

1 概念

以太坊改进提案EIP20（2015-11-19）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。

```
1 | 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。

```
1 | 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。

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

2.4 totalSupply

返回代币的总供应量。

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

2.5 balanceOf

返回给定的owner地址的代币余额。

```
1 | 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。

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

2.7 transferFrom

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。

```
1 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。

```
1 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。

```
1 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。

```
1 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。

```
1 | 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transferFrom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transferFrom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)

```
1 | /**
2 |  *Submitted for verification at Etherscan.io on 2021-02-26
3 |  */
4 |
5 | /**
6 |  *Submitted for verification at Etherscan.io on 2019-08-02
7 |  */
8 |
9 | // File: contracts\open-zeppelin-contracts\token\ERC20\IERC20.sol
10 |
11 | pragma solidity ^0.5.0;
12 |
13 | /**
14 |  * @dev Interface of the ERC20 standard as defined in the EIP. Does not
15 |  * include
16 |  * the optional functions; to access them see `ERC20Detailed`.
17 |  */
18 | interface IERC20 {
19 |     /**
20 |      * @dev Returns the amount of tokens in existence.
21 |      */
22 |     function totalSupply() external view returns (uint256);
23 |
24 |     /**
25 |      * @dev Returns the amount of tokens owned by `account`.
```

```

25     */
26     function balanceOf(address account) external view returns (uint256);
27
28     /**
29     * @dev Moves `amount` tokens from the caller's account to `recipient`.
30     *
31     * Returns a boolean value indicating whether the operation succeeded.
32     *
33     * Emits a `Transfer` event.
34     */
35     function transfer(address recipient, uint256 amount) external returns
36     (bool);
37
38     /**
39     * @dev Returns the remaining number of tokens that `spender` will be
40     * allowed to spend on behalf of `owner` through `transferFrom`. This is
41     * zero by default.
42     *
43     * This value changes when `approve` or `transferFrom` are called.
44     */
45     function allowance(address owner, address spender) external view returns
46     (uint256);
47
48     /**
49     * @dev Sets `amount` as the allowance of `spender` over the caller's
50     tokens.
51     *
52     * Returns a boolean value indicating whether the operation succeeded.
53     *
54     * > Beware that changing an allowance with this method brings the risk
55     * that someone may use both the old and the new allowance by unfortunate
56     * transaction ordering. One possible solution to mitigate this race
57     * condition is to first reduce the spender's allowance to 0 and set the
58     * desired value afterwards:
59     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
60     *
61     * Emits an `Approval` event.
62     */
63     function approve(address spender, uint256 amount) external returns (bool);
64
65     /**
66     * @dev Moves `amount` tokens from `sender` to `recipient` using the
67     * allowance mechanism. `amount` is then deducted from the caller's
68     * allowance.
69     *
70     * Returns a boolean value indicating whether the operation succeeded.
71     *
72     * Emits a `Transfer` event.
73     */
74     function transferFrom(address sender, address recipient, uint256 amount)
75     external returns (bool);

```

```

73     /**
74      * @dev Emitted when `value` tokens are moved from one account (`from`) to
75      * another (`to`).
76      *
77      * Note that `value` may be zero.
78      */
79     event Transfer(address indexed from, address indexed to, uint256 value);
80
81     /**
82      * @dev Emitted when the allowance of a `spender` for an `owner` is set by
83      * a call to `approve`. `value` is the new allowance.
84      */
85     event Approval(address indexed owner, address indexed spender, uint256
value);
86 }
87
88 // File: contracts\open-zeppelin-contracts\math\SafeMath.sol
89
90 pragma solidity ^0.5.0;
91
92 /**
93  * @dev Wrappers over Solidity's arithmetic operations with added overflow
94  * checks.
95  *
96  * Arithmetic operations in Solidity wrap on overflow. This can easily result
97  * in bugs, because programmers usually assume that an overflow raises an
98  * error, which is the standard behavior in high level programming languages.
99  * `SafeMath` restores this intuition by reverting the transaction when an
100  * operation overflows.
101  *
102  * Using this library instead of the unchecked operations eliminates an entire
103  * class of bugs, so it's recommended to use it always.
104  */
105 library SafeMath {
106     /**
107      * @dev Returns the addition of two unsigned integers, reverting on
108      * overflow.
109      *
110      * Counterpart to Solidity's `+` operator.
111      *
112      * Requirements:
113      * - Addition cannot overflow.
114      */
115     function add(uint256 a, uint256 b) internal pure returns (uint256) {
116         uint256 c = a + b;
117         require(c >= a, "SafeMath: addition overflow");
118
119         return c;
120     }
121
122     /**
123      * @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");

```

```

175         uint256 c = a / b;
176         // assert(a == b * c + a % b); // There is no case in which this
doesn't hold
177
178         return c;
179     }
180
181     /**
182     * @dev Returns the remainder of dividing two unsigned integers. (unsigned
integer modulo),
183     * Reverts when dividing by zero.
184     *
185     * Counterpart to Solidity's `%` operator. This function uses a `revert`
186     * opcode (which leaves remaining gas untouched) while Solidity uses an
187     * invalid opcode to revert (consuming all remaining gas).
188     *
189     * Requirements:
190     * - The divisor cannot be zero.
191     */
192     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
193         require(b != 0, "SafeMath: modulo by zero");
194         return a % b;
195     }
196 }
197
198 // File: contracts\open-zeppelin-contracts\token\ERC20\ERC20.sol
199
200 pragma solidity ^0.5.0;
201
202
203
204 /**
205  * @dev Implementation of the `IERC20` interface.
206  *
207  * This implementation is agnostic to the way tokens are created. This means
208  * that a supply mechanism has to be added in a derived contract using
`_mint`.
209  * For a generic mechanism see `ERC20Mintable`.
210  *
211  * *For a detailed writeup see our guide [How to implement supply
212  * mechanisms](https://forum.zeppelin.solutions/t/how-to-implement-erc20-
supply-mechanisms/226).*
213  *
214  * We have followed general OpenZeppelin guidelines: functions revert instead
215  * of returning `false` on failure. This behavior is nonetheless conventional
216  * and does not conflict with the expectations of ERC20 applications.
217  *
218  * Additionally, an `Approval` event is emitted on calls to `transferFrom`.
219  * This allows applications to reconstruct the allowance for all accounts just
220  * by listening to said events. Other implementations of the EIP may not emit
221  * these events, as it isn't required by the specification.
222  *

```

```

223     * Finally, the non-standard `decreaseAllowance` and `increaseAllowance`
224     * functions have been added to mitigate the well-known issues around setting
225     * allowances. See `IERC20.approve`.
226     */
227     contract ERC20 is IERC20 {
228         using SafeMath for uint256;
229
230         mapping (address => uint256) private _balances;
231
232         mapping (address => mapping (address => uint256)) private _allowances;
233
234         uint256 private _totalSupply;
235
236         /**
237          * @dev See `IERC20.totalSupply`.
238          */
239         function totalSupply() public view returns (uint256) {
240             return _totalSupply;
241         }
242
243         /**
244          * @dev See `IERC20.balanceOf`.
245          */
246         function balanceOf(address account) public view returns (uint256) {
247             return _balances[account];
248         }
249
250         /**
251          * @dev See `IERC20.transfer`.
252          *
253          * Requirements:
254          *
255          * - `recipient` cannot be the zero address.
256          * - the caller must have a balance of at least `amount`.
257          */
258         function transfer(address recipient, uint256 amount) public returns (bool)
259         {
260             _transfer(msg.sender, recipient, amount);
261             return true;
262         }
263
264         /**
265          * @dev See `IERC20.allowance`.
266          */
267         function allowance(address owner, address spender) public view returns
268         (uint256) {
269             return _allowances[owner][spender];
270         }
271
272         /**
273          * @dev See `IERC20.approve`.
274          */

```



```

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)
295     public returns (bool) {
296         _transfer(sender, recipient, amount);
297         _approve(sender, msg.sender, _allowances[sender]
298 [msg.sender].sub(amount));
299         return true;
300     }
301
302     /**
303     * @dev Atomically increases the allowance granted to `spender` by the
304 caller.
305     *
306     * This is an alternative to `approve` that can be used as a mitigation
307 for
308     * problems described in `IERC20.approve`.
309     *
310     * Emits an `Approval` event indicating the updated allowance.
311     *
312     * Requirements:
313     * - `spender` cannot be the zero address.
314     */
315     function increaseAllowance(address spender, uint256 addedValue) public
316 returns (bool) {
317         _approve(msg.sender, spender, _allowances[msg.sender]
318 [spender].add(addedValue));
319         return true;
320     }
321
322     /**

```

```

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
397     tokens.
398     *
399     * This is internal function is equivalent to `approve`, and can be used
400     to
401     * e.g. set automatic allowances for certain subsystems, etc.
402     *
403     * Emits an `Approval` event.
404     *
405     * Requirements:
406     *
407     * - `owner` cannot be the zero address.
408     * - `spender` cannot be the zero address.
409     */
410     function _approve(address owner, address spender, uint256 value) internal
{
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

```

```

411         _allowances[owner][spender] = value;
412         emit Approval(owner, spender, value);
413     }
414
415     /**
416     * @dev Destroys `amount` tokens from `account`. `amount` is then deducted
417     * from the caller's allowance.
418     *
419     * See `_burn` and `_approve`.
420     */
421     function _burnFrom(address account, uint256 amount) internal {
422         _burn(account, amount);
423         _approve(account, msg.sender, _allowances[account]
424 [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
452 widely used
453     * @param totalSupply total supply of tokens in lowest units (depending on
454 decimals)
455     * @param tokenOwnerAddress address that gets 100% of token supply
456     */
457     constructor(string memory name, string memory symbol, uint8 decimals,
458 uint256 totalSupply, address payable feeReceiver, address tokenOwnerAddress)
459 public payable {
460         _name = name;
461         _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 standard
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](#)