1 概念

以太坊改进提案EIP20定义了ERC20代币标准，为代币定义了标准接口。

标准接口提供了转移代币的基本功能，并允许代币获得批准，以便另一个链上第三方可以使用他们。

2 ERC20标准API和规范

下面列出的接口和规范使用的是Solidity 0.4.17（或更高版本）的语法，接口使用者必须处理返回的false，不能假设永远不会返回false。

2.1 name

返回代币的名称，如“MyToken”。

为可选项，可以用来提高可用性，但是接口和其它合约不能假定它必须存在。

```
function name() public view returns (string)
```

2.2 symbol

返回代币的符号，如“HIX”。

为可选项，可以用来提高可用性，但是接口和其它合约不能假定它必须存在。

```
function symbol() public view returns (string)
```

2.3 decimals

返回代币的精确度，如“8”代表代币精确到小数点后8位。

为可选项，可以用来提供可用性，但是接口和其它合约不能假定它必须存在。

```
function decimals() public view returns (uint8)
```

2.4 totalSupply

返回代币的总供应量。

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

2.5 balanceOf

返回给定的owner地址的代币余额。

```
function balanceOf(address _owner) public view returns (uint256 balance)
```

2.6 transfer

将value数量的代币转移到地址to，并且必须触发Transfer事件。如果消息调用者的账户余额没有足够的代币，则该函数需要抛出异常。

注意，即使转移代币的数量为0也必须被视为正常调用并触发Transfer事件。

```
function transfer(address _to, uint256 _value) public returns (bool success)
```

2.7 transferForm

将value数量的代币从地址from转移到地址to，并且必须触发Transfer事件。transferFrom方法主要用于代币提现工作流，允许合约代表用户转移代币。例如这可以用于允许合约代表用户转移代币。除非from账户通过某种机制授权消息发送者，否则函数应该抛出异常。

注意，即使转移代币的数量为0也必须被视为正常调用并触发Transfer事件。

```
function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
```

2.8 approve

允许spender多次从用户的账户中提款，最多为value数额。如果再次调用此函数，它会用新的value覆盖当前的允许值allowance。

为了防止该函数受到抢跑攻击(在后面会说明)，用户在为某个spender授权金额时，应当先将他的allowance设为0，然后再设置成你的目标值。

```
function approve(address _spender, uint256 _value) public returns (bool success)
```

2.9 allowance

返回spender还允许从owner账户中提款的数量。

```
function allowance(address _owner, address _spender) public view returns (uint256 remaining)
```

3 ERC20事件(Events)

3.1 Transfer

必须在代币发生转移时触发，包括0 value转移。创建新代币的代币合约应该在创建代币时触发from地址为0x0的Transfer事件。

```
event Transfer(address indexed _from, address indexed _to, uint256 _value)
```

3.2 Approval

任何approve方法的成功调用都应该触发该事件。

```
event Approval(address indexed _owner, address indexed _spender, uint256 _value)
```

4 approve可能存在的抢跑攻击

假定这样一个场景：A通过approval方法给B授权了N个代币的使用权，则此时B可以用transferForm方法将N个代币从A地址转出来使用，若过了一段时间A改变主意，决定变更B可以支配的代币额度为M，则A需要再次调用approval方法给B授权M个代币。

假设在这个过程中B一直在监听A的行为，当他发现A第二次调用approval方法改变额度时，他立刻以更高的手续费调用transferForm函数提前将N个代币转走，则当A的第二次approval方法生效后B还能继续提取M个代币。

在上述场景中，A只想让B使用N或M个代币，但是最终B通过交易抢跑可以提款N+M个代币。抢跑攻击成功的关键是每次approval方法生效时value都会直接覆盖掉allowance而不去判断allowance有无被使用。

5 ERC20代币样例(SHIBA币)

```
/**
 *Submitted for verification at Etherscan.io on 2021-02-26
*/

/**
 *Submitted for verification at Etherscan.io on 2019-08-02
*/

// File: contracts\open-zeppelin-contracts\token\ERC20\IERC20.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see `ERC20Detailed`.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
```

```
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a `Transfer` event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through `transferFrom`. This is
     * zero by default.
     *
     * This value changes when `approve` or `transferFrom` are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * > Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an `Approval` event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a `Transfer` event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

```

```
    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to `approve`. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256
value);
}

// File: contracts\open-zeppelin-contracts\math\SafeMath.sol

pragma solidity ^0.5.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
```

```
124      * overflow (when the result is negative).
125      *
126      * Counterpart to Solidity's `-` operator.
127      *
128      * Requirements:
129      * - Subtraction cannot overflow.
130      */
131     function sub(uint256 a, uint256 b) internal pure returns (uint256) {
132         require(b <= a, "SafeMath: subtraction overflow");
133         uint256 c = a - b;
134 
135         return c;
136     }
137 
138     /**
139      * @dev Returns the multiplication of two unsigned integers, reverting on
140      * overflow.
141      *
142      * Counterpart to Solidity's `*` operator.
143      *
144      * Requirements:
145      * - Multiplication cannot overflow.
146      */
147     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
148         // Gas optimization: this is cheaper than requiring 'a' not being
zero, but the
149         // benefit is lost if 'b' is also tested.
150         // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
151         if (a == 0) {
152             return 0;
153         }
154 
155         uint256 c = a * b;
156         require(c / a == b, "SafeMath: multiplication overflow");
157 
158         return c;
159     }
160 
161     /**
162      * @dev Returns the integer division of two unsigned integers. Reverts on
163      * division by zero. The result is rounded towards zero.
164      *
165      * Counterpart to Solidity's `/` operator. Note: this function uses a
166      * `revert` opcode (which leaves remaining gas untouched) while Solidity
167      * uses an invalid opcode to revert (consuming all remaining gas).
168      *
169      * Requirements:
170      * - The divisor cannot be zero.
171      */
172     function div(uint256 a, uint256 b) internal pure returns (uint256) {
173         // Solidity only automatically asserts when dividing by 0
174         require(b > 0, "SafeMath: division by zero");
```

```
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0, "SafeMath: modulo by zero");
        return a % b;
    }
}

// File: contracts\open-zeppelin-contracts\token\ERC20\ERC20.sol

pragma solidity ^0.5.0;



/**
 * @dev Implementation of the `IERC20` interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using `_mint`.
 * For a generic mechanism see `ERC20Mintable`.
 *
 * *For a detailed writeup see our guide [How to implement supply
 * mechanisms](https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226).*
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an `Approval` event is emitted on calls to `transferFrom`.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
```

```
 * Finally, the non-standard `decreaseAllowance` and `increaseAllowance`
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See `IERC20.approve`.
 */
contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    /**
     * @dev See `IERC20.totalSupply`.
     */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See `IERC20.balanceOf`.
     */
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See `IERC20.transfer`.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public returns (bool)
{
        _transfer(msg.sender, recipient, amount);
        return true;
    }

    /**
     * @dev See `IERC20.allowance`.
     */
    function allowance(address owner, address spender) public view returns
(uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See `IERC20.approve`.
     *
```

```
273         * Requirements:
274         *
275         * - `spender` cannot be the zero address.
276         */
277        function approve(address spender, uint256 value) public returns (bool) {
278            _approve(msg.sender, spender, value);
279            return true;
280        }
281
282        /**
283         * @dev See `IERC20.transferFrom`.
284         *
285         * Emits an `Approval` event indicating the updated allowance. This is not
286         * required by the EIP. See the note at the beginning of `ERC20`;
287         *
288         * Requirements:
289         * - `sender` and `recipient` cannot be the zero address.
290         * - `sender` must have a balance of at least `value`.
291         * - the caller must have allowance for `sender`'s tokens of at least
292         * `amount`.
293         */
294        function transferFrom(address sender, address recipient, uint256 amount)
    public returns (bool) {
295            _transfer(sender, recipient, amount);
296            _approve(sender, msg.sender, _allowances[sender]
    [msg.sender].sub(amount));
297            return true;
298        }
299
300        /**
301         * @dev Atomically increases the allowance granted to `spender` by the
    caller.
302         *
303         * This is an alternative to `approve` that can be used as a mitigation
    for
304         * problems described in `IERC20.approve`.
305         *
306         * Emits an `Approval` event indicating the updated allowance.
307         *
308         * Requirements:
309         *
310         * - `spender` cannot be the zero address.
311         */
312        function increaseAllowance(address spender, uint256 addedValue) public
    returns (bool) {
313            _approve(msg.sender, spender, _allowances[msg.sender]
    [spender].add(addedValue));
314            return true;
315        }
316
317        /**
```

```
318      * @dev Atomically decreases the allowance granted to `spender` by the
caller.
319      *
320      * This is an alternative to `approve` that can be used as a mitigation
for
321      * problems described in `IERC20.approve`.
322      *
323      * Emits an `Approval` event indicating the updated allowance.
324      *
325      * Requirements:
326      *
327      * - `spender` cannot be the zero address.
328      * - `spender` must have allowance for the caller of at least
329      * `subtractedValue`.
330      */
331     function decreaseAllowance(address spender, uint256 subtractedValue)
public returns (bool) {
332         _approve(msg.sender, spender, _allowances[msg.sender]
[spender].sub(subtractedValue));
333         return true;
334     }
335
336     /**
337      * @dev Moves tokens `amount` from `sender` to `recipient`.
338      *
339      * This is internal function is equivalent to `transfer`, and can be used
to
340      * e.g. implement automatic token fees, slashing mechanisms, etc.
341      *
342      * Emits a `Transfer` event.
343      *
344      * Requirements:
345      *
346      * - `sender` cannot be the zero address.
347      * - `recipient` cannot be the zero address.
348      * - `sender` must have a balance of at least `amount`.
349      */
350     function _transfer(address sender, address recipient, uint256 amount)
internal {
351         require(sender != address(0), "ERC20: transfer from the zero
address");
352         require(recipient != address(0), "ERC20: transfer to the zero
address");
353
354         _balances[sender] = _balances[sender].sub(amount);
355         _balances[recipient] = _balances[recipient].add(amount);
356         emit Transfer(sender, recipient, amount);
357     }
358
359     /** @dev Creates `amount` tokens and assigns them to `account`, increasing
360      * the total supply.
361      *
```

```
362      * Emits a `Transfer` event with `from` set to the zero address.
363      *
364      * Requirements
365      *
366      * - `to` cannot be the zero address.
367      */
368     function _mint(address account, uint256 amount) internal {
369         require(account != address(0), "ERC20: mint to the zero address");
370 
371         _totalSupply = _totalSupply.add(amount);
372         _balances[account] = _balances[account].add(amount);
373         emit Transfer(address(0), account, amount);
374     }
375 
376      /**
377      * @dev Destroys `amount` tokens from `account`, reducing the
378      * total supply.
379      *
380      * Emits a `Transfer` event with `to` set to the zero address.
381      *
382      * Requirements
383      *
384      * - `account` cannot be the zero address.
385      * - `account` must have at least `amount` tokens.
386      */
387     function _burn(address account, uint256 value) internal {
388         require(account != address(0), "ERC20: burn from the zero address");
389 
390         _totalSupply = _totalSupply.sub(value);
391         _balances[account] = _balances[account].sub(value);
392         emit Transfer(account, address(0), value);
393     }
394 
395     /**
396      * @dev Sets `amount` as the allowance of `spender` over the `owner`s
    tokens.
397      *
398      * This is internal function is equivalent to `approve`, and can be used
    to
399      * e.g. set automatic allowances for certain subsystems, etc.
400      *
401      * Emits an `Approval` event.
402      *
403      * Requirements:
404      *
405      * - `owner` cannot be the zero address.
406      * - `spender` cannot be the zero address.
407      */
408     function _approve(address owner, address spender, uint256 value) internal
    {
409         require(owner != address(0), "ERC20: approve from the zero address");
410         require(spender != address(0), "ERC20: approve to the zero address");
```

```
411
412         _allowances[owner][spender] = value;
413         emit Approval(owner, spender, value);
414     }
415
416     /**
417      * @dev Destoys `amount` tokens from `account`.`amount` is then deducted
418      * from the caller's allowance.
419      *
420      * See `_burn` and `_approve`.
421      */
422     function _burnFrom(address account, uint256 amount) internal {
423         _burn(account, amount);
424         _approve(account, msg.sender, _allowances[account]
[msg.sender].sub(amount));
425     }
426 }
427
428 // File: contracts\ERC20\TokenMintERC20Token.sol
429
430 pragma solidity ^0.5.0;
431
432
433 /**
434  * @title TokenMintERC20Token
435  * @author TokenMint (visit https://tokenmint.io)
436  *
437  * @dev Standard ERC20 token with burning and optional functions implemented.
438  * For full specification of ERC-20 standard see:
439  * https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
440  */
441 contract TokenMintERC20Token is ERC20 {
442
443     string private _name;
444     string private _symbol;
445     uint8 private _decimals;
446
447     /**
448      * @dev Constructor.
449      * @param name name of the token
450      * @param symbol symbol of the token, 3-4 chars is recommended
451      * @param decimals number of decimal places of one token unit, 18 is
widely used
452      * @param totalSupply total supply of tokens in lowest units (depending on
decimals)
453      * @param tokenOwnerAddress address that gets 100% of token supply
454      */
455     constructor(string memory name, string memory symbol, uint8 decimals,
uint256 totalSupply, address payable feeReceiver, address tokenOwnerAddress)
public payable {
456       _name = name;
457       _symbol = symbol;
```

```
458         _decimals = decimals;
459
460         // set tokenOwnerAddress as owner of all tokens
461         _mint(tokenOwnerAddress, totalSupply);
462
463         // pay the service fee for contract deployment
464         feeReceiver.transfer(msg.value);
465       }
466
467       /**
468        * @dev Burns a specific amount of tokens.
469        * @param value The amount of lowest token units to be burned.
470        */
471       function burn(uint256 value) public {
472         _burn(msg.sender, value);
473       }
474
475       // optional functions from ERC20 stardard
476
477       /**
478        * @return the name of the token.
479        */
480       function name() public view returns (string memory) {
481         return _name;
482       }
483
484       /**
485        * @return the symbol of the token.
486        */
487       function symbol() public view returns (string memory) {
488         return _symbol;
489       }
490
491       /**
492        * @return the number of decimals of the token.
493        */
494       function decimals() public view returns (uint8) {
495         return _decimals;
496       }
497   }
```

6 资料来源

EIP20以太坊改进提案

approve接口可能存在的抢跑攻击

$0.00001085 | SHIBA INU (SHIB) Token Tracker | Etherscan