3 Создание первого децентрализованного приложения

Достаточно теории, давайте займемся практикой. Далее я буду полагать, что у вас уже есть опыт разработки хотя бы одного программного приложения. Создание децентрализованного приложения не намного труднее создания обычного приложения. Основные усложнения заключаются в необходимости думать децентрализованными категориями и в отсутствии большого количества зрелых библиотек, которыми широко пользуются разработчики обычных приложений.

Данная глава проведет вас через создание децентрализованной версии Twitter и познакомит с:

- языком программирования Go;
- децентрализованной архитектурой и распределенным хранилищем данных IPFS;
- Kerala, интерфейсом к IPFS;

- Coinprism, электронным кошельком для окрашенных монет;
- Mikro, децентрализованным приложением для обмена сообщениями, обладающим внутренней экономикой.

Go

Мы напишем наше децентрализованное приложение на языке Go. Go вызвал огромный интерес со стороны разработчиков серверных приложений благодаря простому синтаксису, свободному от функций обратного вызова, высокой скорости и дружественному механизму параллельных вычислений в лице go-сопрограмм (go-routines). Разработчики двух других языков, Erlang и Rust, заявляют, что последние превосходят Go, и, возможно, так и есть в некоторых аспектах, но в отличие от библиотек Go библиотеки Erlang и Rust находятся на очень, очень ранней стадии развития.

JavaScript — еще один довольно популярный в наши дни язык программирования; с появлением Node.js разработчики на JavaScript больше не ограничены областью создания клиентских сценариев. Они могут создавать и развивать целые веб-стеки, пользуясь единственным языком (и, конечно, HTML/CSS). JavaScript — это язык Всемирной паутины, и разработчики на JavaScript могут применять для создания своих веб-приложений большое разнообразие фреймворков для JavaScript. Хотя JavaScript — мощный язык, он имеет свои слабые стороны. Он плохо приспособлен для реализации параллельных вычислений и имеет довольно запутанную объектную модель. Go лишен этих недостатков и лучше приспособлен для создания распределенных систем.

Мне приходилось писать веб-приложения на обоих языках, Go и JavaScript. Оба они имеют свои сильные и слабые стороны, но, на мой взгляд, Go лучше подходит для разработки децентрализованных приложений. Компания Google создала Go, потому что ей требовался язык, позволяющий реализовать максимально эффективные параллельные вычисления с огромными наборами данных в масштабе, характерном для нее. Go решил эту проблему, и с момента выхода его первой версии значительно увеличилось его применение для внутренних нужд Google.

Go обладает широтой возможностей и скоростью компилирующего языка C, а также элегантностью и выразительностью Ruby. Он предназначался для разработки распределенных систем, именно поэтому я возвращаюсь к нему, когда задумываюсь о создании децентрализованных приложений. Большим плюсом является тот факт, что IPFS написана на Go, — это упрощает интеграцию распределенного хранилища файлов в приложение благодаря отсутствию барьеров совместимости. В настоящее время существует множество веб-фреймворков на основе языка Go: Martini, Goji, Gorilla и даже стандартный пакет net/http. Мне нравится, когда стек зависимостей остается максимально легковесным, поэтому в своих приложениях я в основном использую пакет net/http и прибегаю к помощи других библиотек, только когда это действительно необходимо.

Централизованная архитектура

Существуют три распространенные парадигмы создания стандартных клиент-серверных веб-приложений. Поговорим немного о них.

REST

Относительно простая модель клиент-сервер, фактически ставшая стандартным способом организации обмена данными во Всемирной паутине. REST, или Representational State Transfer (передача репрезентативного состояния), — это набор правил и приемов создания масштабируемых веб-приложений, обычно основанных на модели клиент-сервер. REST — название практики (так же как AJAX), а не отдельной технологии. Она поощряет использование возможностей, которые давно имеются в протоколе HTTP, но редко применяются. Пользователь может просто ввести в адресной строке браузера строку URL (Uniform Resource Locator — единообразный локатор (определитель местонахождения) ресурса), в результате чего браузер пошлет HTTP-запрос. Каждый HTTP-запрос несет информацию в форме параметров, которую сервер может использовать, чтобы решить, какой HTTP-ответ послать обратно клиенту.

CRUD

Аббревиатура CRUD расшифровывается как Create-Read-Update-Delete (создать-прочитать-изменить-удалить). Это перечень основных операций, которые могут выполняться с данными в хранилище. С помощью указанных операций вы напрямую работаете с записями или объектами данных; без них записи являются всего лишь пассивными сущностями — как правило, таблицами и записями в базе данных. В то время как REST определяет стиль взаимодействий с действующей системой, CRUD описывает методы управления данными в системе. Обычно для выполнения операций CRUD со своими данными разработчики используют базы данных, такие как MongoDB или MySQL.

MVC

Аббревиатура MVC расшифровывается как Model-View-Controller (модель-представление-контроллер); в настоящее время MVC является наиболее популярной парадигмой программирования. Модели отвечают за логику и данные приложения. Представления отображают пользовательский интерфейс приложения. Контроллеры принимают ввод пользователей и производят необходимые вызовы методов объектов моделей и представлений для выполнения конкретных действий.

Децентрализованная архитектура: введение в IPFS

Итак, что происходит с парадигмами CRUD и REST в децентрализованной архитектуре? Они фактически объединяются в единую парадигму, поскольку данные находятся в децентрализованной сети компьютеров и не принадлежат никому конкретно, как это имеет место в случае с IPFS. Операции с локальными данными выполняются точно так же, как с удаленными. Вы и все остальные одновременно являетесь и серверами, и клиентами. Данное объяснение звучит сложнее, чем есть на самом деле. В качестве решения для организации хранения данных я выбираю IPFS, потому что она продвинулась намного дальше, чем ее конкуренты, и за годы исследований породила множество великолепных идей, подтвержденных на практике.

Децентрализованные приложения выполняются не на сервере, а локально, на всех компьютерах пользователей. Мы все еще не решили проблему децентрализации вычислений, а выгрузка приложения в централизованную виртуальную

машину (Virtual Machine, VM), например Heroku, противоречит цели децентрализации, поэтому правильнее распространять децентрализованное приложение в виде загружаемых двоичных файлов. Пользователи смогут загрузить их на свои компьютеры и получить доступ к децентрализованному приложению либо с помощью веб-браузера, либо напрямую, используя интерфейс клиента, например Spotify или Skype.

Децентрализованным приложениям необходимо хранилище данных в той или иной форме, и как таковые они одновременно будут играть роль узлов распределенного хранилища файлов на основе IPFS. Как вариант для хранения данных можно просто использовать сторонний узел IPFS на сервере, но тогда поставщик облачных услуг станет центральной точкой отказа. Неизбежно кто-то решит купить некоторое пространство на Amazon EC2, развернет там узел и предложит узел-IPFS как услугу, чтобы облегчить жизнь начинающим разработчикам. В этом случае данные будут копироваться оттуда, по мере того как люди будут запрашивать файлы. Облачный узел IPFS мог бы также пригодиться для мобильных децентрализованных приложений, учитывая, что отдельный узел IPFS принимает на себя значительную долю вычислений, связанных с обработкой, и помогает сэкономить значительную долю заряда аккумулятора ноутбука.

Работа узлов может стимулироваться выгружающими данными для хранения путем поощрения в долларах или криптовалюте. Создатель IPFS Хуан Бенет (Juan Benet) опубликовал статью (*http://filecoin.io/filecoin.pdf*) с описанием криптовалюты FileCoin, предназначенной именно для этого, но работа над ее реализацией так и не была начата, поэтому она не может дать нам никаких выгод. Меж-

ду тем любой желающий может предложить свои схемы стимулирования хранения данных в IPFS, чтобы избавиться от необходимости держать узлы постоянно подключенными к сети для доступа к данным, хранящимся на них. Чем выше уровень децентрализации, тем лучше. Даже если сервер узла IPFS отключился, а данные, хранящиеся на нем, пользуются спросом, весьма вероятно, что копия уже хранится у кого-то еще, запрашивавшего эти данные ранее. Таково одно из основных достоинств IPFS и причина, почему создатель называет ее permanent web (постоянная сеть). Теоретически вы можете также оплатить услуги сервера за закрепление ваших данных. Кому-то эти данные не нужны прямо сейчас, но могут понадобиться позже. Пока кому-то нужны ваши данные, они будут существовать.

Было бы здорово создать мобильное приложение, но в данном учебном примере я сосредоточусь на децентрализованном приложении для настольного компьютера, потому что в Swift/ObjC или Android все еще нет надежной обертки для IPFS.

Рассмотрим две ключевые команды в IPFS.

`ADD`

Добавляет данные в IPFS.

`CAT`

Читает данные из IPFS.

Обратите внимание на отсутствие команды `delete`. IPFS — постоянная сеть! После добавления данных в сеть их нельзя удалить, если только вы не являетесь единственным их

хранителем. Это обусловлено тем, что другие узлы будут хранить копию данных, как только они ее получат. Отметьте также отсутствие команды `update`, что связано с тем, что IPFS внутренне использует технологию Git. Изменяя файл, вы не удаляете его, а сохраняете разницу между версиями. Вы можете создать граф merkleDAG для этого файла, чтобы последний хеш соответствовал последней версии файла. Все прежние версии продолжают существовать, и вы при желании сможете извлечь их.

Добавляя данные в сеть, вы фактически рассылаете широковещательное сообщение, что у вас есть некоторые данные, а не посылаете их на другие компьютеры. Передача данных происходит только тогда, когда кто-то их запрашивает. А поскольку данные находятся в сети, манипуляции данными осуществляются командами, посылаемыми в сеть.

IPNS (уровень поддержки имен, располагающийся над IPFS) создает видимость возможности изменения и удаления через изменение имен. С помощью IPNS вы можете опубликовать DAG данных под своим неизменяемым идентификатором, и тогда любой, знающий ваш идентификатор, сможет получить хеш DAG. IPNS может хранить только один вход в DAG для каждого идентификатора, поэтому, если понадобится изменить или удалить данные, достаточно будет просто опубликовать новый DAG для данного идентификатора. Более подробно реализация будет обсуждаться далее в главе.

А что насчет архитектуры MVC?

Она имеет здесь место. Значит, нет никакой новейшей методологии структурирования моего кода? Нет, модели остаются прежними, контроллеры используют IPFS для

сохранения и извлечения данных, а представления — это самый обычный код HTML/CSS/JavaScript.

А умные контракты? Какую роль играют они?

В децентрализованном приложении есть определенные элементы, требующие достижения консенсуса через умные контракты, для чего обычно требуется сервер. Отличный пример — имена пользователей, как и финансовые операции, такие как депонирование и приобретение имущества. Технически умные контракты можно сравнить с моделями — вы можете передавать им свои данные через транзакции — но в действительности они не являются моделями в архитектуре MVC. Они могут действовать наряду с имеющимися моделями, но область их применения распространяется только на конкретные сценарии. Мы еще вернемся к умным контрактам далее и узнаем, как они реализуются. Как говорится, нам нужны умные модели, тонкие контроллеры и простые представления.

Eris Industries создала фреймворк для разработки децентрализованных приложений, который называется Decerver. На веб-сайте проекта можно найти большое количество литературы, описывающей, как пользоваться фреймворком, а также все его революционные методики, помогающие упростить создание децентрализованных приложений. Там говорится, что модели — умные контракты, но проблема в том, что умные контракты действуют по принципу «плати и играй» и не должны зависеть от создания моделей. Это ненужная сложность. Однако архитектура MVC все еще может применяться для создания децентрализованных приложений, и ваши контроллеры будут взаимодействовать с цепочками блоков и DHT вместо серверов.