorData消息的burn函数（burn函数不能携带数据）。

burn和operatorBurn必须被代币合约实现，但是代币合约也可以实现其它用于销毁代币的函数。

ERC777TokensRecipient和tokensReceived（Hook）

tokensReceived函数可以通知给定接收者代币的增加量，任何希望获得该通知的地址（包括合约和账户）可以通过在ERC1820合约实现ERC777TokensRecipient接口以达到目的，接口实现如下：

```
interface ERC777TokensRecipient {
    function tokensReceived(
        address operator,
        address from,
        address to,
        uint256 amount,
        bytes calldata data,
        bytes calldata operatorData
    ) external;
}
```

如果代币接收者是合约地址，且没有注册实现ERC777TokensRecipient接口，则：

- 当tokensReceived被代币铸造和代币发送函数调用时代币合约必须revert
- 当tokensReceived被ERC20标准的transfer和transferFrom函数调用时则继续完成交易

账户地址可以通过合约地址实现ERC777TokensRecipient接口，而合约地址必须通过自身或另一个合约地址实现该接口。

tokensReceived函数

```
function tokensReceived(
    address operator,   //代币增加行为的操作者
    address from,   //发送代币的地址（或0地址，表示代币铸造）
    address to, //接收代币的地址
    uint256 amount, //接收数量
    bytes calldata data,    //持有者数据
    bytes calldata operatorData //操作者数据
) external
```

通知接收者amount数量的代币被operator从from发送到to地址中，该函数不能被铸造代币、发送代币和ERC20 transfer以外的行为调用。

- 调用tokensReceived函数时需要遵循以下规则：
- 必须在所有的代币发送和铸造行为中被调用
- 必须要代币状态更新后被调用（比如余额增加）
- operator必须是触发发送或铸造代币行为的地址
- from必须是持有者地址，若是代币铸造交易则from必须是0地址
- to必须是代币接收者地址
- amount必须是代币接收或铸造的数量
- 如果有的话，data必须包含提供给代币发送或铸造行为的信息，operatorData也相同
- 持有者可以通过revert来阻止代币接收或铸造交易

多个持有者可能使用同样的ERC777TokensRecipient接口实现，一个地址同一时间最多只能为所有的ERC777代币实现一个ERC777TokensRecipient接口，因此该接口必须预期被用于不同的代币合约。代币合约的msg.sender应该是代币合约地址。

hook（tokensReceived）优先级要高于ERC20标准，因此当调用transfer和transferFrom时hook函数也要被调用（如果已经注册的话）。当hook被transfer调用时，operator和from的值必须相同，当被transferFrom调用时，operator必须是调用transferFrom的地址。

Gas消耗相关

Dapps和钱包应该估算用eth_estimateGas提前估算代币发送、铸造、销毁时所需消耗的gas，以避免交易过程中gas耗尽。

Logo

Image					
Color	beige	white	light grey	dark grey	black
Hex	`#C99D66`	`#FFFFFF`	`#EBEFF0`	`#3C3C3D`	`#000000`

可以使用、修改或调整logo以促进有效的ERC777代币和符合ERC777标准的技术实现，例如钱包和Dapp。ERC777代币合约logo可以基于上面的logo进行设计，并且上面的logo不能被用于不符合ERC777标准的技术（比如代币）。该logo可以在/assets/eip-777/logo文件夹中被找到。

基本原理

该提案的目的主要是为了解决ERC20标准的一些短板并且向后兼容ERC20标准，同时避免EIP223提案存在的问题。

ERC777定义了代币从铸造，发送到销毁的整个生命周期，明确定义生命周期对于代币的一致性和准确性来说非常重要。部分ERC20代币的totalSupply返回的值和实际供应量之间存在差异，根源就是ERC20标准中没有明确定义创建和销毁代币的过程。

Hooks

在ERC20标准中人们需要两个步骤才能将代币安全地转入某个目标合约并且不会让代币被永久地锁定在目标合约内，第一个步骤是用户调用approve函数为目标合约授予代币使用权限，第二个步骤是目标合约调用transferFrom函数将代币转移到自身。部分代币持有者会分不清transfer转账模式和approve/transferFrom转账模式的区别，并直接用transfer向目标合约转移代币，从而导致所转移代币被永久锁定在目标合约中并无法取出。

而hooks的出现则允许用户直接通过transfer向目标合约转移代币，tokensReceived hooks能够保障用户转移的代币不会被锁定在目标合约中。

tokensReceived也允许代币持有者拒绝接收某些代笔，这为代币持有者提供了更大的控制权，他们可以根据Data、operatorData或其它参数接收或拒绝某些传入的代币。

兼容性

该提案不会导致向后的不兼容性，并且向后兼容ERC20标准。

该提案不使用transfer和transferFrom而是使用send和operatorSend来转移代币以避免主体的混淆。

该提案允许实现ERC20标准中的transfer、transferFrom、approve和allowance功能，以使得代币与ERC20标准完全兼容，若该提案实现decimals()，则结果必须返回18。

ERC777和ERC20标准可以同时实现，但是ERC777中强制要求实现以下函数：name，symbol，balanceOf和totalSupply。两个标准的状态修改函数是解耦的，可以相互独立运行，但是ERC20标准的函数应该仅限于只能从旧合约中调用。

如果某个ERC777代币实现了ERC20标准，则该代币合约地址必须在ERC1820注册表合约中注册**ERC20Token**接口，若代币合约设置了ERC20标准的开关，则该合约必须同步在注册表合约中注册或注销**ERC20Token**接口。

同时实现ERC777和ERC20的合约与传统ERC20代币合约的不同之处在于tokensToSend和tokensReceived这两个hook的优先级要高于ERC20标准，即即使使用ERC20标准中的transfer和transferFrom，代币合约也必须通过ERC1820的检查，判断from和to是否分别实现了tokensToSend和tokensReceived，如果实现了任意一个hook，则必须调用它。需要注意的是，若调用ERC20转账时发现目标合约没有实现hook函数，转账也应该被目标合约接收（即使存在代币被锁定的风险）。

下面是同时实现ERC777和ERC20标准的合约中面对不同情况两种标准的不同行为：

ERC1820	`to` address	ERC777 Sending And Minting	ERC20 `transfer` / `transferFrom`
`ERC777TokensRecipient` registered	regular address	MUST call `tokensReceived`	
	contract		
`ERC777TokensRecipient` not registered	regular address	continue	
	contract	MUST `revert`	SHOULD continue[1]

在代币发送、铸造和销毁期间，必须触发响应的Sent、Minted和Burned事件。此外，如果代币合约声明它通过ERC1820实现ERC20Token接口，则代币合约应该发出一个代表铸造和销毁的Transfer事件，并且必须发出一个代表代币发扫那个的Transfer事件（ERC20标准中指定的）。在ERC20的transfer或transferFrom事件中，必须触发有效的Sent事件。

因此，对于代币的任何行为，都有可能会触发两个时间：ERC20的Transfer和ERC777的Sent、Minted或Burned。第三方开发人员必须注意不要将这两个事件视为单独的动作，如果应用程序将代币视为ERC20代币，则必须考虑Transfer事件；如果应用程序将代币视为ERC777代币，则必须只考虑Sent、Minted和Burned事件。

资料来源

ERC-777: Token Standard (ethereum.org)