Смарт-контракты

Что такое смарт-контракт
и как его создать в Solidity

Артем Демиденко / ИИ

С подробным разбором кода

Артем Демиденко

Искусственный Интеллект

Артем Демиденко

Смарт-контракты. Что такое смарт-контракт и как его создать в Solidity. С подробным разбором кода

http://www.litres.ru/pages/biblio_book/?art=69557629
SelfPub; 2023

Аннотация

Исчерпывающее руководство для тех, кто стремится овладеть искусством создания и использования смарт-контрактов. С этой книгой вы погрузитесь в захватывающий мир блокчейна и Ethereum, начнете с основных концепций и постепенно продвинетесь к сложным темам и применениям.Преодолевая языковой барьер и технические сложности, книга предоставляет понятное введение в язык программирования Solidity, основной инструмент для разработки смарт-контрактов. Через наглядные примеры и шаг за шагом инструкции, вы научитесь создавать, тестировать и развертывать свои смарт-контракты на Ethereum.Внимание также уделено безопасности смарт-

контрактов, предостерегая от распространенных уязвимостей и атак. Вы узнаете о передовых методах аудита и тестирования, которые помогут обеспечить надежность ваших контрактов.Книга не только предоставляет технические знания, но и исследует практические сценарии использования, такие как создание токенов и участие в децентрализованных финансах (DeFi).

Содержание

Артем Демиденко Смарт-контракты. Что такое смарт-контракт и как его создать в Solidity. С подробным разбором кода

Глава 1: Введение в смарт-контракты

В данной книге в примерах кода стоят .... они показывают количество пробелов, необходимое сделать перед тем или иным участком кода. Это сделано из-за особенностей публикации книги на платформе. В реальности вам надо заменить .... на 4 пробела.

1.1 Что такое смарт-контракты?

Смарт-контракты – это автоматизированные программы, выполнение и условия которых строго определены и выполняются в рамках блокчейн-платформы. В отличие от традиционных юридических контрактов, смарт-контракты существуют и функционируют в цифровом пространстве, что обеспечивает автоматичность и надежность выполнения

условий без участия третьих сторон.

Основная идея смарт-контрактов заключается в том, что они позволяют двум или более участникам совершать сделки и обмениваться ценностями без необходимости доверять друг другу. Вместо этого доверие строится на математических и криптографических принципах блокчейна.

Смарт-контракты могут быть использованы для широкого спектра задач и областей:

1. Управление цифровыми активами: Смарт-контракты могут представлять цифровые токены (токены на платформе Ethereum, например) и автоматически выполнять условия для их передачи владения или других операций.

2. Децентрализованные финансы (DeFi): Смарт-контракты могут использоваться для создания финансовых инструментов, таких как займы, обмены, стейблкоины и другие продукты DeFi.

3. Управление поставками и логистика: Смарт-контракты могут отслеживать передвижение товаров и автоматически выполнять условия выплат или других операций при выполнении определенных событий (например, доставки товара).

4. Голосования и управление сообществами: Смарт-контракты позволяют проводить децентрализованные голосования и управлять решениями внутри сообществ.

5. Юридические и бизнес-соглашения: Смарт-контракты могут автоматически выполнять юридические согла-

шения, такие как выплаты авторских вознаграждений или условия партнерских соглашений.

Однако следует отметить, что смарт-контракты имеют свои ограничения и риски. Важно понимать, что смарт-контракты выполняют только те действия, которые программированы в них, и они могут стать объектами уязвимостей или ошибок в коде. Поэтому создание безопасных и надежных смарт-контрактов требует серьезного понимания технологии, аудита и тестирования.

Ключевым аспектом смарт-контрактов является их децентрализованность и автоматичность, что открывает новые перспективы для автоматизации бизнес-процессов, обеспечения доверия между сторонами и улучшения эффективности в различных отраслях.

1.2: Преимущества и недостатки смарт-контрактов

Смарт-контракты представляют собой программные коды, размещенные на блокчейн-платформах, таких как Ethereum, и предназначенные для автоматизации и обеспечения безопасности выполнения различных соглашений и условий. Они объединяют в себе принципы программирования с принципами децентрализации, обеспечивая надежную и прозрачную систему выполнения сделок и операций без необходимости доверия к посредникам.

Преимущества смарт-контрактов:

1. **Децентрализация и надежность:** Смарт-контракты выполняются на блокчейне, что означает, что они не за-

висят от центральных органов или сторонних посредников. Это обеспечивает уровень надежности, недостижимый в традиционных системах.

2. **Автоматизация:** Смарт-контракты автоматически выполняют условия, определенные в коде. Это сокращает необходимость ручного управления и уменьшает вероятность ошибок.

3. **Прозрачность:** Все события и операции, связанные с смарт-контрактами, регистрируются на блокчейне и доступны для проверки. Это обеспечивает высокую степень прозрачности и отслеживаемости.

4. **Эффективность:** Смарт-контракты могут значительно ускорить процессы, требующие многошаговых соглашений и подтверждений, так как они выполняются автоматически.

5. **Низкие затраты на транзакции:** В сравнении с традиционными финансовыми системами, транзакции на блокчейне могут быть гораздо более дешевыми, особенно в случае смарт-контрактов.

6. **Открытые возможности:** Смарт-контракты позволяют разрабатывать широкий спектр приложений, начиная от децентрализованных финансовых сервисов и заканчивая управлением цепями поставок, голосованиями и даже искусственным интеллектом.

Недостатки смарт-контрактов:

1. **Неизменность:** Код смарт-контракта не может быть

изменен после его развертывания. Если в коде есть ошибки, они могут быть зафиксированы навсегда.

2. **Сложность программирования:** Создание сложных смарт-контрактов может потребовать глубокого понимания программирования и блокчейн-технологий.

3. **Зависимость от блокчейн-платформы:** Смарт-контракты разрабатываются для конкретной блокчейн-платформы, и их переносимость между разными платформами может быть сложной.

4. **Высокая стоимость ошибок:** Несмотря на прозрачность, транзакции и операции с смарт-контрактами не обратимы. Любые ошибки могут иметь серьезные финансовые последствия.

5. **Отсутствие контроля:** Поскольку смарт-контракты автоматически выполняются, нет возможности вмешательства или отмены транзакций в случае конфликта или спора.

6. **Барьеры для внедрения:** Внедрение смарт-контрактов требует изменения подхода к бизнес-процессам и взаимодействию с технологией блокчейна.

Важно понимать, что смарт-контракты – это инструмент, который имеет свои сильные и слабые стороны, и их применение следует рассматривать с учетом конкретных задач и контекста. Несмотря на некоторые ограничения, смарт-контракты остаются мощным средством для повышения эффективности, надежности и прозрачности в различных сферах деятельности.

1.3: Практические примеры использования смарт-контрактов

Смарт-контракты, воплощение децентрализации и автоматизации, находят применение в различных областях, от финансов до логистики, от недвижимости до искусства. Ниже представлены разнообразные практические примеры использования смарт-контрактов, которые позволят вам лучше понять и оценить их потенциал.

1. Децентрализованные финансы (DeFi): Одним из самых динамично развивающихся примеров использования смарт-контрактов являются децентрализованные финансы. Смарт-контракты позволяют создавать и управлять финансовыми инструментами, такими как стейблкоины (стабильные монеты), ликвидностьные пулы, децентрализованные биржи и кредитные платформы. Пользователи могут занимать и предоставлять средства, зарабатывать проценты на хранении, обменивать криптовалюты без посредников и многое другое, всё это работает автоматически на основе заложенных правил в смарт-контрактах.

2. Цифровые ипотеки и недвижимость: Смарт-контракты могут преобразовать процессы покупки, продажи и аренды недвижимости. С помощью контрактов можно автоматизировать выплату и контроль задолженности, а также осуществлять передачу прав собственности по истечении определенных условий (например, после полного погашения кредита).

3. Снабжение и логистика: Смарт-контракты могут облегчить отслеживание и управление поставками товаров. Контракты могут автоматически регулировать оплату и выпуск товаров на основе фактических доставок и приемок.

4. Интеллектуальная собственность и авторские права: Артисты и авторы могут использовать смарт-контракты для автоматизации распределения доходов от своих произведений. Контракты могут гарантировать авторам честное вознаграждение за использование их материалов и упростить процессы лицензирования.

5. Голосования и управление: Смарт-контракты могут обеспечить прозрачные и безопасные системы голосования и управления. Это может быть полезно для выборов, корпоративных решений или общественных голосований.

6. Контроль качества и происхождения товаров: В индустрии пищевых продуктов и других товаров потребители и производители могут использовать смарт-контракты для отслеживания цепочки поставок и убедиться в качестве и безопасности товаров.

7. Музыка и развлечения: Артисты могут использовать смарт-контракты для продажи музыки, видео и другого контента напрямую своим поклонникам. Контракты обеспечивают прозрачные доли доходов и автоматически выплачивают вознаграждения.

8. Страхование: Смарт-контракты позволяют создавать децентрализованные страховые продукты. Подписчики мо-

гут автоматически получать выплаты при наступлении определенных событий, что облегчает процесс оценки убытков и выплат.

9. Образование и сертификация: Смарт-контракты могут использоваться для эффективной и прозрачной выдачи и проверки сертификатов и дипломов. Это может быть полезно в образовательных учреждениях и профессиональных курсах.

10. Экологические инициативы: Смарт-контракты могут помочь отслеживать и подтверждать выполнение экологических обязательств, таких как утилизация отходов или соблюдение стандартов энергоэффективности.

Эти лишь несколько примеров множества возможных сценариев использования смарт-контрактов. Они позволяют улучшить эффективность, уменьшить необходимость посредников, обеспечить прозрачность и автоматизацию в различных сферах, что делает их настоящим инструментом для децентрализованного будущего.

Глава 2: Основы блокчейна и Ethereum

2.1: Основы блокчейна и его роль в смарт-контрактах

Современный мир переживает цифровую революцию, и одним из самых инновационных достижений в этой области является технология блокчейн. Блокчейн – это децентрализованная и надежная система записи данных, которая стала основой для развития смарт-контрактов. Для полного понимания смарт-контрактов важно осознать, что такое блокчейн и как он функционирует.

Основы блокчейна:

Блокчейн можно представить как цепочку блоков, каждый из которых содержит набор транзакций или данных. Каждый блок связан с предыдущим блоком, образуя цепочку, поэтому он называется "блокчейн" – блоки, объединенные в цепь. Эта цепочка блоков распределена по узлам (компьютерам), которые совместно управляют блокчейном. Такой децентрализованный подход обеспечивает высокий уровень безопасности и надежности данных.

Криптография и консенсус:

Блокчейн использует криптографию для обеспечения безопасности данных. Каждый блок содержит хеш (криптогра-

фическую сумму) предыдущего блока, что делает цепочку блоков устойчивой к внесению изменений. Кроме того, блокчейн использует механизмы консенсуса (например, Proof of Work или Proof of Stake), чтобы узлы могли согласовать и подтвердить новые транзакции.

Децентрализация и преимущества:

Главное преимущество блокчейна – это его децентрализованная природа. Вместо того чтобы иметь централизованный орган (например, банк) для проверки и обработки транзакций, блокчейн позволяет участникам сети взаимодействовать напрямую без посредников. Это улучшает прозрачность, снижает затраты и повышает безопасность.

Роль блокчейна в смарт-контрактах:

Смарт-контракты – это автоматизированные программы, которые выполняют условия и соглашения между участниками без необходимости доверять третьей стороне. Блокчейн играет центральную роль в смарт-контрактах, предоставляя среду, в которой они могут функционировать.

1. **Децентрализация:** Блокчейн обеспечивает децентрализованный контроль над смарт-контрактами. Участники сети соглашаются о состоянии контракта и его выполнении, что устраняет необходимость доверия к централизованной организации.

2. **Безопасность:** Блокчейн гарантирует надежность и безопасность смарт-контрактов. Записи в блокчейне не могут быть изменены без согласия большинства участников,

что защищает контракты от подделки и мошенничества.

3. **Надежность:** Благодаря механизмам консенсуса и криптографии, блокчейн обеспечивает надежное выполнение смарт-контрактов без необходимости полагаться на доверие к одному участнику.

4. **Прозрачность:** Все транзакции и состояния смарт-контрактов общедоступны и могут быть проверены любым участником сети, обеспечивая высокий уровень прозрачности.

5. **Отсутствие посредников:** Благодаря использованию блокчейна, смарт-контракты позволяют участникам обходить посредников и сокращать затраты на их услуги.

Все эти аспекты делают блокчейн фундаментальной технологией для реализации смарт-контрактов. Понимание принципов работы блокчейна поможет вам более глубоко осознать, как смарт-контракты функционируют и как их применять на практике.

2.2: Платформа Ethereum и её роль в смарт-контрактах

Ethereum – это платформа, основанная на технологии блокчейна, которая играла ключевую роль в развитии и популяризации смарт-контрактов. Основанная в 2015 году Виталиком Бутериным, Ethereum предложила совершенно новый подход к использованию блокчейн-технологии, позволяя программировать и выполнять смарт-контракты в децентрализованной среде.

Ключевые характеристики Ethereum:

1. **Смарт-контракты:** Одной из основных инноваций Ethereum стали смарт-контракты. Смарт-контракты – это самовыполняющиеся программы, которые выполняются автоматически, когда выполняются определенные условия. Они позволяют создавать и автоматизировать различные виды сделок и операций, включая финансовые транзакции, управление активами, голосования и многое другое.
2. **Тьюринг-полнота:** Ethereum является тьюринг-полной платформой, что означает, что на ней можно создавать сложные и мощные смарт-контракты с помощью полноценных программированных языков, таких как Solidity.
3. **Ether (ETH):** Эфир – это криптовалюта, используемая в сети Ethereum. Она играет роль топлива для выполнения транзакций и выполнения смарт-контрактов. Участники сети платят небольшие суммы эфира за использование вычислительных ресурсов сети (газ) и за транзакции.
4. **Децентрализация:** Ethereum стремится к децентрализации, что означает отсутствие центрального контроля и участие множества независимых узлов в обработке транзакций и поддержке сети. Это делает сеть более устойчивой к цензуре и сбоям.
5. **Ethereum Virtual Machine (EVM):** EVM – это виртуальная машина, которая исполняет смарт-контракты на Ethereum. Она обеспечивает изоляцию и безопасность выполнения контрактов, позволяя разработчикам создавать

комплексные децентрализованные приложения.

Роль Ethereum в смарт-контрактах:

Ethereum предоставляет платформу и инфраструктуру для разработки, развертывания и выполнения смарт-контрактов. Смарт-контракты на Ethereum имеют широкий спектр применений:

1. **Финансовые приложения:** Смарт-контракты Ethereum позволяют создавать децентрализованные финансовые инструменты, такие как криптовалютные кошельки, стейблкоины, децентрализованные биржи и кредитные протоколы DeFi.
2. **Игры и развлечения:** Разработчики могут создавать смарт-контракты для децентрализованных игр, коллекционных предметов и виртуальных миров, предоставляя пользователям большую автономию и контроль над своими активами.
3. **Управление активами:** Смарт-контракты позволяют создавать цифровые активы, такие как токены и жетоны, а также программировать условия и правила их передачи и управления.
4. **Голосования и управление:** Ethereum поддерживает создание смарт-контрактов для голосований, выборов и управления, обеспечивая прозрачность и устойчивость к манипуляциям.
5. **Цифровые идентификаторы:** Смарт-контракты Ethereum могут быть использованы для создания систем

цифровой идентификации, подтверждая личность и авторизацию без необходимости централизованных посредников.

Ethereum оказало огромное влияние на развитие блокчейн-технологии и смарт-контрактов, став основой для множества инновационных проектов и приложений. Понимание роли Ethereum в смарт-контрактах является важным шагом для всех, кто хочет войти в мир децентрализованных приложений и услуг.

2.3: Ether (ETH) и газ: важные понятия для понимания работы смарт-контрактов

Смарт-контракты, основанные на платформе Ethereum, являются ключевой составляющей децентрализованных приложений и услуг. Для того чтобы полностью понять, как работают смарт-контракты, необходимо разобраться в двух важных понятиях: Ether (ETH) и газе. Эти концепции играют критическую роль в процессе создания, развертывания и выполнения смарт-контрактов.

Ether (ETH):

Ether (ETH) – это криптовалюта, которая используется в сети Ethereum как средство обмена и хранения стоимости. Эфир можно сравнить с «топливом» для смарт-контрактов и транзакций в сети. Когда вы выполняете транзакцию или вызываете метод смарт-контракта, вы должны заплатить некоторое количество эфира в качестве вознаграждения для майнеров сети, которые обрабатывают вашу транзакцию. Это обеспечивает инцентив для майнеров поддерживать работу

сети Ethereum.

Газ:

Газ – это ещё одно важное понятие в мире Ethereum. Газ измеряет количество вычислительных ресурсов, необходимых для выполнения определенной операции в смарт-контракте или для обработки транзакции. Каждая операция в смарт-контракте или транзакция имеет свой уровень газа, который указывает на то, сколько вычислительной мощности и времени требуется для её выполнения.

При отправке транзакции или вызове смарт-контракта вы указываете лимит газа и стоимость газа (в эфирах) для этой операции. Если операция требует больше газа, чем указано в лимите, то она автоматически отменяется, но уже заплаченные эфиры за газ не возвращаются.

Стоимость газа:

Стоимость газа определяется рыночной динамикой и зависит от спроса и предложения в сети Ethereum. В нормальных условиях, более сложные операции требуют больше газа для выполнения, и следовательно, их стоимость будет выше. Разработчики смарт-контрактов и пользователи должны следить за стоимостью газа, чтобы оптимизировать расходы при использовании смарт-контрактов.

Заключение:

Понимание Ether и газа является фундаментальным для работы с смарт-контрактами. Ether служит валютой и средством для оплаты транзакционных сборов, а газ позволяет

измерить и оптимизировать вычислительные затраты в сети Ethereum. Разработчики и пользователи должны учитывать стоимость газа и эфира, чтобы обеспечить эффективное функционирование своих смарт-контрактов и транзакций.

Глава 3: Solidity: Язык программирования для смарт-контрактов

3.1 Введение в Solidity

Смарт-контракты – это программные сущности, созданные для автоматизации и выполнения условий соглашений на блокчейне. Чтобы создавать смарт-контракты, разработчику необходимо использовать специальный язык программирования, который позволяет описать логику контракта и его взаимодействие с другими смарт-контрактами и участниками блокчейна. Один из наиболее популярных языков для разработки смарт-контрактов на платформе Ethereum – Solidity.

Solidity – это высокоуровневый язык программирования, специально разработанный для написания смарт-контрактов. Он сочетает в себе элементы известных языков программирования, таких как JavaScript, Python и C++, и обеспечивает разработчикам инструменты для создания сложных и надежных смарт-контрактов.

Основные черты Solidity:

Типизированность: Solidity поддерживает статическую типизацию, что означает, что каждая переменная должна быть объявлена с определенным типом данных (например,

uint, **address**, **bool**, **string** и др.). Это помогает предотвратить ошибки во время выполнения контракта.

Контракты и наследование: В Solidity вы можете создавать контракты, которые могут наследовать свойства и методы других контрактов. Это позволяет создавать модульные и переиспользуемые компоненты.

Функции и модификаторы: Вы можете определять функции в контрактах, которые выполняют определенные действия. Кроме того, модификаторы позволяют применять определенные правила и проверки к функциям.

События: С Solidity вы можете определять события, которые позволяют контракту "сообщать" об определенных действиях или изменениях в блокчейне. Это полезно для мониторинга состояния контракта извне.

Хранилище данных: Контракты могут содержать переменные, которые служат для хранения данных на блокчейне. Эти переменные могут быть публичными, приватными или защищенными.

Модификаторы доступа: Solidity предоставляет модификаторы доступа, такие как **public**, **private**, **internal** и **external**, которые определяют, какие части кода могут взаимодействовать с определенными функциями или переменными.

Основы Solidity представляют собой солидную основу для разработки смарт-контрактов. В дальнейших главах этой книги мы будем более подробно изучать особенности языка

Solidity, понимать, как создавать и тестировать смарт-контракты, а также изучим лучшие практики для обеспечения безопасности и надежности ваших контрактов.

3.2: Синтаксис и структура контрактов на Solidity

Синтаксис и структура смарт-контрактов на языке программирования Solidity играют ключевую роль в создании эффективных и надежных контрактов. В этой части мы разберем основные элементы синтаксиса и структуры контрактов на Solidity.

Контракты и Версии Solidity: Создание контракта начинается с указания версии Solidity. Это важно, так как новые версии языка могут включать дополнительные функции и исправления ошибок. Пример объявления версии:

```
pragma solidity ^0.8.0;
```

Структура контракта: Контракт в Solidity имеет структуру, включающую в себя переменные состояния, функции, события и модификаторы. Основная структура контракта выглядит следующим образом:

```
contract MyContract {
....// Переменные состояния
....uint256 public myVariable;

....// Конструктор контракта (необязателен)
....constructor(uint256 initialValue) {
........myVariable = initialValue;
....}
```

```
    ....// Функции
    ....function setMyVariable(uint256 newValue) public {
    ........myVariable = newValue;
    ....}

    ....function getMyVariable() public view returns (uint256) {
    ........return myVariable;
    ....}

    ....// События
    ....event ValueChanged(uint256 newValue);

    ....// Модификаторы
    ....modifier onlyOwner() {
    ........require(msg.sender == owner, "Only the owner can call
this function");
    ........_;
    ....}
    }
```

Переменные состояния: Переменные состояния хранят данные на блокчейне и являются постоянными для жизни контракта. Они объявляются внутри контракта и могут иметь различные типы данных (uint, int, address, bool и др.), а также модификаторы доступа (public, internal, private).

Функции: Функции представляют собой операции, кото-

рые могут выполняться с контрактом. Они могут иметь входные параметры и возвращать значения. Функции могут изменять состояние контракта или просто возвращать информацию (view функции). Также есть функции, которые изменяют состояние, но не генерируют транзакции (pure функции).

События: События используются для логирования важных событий в контракте. Они позволяют приложениям и внешним сервисам отслеживать изменения в контракте. События объявляются в контракте и могут иметь параметры.

Модификаторы: Модификаторы позволяют вам выполнять проверки перед выполнением функций. Они используются для повышения безопасности и контроля доступа. Например, модификатор "onlyOwner" в приведенном выше примере позволяет вызывать функцию только владельцу контракта.

Конструктор контракта: Конструктор – это специальная функция, которая вызывается при развертывании контракта. Он может принимать параметры и использоваться для инициализации переменных состояния.

С помощью этой структуры и синтаксиса вы можете создавать мощные смарт-контракты на языке Solidity. Основная идея заключается в объявлении переменных состояния, определении функций для управления этим состоянием, использовании событий для логирования событий и применении модификаторов для обеспечения безопасности и кон-

троля доступа к функциям контракта.

3.3: Типы данных, переменные и функции в Solidity

В языке программирования Solidity, который используется для написания смарт-контрактов, основными строительными блоками являются типы данных, переменные и функции. Понимание этих элементов критически важно для успешной разработки и взаимодействия с смарт-контрактами. В этом разделе мы более подробно рассмотрим эти концепции.

Типы данных в Solidity: Solidity поддерживает разнообразные типы данных, которые определяют, какие виды информации могут быть хранены и обрабатываться в смарт-контрактах. Некоторые из основных типов данных:

1. **Целочисленные типы (int, uint):** Позволяют хранить целые числа со знаком (int) и без знака (uint) разных размеров (например, int8, uint256).
2. **Адреса (address):** Используются для представления адресов кошельков или других смарт-контрактов на блокчейне Ethereum.
3. **Логический тип (bool):** Может иметь значение true или false.
4. **Фиксированные и дробные числа (fixed, ufixed):** Позволяют работать с десятичными числами с фиксированной точностью.
5. **Строки (string) и байтовые последовательности (bytes):** Используются для хранения текстовых данных или

последовательностей байтов.

6. **Массивы:** Позволяют группировать однотипные данные в список.

7. **Структуры (struct):** Позволяют объединять различные типы данных в пользовательские типы.

8. **Перечисления (enum):** Позволяют определить список именованных значений.

Переменные: Переменные в Solidity представляют собой именованные контейнеры для хранения данных определенного типа. Они используются для временного хранения информации внутри смарт-контракта. Пример объявления переменной:

uint256 public totalSupply;

В данном примере объявлена публичная переменная **totalSupply** типа **uint256**, которая будет хранить общее количество какой-либо единицы.

Функции: Функции в смарт-контрактах выполняют код и могут иметь параметры и возвращаемые значения. Они позволяют взаимодействовать с данными в контракте и выполнять определенные действия. Пример объявления функции:

function transfer(address _to, uint256 _amount) public returns (bool) {

....// Логика передачи токенов

}

В данном примере объявлена публичная функция **transfer**, принимающая два параметра: **_to** (адрес получате-

ля) и **_amount** (количество токенов для передачи). Функция также объявляет, что она будет возвращать значение типа **bool**.

Модификаторы доступа: Solidity предоставляет модификаторы доступа, которые определяют, как функции могут быть вызваны извне. Некоторые распространенные модификаторы:

1. **public:** Функция может быть вызвана из любого контракта или внешнего аккаунта.
2. **internal:** Функция может быть вызвана только из контракта, где она определена, и из его наследующих контрактов.
3. **private:** Функция может быть вызвана только из контракта, где она определена.

Пример объединения всего вместе:

```
pragma solidity ^0.8.0;

contract MyContract {
....uint256 public myNumber; // Переменная

....constructor(uint256 _initialNumber) {
........myNumber = _initialNumber;
....}

....function setNumber(uint256 _newNumber) public {
........myNumber = _newNumber;
```

....}

....function getNumber() public view returns (uint256) {
.....return myNumber;
....}
}

В этом примере мы создали контракт **MyContract**, который содержит переменную **myNumber**, функцию **setNumber** для обновления значения переменной и функцию **getNumber** для получения значения.

В этой главе мы рассмотрели базовые концепции типов данных, переменных и функций в Solidity. Понимание этих элементов позволит вам начать создавать более сложные смарт-контракты и эффективно взаимодействовать с данными на блокчейне Ethereum.

3.4: Управление данными и хранилищем

В этой части мы погрузимся в детали управления данными и хранилищем в смарт-контрактах, изучив, как хранить и обрабатывать информацию в блокчейне при помощи языка программирования Solidity.

3.4.1 Типы данных и переменные

Для эффективной работы с смарт-контрактами на Solidity важно хорошо понимать различные типы данных и какие возможности они предоставляют. В этом разделе мы подробно рассмотрим основные типы данных в Solidity и примеры их использования.

Целочисленные типы данных (uint и int)

Целочисленные типы данных используются для представления чисел без десятичной части (целых чисел). В Solidity есть беззнаковые и знаковые целочисленные типы данных:

- **uint**: беззнаковое целое число. Например, **uint256** представляет целое число без знака, состоящее из 256 битов (32 байта).
- **int**: знаковое целое число. Например, **int8** представляет знаковое целое число, использующее 8 битов (1 байт).

Пример объявления и использования целочисленных переменных:

```
uint256 public totalSupply;
int8 public temperature;
totalSupply = 100000; // Присвоение значения переменной
temperature = -10;.... // Присвоение другого значения переменной
```

Логический тип данных (bool)

Логический тип данных **bool** может принимать только два значения: **true** (истина) или **false** (ложь). Логические переменные часто используются для контроля потока выполнения программы при помощи условий.

Пример использования логической переменной:

```
bool public isActivated;
isActivated = true;.... // Присвоение значения переменной
if (isActivated) {
....// Выполнить код, если isActivated равно true
```

}

Адресный тип данных (address)

Тип данных **address** предназначен для хранения адресов кошельков Ethereum. С его помощью можно отслеживать владельцев аккаунтов и взаимодействовать с другими смарт-контрактами и адресами.

Пример использования адресного типа данных:

address public owner;

owner = msg.sender;....// Присвоение адреса отправителя транзакции переменной

address recipient = 0xAbCdEf0123456789; //Присвоение адреса переменной

Строковый тип данных (string)

Тип данных **string** используется для хранения переменной длины строковых значений. Обратите внимание, что операции над строками могут потреблять больше газа, чем операции с числами, так как строки более сложные для обработки в блокчейне.

Пример использования строковой переменной:

string public message;

message = "Hello, world!"; //Присвоение строки переменной

Массивы (array)

Массивы в Solidity позволяют объединять несколько значений одного типа в одной переменной. Они могут быть фиксированной длины (размер задается заранее) или динамической длины (размер определяется в процессе выполне-

ния).

Примеры использования массивов:

uint256[5] public numbers;.... // Массив фиксированной длины

string[ ] public names;..... // Динамический массив строк

numbers = [10, 20, 30, 40, 50]; // Присвоение значений фиксированному массиву

names.push("Alice");..... // Добавление значения в динамический массив

Структуры (struct)

Структуры позволяют создавать пользовательские типы данных, объединяя различные поля. Это удобно, когда нужно хранить связанные данные в одной переменной.

Пример использования структуры:

struct Person {
....string name;
....uint256 age;
}

Person public alice;

alice = Person("Alice", 30); // Инициализация структуры

Основные типы данных Solidity позволяют эффективно хранить и обрабатывать информацию в смарт-контрактах. При создании смарт-контрактов важно правильно выбирать тип данных в зависимости от характера данных и требований к работе контракта.

3.4.2 Хранилище данных

В смарт-контрактах управление данными является фундаментальной задачей. Данные могут быть различных типов – от чисел и строк до более сложных структур. Понимание того, как работает хранилище данных, поможет вам создавать эффективные и безопасные смарт-контракты.

Переменные состояния и локальные переменные

Один из ключевых аспектов управления данными – это различие между переменными состояния (state variables) и локальными переменными (local variables).

Переменные состояния хранят данные на блокчейне и сохраняют свои значения между вызовами функций. Они могут быть доступны для всех, кто читает состояние контракта. Эти переменные широко используются для хранения постоянных данных, таких как балансы пользователей, настройки контракта и другие глобальные параметры.

Локальные переменные, с другой стороны, существуют только внутри функции и исчезают после её выполнения. Они используются для временных вычислений и обработки данных внутри функций. Локальные переменные полезны, когда вам нужно временно хранить промежуточные результаты вычислений или выполнять действия внутри функции, не оставляя следов в состоянии контракта.

Пример использования переменных состояния и локальных переменных

Давайте представим, что у нас есть смарт-контракт для

управления простым токеном. Мы хотим хранить общее количество выпущенных токенов и балансы каждого адреса.

```
contract SimpleToken {
....uint256 public totalSupply;
....mapping(address => uint256) public balances;

....constructor(uint256 initialSupply) {
........totalSupply = initialSupply;
........balances[msg.sender] = initialSupply;
....}

....function transfer(address to, uint256 amount) public {
........require(balances[msg.sender] >= amount, "Not enough balance");
........balances[msg.sender] -= amount;
........balances[to] += amount;
```

Конец ознакомительного фрагмента.