1. https://cosmos.network/intro
2. https://docs.cosmos.network/v0.42/basics/app-anatomy.html
3. https://docs.cosmos.network/v0.42/basics/tx-lifecycle.html
4. https://docs.cosmos.network/v0.42/core/grpc_rest.html
5. https://docs.cosmos.network/v0.42/building-modules/intro.html
6. https://docs.cosmos.network/v0.42/building-modules/messages-and-queries.html
7. https://docs.cosmos.network/v0.42/building-modules/query-services.html
8. https://docs.cosmos.network/v0.42/migrations/app_and_modules.html
9. https://docs.cosmos.network/v0.42/ibc/overview.html#
10. https://docs.cosmos.network/v0.42/ibc/integration.html
11. -
12. https://docs.cosmos.network/v0.42/ibc/relayer.html
13. https://docs.tendermint.com/master/introduction/what-is-tendermint.html
14. https://docs.tendermint.com/master/tutorials/go.html
15. https://docs.tendermint.com/master/app-dev/getting-started.html
16. https://docs.cosmos.network/v0.42/ibc/#ibc
17. https://docs.tendermint.com/master/app-dev/indexing-transactions.html
18. https://tutorials.cosmos.network/voter/
19. https://tutorials.cosmos.network/burner-chain/01-welcome.html
20. https://tutorials.cosmos.network/burner-chain/02-buffidao.html

1. Что такое Cosmos?

Вступление

Строго говоря, Cosmos - это децентрализованная сеть независимых параллельных блокчейнов, каждый из которых работает на алгоритмах консенсуса BFT, таких как консенсус Tendermint.

Другими словами, Cosmos - это экосистема блокчейнов, которые могут масштабироваться и взаимодействовать друг с другом. До Cosmos'а блокчейны были изолированными и не могли взаимодействовать друг с другом. Их было сложно построить, и они могли обрабатывать лишь небольшое количество транзакций в секунду. Cosmos решает эти проблемы с помощью нового технического видения. Чтобы понять это видение, нам нужно вернуться к основам технологии блокчейн.

Что такое блокчейн?

Блокчейн можно охарактеризовать как цифровой реестр, поддерживаемый набором валидаторов, который остается работоспособным, даже если некоторые из валидаторов (менее ⅓) являются вредоносными. Каждая сторона хранит копию реестра на своем компьютере и обновляет ее в соответствии с правилами, определенными протоколом, при получении блоков транзакций. Цель технологии блокчейн - убедиться, что реестр правильно воспроизведен, а это означает, что каждая не вредоносная сторона видит одну и ту же версию реестра в любой момент.

Основным преимуществом технологии блокчейн является возможность для сторон совместно использовать реестр, не полагаясь на централизованные органы. Блокчейны децентрализованы. Первое и самое известное применение технологии блокчейн сегодня - это децентрализованная валюта Биткойн.

Теперь, когда у нас есть лучшее понимание того, что такое блокчейн с точки зрения высокого уровня, давайте посмотрим на определение блокчейна с более технического угла. Блокчейн - это детерминированный конечный аппарат, воспроизведенный на полноценных узлах системы, который сохраняет консенсусную безопасность до тех пор, пока менее трети его сопровождающих являются византийцами. Давайте разберемся с этим.

Конечный аппарат - это просто причудливое слово для программы, которая хранит состояние и изменяет его, когда получает входные данные. Существует состояние, которое может представлять разные вещи в зависимости от приложения (например, балансы токенов для криптовалюты) и транзакции, которые изменяют состояние (например, путем вычитания балансов из одной учетной записи и добавления их в другую).

Детерминированный означает, что если вы воспроизводите одни и те же транзакции из одного и того же состояния генезиса, вы всегда будете иметь одно и то же результирующее состояние.

Безопасность консенсуса относится к тому факту, что каждая не вредоносная нода, на котором воспроизводится конечный аппарат, должен видеть одно и то же состояние в одно и то же время. Когда узлы получают блоки транзакций, они проверяют, что они действительны, что означает, что каждая транзакция действительна и что сам блок был проверен более чем ⅔ сопровождающих, называемых валидаторами. Безопасность будет гарантирована, если менее ⅓ валидаторов будут византийскими, то есть вредоносными.

Картинка 1: (сверху вниз)
Уровень логики приложения
Уровень сетевого взаимодействия (нетворкинга)
Уровень консенсуса

С точки зрения архитектуры блокчейны можно разделить на три концептуальных уровня:

- Логика приложения: отвечает за обновление состояния для набора транзакций, то есть за обработку транзакций.
- Сеть (нетворкинг): отвечает за распространение транзакций и сообщений, связанных с консенсусом.
- Консенсус: позволяет узлам согласовывать текущее состояние системы.

Конечный аппарат является схожим с уровнем логики приложения. Он определяет состояние приложения и функции перехода между состояниями. Другие уровни отвечают за репликацию конечного аппаратана всех узлах, которые подключаются к сети.

Как Cosmos дополняет широкую экосистему блокчейнов?

История Биткоина (Блокчейн 1.0)

Чтобы понять, как Cosmos дополняет экосистему блокчейнов, нам нужно вернуться к началу истории блокчейнов. Первым блокчейном был и остается Биткойн - одноранговая цифровая валюта, созданная в 2008 году, в которой использовался новый механизм консенсуса, известный как Proof-of-Work (PoW). Это было первое децентрализованное приложение на блокчейне. Вскоре люди начали осознавать потенциал децентрализованных приложений, и в сообществе появилось желание создавать новые сети.

В то время было два варианта разработки децентрализованных приложений: либо форк кодовой базы Биткойна, либо построение каких-либо приложений на ней. Однако кодовая база Биткоина была весьма монолитной; все три уровня - сеть, консенсус и логика приложения - были смешаны вместе. Кроме того, язык сценариев Биткойна был ограничен и неудобен для пользователя. Нужны были инструменты получше.

История Ethereum (Блокчейн 2.0)

В 2014 году Ethereum представил новое предложение по созданию децентрализованных приложений. Будет единый блокчейн, в котором люди смогут развертывать любые программы. Ethereum добился этого, превратив уровень логики приложения в виртуальную машину, называемую виртуальной машиной Ethereum (EVM). Эта виртуальная машина могла обрабатывать программы, называемые смарт-контрактами, которые любой разработчик мог развернуть в блокчейне Ethereum без дополнительный разрешений от каких-либо централизованных посредников. Этот новый подход позволил тысячам разработчиков начать создавать децентрализованные приложения (dApps). Однако вскоре ограничения этого подхода стали очевидны и сохраняются по сей день.

Ограничение №1: масштабируемость
Первое ограничение - это масштабирование: децентрализованные приложения, созданные на основе Ethereum, ограничиваются общей скоростью 15 транзакций в секунду. Это связано с тем, что Ethereum по-прежнему использует Proof-of-Work и что децентрализованные приложения Ethereum конкурируют за ограниченные ресурсы одного блокчейна.

Ограничение # 2: удобство использования
Второе ограничение - это относительно высокий порог вхождения для разработчиков. Поскольку EVM - это песочница, которая должна учитывать все варианты использования, она оптимизируется для среднего варианта развития приложений. Это означает, что разработчики должны идти на компромиссы в отношении дизайна и эффективности своего приложения (например, требуя использования модели учетной записи в платежной платформе, где модель UTXO может быть предпочтительнее). Среди прочего, они ограничены несколькими языками программирования и не могут реализовать автоматическое выполнение кода.

Ограничение # 3: Обособленность цепи
Третье ограничение заключается в том, что каждое приложение имеет ограниченную обособленность, поскольку все они используют одну и ту же базовую среду. По сути, это создает два уровня управления: уровень логики приложения и уровень базовой среды. Первое ограничено вторым. Если в приложении есть ошибка, с ней ничего нельзя сделать без одобрения руководства самой платформы Ethereum. Если приложению требуется новая функция в EVM, оно снова должно полностью полагаться на управление платформой Ethereum, чтобы принять ее.

Эти ограничения относятся не только к Ethereum, но и ко всем блокчейнам, пытающимся создать единую платформу, подходящую для всех случаев использования. Вот где в игру вступает Cosmos.

История Cosmos (Блокчейн 3.0)

Видение Cosmos состоит в том, чтобы упростить разработчикам создание блокчейнов и преодолеть барьеры между блокчейнами, позволяя им совершать транзакции в общей среде. Конечная цель - создать версию интернета блокчейнов, сеть блокчейнов, способных децентрализованно взаимодействовать друг с другом. С помощью Cosmos блокчейны могут поддерживать обособленное существование, быстро обрабатывать транзакции и взаимодействовать с другими блокчейнами в экосистеме, что делает его оптимальным для различных случаев использования.

Это видение достигается с помощью набора инструментов с открытым исходным кодом, таких как Tendermint Core, Cosmos SDK и протокол IBC, предназначенных для того, чтобы люди могли быстро создавать собственные, безопасные, масштабируемые и совместимые блокчейн-приложения. Давайте подробнее рассмотрим некоторые из наиболее важных инструментов в экосистеме, а также техническую архитектуру сети Cosmos. Обратите внимание, что Cosmos - это проект сообщества с открытым исходным кодом, изначально созданный командой Tendermint. Каждый может создать дополнительные инструменты, чтобы обогатить большую экосистему разработчиков.

Что такое Tendermint BFT и ABCI

До недавнего времени для построения блокчейна требовалось строить все три уровня (сеть, консенсус и логика приложения) с нуля. Ethereum упростил разработку децентрализованных приложений, предоставив блокчейн виртуальной машины, на котором любой мог развернуть пользовательскую логику в форме смарт-контрактов. Однако это не упростило разработку самих блокчейнов. Как и Биткойн, Go-Ethereum остается монолитным технологическим стеком, который сложно разветвить и настроить. Именно здесь появился Tendermint, созданный Джэ Квоном в 2014 году.

Картинка #3: (Сверху-вниз)
Язык программирования / Java, C++ и другие
ABCI
Консенсус
Networking

Tendermint BFT - это решение, которое объединяет сетевой и консенсусный уровни блокчейна в общий механизм, позволяющий разработчикам сосредоточиться на разработке логики приложения, а не на сложном базовом протоколе. В результате Tendermint экономит сотни часов разработки проекта. Обратите внимание, что Tendermint также обозначает название алгоритма византийского отказоустойчивого консенсуса (BFT), используемого в движке Tendermint BFT. Чтобы узнать больше об истории консенсусных протоколов и BFT, вы можете проверить этот крутой подкаст соучредителя Tendermint Итана Бухмана.

Движок Tendermint BFT подключен к приложению через протокол сокетов, называемый Application Blockchain Interface (ABCI). Этот протокол может быть заключен в любой язык программирования, что позволяет разработчикам выбирать язык, соответствующий их потребностям.

Но это еще не все. Вот свойства, которые делают Tendermint BFT современным блокчейн-движком:

- Поддержка частных и публичных блокчейнов: Tendermint BFT обрабатывает только сеть и консенсус для блокчейна, что означает, что он помогает узлам распространять транзакции, а валидаторы согласовывают набор транзакций для добавления в блокчейн. Роль прикладного уровня состоит в том, чтобы определить, как составляется набор валидаторов. Таким образом, разработчики могут создавать как публичные, так и частные блокчейны на базе движка Tendermint BFT. Если приложение определяет, что валидаторы выбираются на основе того, сколько токенов у них содержится в стейке, тогда блокчейн можно охарактеризовать как Proof-of-Stake (PoS). Если, однако, приложение определяет, что только ограниченный набор предварительно авторизованных объектов может быть валидаторами, тогда блокчейн можно охарактеризовать как выборочный или частный. Разработчики имеют полную свободу настраивать правила, определяющие, как изменяется набор валидаторов в их блокчейне.

- Высокая производительность: Tendermint BFT может дсотигать скорости 1 блок / сек и обрабатывать до тысяч транзакций в секунду.

- Мгновенные подтверждения: свойство алгоритма консенсуса Tendermint - мгновенное подтверждение. Это означает, что форки никогда не создаются, если более трети валидаторов являются честными (византийскими). Пользователи могут быть уверены, что их транзакции подтверждены, как только блок будет создан (чего нельзя сказать о блокчейнах Proof-of-Work, таких как Биткойн и Ethereum).

- Безопасность: консенсус Tendermint не только отказоустойчив, но и подотчетен. Если блокчейн разветвляется, всегда найдется способ определить ответственность.

Cosmos SDK и другие платформы уровня логики приложений

Картинка #4: Сверху-вниз, слева-направо
Общественное управление, Стейкинг
Протокол IBC, Штрафы
ABCI
Консенсус
Сетевое взаимодействие

Tendermint BFT сокращает время разработки блокчейна с нескольких лет до недель, но создание безопасного ABCI-приложения с нуля все еще остается сложной задачей. Вот почему существует Cosmos SDK.

Cosmos SDK - это обобщенная структура, которая упрощает процесс создания безопасных приложений блокчейна поверх Tendermint BFT. Он основан на двух основных принципах:

- Модульность: цель Cosmos SDK - создать экосистему модулей, которая позволяет разработчикам легко запускать блокчейны для конкретных приложений без необходимости кодировать каждый бит функциональности своего приложения с нуля. Любой может создать модуль для Cosmos SDK, а использовать готовые встроенные модули в вашем блокчейне так же просто, как импортировать их в свое приложение. Например, команда Tendermint создает набор базовых модулей, необходимых для Cosmos Hub. Эти модули могут использоваться любым разработчиком при создании собственного приложения. Кроме того, разработчики могут создавать новые модули для настройки своего приложения. По мере развития сети Cosmos экосистема модулей SDK будет расширяться, что упростит разработку сложных блокчейн-приложений.

- Безопасность, основанная на возможностях: возможности ограничивают границы безопасности между модулями, позволяя разработчикам лучше разбираться в компоновке модулей и ограничивать объем злонамеренных или неожиданных взаимодействий. Для более подробного ознакомления с возможностями щелкните здесь.

Cosmos SDK также поставляется с набором полезных инструментов разработчика для создания интерфейсов командной строки (CLI), серверов REST и множества других часто используемых служебных библиотек.

И последнее замечание: Cosmos SDK, как и все инструменты Cosmos, спроектирован как модульный. Сегодня он позволяет разработчикам строить поверх Tendermint BFT. Однако его можно использовать с любыми другими механизмами консенсуса, реализующими ABCI. Со временем мы ожидаем появления нескольких SDK, построенных с использованием разных моделей архитектуры и совместимых с несколькими механизмами консенсуса - и все это в рамках единой экосистемы: Cosmos Network.

Чтобы узнать, как кодировать приложения поверх SDK, ознакомитесь с учебными пособиями.

Ethermint

Самым замечательным в Cosmos SDK является то, что его модульность позволяет разработчикам переносить поверх него практически любую существующую кодовую базу блокчейна, уже существующую в Golang. Например, Ethermint - это проект, который портирует виртуальную машину Ethereum в модуль SDK. Ethermint работает точно так же, как Ethereum, но также обладает всеми свойствами Tendermint BFT. Все существующие инструменты Ethereum (Truffle, Metamask и так далее) уже нативно совместимы с Ethermint, и вы можете переносить свои смарт-контракты без дополнительной работы.

Зачем создавать блокчейн с Cosmos SDK, если я могу просто развернуть свое децентрализованное приложение поверх блокчейна виртуальной машины?

Этот вопрос оправдан, учитывая, что большинство децентрализованных приложений сегодня разрабатываются на основе блокчейнов виртуальных машин, таких как Ethereum. Во-первых, следует сказать, что причина этого явления в том, что до сих пор блокчейны было намного сложнее разработать, чем смарт-контракты. Благодаря Cosmos SDK это больше не актуально. Теперь разработчики могут легко разрабатывать целые блокчейны для конкретных приложений, которые имеют несколько преимуществ. Среди прочего, они обеспечивают большую гибкость, безопасность, производительность и обособленность. Чтобы узнать больше о блокчейнах для конкретных приложений, прочтите этот пост. Конечно, если вы не хотите создавать собственный блокчейн, вы все равно можете сделать свои смарт-контракты совместимыми с Cosmos, развернув их на Ethermint.

Соединяем блокчейны вместе - Протокол IBC

Теперь, когда у разработчиков есть способ быстро создавать индивидуальные блокчейны, давайте посмотрим, как соединить эти блокчейны вместе. Связь между блокчейнами достигается через протокол, называемый протоколом межблочной связи (IBC). Протокол IBC использует свойство мгновенного подтверждения консенсуса Tendermint (хотя он может работать с любым механизмом блокчейна «быстрого подтверждения), чтобы позволить гетерогенным блокчейнам передавать ценность (то есть токены) или данные друг другу.

Что такое гетерогенные блокчейны?

По сути, это сводится к двум вещам:

- Разные уровни работы. Гетерогенные блокчейны имеют разные уровни, то есть они могут различаться по способу реализации сетевой, консенсусной и прикладной частей. Чтобы быть совместимой с протоколом IBC, блокчейн должен соответствовать только нескольким требованиям, главное из которых состоит в том, что уровень консенсуса должен иметь свойство быстрого подтверждения. Блокчейны Proof-of-Work (такие как Биткойн и Ethereum) не попадают в эту категорию, поскольку они имеют вероятностное подтверждение.

- Обособленность: каждый блокчейн поддерживается набором валидаторов, чья работа заключается в согласовании следующего блока для утверждения в блокчейне. В блокчейнах Proof-of-Work эти валидаторы называются майнерами. Обособленный блокчейн - это блокчейн со своим собственным набором валидаторов. Во многих случаях важно, чтобы блокчейны были обособленными, поскольку валидаторы в конечном итоге несут ответственность за изменение состояния. В Ethereum все приложения запускаются общим набором валидаторов. Из-за этого каждое приложение имеет ограниченную степень обособленности.

Протокол IBC позволяет гетерогенным блокчейнам передавать токены и данные друг другу, а это означает, что блокчейны с различными приложениями и наборами валидаторов совместимы. Например, он позволяет публичным и частным блокчейнам передавать токены друг другу. В настоящее время никакая другая структура блокчейна не обеспечивает такой уровень взаимодействия.

Как работает протокол IBC?

Принцип, лежащий в основе IBC, довольно прост. Давайте рассмотрим пример, когда учетная запись в блокчейне A хочет отправить 10 токенов (назовем их ATOM) в блокчейн B.

- Отслеживание
  Блокчейн B постоянно получает заголовки блокчейна A, и наоборот. Это позволяет каждому блокчейну отслеживать набор валидаторов другой сети. По сути, каждый блокчейн управляет легким клиентом другой сети.

- Залог
  Когда инициируется передача протоколом IBC, ATOM блокируется (остается в залоге) в блокчейне A.

- Обмен состояниями
  Затем доказательство того, что 10 ATOM остались в залоге, передается от блокчейна A к блокчейну B.

- Проверка
  Доказательство проверяется в блокчейне B по заголовку блокчейна A, и, если он действителен, то в блокчейне B создаются 10 ATOM-ваучеров.

Обратите внимание, что ATOM, которые были созданы в блокчейне B, не являются настоящими ATOM, поскольку ATOM существует только в блокчейне A. Они представляют собой аналог ATOM из блокчейна A на B вместе с доказательством того, что эти ATOM заморожены в блокчейне A.

Аналогичный механизм используется для разблокировки ATOM, когда они возвращаются к своему исходному блокчейну. Более полное описание протокола IBC можно найти в этой спецификации.

Проектирование «Интернета блокчейнов»

IBC - это протокол, который позволяет двум разным блокчейнам передавать токены друг другу. Появляется вопрос, как нам создать сеть блокчейнов?

Одна из идей - связать каждый блокчейн в сети друг с другом через прямые соединения протокола IBC. Основная проблема с этим подходом заключается в том, что количество подключений в сети растет квадратично с количеством блокчейнов. Если в сети 100 блокчейнов, и каждый должен поддерживать соединение IBC друг с другом, это 4950 соединений. Такая система довольно быстро выйдет из-под контроля.

Чтобы решить эту проблему, Cosmos предлагает модульную архитектуру с двумя классами блокчейнов: Базами и Зонами. Зоны - это обычные гетерогенные блокчейны, а базы- это блокчейны, специально предназначенные для соединения зон друг с другом. Когда Зона создает соединение протокола IBC с Базой, она может автоматически получать доступ (т.е. отправлять и получать данные) любой другой Зоне, которая подключена к ней. В результате каждой Зоне необходимо установить ограниченное количество подключений с ограниченным набором Баз. Базы также предотвращают двойные траты между Зонами. Это означает, что когда Зона получает токен от Базы, ей нужно доверять только исходной Зоне этого токена и Базе.

Первой Базой, запущенной в сети Cosmos, - это База Cosmos. Cosmos Hub - это общедоступный блокчейн Proof-of-Stake, чей собственный токен для размещения ставок называется ATOM, а комиссии за транзакции будут оплачиваться несколькими токенами. Запуск Базы также знаменует запуск сети Cosmos.

Поддержка сетей, не относящихся к Tendermint

На данный момент архитектура Cosmos, которую мы представили, показывает, как могут взаимодействовать блокчейны на основе Tendermint. Но Cosmos не ограничивается блокчейнами Tendermint. Фактически, к Cosmos можно подключить любой блокчейн.

У нас есть два случая, которые следует различать: блокчейны с быстрым подтверждением и блокчейны с вероятностным подтверждением.

Блокчейны с быстрым подтверждением

Блокчейны, которые используют любые алгоритмы консенсуса с быстрым подтверждением, могут подключаться к Cosmos, адаптируя протокол IBC. Например, если Ethereum перейдет на Casper FFG (Friendly Finality Gadget), между ним и экосистемой Cosmos можно будет установить прямое соединение, адаптировав протокол IBC для работы с Casper.

Блокчейны с вероятностным подтверждением

Для блокчейнов, которые не имеют быстрого подтверждения, таких как блокчейны Proof-of-Work, все становится немного сложнее. Для этих блокчейнов мы используем специальный вид прокси-блокчейна, называемый Зоной Привязки (Peg-Zone).

Зона Привязки (Peg-Zone) - это блокчейн, которая отслеживает состояние другого блокчейна. Сама Зона Привязки (Peg-Zone) является быстроотверждаемой и поэтому совместима с протоколом IBC. Его роль состоит в том, чтобы установить окончательность подтверждения, который он соединяет. Давайте посмотрим на следующий пример.

Пример: Зона Привязки Ethereum

Мы хотим связать блокчейн Ethereum с алгоритмом Proof-of-Work, чтобы можно было отправлять токены туда и обратно между сетями Ethereum и Cosmos. Поскольку Ethereum не обладает быстрым подтверждением, нам нужно создать Зону Привязки, которая будет служить мостом между ними.

Во-первых, Зона Привязки необходимо определить порог подтверждения исходного блокчейна. Например, он может считать, что данный блок исходного блокчейна является окончательно подтвержденным, если после него были добавлены 100 блоков.

Во-вторых, контракт развертывается на основном блокчейне Ethereum. Когда пользователи хотят отправить токены из Ethereum в Cosmos, они начинают с отправки токенов в этот контракт. Затем контракт замораживает активы, и после 100 блоков представление этих активов публикуется в Зоне Привязки. Аналогичный механизм используется для отправки активов обратно в блокчейн Ethereum.

Интересно, что Зона Привязки также позволяет пользователям отправлять любой токен, который существует в Cosmos, в блокчейн Ethereum (токены Cosmos будут представлены как ERC20 в блокчейне Ethereum). Команда Tendermint в настоящее время работает над реализацией Зоны Привязки для блокчейна Ethereum под названием Peggy.

Зону Привязки необходимо будет настроить для конкретного блокчейна, которую они соединяют. Создание Зоны Привязки Ethereum относительно просто, потому что Ethereum основан на учетной записи и имеет смарт-контракты. Однако создание Зоны Привязки Bitcoin немного сложнее. Объяснение того, как примерно создать Зону Привязки Биткойна, выходит за рамки этого вступления, но знайте, что это теоретически возможно. Если вы хотите узнать больше о Зонах Привязки, то взгляните на эту спецификацию.

Решение Масштабируемости

Теперь, когда мы можем легко создавать и подключать блокчейны, остается решить еще одну последнюю проблему: масштабируемость. Cosmos использует два типа масштабируемости:

- Вертикальная масштабируемость: сюда входят методы масштабирования самого блокчейна. Отойдя от Proof-of-Work и оптимизируя его компоненты, Tendermint BFT может обрабатывать тысячи транзакций в секунду. Самой проблемой чаще всего является само приложение. Например, такое приложение, как виртуальная машина (например, виртуальная машина Ethereum), налагает гораздо более низкий предел пропускной способности транзакции, чем приложение, в которое напрямую встроены типы транзакций и функции перехода между состояниями (например, стандартное приложение Cosmos SDK). Это одна из причин, почему блокчейны для конкретных приложений имеют смысл (подробнее о причинах читайте здесь).

- Горизонтальная масштабируемость: даже если механизм консенсуса и приложение сильно оптимизированы, в какой-то момент пропускная способность транзакций одного блокчейна неизбежно упирается в стену, которую она не может превзойти. Это предел вертикального масштабирования. Чтобы выйти за рамки этого, решение заключается в переходе к многоблокчейновой архитектуре. Идея состоит в том, чтобы иметь несколько параллельных блокчейнов, запускающих одно и то же приложение и управляемых общим набором валидаторов, что теоретически делает блокчейны бесконечно масштабируемыми. Подробности о горизонтальной масштабируемости довольно сложны и выходят за рамки этого вступления.

Cosmos предложит очень хорошую вертикальную масштабируемость при запуске, что само по себе станет значительным улучшением по сравнению с существующими решениями блокчейнов. Позже, после завершения модуля протокола IBC, будут реализованы решения горизонтальной масштабируемости.

Подытожим, что же все-таки есть Cosmos?

Надеюсь, теперь у вас есть более четкое представление о проекте Cosmos. Вот краткое изложение того, что такое Cosmos, в трех кратких деталях:

1. Cosmos делает блокчейны мощными и простыми в разработке с помощью Tendermint BFT и модульности Cosmos SDK.
2. Cosmos позволяет блокчейнам обмениваться ценностями друг с другом через протокол IBC и Зоны Привязки, сохраняя при этом свою обособленность.
3. Cosmos позволяет масштабировать приложения блокчейна для миллионов пользователей с помощью решений горизонтального и вертикального масштабирования.

Прежде всего, Cosmos - это не продукт, а экосистема, построенная на наборе модульных, адаптируемых и взаимозаменяемых инструментов. Разработчикам предлагается присоединиться к усилиям по совершенствованию существующих инструментов и созданию новых, чтобы превратить обещание технологии блокчейн в реальность. Эти инструменты являются основой, необходимой для создания децентрализованного Интернета и глобальной финансовой системы завтрашнего дня.

СЛЕДУЕМ ДАЛЕЕ

Прочитать технический документ Cosmos
Начать разработку в Cosmos

4.Конечные точки gRPC, REST и Tendermint
В данном документе представлен обзор всех конечных точек, которые предоставляет узел: gRPC, REST, а также некоторые другие конечные точки.

Обзор всех конечных точек

Каждый узел предоставляет следующие конечные точки для взаимодействия пользователей с узлом. Каждая конечная точка обслуживается на другом порту. Более подробные сведения о том, как настроить каждую конечную точку приведены в отдельном разделе “конечной точки”.

- сервер gRPC (порт по умолчанию: 9090),
- сервер REST (порт по умолчанию: 1317),
- конечная точка Tendermint RPC (порт по умолчанию: 26657).

ⓘ Также узел предоставляет некоторые другие конечные точки, такие как конечная точка Tendermint P2P или конечная точка Prometheus. Непосредственно с Cosmos SDK они не связаны. Пожалуйста, ознакомьтесь с документацией Tendermint для получения дополнительной информации о данных конечных точках.

gRPC-сервер

ⓘ Патч, который был введен в go-grpc v1.34.0, сделал gRPC несовместимым с библиотекой gogoproto, заставив некоторые запросы gRPC паниковать. Таким образом, SDK требует, чтобы go-grpc <=v1.33.2 был обязательно установлен в вашем go.mod.

Дабы убедиться, что gRPC работает корректно, настоятельно рекомендуем добавить следующую строку в go.mod вашего приложения:

Пожалуйста, смотрите выпуск № 8392 для получения дополнительной информации.

Cosmos SDK v0.40 представил Protobuf в качестве основной библиотеки кодирования. Это приносит широкий спектр инструментов на основе Protobuf, которые могут быть подключены к SDK. Одним из таких инструментов является gRPC, современный высокопроизводительный RPC-фреймворк с открытым исходным кодом. Он имеет достойную клиентскую поддержку на нескольких языках.

Каждый модуль предоставляет службы Msg и Query Protobuf для определения переходов состояний и запросов состояний. К gRPC эти службы подключаются с помощью следующей функции внутри приложения:

https://github.com/cosmos/cosmos-sdk/blob/v0.41.0/server/types/app.go#L39-L41

-это конкретный gRPC-сервер, который порождает и обслуживает любые запросы gRPC. Данный сервер можно настроить внутри ~/.simapp/config/app.toml:

- поле grpc.enable = true|false определяет, должен ли быть включен сервер gRPC. По умолчанию установлено значение true.
- поле grpc.address = {string} определяет адрес (на самом деле порт, так как хост должен быть сохранен на уровне 0.0.0.0), к которому должен привязываться сервер. По умолчанию 0.0.0.0:9090.

ⓘ ~/.simapp-это каталог, в котором хранятся конфигурация узла и базы данных. По умолчанию он установлен в ~/.{app_name}.

После запуска сервера gRPC вы можете отправлять на него запросы с помощью клиента gRPC. В нашем учебнике по взаимодействию с узлом приведены некоторые примеры.

Обзор всех доступных конечных точек gRPC, поставляемых вместе с Cosmos SDK, представлен в документе Protobuf documentation.

Сервер ОТДЫХА

В Cosmos SDK v0.40 узел продолжает обслуживать сервер REST. Однако существующие маршруты, присутствующие в версии v0.39 и более ранних версиях, теперь помечены как устаревшие, и новые маршруты были добавлены через gRPC-gateway.

Все маршруты настраиваются в следующих полях ~/.simapp/config/app.toml:

- поле api.enable = true|false определяет, должен ли быть включен сервер REST. По умолчанию поле false.
- api.address = {string} определяет адрес (на самом деле порт, так как хост должен быть сохранен на уровне 0.0.0.0), к которому должен привязываться сервер. По умолчанию используется tcp://0.0.0.0:1317.
- некоторые дополнительные параметры конфигурации API определены в ~/.simapp/config/app.toml, вместе с комментариями, пожалуйста, обратитесь непосредственно к этому файлу.

#gRPC-маршруты отдыха шлюза

Если вы по каким-либо причинам не можете использовать gRPC (например, вы создаете веб-приложение, а браузеры не поддерживают HTTP2, на котором построен gRPC), то SDK предлагает REST-маршруты через gRPC-шлюз.

gRPC-gateway - это инструмент для предоставления конечных точек gRPC в качестве конечных точек REST. Для каждой конечной точки RPC, определенной в службе Protobuf, SDK предлагает эквивалент REST. Например, запрос баланса можно выполнить через файл /cosmos.bank.v1beta1.Запрос/AllBalances gRPC endpoint или альтернативно через gRPC-шлюз "/cosmos/bank/v1beta1/balances/{address}" REST endpoint: оба запроса выведут один и тот же результат. Для каждого метода RPC, определенного в службе Protobuf, соответствующая конечная точка REST определяется как опция:

Для разработчиков приложений маршруты REST gRPC-gateway должны быть подключены к серверу REST, это делается путем вызова функции RegisterGRPCGatewayRoutes в ModuleManager.

Устаревшие Маршруты REST API

ОСТАЛЬНЫЕ маршруты, присутствующие в Cosmos SDK v0.39 и более ранних версиях, помечаются как устаревшие с помощью заголовка HTTP deprecation. Они по-прежнему поддерживаются для обеспечения обратной совместимости, но будут удалены в версии v0.41. Для обновления с устаревших маршрутов REST на новые маршруты REST gRPC-gateway обратитесь к нашему руководству по миграции.

Для разработчиков приложений устаревшие маршруты REST API должны быть подключены к серверу REST. Делается это путем вызова функции RegisterRESTRoutes в ModuleManager.

Swagger

Файл спецификации Swagger (или OpenAPIv2) открывается под маршрутом /swagger на сервере API. Swagger-это открытая спецификация, которая описывает конечные точки API, обслуживаемые сервером, включая описание, входные аргументы, типы возвращаемых значений и многое другое о каждой конечной точке.

Включение конечной точки /swagger настраивается внутри ~/.simapp/config/app.toml через поле api.swagger, которое по умолчанию имеет значение true.

Для разработчиков приложений вы можете создать свои собственные определения Swagger на основе ваших пользовательских модулей. Сценарий генерации Swagger SDK - это хорошее место для начала.

Tendermint RPC

Независимо от Cosmos SDK, Tendermint также предоставляет RPC-сервер. Этот RPC-сервер можно настроить, настроив параметры в таблице rpc в файле ~/.simapp/config/config.toml, адрес прослушивания по умолчанию-tcp://0.0.0.0:26657. Спецификация OpenAPI всех конечных точек Tendermint RPC доступна здесь.

Некоторые конечные точки Tendermint RPC напрямую связаны с Cosmos SDK:

- /abci_query: эта конечная точка будет запрашивать состояние приложения. В качестве параметра path можно отправить следующие строки:

○ любой Протобуф полностью квалифицированный метод обслуживания, например /cosmos.bank.v1beta1.Запрос/AllBalances. Затем поле данных должно включать параметр(ы) запроса метода, закодированный в байтах с помощью Protobuf.
○ /app/simulate: это будет имитировать транзакцию и возвращать некоторую информацию, такую как используемый газ.
○ /app/version: это вернет версию приложения.
○ /store/{path}: это вызовет прямой запрос к хранилищу.
○ /p2p/filter/addr/{port}: это вернет отфильтрованный список одноранговых узлов P2P узла по адресу порта.
○ /p2p/filter/id/{id}: это вернет отфильтрованный список одноранговых узлов P2P по идентификатору.

- /broadcast_tx_{aync,async,commit}: эти 3 конечные точки будут транслировать транзакцию другим узлам. CLI, gRPC и REST предоставляют способ широковещательных транзакций, но все они используют эти 3 Tendermint RPC под капотом.

#Сравнительная таблица

Имя	Преимущества	Недостатки
gRPC	- может использовать сгенерированные кодом заглушки на различных языках -поддерживает потоковое и двунаправленное общение (HTTP2) -небольшие двоичные размеры провода, более быстрая передача	- на основе HTTP2, не доступны в браузерах - кривая обучения (в основном из-за Protobuf)
Остальные	-вездесущие - клиентские библиотеки на всех языках, более быстрая реализация	- поддерживает только унарную связь запроса-ответа (HTTP1.1) - большие размеры сообщений по проводам (JSON)
Tendermint RPC	-прост в использовании	- большие размеры проводных сообщений (JSON)

5.
Введение в модули SDK
Модули определяют большую часть логики приложений SDK. Разработчики составляют модуль вместе, используя Cosmos SDK для создания своих пользовательских блокчейнов для конкретных приложений. В этом документе описываются основные концепции, лежащие в основе модулей SDK, и способы подхода к управлению модулями.

#Роль модулей в приложении SDK
Cosmos SDK можно рассматривать как Ruby-on-Rails разработки блокчейна. Он поставляется с ядром, которое обеспечивает основные функциональные возможности, необходимые каждому блокчейн-приложению, такие как шаблонная реализация ABCI для связи с базовым механизмом консенсуса, мультистор для сохранения состояния, сервер для формирования полного узла и интерфейсы для обработки запросов.

В дополнение к этому ядру Cosmos SDK позволяет разработчикам создавать модули, реализующие бизнес-логику их приложения. Другими словами, модули SDK реализуют основную часть логики приложений, в то время как ядро выполняет проводку и позволяет модулям быть составленными вместе. Конечная цель состоит в том, чтобы создать надежную экосистему модулей SDK с открытым исходным кодом, что делает все более простым создание сложных блокчейн-приложений.

Модули SDK можно рассматривать как небольшие машины состояний внутри машины состояний. Они обычно определяют подмножество состояния, используя один или несколько KVStores в главном мультисторе, а также подмножество типов сообщений. Эти сообщения направляются одним из основных компонентов SDK core, BaseApp, в службу Msg модуля, который их определяет.

В результате этой архитектуры создание приложения SDK обычно вращается вокруг написания модулей для реализации специализированной логики приложения и составления их с существующими модулями для завершения приложения. Разработчики обычно работают над модулями, реализующими логику, необходимую для их конкретного случая использования, который еще не существует, и будут использовать существующие модули для более общих функций, таких как стейкинг, учетные записи или управление токенами.

#Как подойти к построению модулей в качестве разработчика
Хотя нет никаких окончательных рекомендаций по написанию модулей, вот некоторые важные принципы проектирования, которые разработчики должны иметь в виду при их создании:

Композиционность: Приложения SDK почти всегда состоят из нескольких модулей. Это означает, что разработчикам необходимо тщательно продумать интеграцию своего модуля не только с ядром Cosmos SDK, но и с другими модулями. Первое достигается путем следования стандартным схемам проектирования, описанным здесь, в то время как второе достигается путем правильного раскрытия хранилища(хранилищ) модуля через хранитель.

Специализация: Прямым следствием функции композиционности является то, что модули должны быть специализированными. Разработчики должны тщательно определить область применения своего модуля, а не паковать несколько функций в один и тот же модуль. Такое разделение задач позволяет повторно использовать модули в других проектах и повышает возможность обновления приложения. Специализация также играет важную роль в объектно-ориентированной модели Cosmos SDK.

Возможности: Большинство модулей нуждаются в чтении и/или записи в хранилище(хранилища) других модулей. Однако в среде с открытым исходным кодом некоторые модули могут быть вредоносными. Именно поэтому разработчикам модулей необходимо тщательно продумать не только то, как их модуль взаимодействует с другими модулями, но и то, как предоставить доступ к хранилищу(хранилищам) модуля. Cosmos

SDK использует ориентированный на возможности подход к межмодульной безопасности. Это означает, что доступ к каждому хранилищу, определяемому модулем, осуществляется с помощью ключа, который хранится хранителем модуля. Этот хранитель определяет, как получить доступ к магазину(магазинам) и при каких условиях. Доступ к хранилищу(хранилищам) модуля осуществляется путем передачи ссылки хранителю модуля.
#Основные компоненты модулей SDK
Модули по соглашению определяются в подпапке ./x/ (например, банковский модуль будет определен в папке ./x/bank). Они обычно имеют одни и те же основные компоненты:

Хранитель, используемый для доступа к хранилищу(хранилищам) модуля и обновления состояния.

Служба Msg, используемая для обработки сообщений, когда они направляются в модуль BaseApp, и запуска переходов состояний.

Служба запросов, используемая для обработки запросов пользователей, когда они направляются в модуль BaseApp.

Интерфейсы, позволяющие конечным пользователям запрашивать подмножество состояния, определенного модулем, и создавать сообщения пользовательских типов, определенных в модуле.

В дополнение к этим компонентам модули реализуют интерфейс AppModule для управления менеджером модулей.

Пожалуйста, обратитесь к документу structure, чтобы узнать о рекомендуемой структуре каталога модуля.

11. Интеграция протокола IBC

Вступление

Узнайте, как интегрировать протокол IBC в ваше приложение и отправлять пакеты данных в другие сети.

В этом документе описаны необходимые шаги для интеграции и настройки модуля IBC в приложение Cosmos SDK и отправки взаимозаменяемых токенов в другие цепочки.

Интеграция модуля IBC

Интегрировать модуль IBC в ваше приложение на основе SDK несложно. Общие изменения можно резюмировать в следующих этапах:

- Добавьте необходимые модули в `module.BasicManager`.
- Определите дополнительные поля Keeper для новых модулей в типе App
- Добавьте StoreKeys из модуля и инициализируйте Keepers
- Настройте соответствующие маршрутизаторы и маршруты для модулей ibc и evidence.
- Добавьте модули в модуль Manager
- Добавить модули в Begin/EndBlockers и InitGenesis
- Обновите модуль SimulationManager, чтобы включить симуляции

Разрешения модулей BasicManager и ModuleAccount

Первым шагом является добавление в BasicManager следующих модулей: x/features, x/ibc, x/inventory и x/ibc-transfer. После этого нам нужно предоставить разрешения Minter и Burner для учетной записи ibc-transfer ModuleAccount для создания и записи транслируемых токенов.

Области применения

Затем нам нужно зарегистрировать Keepers следующим образом:

Настройка Keepers

Во время инициализации, помимо инициализации IBC Keepers (для модулей x/ibc и x/ibc-transfer), нам необходимо предоставить определенные возможности через модуль возможностей ScopedKeepers, чтобы мы могли аутентифицировать права доступа к объектам для каждого из каналов IBC.

Регистрация Routers

IBC необходимо знать, какой модуль привязан к какому порту, чтобы он мог направлять пакеты в соответствующий модуль и вызывать соответствующие обратные вызовы. Отображение порта на имя модуля обрабатывается портом IBC Keeper. Однако сопоставление имени модуля с соответствующими обратными вызовами выполняется маршрутизатором портов на модуле IBC.

Добавление маршрутов модуля позволяет обработчику IBC вызывать соответствующий обратный вызов при обработке квитирования канала или пакета.

Второй необходимый Router - это маршрутизатор модуля свидетельства. Этот маршрутизатор обрабатывает общую отправку свидетельств и направляет бизнес-логику каждому зарегистрированному обработчику свидетельств. В случае IBC требуется предоставить доказательства ненадлежащего поведения легкого клиента, чтобы заблокировать клиента и предотвратить отправку / получение дальнейших пакетов данных.

В настоящее время Router является статическим, поэтому он должен быть инициализирован и правильно настроен при инициализации приложения. После настройки Router новые маршруты добавлять нельзя.

Менеджеры модулей

Чтобы использовать IBC, нам необходимо добавить новые модули в диспетчер Manager и в SimulationManager, если ваше приложение поддерживает моделирование.

Упорядочивание приложения ABCI

Одним из дополнений от IBC является концепция HistoricalEntries, которая хранится в модуле ставок. Каждая запись содержит историческую информацию для Headers и ValidatorSet этой цепочки, которая сохраняется на каждой высоте во время вызова BeginBlock. Историческая информация требуется для самоанализа прошлой исторической информации на любой заданной высоте, чтобы проверить ConsensusState легкого клиента во время установления соединения.

Модуль IBC также имеет логику BeginBlock. Это необязательно, так как требуется только в том случае, если ваше приложение использует клиент localhost для подключения двух разных модулей из одной цепочки.

Внимание: Регистрируйте модуль ibc в SetOrderBeginBlockers только в том случае, если ваше приложение будет использовать клиент localhost (он же loopback).

ВАЖНО: модуль возможностей **должен** быть объявлен первым в SetOrderInitGenesis.

Вот и все! Вы подключили модуль IBC и теперь можете отправлять взаимозаменяемые токены по разным цепочкам. Если вы хотите получить более широкое представление об изменениях, загляните в SimApp SDK.

12. Ретранслятор

Предварительные статьи

События

События генерируются для каждой транзакции, обрабатываемой базовым приложением, чтобы указать, что выполнение некоторой логики, о которой клиенты могут захотеть знать. Это чрезвычайно полезно при ретрансляции пакетов IBC. Любое сообщение, использующее IBC, будет генерировать события для соответствующей логики TAO, выполняемой, как определено в спецификации событий IBC.

В SDK можно предположить, что для каждого сообщения существует событие, отправляемое с типом message, ключом атрибута action и значением атрибута, представляющим тип отправленного сообщения (channel_open_init будет значением атрибута для MsgChannelOpenInit). Если ретранслятор запрашивает события транзакции, он может разделить события сообщения, используя эту пару «Тип события / ключ атрибута».

Тип события message с ключом атрибута module может быть отправлено несколько раз для одного сообщения из-за обратных вызовов приложения. Можно предположить, что любая выполненная логика TAO приведет к генерации события модуля со значением атрибута ibc_<submodulename> (02-client испускает ibc_client).

Подписи с Tendermint

Вызов метода Tendermint RPC Subscribe через Tendermint Websocket вернет события, используя их внутреннее представление Tendermint. Вместо получения списка событий в том виде, в каком они были отправлены, Tendermint вернет строку типа map [string] [], которая отображает строку в форме <event_type>.<attribute_key> на attribute_value. Это делает извлечение порядка событий нетривиальным, но Все же возможным.

Ретранслятор должен использовать ключ message.action для извлечения количества сообщений в транзакции и типа отправленных транзакций IBC. Для каждой транзакции IBC в массиве строк для message.action необходимая информация должна быть извлечена из других полей событий. Если send_packet появляется с индексом 2 в значении message.action, ретранслятор должен будет использовать значение с индексом 2 ключа send_packet.packet_sequence. Этот процесс следует повторять для каждой части информации, необходимой для ретрансляции пакета.

Примеры использования

Ретранслятор Golang

13. Что такое Tendermint?

Tendermint - это программное обеспечение для безопасной и последовательной репликации приложения на многих машинах. Под безопасностью подразумевается то, что Tendermint будет работать, даже если из строя произвольным образом выходят вплоть до 1/3 машин. Под последовательностью подразумевается, что каждая исправная машина видит один и тот же журнал транзакций и вычисляет одно и то же состояние. Безопасная и последовательная репликация - фундаментальная проблема в распределенных системах; она играет решающую роль в обеспечении безотказности широкого спектра приложений, от валют до выборов, управление инфраструктурой и многих других.

Способность противостоять сбоям машин, выходящим из строя произвольным образом, в том числе становясь вредоносными, известна как византийская отказоустойчивость (BFT). Теории BFT уже несколько десятилетий, но реализация программных обеспечений стала популярна не так давно, во многом благодаря успеху «технологии блокчейн», такой как Bitcoin и Ethereum. Технология блокчейн - это просто формализация BFT в более современных условиях с акцентом на одноранговые сети и криптографическую аутентификацию. Название происходит от способа пакетной обработке транзакций в блоки, где каждый блок содержит криптографический хэш предыдущего, образуя цепочку. На практике структура данных блокчейна фактически оптимизирует дизайн BFT.

Tendermint состоит из двух основных технических компонентов: движка консенсуса блокчейна и универсального интерфейса приложения. Консенсус-движок, так называемый Tendermint Core, гарантирует, что одинаковые транзакции регистрируются на каждой машине в одном и том же порядке. Интерфейс приложения, называемый Application BlockChain Interface (ABCI), позволяет обрабатывать транзакции на любом языке программирования. В отличие от других блокчейн-решений и консенсусных решений, которые поставляются предварительно упакованными со встроенными конечными аппаратами (такими как необычное хранилище ключей или необычный скриптовый язык), разработчики могут использовать Tendermint для репликации государственных машин BFT приложений, написанных на любом подходящем для них языке программирования и среде разработки.

Tendermint разработан таким образом, чтобы быть простым и понятным в использовании, в тоже время являясь высокопроизводительным и полезным для широкого спектра распределенных приложений.

Tendermint в сравнении с X

Tendermint в целом похож на два класса программного обеспечения. Первый класс состоит из распределенных хранилищ ключей и значений, таких как Zookeeper, etcd и consul, которые не используют консенсус BFT. Второй класс известен «технология блокчейна» и состоит из криптовалют, таких как Bitcoin и Ethereum, так и из альтернативных конструкций распределенных реестров, например Hyperledger's Burrow.

Zookeeper, etcd, consul

Zookeeper, etcd и consul - все это способы реализации хранилища ключей и значений на основе классического алгоритма, отличного от BFT. Zookeeper использует версию Paxos под названием Zookeeper Atomic Broadcast, а etcd и consul используют алгоритм консенсуса Raft, который намного моложе и проще. Типичный кластер состоит из 3-5 машин и может выдерживать сбои в работе до 1/2 машин, однако даже один византийский сбой способен разрушить систему.

Каждое предложение обеспечивает несколько иную реализацию функционального хранилища ключей и значений, но все они, как правило, сосредоточены на предоставлении базовых услуг распределенным системам, таких как: динамическая конфигурация, обнаружение служб, блокировка, выбор лидера и так далее.

Tendermint по сути аналогичное программное обеспечение, но с двумя ключевыми отличиями:

- Это византийская отказоустойчивость, то есть он может терпеть только до 1/3 сбоев, но эти сбои могут вызывать произвольное поведение, включая взлом и вредоносные атаки.
- Он не указывает конкретное приложение, например, необычное хранилище ключей и значений. Вместо этого он фокусируется на произвольной репликации конечного аппарата, чтобы разработчики могли выстроить логику приложения, которая им подходит, от хранилища ключей до криптовалюты, платформу для электронного голосования и так далее.
-

Bitcoin, Ethereum и другие

Tendermint возник в традициях криптовалют, таких как Bitcoin, Ethereum и других, с целью обеспечения более эффективного и безопасного алгоритма консенсуса, чем Proof-of-Work от Bitcoin. В первые дни своего существования Tendermint имел простую встроенную валюту, и для участия в достижении консенсуса пользователи должны были “закладывать” единицы валюты в гарантийный депозит, который мог быть отозван, если они совершали вредоносные действия - это как раз то, что составляет из себя алгоритм Proof-of-Stake от Tendermint.

С тех пор Tendermint превратился в механизм консенсуса блокчейна общего назначения, который способен поддерживать произвольные состояния приложений. Это означает, что он может быть использован в качестве замены plug-and-play для движков консенсуса и другого программного обеспечения блокчейнов. Таким образом, можно взять текущую базу кода Ethereum, будь то Rust, Go или Haskell, и запустить ее как приложение ABCI, используя консенсус Tendermint. Действительно, мы сделали это с Ethereum . У нас в планах сделать то же самое для Bitcoin, ZCash и различных других детерминированных приложений.

Еще одним примером криптовалютного приложения, построенного на Tendermint, является сеть Cosmos.

Другие проекты Blockchain

Fabric использует аналогичный подход к Tendermint, но более категоричен в отношении управления состоянием и требует, чтобы поведение приложения полностью выполнялось в потенциально многих докерах, модулях, которые называются блокчейн-кодом». Он использует реализацию PBFT от команды IBM, которая расширена для обработки потенциально недетерминированного блокчейн-кода. Это поведение на основе докеров можно реализовать как приложение ABCI в Tendermint, хотя расширение Tendermint для обработки недетерминированного кода планируется разработать позже.

Burrow - это реализация виртуальной машины Ethereum и механики транзакций Ethereum с дополнительными функциями для реестра имен, разрешений и собственных контрактов, а также альтернативного API блокчейна. Он использует Tendermint в качестве механизма консенсуса и обеспечивает определенное состояние приложения.

Обзор ABCI

Интерфейс Application BlockChain (ABCI) позволяет выполнять византийскую отказоустойчивую репликацию приложений, написанных на любом языке программирования.

Мотивация

До сих пор все «стеки» блокчейнов (например, Биткойн) имели монолитную конструкцию. То есть каждый стек блокчейна - это единая программа, которая обрабатывает все задачи децентрализованного реестра; это включает в себя P2P-подключение, трансляцию транзакций в «mempool», консенсус по самому последнему блоку, балансы счетов, контракты по-Тьюрингу, разрешения на уровне пользователя и так далее.

Использование монолитной архитектуры обычно является плохой практикой в компьютерных науках. Это затрудняет повторное использование компонентов кода, и попытки сделать это приводят к сложным процедурам обслуживания форков кодовой базы. В особенности это видно, когда кодовая база не имеет модульной конструкции и страдает от «спагетти-кода».

Еще одна проблема с монолитным дизайном заключается в том, что он ограничивает вас языком стека блокчейна (или наоборот). В случае Ethereum, который поддерживает виртуальную машину с полным байт-кодом по-Тьюрингу, он ограничивает вас языками, которые компилируются до этого байт-кода; сегодня это Serpent и Solidity.

Напротив, наш подход заключается в том, чтобы отделить механизм консенсуса и уровни P2P от деталей состояния конкретного приложения блокчейна. Мы делаем это, абстрагируя детали приложения в интерфейс, который реализован как протокол сокета.

Таким образом, у нас есть интерфейс Application BlockChain Interface (ABCI) и его основная реализация, Tendermint Socket Protocol (TSP или Teaspoon).

Введение в ABCI

Tendermint Core («механизм консенсуса») взаимодействует с приложением по протоколу сокета, удовлетворяющему требованиям ABCI.

Bitcoin - это блокчейн криптовалюты, где каждая нода поддерживает полностью проверенную базу данных неизрасходованных транзакций (UTXO). Если кто-то хотел бы создать систему, подобную Bitcoin, поверх ABCI, Tendermint Core отвечал бы за:

- Совместное использование блоков и транзакций между узлами
- Установление канонического / неизменного порядка транзакций (блокчейн)

Приложение будет нести ответственность за:

- Ведение базы данных UTXO
- Проверка криптографических подписей транзакций
- Предотвращение транзакций от расходования несуществующих транзакций
- Разрешение клиентам запрашивать базу данных UTXO.

Tendermint может декомпозировать структуру блокчейна, предлагая очень простой API (например, ABCI) между процессом подачи заявки и процессом консенсуса.

ABCI состоит из 3 основных типов сообщений, которые доставляются из ядра в приложение. Приложение отвечает соответствующими ответными сообщениями.

Здесь указываются сообщения: Типы сообщений ABCI.

Сообщение DeliverTx - это рабочая лошадка приложения. Каждая транзакция в блокчейне доставляется с этим сообщением. Приложению необходимо проверять каждую транзакцию, полученную с помощью сообщения DeliverTx, на соответствие текущему состоянию, протоколу приложения и криптографическим учетным данным транзакции. Затем проверенной транзакции необходимо обновить состояние приложения - например, привязав значение к хранилищу значений ключей или обновив базу данных UTXO.

Сообщение CheckTx похоже на сообщение DeliverTx, но только для проверки транзакций. Мемпул Tendermint Core сначала проверяет действительность транзакции с помощью CheckTx и передает только действительные транзакции своим партнерам. Например, приложение может проверять увеличивающийся порядковый номер в транзакции и возвращать ошибку при CheckTx, если порядковый номер старый. В качестве альтернативы они могут использовать систему, основанную на возможностях, которая требует обновления возможностей с каждой транзакции.

Сообщение Commit используется для вычисления криптографического обязательства текущему состоянию приложения, которое должно быть помещено в следующий заголовок блока. Это имеет ряд полезных свойств. Несоответствия в обновлении этого состояния теперь будут отображаться как блокчейн-форки, которые выявляют целый класс программных ошибок. Это также упрощает разработку защищенных легких клиентов, поскольку доказательства хэша Меркла можно проверить, сверившись с хешем блока, и что хэш блока подписан кворумом.

К приложению может быть подключено несколько сокетов ABCI. Tendermint Core создает три соединения ABCI с приложением; один для проверки транзакций при трансляции в пуле памяти, один для механизма консенсуса для запуска блочных предложений и еще один для запроса состояния приложения.

Вероятно, очевидно, что разработчикам приложений необходимо очень тщательно разрабатывать свои обработчики сообщений, чтобы создать блокчейн, который делает что-нибудь полезное, но эта архитектура предоставляет место для начала. На диаграмме ниже показан поток сообщений через ABCI.

Заметка о детерминизме

Логика обработки транзакций блокчейна должна быть детерминированной. Если бы логика приложения не была детерминированной, консенсус между узлами реплик Tendermint Core не был бы достигнут.

Solidity на Ethereum - отличный язык для приложений блокчейна, потому что, среди прочего, это полностью детерминированный язык программирования. Однако также возможно создавать детерминированные приложения с использованием существующих популярных языков, таких как Java, C ++, Python или Go. Программисты игр и разработчики блокчейнов уже знакомы с созданием детерминированных программ, избегая источников недетерминизма, таких как:

- генераторы случайных чисел (без детерминированного заполнения)
- условия гонки на потоках (или полное избегания потоков)
  системные часы
- неинициализированная память (в небезопасных языках программирования, таких как C или C ++)
- арифметика с плавающей запятой
- случайные языковые особенности (например, итерация карты в Go)

В то время как программисты могут избежать недетерминизма, проявляя осторожность, также можно создать специальный линтер или статический анализатор для каждого языка, чтобы проверять его на детерминизм. В будущем мы можем работать с партнерами над созданием таких инструментов.

Обзор консенсуса
Tendermint - это простой для понимания, в основном асинхронный протокол консенсуса BFT. Протокол следует за простым конечным автоматом, который выглядит так:

Участники протокола называются валидаторами; они по очереди предлагают блоки транзакций и голосуют за них. Блоки фиксируются в блокчейне, по одному блоку на каждой высоте. Блок может не быть зафиксирован, и в этом случае протокол переходит к следующему раунду, и новый валидатор может предложить блок для этой высоты. Для успешной фиксации блока требуется два этапа голосования; мы называем их предварительным голосованием и предварительной фиксацией. Блок фиксируется, когда более 2/3 валидаторов предварительно фиксируют один и тот же блок в одном и том же раунде.

Есть фотография пары, исполняющей польку, потому что валидаторы исполняют что-то вроде танца польки. Когда более двух третей валидаторов предварительно голосуют за один и тот же блок, мы называем это полькой. Каждая предварительная фиксация должна быть оправдана полькой в том же раунде.

Валидаторы могут не зафиксировать блок по ряду причин; текущий претендент может быть не в сети, или сеть может быть медленной. Tendermint позволяет им установить, что валидатор должен быть пропущен. Валидаторы ждут немного времени, чтобы получить полный блок предложения от претендента, прежде чем проголосовать, чтобы перейти к следующему раунду. Эта зависимость от тайм-аута делает Tendermint скорее слабо синхронным протоколом, чем асинхронным. Однако остальная часть протокола является асинхронной, и валидаторы добиваются прогресса только после того, как услышат более двух третей набора валидаторов. Упрощающим элементом Tendermint является то, что он использует тот же механизм для фиксации блока, что и для перехода к следующему раунду.

Предполагая, что менее одной трети валидаторов являются византийскими, Tendermint гарантирует, что безопасность никогда не будет нарушена, то есть валидаторы никогда не будут фиксировать конфликтующие блоки на одной высоте. Для этого он вводит несколько правил блокировки, которые модулируют пути, по которым можно следовать на блок-схеме. Как только валидатор предварительно фиксирует блок, он блокируется на этом блоке. Потом,

1. он должен проголосовать за блок, на котором он заблокирован
2. он может только разблокировать и предварительно зафиксировать новый блок, если будет полька для этого блока в более позднем раунде

Стейкинг

Во многих системах не все валидаторы будут иметь одинаковый «вес» в протоколе консенсуса. Таким образом, нас интересуют не только одна треть или две трети валидаторов, но и те доли от общего числа голосов, которые могут быть неравномерно распределены между отдельными валидаторами.

Поскольку Tendermint может воспроизводить произвольные заявки, можно определить валюту и определить право голоса в этой валюте. Когда количество голосов выражается в национальной валюте, систему часто называют Proof-of-Stake. Логика приложения может заставить валидаторов “заложить” свои валютные резервы с помощью гарантийного депозита, который может быть уничтожен, если обнаружится, что они неправильно себя ведут в протоколе консенсуса. Это добавляет экономический элемент к безопасности протокола, позволяя количественно оценить стоимость нарушения предположения о том, что менее одной трети голосов принадлежит Византии.

Сеть Cosmos предназначена для использования этого механизма Proof-of-Stake для множества криптовалют, реализованных в виде приложений ABCI.

Следующая диаграмма представляет собой краткую (техническую) схему работы Tendermint.

14. Создание приложения на Go

Ознакомление с руководством

Это руководство предназначено для новичков, которые хотели бы начать работу с приложением Tendermint Core с нуля. Мы не рассчитываем, что у вас имеется какой-либо опыт работы с Tendermint Core.

Tendermint Core - это византийское (отказоустойчивое) промежуточное программное обеспечение Byzantine Fault Tolerant (BFT), которое берет аппарат перехода состояний, написанный на любом из языков программирования, и надежно производит его репликацию на многих машинах.

Хотя Tendermint Core создано на языке программирования Golang, вам не потребуется предварительное знание этого языка для понимания этого руководства. Изучить его можно в процессе работы, благодаря его простоте. Однако, при желании, вы можете сначала пройти обучение X за Y минут. Где X = Go, чтобы ознакомиться с синтаксисом.

Следуя этому руководству, вы сможете создать проект Tendermint Core, который будет иметь название - kvstore, (очень) простое распределенное хранилище ключей и значений BFT.

Встроенное приложение и внешнее приложение

Лучше запускать ваше приложение вместе с Tendermint Core, дабы получить максимальную производительность. Как раз таким образом написан Cosmos SDK. Пожалуйста, обратитесь к руководству по написанию встроенного приложения Tendermint Core в Go.

Наличие отдельного приложения дает вам наилучшие гарантии безопасности. Это обусловлено тем, что оба процесса будут взаимодействовать через установленный бинарный протокол. Tendermint Core не будет иметь доступа к состоянию приложения.

1.1 Установка Go

См. Официальное руководство по установке Go.

Убедитесь, что у вас установлена последняя версия Go:

1.2 Создание нового проекта Go

Начнем с создания нового проекта Go.

Внутри каталога примера следует создать файл main.go, который будет содержать следующее:
При запуске на стандартный вывод должно быть выведено сообщение «Hello, Tendermint Core».

1.3 Написание приложения Tendermint Core

Tendermint Core взаимодействует с приложением через интерфейс Application BlockChain (ABCI). protobuf -это файл, где определены все типы сообщений. Это позволяет Tendermint Core запускать любые приложения, в независимости от того на каком языке программирования они написаны.

Далее следует создать файл app.go со следующим содержимым:

Теперь рассмотрим каждый метод, объясняя, в каких случаях он вызывается. Также добавим необходимую бизнес-логику.

1.3.1 CheckTx

Каждый раз, когда в Tendermint Core добавляется новая транзакция, она просит приложение проверить ее (проверить формат, подписи и все остальное).

Не переживайте, если он еще не компилируется.

Если транзакция не имеет формы `{bytes}={bytes}`, мы возвращаем код 1. Когда такой же ключ = значение уже имеется (тот же ключ и значение), мы возвращаем код 2. Для прочих мы возвращаем нулевой код, который говорит нам о том, что они действительны.

Обратите внимание, что абсолютно все с ненулевым кодом будет считаться Tendermint Core недействительным (-1, 100 и другие).

В результате будут совершены только действительные транзакции, при условии, что они не слишком велики и в них достаточно газа. Чтобы узнать больше информации о газе, ознакомьтесь с «спецификацией».

Для базового хранилища ключей и значений мы будем использовать badger. Он представляет собой встраиваемую, постоянную и быструю базу данных «ключ-значение» (KV).

1.3.2 BeginBlock -> DeliverTx -> EndBlock -> Commit

Во время того, как Tendermint Core принимает решение о блоке, он передается приложению в трех составляющих: BeginBlock, один DeliverTx на транзакцию и EndBlock в конце. DeliverTx передаются асинхронно, но предполагается, что ответы будут идти в правильном порядке.

Тут мы создаем пакет. В нем будут храниться транзакции блока.

В том случае, если транзакция отформатирована плохо или встречается такой же ключ = значение уже имеется, мы опять возвращаем ненулевой код. В противном случае мы добавляем его в текущий пакет.

В текущем виде блок может включать в себя неверные транзакции (те, которые прошли CheckTx, но не прошли DeliverTx, или транзакции, включенные непосредственно заявителем). Это сделано для получения большей производительности.

Следует отметить, что мы не можем фиксировать транзакции внутри DeliverTx, так как в этом случае Query, который может быть вызван параллельно, будет возвращать несогласованные данные (т.е. он сообщит, что определенное значение уже имеется, даже если фактический блок еще не был зафиксирован).

Commit предписывает приложению сохранять новое состояние.

1.3.3 Запрос

Теперь, если клиенту захочется узнать, существует ли конкретный ключ / значение, он может вызвать конечную точку Tendermint Core RPC / abci_query. А она, в свою очередь, вызовет метод Query приложения.

Приложения могут свободно предоставлять свои собственные API. Однако, используя Tendermint Core в качестве прокси, клиенты (включая легкий клиентский пакет) могут использовать унифицированный API для различных приложений. Помимо этого, им не нужно будет вызывать отдельный API Tendermint Core для дополнительных доказательств.

Прошу обратить внимание, что здесь доказательства мы не приводим.

Спецификацию в полном виде можно найти здесь.

1.4 Запуск приложения и экземпляров Tendermint Core

Следующий код необходимо поместить в файл "main.go":

Это очень большая часть кода, поэтому следует разобрать его на части.

Для начала мы инициализируем базу данных Badger и создаем экземпляр приложения:

Для тех, кто использует Windows перезапуск этого приложения приведет к тому, что Badger выдаст ошибку, так как для этого необходимо усечение журнала значений. Для получения более подробной информации посетите здесь. Этого возможно избежать, установив для параметра truncate значение true, к примеру:

После этого мы запускаем сервер ABCI и добавляем некоторую обработку сигналов, чтобы корректно остановить его при получении SIGTERM или Ctrl-C. Tendermint Core будет выступать в роли клиента, который будет подключаться к нашему серверу, отправлять нам транзакции и прочие сообщения.

1.5 Запуск

Тут мы планируем использовать модули Go для управления зависимостями.

Это должно создать файл go.mod. Текущее руководство работает только с основной веткой Tendermint, поэтому обязательно нужно убедиться, что мы используем последнюю версию приложения:

Это заполнит go.mod номером версии. За ним следует хэш Tendermint.

Теперь у нас есть возможность собрать бинарный файл:

Для того, чтобы создать конфигурацию по умолчанию, nodeKey и частные файлы валидатора, нужно выполнить tendermint init. Но прежде чем мы это сделаем, нам требуется установить Tendermint Core. См. Официальное руководство. Если вы производите установку из исходного кода, не забудьте проверить последнюю версию (git checkout vX.Y.Z). Также не забудьте сравнить, использует ли приложение ту же основную версию.

Не стесняйтесь исследовать те файлы, которые были сгенерированны. Их можно найти в каталоге /tmp/example/config. Документацию по конфигу можно изучить здесь.

Мы готовы к запуску нашего приложения:

Далее нам надо запустить Tendermint Core и указать его на наше приложение. Оставаясь в каталоге приложения, выполнить:

Это должно запустить полную ноду и подключиться к нашему приложению ABCI.

Далее откройте новую вкладку в вашем терминале и попытайтесь отправить транзакцию:

Ответ должен включать в себя высоту, на которой была зафиксирована эта транзакция.

Теперь давайте убедимся, существует ли данный ключ и его значение:

“dGVuZGVybWludA ==” и “cm9ja3M =” являются кодировкой base64 ASCII «Tendermint» и “rocks ”соответственно.

Заключение

Я очень надеюсь, что у вас все получилось и ваше первое, но, надеюсь, не последнее приложение Tendermint Core запущено и работает. Если у вас что-либо не получилось или появились дополнительные вопросы, откройте вопрос на Github. Чтобы изучить подробнее, прочтите документацию.

15. Начинаем работу

Первое приложение Tendermint

Tendermint является блокчейн-движком общего назначения. Он не зависит от приложения, которое вы хотели бы запустить. Итак, для того, чтобы запустить полный блокчейн, который будет приносить какую-либо пользу, вам потребуется запустить две программы: первая - это Tendermint Core, а вторая - ваше приложение, написанное на любом языке программирования. Вспомните вступление к ABCI, где говорится что Tendermint Core способен обрабатывать все p2p и консенсусные вещи. Tendermint Core перенаправляет транзакции в приложение, когда они нуждаются в проверке или когда они готовы для коммита в блоке.

В данном руководстве мы продемонстрируем вам несколько примеров того, как можно запустить приложение при помощи Tendermint.

Установка

Первые приложения, с которыми мы будем работать, написаны на Go. Для того чтобы их установить нам необходимо будет сначала выполнить установку Go, поместить $GOPATH/bin в свой $PATH и включить модули go, следуя этим инструкциям:

Затем запустить

Теперь у вас должен быть установлен abci-cli; вы можете видеть пару команд (counter и kvstore). Они являются примерами приложений, написанных на Go. См. Ниже приложение, написанное на JavaScript.

А теперь давайте запустим несколько приложений!

KVStore - первый пример

Приложение kvstore представляет собой дерево Меркла, в котором просто хранится информация о всех транзакциях. Если транзакция содержит знак =, например key = value, это означает то, что значение сохраняется под ключом в дереве Меркла. В ином случае полные байты транзакции сохраняются как ключ и значение.

Запустим приложение kvstore.

В другом терминале мы можем запустить Tendermint. Здесь у вас уже должен быть установлен бинарный файл Tendermint. Если он у вас не установлен, следуйте инструкциям отсюда. Если до этого вы никогда не запускали Tendermint, используйте:

Если вы уже использовали Tendermint, вы можете сбросить данные для нового блокчейна. Для этого нужно запустить tendermint unsafe_reset_all. Далее вы можете запустить ноду tendermint, чтобы запустить Tendermint, и выполнить подключение к приложению. Если вам требуется дополнительная информация см. Руководство по использованию Tendermint.

Вы должны увидеть, как Tendermint делает блоки! Для того, чтобы сделать это, мы можем получить статус нашей ноды Tendermint. Делается это следующим образом:

-S просто заглушает curl. Для того, чтобы вывод был более удобный перенесите результат в такой инструмент, как jq или json_pp.

Теперь отправим несколько транзакций в kvstore.

Обратите внимание на одинарную кавычку (') вокруг URL-адреса, которая гарантирует, что двойные кавычки (") не будут экранированы с помощью bash. Данная команда отправила транзакцию с байтами abcd, поэтому abcd будет сохранен как ключ и значение в дерево Меркла. Ответ должен иметь примерно следующий вид:

Мы можем подтвердить, что наша транзакция прошла успешно и значение было сохранено. Для этого нужно запросить приложение:

Результат будет выглядеть примерно таким образом:

Обратите внимание на значение в результате (YWJjZA ==); это кодировка base64 ASCII abcd. Вы можете проверить это в оболочке python 2, запустив «YWJjZA ==». Decode ('base64'), или в оболочке python 3, запустив кодеки импорта; codecs.decode (b "YWJjZA ==", 'base64'). decode ('ascii'). Следите за дальнейшими выпусками, которые сделают эти выходные данные более удобочитаемыми.

Теперь попробуем установить другой ключ и значение:

Теперь, запросив имя, в ответ мы должны получить сатоши или c2F0b3NoaQ == в base64:

Попробуйте выполнить то же самое для других транзакций и запросов, чтобы удостовериться, что все работает!

Счетчик - еще один пример

Теперь, когда мы смогли разобраться с этим, давайте опробуем другое приложение - приложение счетчика.

Данное приложение не использует дерево Меркла, оно просто подсчитывает, сколько раз мы отправляли транзакцию или фиксировали состояние.

У данного приложения имеются два режима: serial = off и serial = on.

В том случае, если serial = on, транзакции должны быть увеличивающимся целым числом с прямым порядком байтов, начиная с 0.

Если serial = off, ограничения на транзакции не имеются.

В живом блокчейне транзакции накапливаются в памяти, прежде чем они будут объединены в блоки. Дабы не тратить ресурсы на недействительные транзакции, ABCI предоставляет сообщение CheckTx, которое разработчики приложений в праве использовать для того, чтобы принять или отклонить транзакции, прежде чем они будут сохранены в памяти или переданы другим партнерам.

В этом экземпляре приложения счетчика с serial = on CheckTx допускает только транзакции, целое число которых больше последнего зафиксированного.

Уберем предыдущий экземпляр softmint и приложение kvstore. Далее запустим приложение counter. Мы можем включить serial = on с помощью флага:

В другом окне сбросьте, затем запустите Tendermint:

И снова вы можете наблюдать, как блоки проходят мимо. Отправим несколько транзакций. Так как мы установили serial = on, первая транзакция обязана быть под номером 0:

Просим обратить внимание на пустой (следовательно, успешный) ответ. Следующая транзакция должна быть под номером 1. Если вместо этого мы попытаемся отправить 5, в ответ мы получим ошибку:

Но если мы отправим 1, все будет работать:

Если вам интересно изучить дополнительные сведения об broadcast_tx API см. В руководстве по использованию Tendermint.

CounterJS - Пример на другом языке

Мы также хотим показать пример того, как запускать приложения на другом языке - в данном случае мы запустим Javascript-версию счетчика. Для его запуска вам нужно установить ноду.

Также вам необходимо получить отсюда соответствующий репозиторий, а затем выполнить его установку:

Очистите предыдущие процессы counter и tendermint. Далее запустите приложение:

В другом окне нужно выполнить сброс и запустить tendermint:

И снова вы можете наблюдать потоковую передачу блоков, однако в данном случае наше приложение уже написано на Javascript! Попытайтесь отправить несколько транзакций. Результат должен быть таким же, как и ранее:

Неплохо, да?

16. Application Architecture Guide

Обзор

Вступление

Узнайте что такое протокол IBC, с какой целью он создан и его методы применения.

Что такое протокол межблокчейновой связи (IBC)?

Этот документ является руководством для разработчиков, которые хотели бы написать свои собственные приложения протокола связи между блокчейнами (IBC) для пользовательских сценариев использования.

Протокол IBC имеет модульную конструкцию. Благодаря этому разработчикам приложений IBC не следует волноваться о низкоуровневых деталях клиентов, подключениях и проверке доказательств. Тем не менее, имеется краткое объяснение нижних уровней стека. Это сделано для того, чтобы разработчики приложений имели общее представление о протоколе IBC. Далее в руководстве довольно подробно описывается уровень абстракции, который наиболее подходит для разработчиков приложений (каналы и порты). Там следует описание того, как определять свои собственные пользовательские пакеты и обратные вызовы IBCModule.

Для того, чтобы ваш модуль взаимодействовал через IBC, вам нужно сделать следующее: привязаться к порту (портам), определить свои собственные структуры пакетных данных (и, возможно, подтверждения), а также способы их кодирования / декодирования. Ко всему прочему нужно будет реализовать интерфейс IBCModule. Ниже вы можете ознакомиться с более подробным объяснением того, как правильно написать модуль приложения IBC.

Обзор компонентов

Клиенты

Клиенты протокола IBC - это легкие клиенты. Идентифицируются они уникальным идентификатором клиента, которые в свою очередь отслеживают согласованные состояния других блокчейнов вместе со спецификацией доказательства. Данная спецификация необходима для правильной проверки доказательств относительно согласованного состояния клиента. Сам клиент может быть связан с любым количеством подключений к нескольким блокчейнам. Клиенты, поддерживаемые IBC:

- Легкий клиент Solo Machine: такие устройства, как телефоны, браузеры или ноутбуки.
- Легкий клиент Tendermint: по умолчанию для блокчейнов на основе SDK,
- Клиент Localhost (loopback): полезен для тестирования, моделирования и ретрансляции пакетов модулям в одном приложении.

Соединения

Соединения размещают в одном компоненте данные и методы, которые с ними работают. Два компонента ConnectionEnd в двух отдельных блокчейнах. Каждый ConnectionEnd связан с клиентом другого блокчейна (то есть блокчейнами контрагентов). В свою очередь соединения отвечают за проверку того, что легкие клиенты в каждом блокчейне соответствуют их контрагентам. Ко всему прочему, установленные соединения несут ответственность за облегчение проверки состояния протокола IBC между цепями. Соединение может быть связано с любым количеством каналов.

Proofs и Paths

В протоколе IBC блокчейны не передают друг другу сообщения напрямую по сети. Вместо этого для связи блокчейн фиксирует некоторое состояние по специально определенному пути. Путь этот зарезервирован для определенного типа сообщения и определенного контрагента (возможно, сохраняя конкретное ConnectionEnd как часть соединения или пакет, который предназначен для ретрансляции на блокчейны контрагентов на прямую в модуль). Процесс работы ретранслятора заключается в отслеживании обновлений этих путей и ретранслировании сообщения. Это делается за счет отправления данных, хранящихся в этом пути, вместе с доказательством в блокчейн контрагентов. Все пути, которые реализованы в протоколе IBC должны использовать для фиксации сообщений IBC, определены в ICS-24. В свою очередь формат подтверждения, в которых все реализации должны иметь возможность создавать и проверять, определен в этой реализации ICS-23.

Возможности

Протокол IBC предназначен для работы в среде исполнения, где модули не обязательно доверяют друг другу. Таким образом, IBC должен аутентифицировать действия модуля на портах и каналах, чтобы их могли использовать только модули с соответствующими разрешениями. Это достигается с помощью динамических возможностей. Привязка к порту или создание канала для модуля IBC вернет динамическую возможность, которую требует модуль, чтобы использовать данный порт или канал. Это помогает предотвратить использование этого порта или канала другими модулями, так как они не будут обладать соответствующими возможностями.

Приведенный выше абзац является в основном полезной справочной информацией, так как модули IBC не нуждаются во взаимодействии с этими абстракциями нижнего уровня. уровень, который соответствует абстракции для разработчиков приложений IBC - это уровень каналов и портов. Приложения IBC должны быть написаны как независимые друг от друга, полностью автономные модули. Таким образом, модуль в одном блокчейне может взаимодействовать с другими модулями в других блокчейнах, отправляя, получая и подтверждая пакеты по каналам, которые однозначно идентифицируются парой (channelID, portID). Давайте рассмотрим модули IBC, как полезную аналогию. К примеру

как интернет-приложения на компьютере. Канал может быть в концепции, как IP-соединение, при этом IBC portID аналогичен IP-порту, а IBC channelID аналогичен IP-адресу. Таким образом, один экземпляр модуля IBC способен обмениваться данными по одному и тому же порту с любым количеством других модулей. В IBC будет правильно направлять все пакеты в соответствующий модуль, с помощью использования (channelID, portID tuple). Модуль IBC может также связываться с другим модулем IBC

через несколько портов. При этом каждый поток пакетов (portID <-> portID) отправляется по другому уникальному каналу.

Порты

Модуль IBC имеет возможность подключаться к любому количеству портов. Каждый порт должен иметь уникальный идентификатор порта. IBC спроектирован таким образом, чтобы обеспечивать безопасность с взаимно недоверенными модулями, работающими в одном реестре. Привязка порта вернет возможность предоставления динамического объекта. Для того, чтобы выполнить действие на конкретном порту (например, открыть канал с его portID), модуль должен предоставить динамический объект обработчику IBC. Это предотвращает открытие вредоносным модулем каналов с портами, которыми он не владеет. Таким образом, модули IBC отвечают за использование возможностей, возвращаемых BindPort.

Каналы

Канал IBC может быть установлен между 2 портами IBC. В данный момент порт исключительно принадлежит одному модулю. Пакеты IBC отправляются по каналам. Так же, как IP-пакеты содержат IP-адрес и IP-порт назначения, а также IP-адрес источника и IP-порт источника, пакеты IBC будут содержать в себе идентификатор порта назначения и channelID, а также исходный portID и channelID. Это позволяет IBC правильно маршрутизировать пакеты к модулю назначения, а также позволяет модулям, принимающим пакеты, знать модуль отправителя.

Канал может быть упорядочен. В этом случае пакеты от отправляющего модуля должны обрабатываться принимающим модулем в том порядке, в котором они были отправлены. В том случае, если канал не упорядочен, пакеты от отправляющего модуля обрабатываются в том порядке, в котором они поступают (возможно, не в том порядке, в котором они были отправлены).

Модули могут выбирать, с какими каналами у них есть желание взаимодействовать, поэтому протокол IBC подразумевает, что модули будут реализовывать обратные вызовы, которые вызываются во время того, как устанавливается связь в канале. Эти обратные вызовы способны выполнять логику (настраиваемую) инициализации канала, если какой-либо из них возвращает ошибку, соединение канала завершится ошибкой. В конечном итоге, возвращая ошибки при обратном вызове, модули способны программно принимать и отклонять каналы.

Успешное соединение канала происходит в четыре этапа. Вкратце, если блокчейн A хочет открыть канал с блокчейном B, используя уже установленное соединение. Происходит это следующим образом:

1. блокчейн A отправляет сообщение ChanOpenInit, чтобы сигнализировать о попытке инициализации канала блокчейна B.
2. блокчейн B отправляет сообщение ChanOpenTry, чтобы попытаться открыть канал в блокчейн A.
3. блокчейн A отправляет сообщение ChanOpenAck, чтобы пометить свой конечный статус канала как открытый.
4. блокчейн B отправляет сообщение ChanOpenConfirm, чтобы пометить конечный статус своего канала как открытый.

Если все это сложится успешно, то канал будет открыт с обеих сторон. На каждом этапе соединения модуль, связанный с ChannelEnd, будет выполнять обратный вызов для этого этапа соединения. Таким образом, в ChanOpenInit модуль в блокчейне A будет выполнять обратный вызов OnChanOpenInit.

Подобно тому, как порты поставлялись с динамическими возможностями, инициализация канала вернет динамическую возможность, которую должен требовать модуль, чтобы они передать возможность аутентификации действий канала, например как отправка пакетов. Возможности канала передаются в обратный вызов в первой части этапа соединения; либо OnChanOpenInit в инициализирующем блокчейне, либо OnChanOpenTry в другом блокчейне.

Пакеты

Модули взаимодействуют друг с другом путем отправки пакетов по каналам IBC. Как упоминалось ранее, все пакеты IBC содержат идентификатор порта назначения и идентификатор канала вместе с идентификатором порта источника и идентификатором канала. Это позволяет модулям узнать модуль отправителя данного пакета. Пакеты IBC содержат последовательность для необязательного упорядочивания. Пакеты IBC также содержат TimeoutTimestamp и TimeoutHeight. Если TimeoutTimestamp и TimeoutHeight не равны нулю, то они будут определять крайний срок, до которого принимающий модуль должен обработать пакет. Если по истечении тайм-аута пакет не был успешно обработан, отправляющий модуль в праве приостановить пакет и предпринять соответствующие действия.

Модули отправляют друг другу данные пользовательского приложения в поле Data []byte пакета IBC. Таким образом, пакетные данные полностью недоступны для обработчиков IBC. Модуль-отправитель должен кодировать информацию о пакете, зависящую от приложения, в поле пакетов Data. Модуль-получатель в свою очередь должен декодировать поле Data обратно в исходные данные приложения.

Квитанции и таймауты

Так как IBC работает в распределенной сети и полагается на потенциально неисправные ретрансляторы для ретрансляции сообщений между регистрами, IBC должна обрабатывать случай, когда пакет не отправляется к месту назначения своевременно, либо не оправляется вовсе. Таким образом, пакеты должны указывать высоту тайм-аута или временную метку тайм-аута, после которой пакет больше не может быть успешно принят в блокчейне назначения.

В том случае, если тайм-аут действительно достигнут, то доказательство тайм-аута пакета может быть отправлено в исходный блокчейн, который после этого может выполнить логику приложения для тайм-аута пакета. Сделать он это может путем отката изменений отправки пакета (возмещение отправителям любых заблокированных средств и так далее) .

В упорядоченных каналах тайм-аут всего одного пакета приведет к закрытию канала. Если для последовательности пакетов n истекает время ожидания, то ни один пакет с последовательностью k> n не может быть успешно принят без нарушения контракта упорядоченных каналов, согласно которому пакеты обрабатываются в том порядке, в котором они отправлены. Так как упорядоченные каналы обеспечивают соблюдение этого правила, то доказательства того, что последовательность n не была получена в блокчейне назначения по истечении указанного тайм-аута пакета n, будет достаточно для тайм-аута пакета n и для закрытия канала.

Давайте рассмотрим неупорядоченные каналы, где пакеты могут приниматься в любом порядке. Таким образом, IBC будет писать квитанцию о получении пакета для каждой последовательности, полученной в неупорядоченном канале. Эта квитанция не содержит информации, это просто маркер, который предназначен для обозначения того, что неупорядоченный канал получил пакет в указанной последовательности. На неупорядоченном канале для тайм-аута пакета необходимо предоставить доказательство того, что получение пакета не существует для последовательности пакетов к указанному тайм-ауту. Важно отметить, что тайм-аут пакета на неупорядоченном канале просто запустит логику тайм-аута приложения для этого пакета и не закроет канал.

По этой причине большинству модулей приходится использовать неупорядоченные каналы. Это связано с тем, что они требуют меньше гарантий живучести для эффективного функционирования пользователей этого канала.

Подтверждения

Для конкретного приложения при обработке пакета модули могут выбрать запись подтверждений. Это может быть выполнено синхронно на OnRecvPacket, если модуль обрабатывает пакеты в тот самый момент, как они получены от модуля IBC. Также они могут выполняться асинхронно в том случае, если модуль обрабатывает пакеты позже, то есть уже после получения пакета.

Несмотря на это, эти данные подтверждения непрозрачны для IBC, как и пакеты Data. Они будут обрабатываться IBC как простая строка байтов [] byte. Модули-получатели обязательно должны кодировать свою сеть подтверждения таким образом, чтобы модуль-отправитель мог ее правильно декодировать. Это должно быть реализовано путем согласования версии во время подтверждения канала.

Подтверждение может закодировать, была ли обработка пакета успешной или неудачной, а также дополнительную информацию, которая позволит модулю-отправителю предпринять соответствующие действия.

Как только подтверждение будет записано принимающей цепочкой, ретранслятор будет ретранслировать подтверждение обратно в исходный модуль отправителя. Тот в свою очередь выполнит логику подтверждения для конкретного приложения, используя содержимое подтверждения. Это может включать откат изменений отправки пакетов при неудачном подтверждении (возврат отправителям).

После успешного получения подтверждения от исходного отправителя в цепочке, модуль IBC удаляет соответствующее подтверждение пакета, потому что оно больше не требуется.

Дальнейшая документация и спецификации

Если вы хотите получить больше информации о протоколе IBC, ознакомьтесь со следующими спецификациями:

Обзор спецификации IBC

Спецификация протокола IBC SDK

17. Индексация транзакций

Tendermint дает возможность индексировать транзакции, а после - запрашивать их результаты или подписываться на них.

События могут использоваться для индексации транзакций и блоков в соответствии с тем, что происходило во время их выполнения. Просим обратить внимание на то, что набор событий, возвращаемых для блока из BeginBlock и EndBlock, объединяются. Если оба метода возвращают один и тот же тип, то используются только пары ключ-значение, определенные в EndBlock.

Каждое из событий содержит в себе тип и список атрибутов. Они представляют собой пары ключ-значение, которые обозначают некую информацию о том, что происходило в то время, когда выполнялся метод. Если вы хотите изучить дополнительные сведения об Events см. В документации ABCI.
Каждое выполняемое событие имеет связанный с ним составной ключ. compositeKey создается по его типу и ключу, разделенным точкой.

Например:

будет равен составному ключу jack.account.number.

Давайте обратим внимание на секцию конфигурации [tx_index]:

По умолчанию Tendermint индексирует все транзакции по их соответствующим хэшам и высоте. Это происходит при помощи встроенного простого индексатора.

Вы в праве полностью отключить индексацию. Для этого потребуется установить для tx_index значение null.

Добавление событий

Приложения могут самостоятельно определить, какие события индексировать. Tendermint не предоставляет функциональные возможности для определения того, какие события индексировать, а какие игнорировать. В методе DeliverTx вашего приложения добавьте поле Events с парами строк в кодировке UTF-8 (например, «transfer.sender»: «Bob», «transfer.recipient»: «Alice», «transfer.balance»: «100») .

Пример:

Транзакция будет проиндексирована (при условии, что индексатор не равен нулю) с определенным атрибутом, если в поле Index атрибута установлено значение true. В примере, приведенном выше будут проиндексированы все атрибуты.

Запрос транзакций
Вы самостоятельно можете запросить результаты транзакции. Для этого нужно вызвать конечную точку RPC/tx_search:

Если вам потребуется дополнительная информация о синтаксисе запроса и других параметрах, вы можете подробнее ознакомиться с документацией API.

Подписка на транзакции

Пользователи могут подписаться на транзакции с указанными тегами посредством WebSocket. Для этого нужно отправить запрос/подписаться на конечную точку RPC.

Для получения дополнительной информации о синтаксисе запроса и других параметрах, ознакомьтесь с документацией API.

18. Приложение для опроса

Мы будем создавать простое приложение для опросов. В этом приложении у пользователя будут следующие возможности: входить в систему, создавать опросы, отдавать голоса и видеть результаты голосования. Стоимость создания вопроса - 200 токенов, голосование бесплатное. Оба действия будут доступны только для пользователей, прошедших авторизацию.

Требования

В данном руководстве мы будем использовать Starport v0.14.0. Это инструмент для создания блокчейнов, который довольно прост в использовании. Чтобы установить звездный порт в /usr/local/bin, нужно выполнить следующую команду:

Также вы можете использовать Starport v0.14.0 в Интернете в среде IDE на основе браузера. Если у вас имеется желание узнать больше о других способах установки Starport.

Создание блокчейна

Чтобы создать проект избирателя, выполните следующую команду:

Команда Starport app создаст структуру проекта для вашего приложения в каталоге voter. Не забудьте заменить alice своим именем пользователя GitHub.

Внутри каталога избирателя мы наблюдаем несколько файлов и других каталогов:

- app содержит файлы, которые соединяют все движущиеся части вашего приложения.
- cmd отвечает за программы voterd и votercli, которые соответственно позволяют запускать приложение и выполнять с ним какие-либо взаимодействия.
- vue содержит пользовательский веб-интерфейс для вашего приложения, который отвечает за все, что вы видите на скриншоте выше.
- x содержит основные конструктивные блоки вашего приложения: модули. В данном случае у нас имеется только один: voter.

Каталог нашего проекта содержит весь код, необходимый для создания и запуска приложения на основе блокчейна. Давайте попытаемся запустить наше приложение. Для этого следует запустить starport serve внутри нашего проекта:

Тут вы должны увидеть следующий вывод, а также любые ошибки, которые могут появиться в вашем приложении.

Примечание: используйте starport serve --verbose для визуализации подробных операций, которые происходят в фоновом режиме.

Поздравляю! Теперь у вас есть собственное блокчейн-приложение, запущенное на вашем компьютере всего двумя командами. Но приложение пока что не выполняет каких-либо функций, так что давайте мы над этим поработаем.

В наших приложениях для голосования имеются два типа объектов: опросы и голоса. у типа “Опрос” имеется заголовок и список вариантов.

Добавление опросов

Откройте новый терминал в каталоге избирателей и выполните следующее:

Данная команда сгенерировала код, который служит для обработки создания элементов poll. Если мы сейчас выполним запуск starport serve и перейдем по адресу http://localhost:8080, мы увидим форму для создания опросов. Перестройка приложения может занять некоторое время, поэтому подождите несколько секунд.

Выполните вход в систему, используя один из паролей, который отображается в консоли. Далее попытайтесь создать опрос. Вы должны увидеть новый объект, созданный и отображаемый над формой. Вы успешно создали объект и сохранили его в блокчейне!

Однако это выглядит и работает не так, как нам нужно. Мы должны иметь возможность добавлять поля параметров (и сохранять их в виде массива), и они должны отображаться в виде интерактивных кнопок.

Давайте взглянем на некоторые файлы, измененные командой типа starport.

#x/voter/types/TypePoll.go

Этот файл содержит определение типа Poll. Мы видим, что в опросе имеются два поля (создатель и идентификатор), которые будут сгенерированы автоматически. Также имеются два поля (заголовок и параметры), которые определены нами. Поскольку мы хотим, чтобы Options были списком строк, замените string на []string

x/voter/types/MsgCreatePoll.go

Этот файл определяет сообщение, которое создает опрос.

Нам нужно сделать так, чтобы параметры сохранялись в виде списка, а не строк. Замените Options string на строку Options []string в структуре MsgCreatePoll и options string на options []string в аргументах функции NewMsgCreatePoll.

Для того, чтобы записать что-либо в блокчейн или выполнить любой другой переход состояния, клиент (в нашем случае веб-приложение) отправляет HTTP-запрос POST с заголовком и параметрами http://localhost:1317/voter/poll обработчику конечной точки, для которого определено в x/voter/client/rest/txPoll.go. Обработчик создает неподписанную транзакцию, которая содержит массив сообщений. Далее клиент подписывает транзакцию и отправляет ее по адресу http://localhost:1317/txs. Затем приложение выполняет обработку этой транзакции, отправляя каждое сообщение соответствующему обработчику, в нашем случае x/voter/handlerMessageCreatePoll.go. Затем обработчик вызывает функцию CreatePoll, определенную в x/voter/keeper/poll.go. В свою очередь она записывает данные опроса в хранилище.

x/voter/types/MsgSetPoll.go

Также в MsgSetPoll нам нужно изменить наш Options string на []string

x/voter/client/rest/txPoll.go

Замените Options string на Options []string в структуре createPollRequest.

Также ниже в структуре setPollRequest.

x/voter/client/cli/txPoll.go

Пользователь также сможет взаимодействовать с нашим приложением через интерфейс командной строки.

Данная команда сгенерирует транзакцию с сообщением «создать опрос», после этого подпишет ее, используя закрытый ключ пользователя user1 (один из двух пользователей, созданных по умолчанию), и передаст ее в блокчейн.

Нам нужно сделать следующую модификацию, а именно - изменить строку, которая считывает аргументы из консоли.

В функции GetCmdCreatePoll

replace

на

Переменная msg определена для чтения строки argOptions и удаления привязки строк

В итоге получаем следующую функцию

Такие же изменения нужно будет сделать для функции GetCmdSetPoll.

И в GetCmdSetPoll мы меняем

на

Предполагается, что все аргументы, которые идут после первого, представляют собой список опций.

Получаем следующую функцию

Теперь, когда мы внесли все необходимые изменения в наше приложение, давайте посмотрим на клиентское приложение.

Фронтенд приложение

Компания Starport создала базовый интерфейс для нашего приложения. Для удобства фреймворк Vue.js используется с Vuex для управления состоянием. Но поскольку через HTTP API доступны все функции нашего приложения, клиенты могут быть созданы с использованием любого языка или фреймворка.

В основном нас будет интересовать каталог vue src/views. В нем содержатся шаблоны страниц нашего приложения. vue/src/store/index.js обрабатывает отправку транзакций и получение данных из нашего блокчейна и @tendermint/vue каталог, в котором содержатся компоненты, такие как кнопки и формы.

Внутри vue/src/store/index.js мы импортируем @tendermint/vue/src/store/cosmos, который использует CosmJS, библиотеку для обработки кошельков, создания, подписания и трансляции транзакций. Также определяем магазин Vuex. Мы будем пользоваться функцией entitySubmit. Она служит для отправки данных в наш блокчейн (например, JSON, представляющий только что созданный опрос), entityFetch служит для запроса списка опросов и accountUpdate для получения информации о балансе наших токенов.

vue/src/views/Index.vue

Поскольку компонент формы по умолчанию нам не нужен, замените

внутри vue/src/views/Index.vue с новым компонентом и заголовком

В тегах <script> </script> ниже мы импортируем наш компонент следующим образом

Для нашего PollForm нам следует создать новые компоненты каталога в нашем vue/src/path. В данном каталоге мы создаем новый файл PollForm.vue и наполняем приложение нашим первым компонентом.

vue/src/components/PollForm.vue

Создаем наш компонент PollForm. У него будет заголовок и две кнопки.

между тегами <script> </script> под этим:

Нам также обязательно нужно настроить наш магазин Vue.

vue/src/store/index.js

В нашем основном файле App.vue мы обязательно инициализируем функции Cosmos Wallet, которые созданы нами. Делается это следующим образом:

vue/src/App.vue
В теге <script> </script> в конце файла мы отправляем для инициализации нашего собственного приложения и фреймворка cosmos Vue.

Далее обновляем страницу, входим с паролем и создаем новый опрос. Обработка транзакции занимает несколько секунд. Теперь, если вы посетите http://localhost:1317/voter/poll, вы должны увидеть список опросов (эта конечная точка определена в x/voter/rest/queryPoll.go):

Добавление голосов

Тип голосования содержит в себе идентификатор опроса и значение (строковое представление выбранной опции).

vue/src/views/Index.vue

Удалите только что загруженный для нас <sp-type-form type = "vote": fields = "['pollID', 'value',]" module = "voter" />. Добавьте <poll-list /> в файл vue/src/view/Index.vue после только что созданного компонента формы опроса.

Обновите импортированные ниже теги <script> </script>. Сделать это можно следующим образом:

Далее создайте новый компонент в vue/src/components/PollList.vue и добавьте следующее:

vue/src/components/PollList.vue

между тегами <script> </script> под этим:

Компонент PollList перечисляет для каждого опроса все параметры. Делается это в виде кнопок. Выбор опции запускает метод отправки, который в свою очередь транслирует транзакцию с сообщением «создать голосование» и получает данные обратно из нашего приложения.

В нашем приложении по-прежнему не достает двух компонентов. Для того, чтобы оно выглядело лучше, добавим AppRadioItem.vue и AppText.vue.

vue/src/components/AppRadioItem.vue
между тегами <script> </script> под этим:

vue/src/components/AppText.vue

между тегами <script> </script> под этим:

Теперь в нашем App.vue нам нужно выполнить обновление, чтобы получить наши голоса. В теге <script> </script> у нас должен иметься результирующий код, который должен выглядеть следующим образом:

vue/src/App.vue

К этому моменту должен быть виден тот же интерфейс, что и на первом снимке экрана. Попробуйте создавать опросы и отдавать голоса. Вы можете обратить внимание на то, что в одном опросе можно отдать сразу несколько голосов. Это не то, что нам нужно. Давайте же исправим это.

Единоразовое голосование

Чтобы решить эту проблему, для начала следует понять, как хранятся данные в нашем приложении.

Мы можем считать наше хранилище данных лексикографическим упорядоченным хранилищем значений ключа. Вы можете перемещаться по записям в цикле, фильтровать по префиксу ключа, добавлять, обновлять и удалять записи. Визуализировать магазин проще в формате JSON:

И poll-, и vote- являются префиксами. Они добавлены к клавишам для более удобной фильтрации. По соглашению префиксы определены в x/voter/types/key.go.

x/voter/keeper/vote.go

Каждый раз, когда пользователь голосует, новое сообщение «создать голос» обрабатывается обработчиком и передается хранителю. Keeper принимает префикс vote-, добавляет UUID (уникальный для каждого сообщения) и использует эту строку в качестве ключа.
x/voter/keeper/vote.go:

Эти строки уникальны, и мы получаем голоса, которые дублируются. Дабы этого избежать, нам следует убедиться, что хранитель производит запись каждого голоса только один раз, выбирая правильный ключ. Для нашего случая мы можем использовать идентификатор опроса и адрес создателя. Это нужно для того, чтобы один пользователь мог отдать только один голос за опрос.

Перезапустите приложение и попробуйте проголосовать несколько раз в одном опросе. Как вы могли заметить, вы можете голосовать столько раз, сколько хотите, однако засчитывается только ваш последний голос.

Введение комиссии за создание опросов
Давайте сделаем так, чтобы создание опроса стоило к примеру 200 токенов.

Данную функцию добавить очень просто. Мы уже требуем, чтобы у пользователей были зарегистрированные учетные записи, и у каждого пользователя имеются токены на балансе. Единственное, что нам предстоит сделать, это наладить отправку монет из учетной записи пользователя в учетную запись модуля, прежде чем нами будет создан опрос.

x/voter/handlerMsgCreatePoll.go:

Оплата комиссии происходит до k.CreatePoll (ctx, poll). Таким образом, если у пользователя недостаточно токенов для оплаты, приложение выдаст ошибку и не перейдет к созданию опроса. Не забудьте добавить «github.com/tendermint/tendermint/crypto» в оператор импорта (если ваш текстовый редактор не сделал это за вас).

Теперь перезапустите приложение и снова попробуйте создать несколько опросов, чтобы увидеть, как это повлияет на баланс вашего токена.

Поздравляем, вы создали приложение для голосования на блокчейне.

19. Tutorilals: Burner chain, welcome

Добро пожаловать

Вступление

Данное обучение изначально было опубликовано Билли Реннекампом на воркшопе на хакатоне ETH Denver в 2020 году
Добро пожаловать в первую официальную сеть Cosmos Burner Chain! Сохраняя традиции проекта Burner Wallet (и других временных цепочек с низким уровнем безопасности), мы разработали и с успехом запустили игровой блокчейн, который будет использоваться во время Eth Denver Hackathon 2020. Цель этого блокчейна - предоставить план для вашего собственного блокчейна, который будет разработан для конкретного приложения. Он так же будет взаимодействовать с другими сетями на основе EVM. Это такие сети, как Ethereum и xDai.

Цепь Cosmos Burner состоит из нескольких важных составляющих. Первая из них - это сам Cosmos SDK. Это фреймворк, очень похожий на Ruby-on-Rails. Он служит для создания блокчейнов для конкретных приложений. Поставляется он с набором стандартных модулей, которые служат для обработки общего набора функций: Банк (взаимозаменяемые токены, такие как ERC-20), Аутентификация (учетные записи, защищенные криптографией с открытым ключом), Стейкинг + Слэшинг (благодаря совместной работе эти модули обрабатывают набор валидаторов для подтверждения ставки, используя Tendermint Consensus с условиями обрезания для двусмысленности и простоя), а также Params (текущие обновления цепочки). Ко всему прочему, имеется модуль управления Gov, который можно использовать для текстовых предложений, изменений параметров и управления пулом средств сообщества и некоторых иных задач более низкого уровня.

Помимо основных модулей SDK, наш Burner Chain использует Bank, Auth, Staking и Slashing. Также помимо стандартного тарифа, он использует Peggy. Этот новый модуль был разработан Swish Labs при поддержке Intechain Foundation. По своей сути, он является мостом между несколькими цепочками EVM, такими как Ethereum и xDai. Peggy может быть использован для транспортировки взаимозаменяемых токенов. Он вполне может быть расширен для передачи NFT, а также произвольных сообщений.
Заключительной частью нашего блокчейна является Scavenge. Это модуль, который является частью репозитория sdk-tutorials. Он позволяет опубликовать загадку с наградой за того, кто сможет ее разгадать.

В конечном итоге получаем следующие части:

- Банк
- Аутентификация
- Стейкинг
- Слэшинг
- Peggy
- Scavenge

20. Peggy/Scavenge + BuffiDao

Цель блокчейна - продвижение игры "Охота на мусорщиков", чтобы у нее была возможность взаимодействовать с игрой Eth Denver BuffiDao. У Cosmos для выдачи имеются 1200 XP и 2 значка NFT (Big Brain и Cosmic Brain). Мы заблокировали их всех в рамках контракта Peggy Bridge, который развернут в цепочке xDai. Этот контракт контролируется валидаторами нашего блокчейна записи, которые в свою очередь ретранслируют транзакции между двумя блокчейнами. Для получения опыта в сети Cosmos Burner Chain, сначала вам необходимо зарегистрироваться, для того, чтобы вы могли участвовать в нашем опросе. Скорее всего вы уже прошли регистрацию, если вы находитесь здесь, но если вы этого еще не сделали, то заходите сюда. После этого вы в праве запросить свои XP в Burner Chain Cosmos, где в дальнейшем вы сможете переместить XP обратно в контракт BuffiDai на xDai, используя мост Peggy ETH.

Для того, чтобы выиграть больше БОЛЬШЕ XP, а также NFT с бронзовым и серебряным значками, вам потребуется сыграть в игру Scavenge! Эта игра основана на серии загадок и охоты за мусорщиками. Результат успешной игры - секретные ответы, которые отправляются через commit-reveal в Cosmos Chain. Пользователи с правильными результатами получают свое вознаграждение ввиде XP, а также собственным токеном блокчейна записи, называемым brain. Два лучших обладателя brain по итогам хакатона получат два значка NFT - Big Brain и Cosmos Brain.