

Работа с токенами Ethereum

Основы разработки на Solidity



Оглавление

Введение	3
Словарь терминов	3
Модификаторы	4
Что такое токены	5
Примеры того, что можно сделать с помощью токенов:	6
Стандарты ERC	8
Основные типы стандартов токенов	9
ERC-20	10
Кратко	11
Спецификация ERC-20	13
Пример простого ERC-20 токена	13
Перевод	15
Десятичные знаки	17
Allowance	17
Расширения ERC-20	18
ERC-223	18
ERC-777	19
ERC-827	19
ERC-844	20
ERC-865	21
Стейблкоины	22
Взаимозаменяемые и невзаимозаменяемые токены	22
Что такое взаимозаменяемый токен	22
Варианты использования взаимозаменяемых токенов	23
Что такое невзаимозаменяемый токен	24
Варианты использования NFT	24
ERC-721	25
Спецификация ERC-721	25
Функции ERC-721	25
Пример кода	26
Задание 1: Создание и деплой собственного ERC-20 токена	28
Цель:	28
Задачи:	28
Задание 2: Разработка смарт-контракта для голосования на основе токенов	28
Цель:	28
Задачи:	28
Подсказка:	28
Использованная литература	29

Введение

Добрый день, друзья! Рады приветствовать вас на четвертой лекции курса, посвященной работе с токенами Ethereum.

На этом уроке Вы узнаете:

- Что такое токены Ethereum и почему они стали фундаментальной частью блокчейн-проектов.
- Основные стандарты токенов, включая ERC-20 и ERC-721, их отличия и применение.
- Почему понимание этих стандартов критично для разработчиков и инвесторов в сфере криптовалют.
- Реальные примеры реализации токенов по стандартам ERC-20 и ERC-721, подробный разбор их функций и механизмов.

Изучение токенов Ethereum не только теоретически важно, но и крайне полезно с практической точки зрения. Понимание этих концепций открывает двери к созданию собственных криптовалютных активов, участию в разработке и анализе ICO, и даже к возможности разработки уникальных цифровых коллекционных предметов.

Эта лекция поможет вам не только усвоить теорию, но и понять, как эти знания могут быть применены в реальных проектах – от создания цифровых активов до разработки уникальных решений в блокчейн-индустрии.

Словарь терминов

OpenZeppelin – это библиотека безопасных смарт-контрактов, разработанных для блокчейн-платформы Ethereum. Библиотека предоставляет стандартизированные, проверенные и защищенные компоненты (например, токены ERC20 и ERC721), которые разработчики могут использовать для создания своих собственных децентрализованных приложений. OpenZeppelin помогает уменьшить риск ошибок и уязвимостей в смарт-контрактах, предоставляя тщательно протестированные и аудированные решения.

Краудсейл (crowdsale) – продажа криптоактивов неограниченному кругу лиц, обычно применяется для сбора средств. Этот метод часто используется стартапами в блокчейн-сфере для привлечения капитала и финансирования своих проектов. Участники краудсейла покупают токены в надежде, что их стоимость вырастет по мере развития проекта.

Стейблкоин (stablecoin) – криптовалюта с минимальной волатильностью (колебаниями цены). Стейблкоины обычно привязаны к стабильным активам, таким как валюты (например, доллар США, евро) или драгоценные металлы (например, золото), чтобы поддерживать постоянную стоимость токена. Это делает стейблкоины идеальным инструментом для транзакций, сбережений или хеджирования от волатильности рынка криптовалют.

Эксплорер (explorer) – веб-сайт с детальной информацией о блокчейне (блоках, транзакциях, аккаунтах и т. д.), например, Etherscan - эксплорер сети Ethereum.

Цифровые акции – токены могут представлять собой долю владения каким-либо активом, при этом дивиденды между держателями будут начисляться программным способом автоматически.

Голосование – токены могут использоваться для голосования с разным весом участников, когда количество токенов означает вес голоса.

Модификаторы

Модификаторы являются ключевым элементом языка Solidity и используются для изменения поведения функций или контрактов. Они представляют собой специальные функции, которые позволяют добавить дополнительную функциональность к другим функциям.

Модификаторы могут быть применены к функциям и могут проверять определенные условия перед выполнением целевой функции. Если условия, заданные модификатором, выполняются, то функция будет вызвана, в противном случае, выполнение функции будет прервано с ошибкой.

Модификаторы могут быть полезны для множества задач, таких как проверка прав доступа, проверка условий, обработка ошибок, логирование и другие. Они помогают повысить безопасность и эффективность кода, а также упростить его чтение и понимание.

Знак подчёркивания – это специальный символ, используемый внутри модификаторов. Он используется, чтобы связать модификатор с кодом основной функции и указать, где должен быть выполнен код основной функции.

Пример :

```
1 // Указываем версию Solidity
2 pragma solidity ^0.8.0;
3 •
4 // Объявляем контракт
5 contract MyContract {
6     // Адрес владельца контракта
7     address public owner;
8     •
9     // Конструктор контракта, устанавливающий владельца
10    constructor() {
11        owner = msg.sender;
12    }
13    •
14    // Модификатор для проверки, что вызывающий является владельцем
15    modifier onlyOwner() {
16        require(msg.sender == owner, "Вы не владелец!");
17        _;
18    }
19    •
20    // Пример функции, использующей модификатор
21    function someSensitiveOperation() public onlyOwner {
22        // Логика, доступная только владельцу контракта
23    }
24 }
25
```

В этом примере:

- owner хранит адрес владельца контракта.
- Конструктор constructor() устанавливает владельца контракта в момент его создания.
- Модификатор onlyOwner() использует require() для проверки, что msg.sender (адрес, инициирующий текущую функцию) является owner. Если условие не выполняется, транзакция отклоняется с ошибкой "Вы не владелец!".

- Функция `someSensitiveOperation()` использует модификатор `onlyOwner`, делая её выполнение доступным только владельцу контракта.

Что такое токены

Токены в Ethereum представляют собой цифровые активы, которые можно создавать на этом блокчейне. Токены содержат информацию о владении определенным количеством активов или представляют некоторое право или привилегию в сети. Токены можно использовать для представления различных активов, таких как криптовалюта, акции, недвижимость или даже физические предметы.

Создание токена может быть полезно для запуска собственной криптовалюты, проведения ICO (Initial Coin Offering) или для других целей, связанных с децентрализованными приложениями на блокчейне Ethereum.

Одним из преимуществ токенов Ethereum является гибкость их реализации, т. е. возможность программно настроить их на автоматическое выполнение определённых действий или соответствие определённым условиям.

Примеры того, что можно сделать с помощью токенов:

Гезелевские деньги – деньги с демереджем, т. е. особым налогом на хранение. Такие деньги автоматически "сгорают" с заранее заданной скоростью. По замыслу создателей это увеличивает скорость их обращения, т. к. держатели стремятся избавиться от денег. Гезелевские деньги, названные в честь экономиста Сильвио Гезеля, представляют собой концепцию денег с "демереджем" — штрафом за хранение. Это предназначено для стимулирования расходов и инвестиций, предотвращения хамстерства и борьбы с дефляцией, заставляя людей быстрее тратить свои деньги, поскольку их стоимость со временем уменьшается.

Реализация Гезелевских денег в Solidity.

Реализация такой системы в смарт-контракте на Ethereum с использованием языка программирования Solidity может включать следующие шаги:

Определение переменных:

В контракте нужно определить переменные, которые будут отслеживать балансы пользователей и время последнего обновления баланса для каждого пользователя.

Механизм демереджа:

Реализация функции, которая автоматически уменьшает балансы в зависимости от времени, прошедшего с момента последнего обновления баланса пользователя.

Обновление балансов:

При любых транзакциях, таких как переводы или платежи, баланс пользователя должен быть автоматически скорректирован с учетом демереджа перед выполнением транзакции.

Функция для обновления баланса:

Пользователи могут вызывать функцию для обновления своих балансов, чтобы применить демередж до совершения транзакции.

Пример такой функции:

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ≥0.6.12 <0.9.0;
3 •
4 •
5 contract GesellMoney {
6     mapping(address ⇒ uint256) public balance;
7     mapping(address ⇒ uint256) private lastUpdated;
8 •
9     uint256 public demurrageRate = 1; // Демередж в процентах за год
10    uint256 public demurrageInterval = 365 days; // Интервал
        обновления
11 •
12    event Transfer(address indexed from, address indexed to, uint256
        value);
13 •
14    function updateBalance(address user) public {
15        uint256 heldDuration = block.timestamp - lastUpdated[user];
16        if (heldDuration ≥ demurrageInterval) {
17            uint256 years = heldDuration / demurrageInterval;
18            uint256 decayAmount = balance[user] * demurrageRate / 100
        * years;
19            balance[user] -= decayAmount;
20            lastUpdated[user] = block.timestamp;
21        }
22    }
23 •
24    function transfer(address to, uint256 amount) public {
25        updateBalance(msg.sender);
26        updateBalance(to);
27        require(balance[msg.sender] ≥ amount, "Insufficient
        balance");
28        balance[msg.sender] -= amount;
29        balance[to] += amount;
30        emit Transfer(msg.sender, to, amount);
31    }
32 •
33    function deposit() public payable {
34        updateBalance(msg.sender);
35        balance[msg.sender] += msg.value;
36    }
37 •
38    function withdraw(uint256 amount) public {
39        updateBalance(msg.sender);
40        require(balance[msg.sender] ≥ amount, "Insufficient
        balance");
41        balance[msg.sender] -= amount;
42        payable(msg.sender).transfer(amount);
43    }
44 }
45
46

```


Стандарты ERC

ERC означает «Ethereum Request for Comments». По сути, это набор технической документации, включающей руководящие принципы и стандарты для создания и управления смарт-контрактами в блокчейне Ethereum. Стандарты ERC устанавливают определенные правила и рекомендации, которым необходимо следовать разработчикам, чтобы обеспечить совместимость между приложениями и сетью.

Хотя Ethereum позволяет разработчикам создавать токены, которые не соответствуют никаким стандартам крипто-токенов, таким токенам не хватает функциональности и совместимости с другими функциями сети. Поэтому для реализации таких функций, как обработка транзакций, взаимозаменяемость/невзаимозаменяемость, безопасность и т. д., важно соблюдать стандарты ERC.

Любой может создать токен ERC. Однако для этого нужно предоставить документ с предлагаемыми функциями и процессами для улучшения сети блокчейна Ethereum.

Как только разработчик направляет свое предложение, оно оценивается и изучается основными разработчиками Ethereum. Если сообщество считает это важным дополнением к экосистеме блокчейна, предложение принимается, дорабатывается и реализуется.

Как только этот процесс будет завершен, исходный документ станет стандартом ERC, который другие разработчики смогут использовать для создания своих собственных токенов.

Основные типы стандартов токенов

Стандарты токенов Ethereum, такие как ERC-20, ERC-721, и другие, играют критически важную роль в экосистеме Ethereum, обеспечивая стандартизацию, безопасность и взаимодействие между различными децентрализованными

приложениями (dApps) и контрактами. Стандартизация токенов позволяет токенам легко взаимодействовать с различными приложениями и сервисами, такими как кошельки, обменные платформы и финансовые инструменты.

ERC-20

ERC20 был одним из первых протоколов токенов Ethereum, предложенных в ноябре 2015 года Фабианом Фогельстеллером, который реализует стандарты для взаимозаменяемых токенов.

ERC-621

Стандарт ERC-621 играет ключевую роль в увеличении и уменьшении общего объема обращения. Этот стандарт расширяет функциональность токенов ERC20, позволяя их сжигать или обменивать на другой актив.

ERC-721

Стандарт ERC-721 является основой невзаимозаменяемых токенов. В отличие от ERC-20, токены ERC-721 отличаются редкостью и уникальностью. Это означает, что ни один токен ERC-721 не совпадает с другим токеном ERC-721. Каждый NFT состоит из уникального адреса и метаданных (имени или изображения), хранящихся в сети Ethereum, что упрощает их проверку и передачу. Назначение: ERC-721 используется для создания невзаимозаменяемых токенов, где каждый токен уникален и может представлять различные активы, такие как произведения искусства, коллекционные предметы или даже недвижимость. Этот стандарт позволяет отслеживать владение уникальными активами на блокчейне, что открывает множество возможностей для индустрии цифровых товаров и цифрового искусства.

ERC-777

ERC-777 предлагает дополнительные функции по сравнению с токенами ERC-20, такие как функция аварийного восстановления в случае потери закрытого ключа и контракт микшера для повышения конфиденциальности транзакций.

ERC-865

Стандарт ERC-865 планирует снизить комиссию за транзакцию, взимаемую пользователями, путем введения функции под названием «transferPreSigned», позволяющей третьей стороне оплачивать комиссию за газ. Это полезно для dApps, работающих с многочисленными передачами токенов.

ERC-884

ERC-884 — это предлагаемый стандарт Ethereum, который допускает частичное владение активами в блокчейне. Поскольку дробные акции экономичны, все больше пользователей будут использовать сеть для покупки активов.

ERC1155

Стандарт ERC-1155, предложенный в 2018 году Витеком Радомски (техническим директором Enjin), позволяет разрабатывать взаимозаменяемые, полувзаимозаменяемые и невзаимозаменяемые токены в рамках одного контракта. Это привело к значительному сокращению количества транзакций, необходимых для управления несколькими типами токенов ERC, и, следовательно, значительно повысило эффективность сети. Назначение: ERC-1155 позволяет одному контракту управлять несколькими типами токенов — как взаимозаменяемыми, так и невзаимозаменяемыми. Это сокращает потребление ресурсов (газ) и упрощает управление различными токенами, что делает этот стандарт идеальным для игровых приложений и платформ, где необходимо множество типов активов.

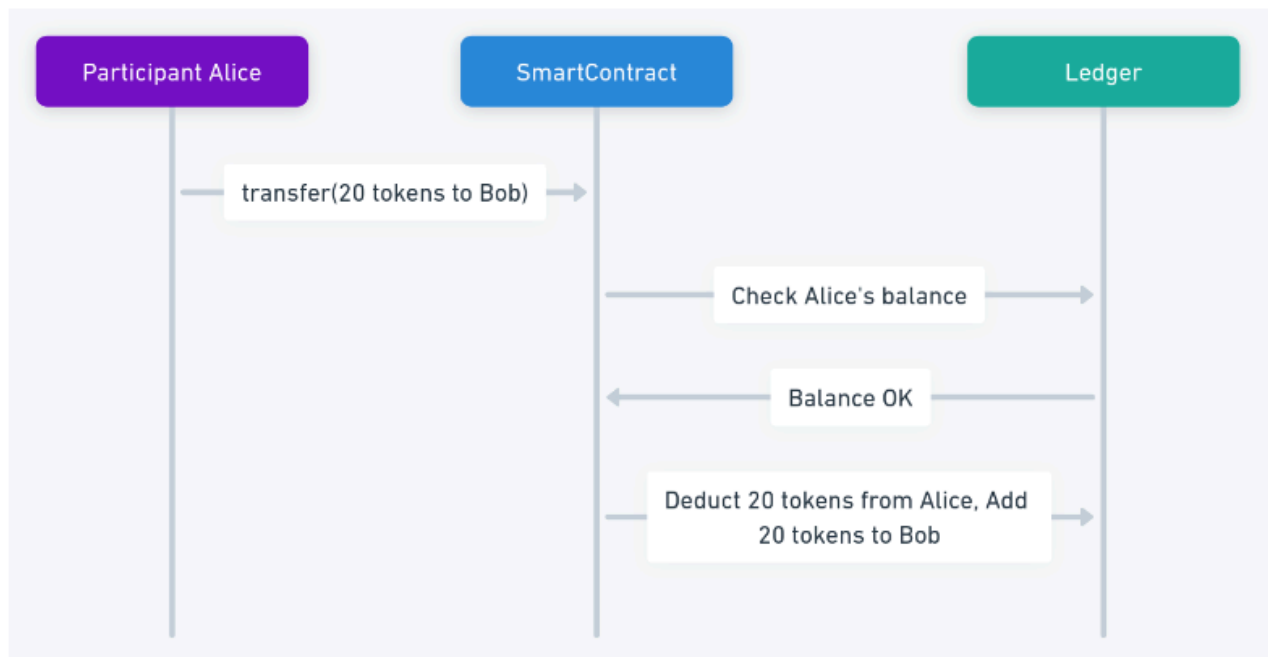
ERC-20

Одной из наиболее широко используемых спецификаций токенов в сети Ethereum является ERC-20. Благодаря своей простоте и адаптируемости он был быстро принят после того, как его представил Фабиан Фогельстеллер в ноябре 2015 года. Токены ERC-20 взаимозаменяемы и просты в обслуживании благодаря смарт-контрактам, которые устанавливают набор правил, которым должны следовать токены.

С момента своего создания, тысячи токенов в сети Ethereum, включая некоторые из самых крупных и ценных, таких как Chainlink, Binance Coin и Tether, были созданы на основе стандарта ERC-20. Этот стандарт также повлиял на несколько других стандартов токенов, включая ERC721 (для невзаимозаменяемых токенов) и ERC-1155 (один смарт-контракт, содержащий несколько токенов). ERC20 сыграл важную роль в росте и совершенствовании экосистемы Ethereum.

ERC-20 описывает интерфейс смарт-контракта, который будет обладать некоторыми свойствами. По сути дела этот смарт контракт будет поддерживать

ledger, содержащий балансы каждого владельца токенов. Для перевода токенов будет использоваться функция, изменяющая соответствующие балансы.



На данном рисунке представлено:

Передача токенов в контексте стандарта ERC-20 на блокчейне Ethereum. Видим следующие этапы:

До транзакции: у Alice есть 100 токенов, что отражено в учетной книге смарт-контракта (ledger). Это первоначальное состояние балансов.

Транзакция: Alice инициирует транзакцию, вызывая функцию `transfer`, чтобы отправить 20 токенов пользователю с именем Bob.

После транзакции: смарт-контракт обрабатывает этот запрос и, если у Alice достаточно токенов и нет других ограничений на выполнение транзакции, переводит 20 токенов из баланса Alice на баланс Bob. Теперь в учётной книге отражено, что у Alice осталось 80 токенов, а у Bob – 20 токенов.

Этот процесс демонстрирует базовую функцию `transfer` стандарта ERC-20, который является основой для большинства токенов, используемых в Ethereum. Стандарт определяет набор обязательных и необязательных функций, которые смарт-контракт должен реализовать, включая `balanceOf` для получения баланса, `transfer` для передачи токенов, и `transferFrom` для разрешения и передачи токенов от одного пользователя другому.

Спецификация ERC-20

Чтобы токен соответствовал стандарту ERC-20 он должен реализовывать следующий интерфейс:

Name: официальное название токена.

Symbol: идентифицирующая аббревиатура токена (тикер)

Decimals: количество десятичных знаков, используемых для представления токена. Например, если токен имеет 18 десятичных знаков, один токен равен 1000000000000000000 (10¹⁸) единиц.

Total supply: общее количество токенов, которые будут выпущены.

BalanceOf: возвращает количество токенов у конкретного держателя

Transfer: эта функция позволяет перемещать токены с одного адреса на другой

TransferFrom: эта функция позволяет авторизованным адресам отправлять токены с других адресов

Approve: эта функция позволяет одному адресу разрешить другому адресу тратить токены от его имени.

Allowance: эта функция указывает количество токенов, которые разрешено использовать одному адресу от имени другого (как показано выше).

Пример простого ERC-20 токена

Код основан на примере 0xProject ERC-20

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.19;

contract ERC20Token {
    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);

    mapping(address => uint256) internal balances;
    mapping(address => mapping(address => uint256)) internal allowed;
```

```
uint256 internal __totalSupply;  
string public name;  
string public symbol;  
uint256 public decimals;
```

```
function transfer(address _to, uint256 _value) external returns (bool) {  
    require(balances[msg.sender] >= _value, "ERC20_INSUFFICIENT_BALANCE");  
    require(balances[_to] + _value >= balances[_to], "UINT256_OVERFLOW");
```

```
    balances[msg.sender] -= _value;  
    balances[_to] += _value;
```

```
    emit Transfer(msg.sender, _to, _value);
```

```
    return true;  
}
```

```
function transferFrom(address _from, address _to, uint256 _value) external returns  
(bool) {
```

```
    require(balances[_from] >= _value, "ERC20_INSUFFICIENT_BALANCE");  
    require(allowed[_from][msg.sender] >= _value,  
"ERC20_INSUFFICIENT_ALLOWANCE");  
    require(balances[_to] + _value >= balances[_to], "UINT256_OVERFLOW");
```

```
    balances[_to] += _value;  
    balances[_from] -= _value;  
    allowed[_from][msg.sender] -= _value;
```

```
    emit Transfer(_from, _to, _value);
```

```
    return true;  
}
```

```
function approve(address _spender, uint256 _value) external returns (bool) {  
    allowed[msg.sender][_spender] = _value;  
    emit Approval(msg.sender, _spender, _value);  
    return true;  
}
```

```
function totalSupply() external view returns (uint256) {  
    return __totalSupply;  
}
```

```
function balanceOf(address _owner) external view returns (uint256) {
```

```

        return balances[_owner];
    }

    function allowance(address _owner, address _spender) external view returns
(uint256) {
        return allowed[_owner][_spender];
    }

    constructor (
        string memory _name,
        string memory _symbol,
        uint256 _decimals,
        uint256 _totalSupply
    )
    public
    {
        name = _name;
        symbol = _symbol;
        decimals = _decimals;
        __totalSupply = _totalSupply;
        balances[msg.sender] = _totalSupply;
    }
}

```

При деплое контракта надо указать параметры конструктора - имя, символ, десятичные знаки и общую эмиссию (supply). Приведённая выше реализация не позволяет изменять эмиссию. Вся созданная эмиссия будет присвоена создателю токена.

Леджер токена хранится в виде отображения balances. Баланс каждого конкретного адреса можно проверить методом `balanceOf(_owner)`, где `owner` - проверяемый адрес.

Перевод

Чтобы отправить токены на другой адрес, надо использовать функцию `transfer(address _to, uint256 _value)`, передав адрес получателя и количество передаваемых токенов. Функция проверяет наличие необходимого количества токенов на балансе отправителя, отсутствие переполнения при вычислении количества у получателя, изменяет балансы отправителя и получателя и регистрирует событие `Transfer`. Событие `Transfer` служит индикатором того, что передача токенов выполнена успешно. Функция `transfer` переводит токены с адреса отправителя транзакции.

Пусть Alice хочет запустить ICO и продавать токены за эфир. Тогда ей надо создать токен, задеплоить смарт-контракт с ICO и перевести туда свои токены. Вот пример простого контракта ICO, который будет переводить со своего счёта токены всем, кто отправит на него эфир. При деплое контракта нужно указать адрес токена.

```
contract Plain_ICO{
    ERC20Token public token;
    address public owner;

    event TokensPurchased(address buyer, uint256 amount);

    constructor(address tokenAddress) {
        token = ERC20Token(tokenAddress);
        owner = msg.sender;
    }

    receive() external payable {
        buyTokens();
    }

    function buyTokens() public payable {
        uint256 ethAmount = msg.value;
        uint256 tokenAmount = ethAmount;

        token.transfer(msg.sender, tokenAmount);

        emit TokensPurchased(msg.sender, tokenAmount);
    }

    function withdraw() external {
        require(msg.sender == owner, "Only the contract owner can call this function");

        uint256 balance = address(this).balance;
        payable(msg.sender).transfer(balance);
    }
}
```

При отправке на контракт эфира вызовется обработчик `receive()`, который выполнит метод `buyTokens()`. Метод `buyTokens` считает количество отправленного эфира в `wei`, считает адрес отправителя транзакции и вызовет функцию `transfer()` токена ERC20.

Весь эфир, лежащий на смарт-контракте, владелец смарт-контракта сможет забрать, вызвав функцию `withdraw()`.

Для проверки кода надо:


- С аккаунта Alice создать токен ERC20Token с не нулевой эмиссией.
- С аккаунта Alice создать контракт Plain_ICO.
- С аккаунта Alice отправить на адрес контракта Plain_ICO нужное количество токенов.
- С аккаунта Bob отправить на адрес контракта эфир, и убедиться что балансы контракта и отправителя эфира изменились.

Десятичные знаки

Десятичные знаки (decimals) в ERC-20 токенах используются для определения количества дробных разрядов, которые могут быть представлены в балансе токена. Они помогают в определении точности и деления токена на более мелкие единицы.

Когда создается токен ERC-20, автор контракта определяет значение десятичных знаков. Обычно это число от 0 до 18, где 0 означает, что токен не может быть разделен на более мелкие единицы (аналогично Bitcoin), а 18 означает, что токен поддерживает максимальное количество десятичных знаков и может быть разделен на маленькие доли.

Знание значения десятичных знаков для токена позволяет кошелькам и другим приложениям правильно отображать баланс токена и проводить вычисления с ним. Без правильного определения десятичных знаков вы можете столкнуться с ошибками при показе баланса или отправке токенов, так как баланс может быть неправильно интерпретирован из-за неправильного числа десятичных знаков.

 Все значения передаются в виде целых чисел! Decimals служит только для корректного отображения количества в пользовательском интерфейсе. Например, если у пользователя есть 1220 токенов с decimals = 2, то в пользовательском интерфейсе отобразится значение 12.20.

Allowance

Популярным сценарием является передача управления токенами другому адресу. Например, вы хотите сделать web3 приложение, которое будет в соответствии с внутренней логикой переводить токены на другой адрес. При внешних вызовах msg.sender изменится на адрес вызывающего контракта и функция transfer() будет переводить токены контракта приложения, а не пользователя. Для решения этой проблемы используется пара функций: approve() и transferTo().

Например, приведенный выше контракт Plain_ICO можно модифицировать таким образом, что токены будут продаваться напрямую с кошелька продавца. Таким образом продавцу не придётся изымать из оборота свои токены, передавая их контракту.

Для этого в примере Plain_ICO заменим код метода buyTokens() на следующий.

```
function buyTokens() public payable {
    uint256 ethAmount = msg.value;
    uint256 tokenAmount = ethAmount;

    address ethSender = msg.sender;

    token.transferFrom(owner, ethSender, tokenAmount);

    emit TokensPurchased(ethSender, tokenAmount);
}
```

В этом примере токены всё время находятся на кошельке Alice, и она должна разрешить контракту их использование с помощью метода approve(). Тогда последовательность действий будет следующей:

- С аккаунта Alice создать токен ERC20Token с ненулевой эмиссией
- С аккаунта Alice создать контракт Plain_ICO
- С аккаунта Alice отправить вызвать метод approve() контракта ERC20Token, передав ему адрес контракта Plain и количество токенов
- С аккаунта Bob отправить на адрес контракта эфир, и убедиться что балансы контракта и отправителя эфира изменились, а allowance токена уменьшился

Расширения ERC-20

ERC-223

ERC-223 является расширением протокола ERC-20 и был создан как решение ряда проблем ERC-20. Хотя стандарт ERC-20 чрезвычайно полезен, он далек от идеала. Например, один особенно вопиющий недостаток дизайна в ERC-20 позволял токенам теряться, когда люди по ошибке отправляли их в смарт-контракт, используя тот же процесс, который они использовали бы для отправки токенов в обычный кошелек.

К сожалению, это уже привело к потере более 3 миллионов долларов в токенах ERC-20.

Этот недостаток конструкции устраняется стандартом ERC-223, который позволяет пользователям передавать токены в смарт-контракты и кошельки, выполняющие ту же функцию. Кроме того, токены ERC-223 превосходят токены ERC-20 с точки зрения эффективности, поскольку для транзакций требуется только один шаг, а не два.

ERC-223 сохраняет все исходные функции, устраняя вышеупомянутые ошибки. Новый стандарт представляет улучшения и возможности, которые решают некоторые из наиболее серьезных проблем ERC-20, особенно, при взаимодействии с другими смарт-контрактами.

Функция передачи ERC-223 включает параметр, который гарантирует, что адрес назначения является смарт-контрактом. В этом случае транзакция вызывает функцию возврата токена в смарт-контракте, и ее можно вернуть на счет отправителя с помощью этой функции, после чего токены передаются в смарт-контракт.

Никакие токены не теряются, поскольку обновленная функция передачи теперь работает и для смарт-контрактов.

ERC-777

ERC-777 — это стандарт токенов, который позволяет создавать токены на блокчейне Ethereum, аналогичный стандарту токенов ERC-20. Этот стандарт преобразует существующие модели взаимозаменяемых токенов и предлагает более безопасный и эффективный подход к обмену токенами.

Несмотря на то, что контракты ERC-777 взаимодействуют так же, как контракты ERC-20, отличительной особенностью первых является механизм перехвата (hook). Это упрощает обмен токенами между аккаунтами и контрактами. Механизм перехвата задействуется, как только токены отправляются в смарт контракт. Это позволяет обрабатывать получение токенов аналогично функции `receive()` при получении эфира.

Помимо этого, стандарт токена ERC-777 имеет еще много преимуществ. Например, вы можете добавить механизм аварийного восстановления, который поможет вам, если вы потеряете свои секретные ключи.

ERC-827

Одним из основных преимуществ ERC-827 является то, что он позволяет выполнять более сложные транзакции, включающие передачу токенов и дополнительных

данных. С ERC-20 и ERC-223 транзакции ограничиваются только передачей токенов. Однако ERC-827 включает дополнительные поля данных, которые можно использовать для указания дополнительной информации о транзакции, например ссылочного номера или идентификатора счета-фактуры. Это упрощает интеграцию токенов в существующие финансовые системы и приложения.

ERC-827 также включает поддержку функции `approveAndCall`, которая позволяет держателям токенов утверждать передачу токенов и инициировать вызов функции в одной транзакции. Это может быть полезно для создания более сложных смарт-контрактов и приложений, требующих выполнения нескольких действий в одной транзакции.

ERC-827 предназначен для обеспечения более гибкого и расширяемого стандарта токенов для сети Ethereum и призван облегчить интеграцию токенов в существующие финансовые системы и приложения. Хотя он еще не получил широкого распространения, в будущем он может стать популярным стандартом для создания токенов.

ERC-844

Каждый токен ERC-884 в соответствии со стандартом определяет определенную долю в компании штата Делавэр. Стандарт предназначен для транзакций с акциями, и владелец токена должен быть внесен в белый список, который является методом, встроенным в смарт контракт. Кроме того, эмитенты ERC-884 должны создать частную базу данных вне сети, чтобы соответствовать законодательству о ценных бумагах.

Благодаря новым законам компании в американском штате Делавэр теперь могут использовать технологию блокчейна для управления регистрацией акций. ERC-884 направлен на его использование, где каждый токен будет обозначаться как доля корпорации, созданной в штате Делавэр. Для соответствия законодательству штата токены реализуют дополнительные методы.

Личности владельцев токенов должны быть проверены и внесены в белый список. Отдельного контракта на краудсейл для обеспечения внесения в белый список владельцев токенов не существует, а ERC-884 требует внесения в белый список всех владельцев токенов. Хотя включение краудсейл-контрактов может использовать белый список для проверки права на участие, белый список по-прежнему является важным элементом контракта токена.

Корпорация может составить список акционеров. Согласно нормативным положениям, оно должно разрешить корпорации создавать список акционеров.

Кроме того, регулирующие органы требуют регистрации информации и регистрации передачи акций.

Акционеры, потерявшие свои закрытые ключи или токены, должны иметь возможность перенести их на новый адрес. Должна быть внедрена система, позволяющая акционерам, которые потеряли свой закрытый ключ или потеряли доступ к своим токенам, отозвать свой адрес и перевыпустить свои токены на новый адрес.

Внедрение ERC-884 требует использования автономной базы данных для соответствия определенным критериям «Знай своего клиента» (KYC).

Поскольку токены ERC-884 совместимы с ERC-20, их можно обменять на любой криптобирже, которая поддерживает торговлю токенами ERC-20.

ERC-865

Одним из основных преимуществ ERC-865 является то, что он позволяет оплачивать переводы токенов кем-то, кроме отправителя, что невозможно в стандартах ERC-20 и ERC-223. Это означает, что пользователи могут отправлять токены без необходимости самостоятельно платить за газ, что может быть особенно полезно для децентрализованных приложений и сервисов, требующих больших объемов передачи токенов.

ERC-865 достигает этого за счет введения новой функции под названием `transferPreSigned`, которая позволяет третьей стороне оплачивать комиссию за газ, связанную с передачей токена. Эта функция требует, чтобы владелец токена подписал сообщение, содержащее детали перевода, которые затем могут быть отправлены в сеть третьей стороной. Это уменьшает количество газа, необходимого для перевода, поскольку плата за газ может быть оплачена более эффективным способом.

ERC-865 разработан для обеспечения более удобного и экономичного процесса передачи токенов в сети Ethereum. Хотя он еще не получил широкого распространения, он потенциально может снизить бремя платы за газ для пользователей и децентрализованных приложений, которые требуют передачи больших объемов токенов.

Стейблкоины

Стейблкоин - это тип криптовалюты, который разработан для минимизации волатильности своей стоимости в сравнении с другими криптовалютами, такими как Биткоин или Эфириум. Они стремятся обеспечить стабильность своей цены, привязываясь к другим активам, таким как фиатные деньги (например, доллар США) или сырьевые товары (например, золото).

Стейблкоины пытаются решить проблему волатильности, которая является одной из основных преград для массового принятия криптовалют в повседневной жизни и сфере коммерции. Поскольку их цена сравнительно стабильна, стейблкоины могут быть использованы для хранения стоимости и проведения транзакций, не подвергаясь значительным колебаниям.

При создании стейблкоинов используются различные механизмы обеспечения стабильности и привязки к активам. Например, некоторые стейблкоины основаны на централизованной модели, где эмитент (обычно компания) обязуется хранить резерв активов, эквивалентный общей эмиссии стейблкоинов в обороте. Другие стейблкоины используют децентрализованные механизмы, такие как смарт-контракты, чтобы автоматически регулировать предложение и спрос на основе изменений цены.

Стейблкоины имеют потенциал повысить удобство использования криптовалюты в повседневной жизни, однако они также вызывают вопросы о прозрачности, регулировании и доверии к эмитенту. Важно знать, что все стейблкоины могут в значительной степени отличаться по своим техническим характеристикам, модели консенсуса и правилам функционирования.

Взаимозаменяемые и невзаимозаменяемые токены

Что такое взаимозаменяемый токен

Взаимозаменяемые токены — это цифровые активы, которые взаимозаменяемы и неотличимы друг от друга. Каждая единица взаимозаменяемого токена идентична по своим свойствам, стоимости и характеристикам. Это означает, что одну единицу взаимозаменяемого токена можно обменять по принципу «один к одному» на любую другую единицу того же типа, и они имеют равную стоимость.

Ключевые характеристики взаимозаменяемых токенов включают в себя:

- **Взаимозаменяемость:** взаимозаменяемые токены полностью взаимозаменяемы, что означает, что один токен так же хорош, как и любой другой того же типа. Например, если у вас есть один биткойн (BTC), он эквивалентен по стоимости любому другому отдельному биткойну.
- **Делимость:** взаимозаменяемые токены часто делятся на более мелкие единицы. Например, вы можете отправлять или владеть частью биткойна, например 0,5 BTC, так же легко, как и целым биткойном.
- **Единообразие:** взаимозаменяемые токены однородны и последовательны и не имеют уникальных отличительных особенностей. Каждая единица взаимозаменяемого токена такая же, как и остальные.

Варианты использования взаимозаменяемых токенов

Взаимозаменяемые токены, однородные по стоимости и обмениваемые, имеют широкий спектр применений:

- **Криптовалюта:** Биткойн (BTC) и Эфириум (ETH) — это цифровые деньги для транзакций и хранения ценностей.
- **Стейблкоины:** обеспечивая стабильность, они используются для надежных транзакций на волатильных рынках.
- **Бонусные баллы и лояльность:** обычно используются компаниями для получения скидок и льгот.
- **Токенизированные активы:** реальные активы, такие как недвижимость, акции и товары, могут быть представлены и принадлежать им частично.
- **Игры:** внутриигровая валюта, предметы и активы игровой индустрии.
- **DeFi:** взаимозаменяемые токены лежат в основе децентрализованного финансирования кредитования, пулов ликвидности и управления.
- **Денежные переводы:** экономически эффективные и быстрые трансграничные платежи.
- **Краудфандинг:** привлечение капитала посредством ICO и продажи токенов.
- **Монеты конфиденциальности:** обеспечивают повышенную конфиденциальность транзакций, как Monero (XMR) и Zcash (ZEC).
- **Благотворительные пожертвования:** прозрачные пожертвования с возможностью отслеживания.
- **Монетизация контента:** используется создателями контента для продажи эксклюзивного контента или услуг.

Что такое невзаимозаменяемый токен

Невзаимозаменяемый токен (NFT) — это тип цифрового токена, который представляет собой право собственности и доказательство подлинности уникального цифрового или физического предмета. В отличие от взаимозаменяемых токенов, которые взаимозаменяемы и имеют одинаковую стоимость (например, криптовалюты, такие как Биткойн или Эфириум), каждый NFT уникален и не может быть обменян по принципу «один к одному» с другим NFT. NFT неделимы, то есть вы не можете отправлять части NFT; вы можете передать только весь токен.

Варианты использования NFT

Поскольку NFT в основном используются для установления права собственности, происхождения и аутентичности цифровых или физических активов, они обычно ассоциируются с миром цифрового искусства, предметов коллекционирования, игр и виртуальной недвижимости. Вот некоторые из примечательных случаев использования NFT.

- Цифровое искусство: художники и создатели могут токенизировать свои цифровые произведения искусства, что позволяет им продавать уникальные произведения и доказывать право собственности покупателям.
- Предметы коллекционирования: NFT завоевали популярность в мире цифровых предметов коллекционирования, где можно покупать, продавать и обменивать редкие и уникальные предметы, такие как коллекционные карточки или виртуальных домашних животных.
- Игры: NFT используются в игровой индустрии для представления внутриигровых активов, персонажей и виртуальной недвижимости. Игроки могут иметь истинное право собственности на эти активы и торговать ими в играх и вне их.
- Виртуальная недвижимость. В виртуальных мирах и платформах метавселенной NFT используются для покупки и продажи виртуальной земли и недвижимости, что позволяет пользователям устанавливать право собственности и монетизировать свои виртуальные активы.
- Токенизация физических активов: NFT также можно использовать для обозначения владения физическими активами, такими как недвижимость, предметы искусства или предметы роскоши. Эти токены связывают физический и цифровой миры, упрощая торговлю, проверку и установление права собственности на реальные активы.

ERC-721

Подобно ERC-20, ERC-721 это стандарт интерфейса смарт контракта, содержащего леджер. Основное отличие токенов ERC-721 заключается в том, что каждый из них уникален. Когда создается токен ERC-721, существует один и только один из этих токенов.

Спецификация ERC-721

Функции ERC-721

BalanceOf: эта функция используется для возврата количества NFT, принадлежащих определенному адресу.

OwnerOf: эта функция возвращает адрес владельца определенного токена. Каждый токен ERC-721 уникален и представлен идентификатором. Эта функция позволяет пользователям или приложениям определять владельца токена на основе его уникального идентификатора.

SafeTransferFrom (без данных): эта функция безопасно передает право собственности на определенный токен с одного адреса на другой. Эта функция проверяет, является ли получатель смарт-контрактом. Если да, то для принятия передачи он должен реализовать специальную функцию (onERC721Received).

TransferFrom: эта функция используется для передачи владения токеном с одного адреса на другой. Обычно он используется, когда отправителю было разрешено передать токен.

Approve: эта функция используется для одобрения адреса для передачи определенного токена. Это позволяет осуществлять делегированную передачу, когда владелец может разрешить другой стороне передавать токен от его имени.

getApproved: эта функция используется для получения утвержденного адреса для определенного токена. Если для токена нет утвержденного адреса, эта функция вернет нулевой адрес.

setApprovalForAll: эта функция позволяет владельцу одного или нескольких токенов одобрить или отозвать разрешение оператора на управление всеми своими токенами.

isApprovedForAll: эта функция используется для проверки того, разрешено ли оператору управлять всеми токенами владельца.

SafeTransferFrom (с данными): эта функция аналогична SafeTransferFrom (без данных), но с дополнительным параметром данных. Эти дополнительные данные могут использоваться для передачи дополнительной информации во время передачи, если получателем является смарт-контракт. Эта функция также проверяет, является ли получатель смарт-контрактом и реализует ли он функцию onERC721Received.

Пример кода

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.1;
3
4 import "@openzeppelin/contracts@5.0.0/token/ERC721/ERC721.sol";
5 import
  "@openzeppelin/contracts@4.9.0/token/ERC721/extensions/ERC721URIStorage.sol";
6 import
  "@openzeppelin/contracts@4.9.0/token/ERC721/extensions/ERC721Burnable.sol";
7 import "@openzeppelin/contracts@5.0.0/access/Ownable.sol";
8
9 contract MyToken is ERC721, ERC721URIStorage, ERC721Burnable,
  Ownable {
10     constructor(address initialOwner)
11         ERC721("MyToken", "MTK")
12         Ownable(initialOwner)
13     {}
14
15     function safeMint(address to, uint256 tokenId, string memory
  uri)
16     public
17     onlyOwner
18     {
19         _safeMint(to, tokenId);
20         _setTokenURI(tokenId, uri);
21     }
22
23     // The following functions are overrides required by Solidity.
24
25     function tokenURI(uint256 tokenId)
26     public
27     view
28     override(ERC721, ERC721URIStorage)
29     returns (string memory)
30     {
31         return super.tokenURI(tokenId);
32     }
33
34     function supportsInterface(bytes4 interfaceId)
35     public
36     view
37     override(ERC721, ERC721URIStorage)
38     returns (bool)
39     {
40         return super.supportsInterface(interfaceId);
41     }
42 }
43
```

Этот смарт-контракт использует библиотеку OpenZeppelin для создания токена. Помимо целочисленного идентификатора экземпляра токена содержит ссылку на описание. Описание представляет собой JSON файл с заданной схемой.

```
1 {
2   "title": "Asset Metadata",
3   "type": "object",
4   "properties": {
5     "name": {
6       "type": "string",
7       "description": "Identifies the asset to which this NFT
represents"
8     },
9     "description": {
10      "type": "string",
11      "description": "Describes the asset to which this NFT
represents"
12    },
13    "image": {
14      "type": "string",
15      "description": "A URI pointing to a resource with mime
type image/* representing the asset to which this NFT represents.
Consider making any images at a width between 320 and 1080 pixels
and aspect ratio between 1.91:1 and 4:5 inclusive."
16    }
17  }
18 }
19
20
```

При корректном описании формата изображение будет отображаться в криптокошельках и в эксплорерах.

Практические задания:

Задание 1: Создание и деплой собственного ERC-20 токена

Цель:

Вы должны разработать и задеплоить на тестовой сети Ethereum (например, Rinkeby) собственный ERC-20 токен. А также настроить параметры токена, такие как имя, символ и общее количество токенов.

Задачи:

Использовать OpenZeppelin библиотеку для создания стандартного ERC-20 токена.

Задеплоить контракт на тестовую сеть.

Провести тестовые транзакции между различными аккаунтами.

Ответ:

Для создания стандартного ERC-20 токена с использованием OpenZeppelin библиотеки, нам сначала потребуется установить библиотеку и написать контракт токена на языке Solidity. Затем мы задеплоим контракт на тестовую сеть и проведем тестовые транзакции между различными аккаунтами.

1. Установим OpenZeppelin библиотеку:

```
bash
```

Копировать

```
npm install @openzeppelin/contracts
```

2. Напишем контракт стандартного ERC-20 токена:

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5
6 contract MyToken is ERC20 {
7     constructor() ERC20("MyToken", "MTK") {
8         _mint(msg.sender, 1000000 * 10 ** uint(decimals()));
9     }
10 }
11

```

3. Деплоим контракт на тестовую сеть, например, на Rinkeby. Для этого нам понадобится скомпилированный байт-код контракта.

4. Проведем тестовые транзакции между различными аккаунтами, используя веб-интерфейс вроде MyEtherWallet или MetaMask.

Чтобы провести тестовые транзакции, вы можете использовать один из тестовых аккаунтов с Rinkeby тестовой криптовалютой, отправив токены с созданного контракта на другие аккаунты или обратно.

Для успешного деплоя контракта и проведения транзакций необходимо наличие достаточного количества эфира на вашем тестовом аккаунте для оплаты газа.

Задание 2: Разработка смарт-контракта для голосования на основе токенов

Цель:

Создать смарт-контракт для голосования, где владение токенами определяет количество голосов пользователя. Это позволит понять, как токены могут использоваться для управления проектами и принятия решений в сообществе.

Задачи:

Создать ERC-20 токен, который будет использоваться в качестве голоса.
Разработать контракт голосования, который позволяет токен-холдерам голосовать по предложенным вопросам.

Реализовать функции для подсчета голосов и обнародования результатов голосования.

Подсказка:

Можно использовать функцию `balanceOf` из стандарта ERC-20, чтобы определить количество токенов у пользователя, что позволит определить его голосовые права. Функция `transfer` поможет перемещать токены на специальный контрактный адрес для учета голосов.

Ответ:

Для создания ERC-20 токена, который будет использоваться в качестве голоса, мы можем начать с написания контракта на языке Solidity:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5
6 contract VotingToken is ERC20 {
7
8     constructor() ERC20("VotingToken", "VOTE") {
9         _mint(msg.sender, 10000 * 10 ** uint(decimals()));
10    }
11 }
12
```

Этот контракт создает ERC-20 токен с именем "VotingToken" и символом "VOTE". Токены начисляются создателю контракта при его развёртывании.

Далее, нам необходимо создать контракт голосования, который позволит токен-холдерам голосовать:

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 import "./VotingToken.sol";
5
6 contract Voting {
7
8     struct Proposal {
9         string description;
10        uint forVotes;
11        uint againstVotes;
12        bool isOpen;
13    }
14
15    Proposal[] public proposals;
16    address public admin;
17    VotingToken public votingToken;
18
19    mapping(address => bool) public hasVoted;
20
21    constructor(address _votingToken) {
22        admin = msg.sender;
23        votingToken = VotingToken(_votingToken);
24    }
25
26    function createProposal(string memory _description) public {
27        require(msg.sender == admin, "Only admin can create
proposals");
28        proposals.push(Proposal(_description, 0, 0, true));
29    }
30
31    function vote(uint _proposalIndex, bool _supports) public {
32        require(_proposalIndex < proposals.length, "Invalid proposal
index");
33        require(votingToken.balanceOf(msg.sender) > 0, "Must have
tokens to vote");
34
35        Proposal storage proposal = proposals[_proposalIndex];
36        require(proposal.isOpen, "Voting for this proposal is
closed");
37        require(!hasVoted[msg.sender], "You have already voted");
38
39        hasVoted[msg.sender] = true;
40
41        if (_supports) {
42            proposal.forVotes += votingToken.balanceOf(msg.sender);
43        } else {
44            proposal.againstVotes +=
votingToken.balanceOf(msg.sender);
45        }
46    }
47
48    function closeVoting(uint _proposalIndex) public {
49        require(msg.sender == admin, "Only admin can close voting");
50
51        Proposal storage proposal = proposals[_proposalIndex];
52        require(proposal.isOpen, "Voting for this proposal is
already closed");
53
54        proposal.isOpen = false;
55    }
56
57    function getResults(uint _proposalIndex) public view returns
(uint, uint, bool) {
58        Proposal storage proposal = proposals[_proposalIndex];
59        return (proposal.forVotes, proposal.againstVotes,
proposal.isOpen);
60    }
61 }
62

```


В этом контракте наряду с структурой Proposal для предложений, мы определяем функции для создания предложений, голосования, закрытия голосования и получения результатов голосования.

Будьте уверены, что контракты прошли все необходимые аудиты перед развертыванием в сети Ethereum.

Заключение:

Мы подошли к концу нашего путешествия по изучению токенов Ethereum и их стандартов. На протяжении этой лекции мы обсудили ключевые концепции и стандарты, такие как ERC-20 и ERC-721, их функциональные возможности и потенциальное применение. Вы узнали о важности этих стандартов для обеспечения совместимости и безопасности в экосистеме Ethereum и о том, как разнообразны области применения токенов - от взаимозаменяемых токенов, используемых в качестве цифровых валют, до невзаимозаменяемых токенов (NFT), которые изменяют правила игры в мире цифрового искусства и собственности.

Мы также рассмотрели различные расширения стандарта ERC-20, предлагающие улучшения в области управления, безопасности и взаимодействия с пользовательскими интерфейсами. В дополнение, мы изучили практические аспекты создания и развертывания смарт-контрактов, включая практические упражнения по созданию собственного ERC-20 токена и разработке контракта для голосования на основе токенов.

Как мы видели, возможности блокчейна и токенов безграничны. Теперь, когда вы оснащены знаниями о токенах и стандартах ERC, вы можете применить их для решения реальных задач, от разработки уникальных цифровых активов до создания новаторских решений для децентрализованных систем. Не останавливайтесь на достигнутом, продолжайте изучать, экспериментировать и разрабатывать, и помните, что блокчейн предлагает многообещающее и захватывающее будущее для всех, кто готов идти в ногу со временем и технологиями.

Использованная литература

1. <https://ethereum.org/en/developers/docs/standards>
2. <https://docs.openzeppelin.com/contracts>