### IERC20

EIP中定义的ERC20标准的接口

Interface of the ERC20 standard as defined in the EIP

EIP:[Ethereum Improvement Proposals](https://eips.ethereum.org/)

#### 函数

[totalSupply()](https://docs.openzeppelin.com/contracts/3.x/api/token/erc20" \l "IERC20-totalSupply--)

返回存在的代币总量

Returns the amount of tokens in existence.

function totalSupply() external view returns (uint256);

[balanceOf(account)](https://docs.openzeppelin.com/contracts/3.x/api/token/erc20" \l "IERC20-balanceOf-address-)

返回账户拥有的代币数量

Returns the amount of tokens owned by account.

function balanceOf(address account) external view returns (uint256);

[transfer(recipient, amount)](https://docs.openzeppelin.com/contracts/3.x/api/token/erc20" \l "IERC20-transfer-address-uint256-)

将amount数量的代币从调用者的账户移动到recipient账户，即转账

返回一个布尔值，指示操作是否成功

触发一个Transfer事件

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);

[allowance(owner, spender)](https://docs.openzeppelin.com/contracts/3.x/api/token/erc20" \l "IERC20-allowance-address-address-)

返回spender将被允许通过transferFrom代表owner消费的剩余代币数量，即授权额度，默认为0

当调用approve或transferFrom时，该值会发生变化

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);

[approve(spender, amount)](https://docs.openzeppelin.com/contracts/3.x/api/token/erc20" \l "IERC20-approve-address-uint256-)

设置amount作为调用者给spender账户的额度，即授权

返回一个布尔值，指示操作是否成功

触发一个Approval事件

Sets amount as the allowance of spender over the caller’s tokens.

Returns a boolean value indicating whether the operation succeeded.

Emits an Approval event.

function approve(address spender, uint256 amount) external returns(bool);

[transferFrom(sender, recipient, amount)](https://docs.openzeppelin.com/contracts/3.x/api/token/erc20" \l "IERC20-transferFrom-address-address-uint256-)

通过授权机制，从sender账户向recipient账户转账amount数量代币，转账的部分会从调用者的授权额度中扣除

返回一个布尔值，指示操作是否成功

触发一个Transfer事件

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);

#### 事件

[Transfer(from, to, value)](https://docs.openzeppelin.com/contracts/3.x/api/token/erc20" \l "IERC20-Transfer-address-address-uint256-)

当value数量的代币从账户from转账到另一个账户to时触发

该值可能为0

Emitted when value tokens are moved from one account (from) to another (to).

Note that value may be zero.

event Transfer(address from, address to, uint256 value);

[Approval(owner, spender, value)](https://docs.openzeppelin.com/contracts/3.x/api/token/erc20" \l "IERC20-Approval-address-address-uint256-)

当value数量的代币从账户owner授权给另一个账户spender时触发

value是新的授权额度

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 owner, address spender, uint256 value);

#### OpenZeppelin IERC20.sol

// SPDX-License-Identifier: MIT

// OpenZeppelin Contracts (last updated v4.6.0) (token/ERC20/IERC20.sol)

pragma solidity ^0.8.0;

/\*\*

\* @dev Interface of the ERC20 standard as defined in the EIP.

\*/

interface IERC20 {

/\*\*

\* @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);

/\*\*

\* @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 `to`.

\*

\* Returns a boolean value indicating whether the operation succeeded.

\*

\* Emits a {Transfer} event.

\*/

function transfer(address to, 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.

\*

\* IMPORTANT: 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 `from` to `to` 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 from,

address to,

uint256 amount

) external returns (bool);

}

### ERC20

IERC20接口的实现

#### 状态变量

// 账户余额

mapping(address => uint256) public override balanceOf;

// 授权额度

mapping(address => mapping(address => uint256)) public override allowance;

// 代币总供给

uint256 public override totalSupply;

// 代币名称

string public name;

// 代号

string public symbol;

// 小数位数

uint8 public decimals = 18;

#### 函数

// 构造函数，初始化代币名称，代号

constructor(string memory name\_, string memory symbol\_){

name = name\_;

symbol = symbol\_;

}

// 转账

function transfer(address recipient, uint amount) external override returns (bool) {

balanceOf[msg.sender] -= amount;

balanceOf[recipient] += amount;

emit Transfer(msg.sender, recipient, amount);

return true;

}

// 授权

function approve(address spender, uint amount) external override returns (bool) {

allowance[msg.sender][spender] = amount;

emit Approval(msg.sender, spender, amount);

return true;

}

// 授权转账

function transferFrom(

address sender,

address recipient,

uint amount

) external override returns (bool) {

allowance[sender][msg.sender] -= amount;

balanceOf[sender] -= amount;

balanceOf[recipient] += amount;

emit Transfer(sender, recipient, amount);

return true;

}

// 铸造代币

function mint(uint amount) external {

balanceOf[msg.sender] += amount;

totalSupply += amount;

emit Transfer(address(0), msg.sender, amount);

}

// 销毁代币

function burn(uint amount) external {

balanceOf[msg.sender] -= amount;

totalSupply -= amount;

emit Transfer(msg.sender, address(0), amount);

}

#### OpenZeppelin ERC20.sol

// SPDX-License-Identifier: MIT

// OpenZeppelin Contracts (last updated v4.7.0) (token/ERC20/ERC20.sol)

pragma solidity ^0.8.0;

import "./IERC20.sol";

import "./extensions/IERC20Metadata.sol";

import "../../utils/Context.sol";

/\*\*

\* @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 {ERC20PresetMinterPauser}.

\*

\* TIP: For a detailed writeup see our guide

\* https://forum.openzeppelin.com/t/how-to-implement-erc20-supply-mechanisms/226[How

\* to implement supply mechanisms].

\*

\* We have followed general OpenZeppelin Contracts guidelines: functions revert

\* instead 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 Context, IERC20, IERC20Metadata {

mapping(address => uint256) private \_balances;

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

uint256 private \_totalSupply;

string private \_name;

string private \_symbol;

/\*\*

\* @dev Sets the values for {name} and {symbol}.

\*

\* The default value of {decimals} is 18. To select a different value for

\* {decimals} you should overload it.

\*

\* All two of these values are immutable: they can only be set once during

\* construction.

\*/

constructor(string memory name\_, string memory symbol\_) {

\_name = name\_;

\_symbol = symbol\_;

}

/\*\*

\* @dev Returns the name of the token.

\*/

function name() public view virtual override returns (string memory) {

return \_name;

}

/\*\*

\* @dev Returns the symbol of the token, usually a shorter version of the

\* name.

\*/

function symbol() public view virtual override returns (string memory) {

return \_symbol;

}

/\*\*

\* @dev Returns the number of decimals used to get its user representation.

\* For example, if `decimals` equals `2`, a balance of `505` tokens should

\* be displayed to a user as `5.05` (`505 / 10 \*\* 2`).

\*

\* Tokens usually opt for a value of 18, imitating the relationship between

\* Ether and Wei. This is the value {ERC20} uses, unless this function is

\* overridden;

\*

\* NOTE: This information is only used for \_display\_ purposes: it in

\* no way affects any of the arithmetic of the contract, including

\* {IERC20-balanceOf} and {IERC20-transfer}.

\*/

function decimals() public view virtual override returns (uint8) {

return 18;

}

/\*\*

\* @dev See {IERC20-totalSupply}.

\*/

function totalSupply() public view virtual override returns (uint256) {

return \_totalSupply;

}

/\*\*

\* @dev See {IERC20-balanceOf}.

\*/

function balanceOf(address account) public view virtual override returns (uint256) {

return \_balances[account];

}

/\*\*

\* @dev See {IERC20-transfer}.

\*

\* Requirements:

\*

\* - `to` cannot be the zero address.

\* - the caller must have a balance of at least `amount`.

\*/

function transfer(address to, uint256 amount) public virtual override returns (bool) {

address owner = \_msgSender();

\_transfer(owner, to, amount);

return true;

}

/\*\*

\* @dev See {IERC20-allowance}.

\*/

function allowance(address owner, address spender) public view virtual override returns (uint256) {

return \_allowances[owner][spender];

}

/\*\*

\* @dev See {IERC20-approve}.

\*

\* NOTE: If `amount` is the maximum `uint256`, the allowance is not updated on

\* `transferFrom`. This is semantically equivalent to an infinite approval.

\*

\* Requirements:

\*

\* - `spender` cannot be the zero address.

\*/

function approve(address spender, uint256 amount) public virtual override returns (bool) {

address owner = \_msgSender();

\_approve(owner, spender, amount);

return true;

}

/\*\*

\* @dev See {IERC20-transferFrom}.

\*

\* Emits an {Approval} event indicating the updated allowance. This is not

\* required by the EIP. See the note at the beginning of {ERC20}.

\*

\* NOTE: Does not update the allowance if the current allowance

\* is the maximum `uint256`.

\*

\* Requirements:

\*

\* - `from` and `to` cannot be the zero address.

\* - `from` must have a balance of at least `amount`.

\* - the caller must have allowance for ``from``'s tokens of at least

\* `amount`.

\*/

function transferFrom(

address from,

address to,

uint256 amount

) public virtual override returns (bool) {

address spender = \_msgSender();

\_spendAllowance(from, spender, amount);

\_transfer(from, to, amount);

return true;

}

/\*\*

\* @dev Atomically increases the allowance granted to `spender` by the caller.

\*

\* This is an alternative to {approve} that can be used as a mitigation for

\* problems described in {IERC20-approve}.

\*

\* Emits an {Approval} event indicating the updated allowance.

\*

\* Requirements:

\*

\* - `spender` cannot be the zero address.

\*/

function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {

address owner = \_msgSender();

\_approve(owner, spender, allowance(owner, spender) + addedValue);

return true;

}

/\*\*

\* @dev Atomically decreases the allowance granted to `spender` by the caller.

\*

\* This is an alternative to {approve} that can be used as a mitigation for

\* problems described in {IERC20-approve}.

\*

\* Emits an {Approval} event indicating the updated allowance.

\*

\* Requirements:

\*

\* - `spender` cannot be the zero address.

\* - `spender` must have allowance for the caller of at least

\* `subtractedValue`.

\*/

function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {

address owner = \_msgSender();

uint256 currentAllowance = allowance(owner, spender);

require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");

unchecked {

\_approve(owner, spender, currentAllowance - subtractedValue);

}

return true;

}

/\*\*

\* @dev Moves `amount` of tokens from `from` to `to`.

\*

\* This internal function is equivalent to {transfer}, and can be used to

\* e.g. implement automatic token fees, slashing mechanisms, etc.

\*

\* Emits a {Transfer} event.

\*

\* Requirements:

\*

\* - `from` cannot be the zero address.

\* - `to` cannot be the zero address.

\* - `from` must have a balance of at least `amount`.

\*/

function \_transfer(

address from,

address to,

uint256 amount

) internal virtual {

require(from != address(0), "ERC20: transfer from the zero address");

require(to != address(0), "ERC20: transfer to the zero address");

\_beforeTokenTransfer(from, to, amount);

uint256 fromBalance = \_balances[from];

require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");

unchecked {

\_balances[from] = fromBalance - amount;

// Overflow not possible: the sum of all balances is capped by totalSupply, and the sum is preserved by

// decrementing then incrementing.

\_balances[to] += amount;

}

emit Transfer(from, to, amount);

\_afterTokenTransfer(from, to, amount);

}

/\*\* @dev Creates `amount` tokens and assigns them to `account`, increasing

\* the total supply.

\*

\* Emits a {Transfer} event with `from` set to the zero address.

\*

\* Requirements:

\*

\* - `account` cannot be the zero address.

\*/

function \_mint(address account, uint256 amount) internal virtual {

require(account != address(0), "ERC20: mint to the zero address");

\_beforeTokenTransfer(address(0), account, amount);

\_totalSupply += amount;

unchecked {

// Overflow not possible: balance + amount is at most totalSupply + amount, which is checked above.

\_balances[account] += amount;

}

emit Transfer(address(0), account, amount);

\_afterTokenTransfer(address(0), account, amount);

}

/\*\*

\* @dev Destroys `amount` tokens from `account`, reducing the

\* total supply.

\*

\* Emits a {Transfer} event with `to` set to the zero address.

\*

\* Requirements:

\*

\* - `account` cannot be the zero address.

\* - `account` must have at least `amount` tokens.

\*/

function \_burn(address account, uint256 amount) internal virtual {

require(account != address(0), "ERC20: burn from the zero address");

\_beforeTokenTransfer(account, address(0), amount);

uint256 accountBalance = \_balances[account];

require(accountBalance >= amount, "ERC20: burn amount exceeds balance");

unchecked {

\_balances[account] = accountBalance - amount;

// Overflow not possible: amount <= accountBalance <= totalSupply.

\_totalSupply -= amount;

}

emit Transfer(account, address(0), amount);

\_afterTokenTransfer(account, address(0), amount);

}

/\*\*

\* @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.

\*

\* This internal function is equivalent to `approve`, and can be used to

\* e.g. set automatic allowances for certain subsystems, etc.

\*

\* Emits an {Approval} event.

\*

\* Requirements:

\*

\* - `owner` cannot be the zero address.

\* - `spender` cannot be the zero address.

\*/

function \_approve(

address owner,

address spender,

uint256 amount

) internal virtual {

require(owner != address(0), "ERC20: approve from the zero address");

require(spender != address(0), "ERC20: approve to the zero address");

\_allowances[owner][spender] = amount;

emit Approval(owner, spender, amount);

}

/\*\*

\* @dev Updates `owner` s allowance for `spender` based on spent `amount`.

\*

\* Does not update the allowance amount in case of infinite allowance.

\* Revert if not enough allowance is available.

\*

\* Might emit an {Approval} event.

\*/

function \_spendAllowance(

address owner,

address spender,

uint256 amount

) internal virtual {

uint256 currentAllowance = allowance(owner, spender);

if (currentAllowance != type(uint256).max) {

require(currentAllowance >= amount, "ERC20: insufficient allowance");

unchecked {

\_approve(owner, spender, currentAllowance - amount);

}

}

}

/\*\*

\* @dev Hook that is called before any transfer of tokens. This includes

\* minting and burning.

\*

\* Calling conditions:

\*

\* - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens

\* will be transferred to `to`.

\* - when `from` is zero, `amount` tokens will be minted for `to`.

\* - when `to` is zero, `amount` of ``from``'s tokens will be burned.

\* - `from` and `to` are never both zero.

\*

\* To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].

\*/

function \_beforeTokenTransfer(

address from,

address to,

uint256 amount

) internal virtual {}

/\*\*

\* @dev Hook that is called after any transfer of tokens. This includes

\* minting and burning.

\*

\* Calling conditions:

\*

\* - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens

\* has been transferred to `to`.

\* - when `from` is zero, `amount` tokens have been minted for `to`.

\* - when `to` is zero, `amount` of ``from``'s tokens have been burned.

\* - `from` and `to` are never both zero.

\*

\* To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].

\*/

function \_afterTokenTransfer(

address from,

address to,

uint256 amount

) internal virtual {}

}

https://docs.openzeppelin.com/contracts/3.x/api/token/erc20

https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/token/ERC20