合約代碼
pragma solidity ^0.5.0; import "./IERC20.sol"; import "./SafeMath.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 {ERC20Mintable}. * * TIP: For a detailed writeup see our guide * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How * to implement supply mechanisms]. * * 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}. * * Requirements: * * - `spender` cannot be the zero address. */ function approve(address spender, uint256 value) public returns (bool) { _approve(msg.sender, spender, value); 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}; * * Requirements: * - `sender` and `recipient` cannot be the zero address. * - `sender` must have a balance of at least `value`. * - the caller must have allowance for `sender`'s tokens of at least * `amount`. */ function transferFrom(address sender, address recipient, uint256 amount) public returns (bool) { _transfer(sender, recipient, amount); _approve(sender, msg.sender, _allowances[sender] [msg.sender].sub(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 returns (bool) { _approve(msg.sender, spender, _allowances[msg.sender] [spender].add(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 returns (bool) { _approve(msg.sender, spender, _allowances[msg.sender]

[spender].sub(subtractedValue)); return true; } /** * @dev Moves tokens `amount` from `sender` to `recipient`. * * This is 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: * * - `sender` cannot be the zero address. * - `recipient` cannot be the zero address. * - `sender` must have a balance of at least `amount`. */ function _transfer(address sender, address recipient, uint256 amount) internal { require(sender != address(0), "ERC20: transfer from the zero address"); require(recipient != address(0), "ERC20: transfer to the zero address"); _balances[sender] = _balances[sender].sub(amount); _balances[recipient] = _balances[recipient].add(amount); emit

Transfer(sender, recipient, 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 * * - `to` cannot be the zero address. */ function _mint(address account, uint256 amount) internal { require(account != address(0), "ERC20: mint to the zero address"); _totalSupply = _totalSupply.add(amount); _balances[account] = _balances[account].add(amount); emit Transfer(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 value) internal { require(account != address(0), "ERC20: burn from the zero address"); _totalSupply = _totalSupply.sub(value); _balances[account] = _balances[account].sub(value); emit Transfer(account, address(0), value); } /** * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens. * * This is 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 value) internal { require(owner != address(0), "ERC20: approve from the zero address"); require(spender != address(0), "ERC20: approve to the zero address");

_allowances[owner][spender] = value; emit Approval(owner, spender, value); } /** * @dev Destoys `amount` tokens from `account`.`amount` is then deducted * from the caller's allowance. * * See {_burn} and {_approve}. */ function _burnFrom(address account, uint256 amount) internal { _burn(account, amount); _approve(account, msg.sender, _allowances[account] [msg.sender].sub(amount)); } }