ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет об программном проекте на тему:

Серверное приложение для предоставления данных по протоколу WebDAV

Выполнил:

студент группы БПМИ205
Шитов Даниил Сергеевич

(подпись) 18.05.2023 (дата)

Принял руководитель проекта:

Родригес Залепинос Рамон Антонио
Доцент
Департамента программной инженерии
ФКН НИУ ВШЭ

Rodriges (подпись) 18.05.2023 (дата)

Москва 2023

Содержание

Аннотация

Проект про создание серверного приложения, которое взаимодействует с клиентом по протоколу `WebDAV`. Протокол `WebDAV` по сути является расширением протокола `HTTP`, он открывает много новых возможностей для работы с сервером несколькими пользователями одновременно. Сейчас существует множество приложений, которые используют этот протокол, но в основном работа сервера сильно заточена под какую-то конкретную задачу. Основная цель этого проекта – создать простое, понятное и легкое в использовании серверное приложение, которое позволит работать с файлами по протоколу `WebDAV`, а также дополнит его функциональность возможностью версионирования.

Ключевые слова

WebDAV, DeltaV, HTTP, Версионирование, URI, Git

1 Введение

Наверное, самый популярным протоколом прикладного уровня (L7), по которому сервер и клиент обмениваются информацией, сейчас является `HTTP`. Все потому, что он предоставляет простой и прозрачный формат взаимодействия – небольшой набор примитивных методов: `GET`, `HEAD`, `POST`, `PUT`, и т.д. Во многих случаях этого хватает для работы с ресурсами на сервере. Естественно, со временем появилась потребность в создании протокола, который бы обеспечивал более гибкую функциональность. В качестве оправдания можно привести ситуацию, когда несколько пользователей хотят одновременно работать с ресурсом на сервере, в таком случае во избежании порчи ресурса распараллелить можно только идемпотентные `HTTP` методы, то есть те, которые при повторном выполнении будут давать тот же результат, что и при первом. Таким образом целью при создании протокола `WebDAV` было дополнить уже знакомый всем протокол `HTTP` так, чтобы у клиентов появилась возможность распределенно работать с ресурсами на сервере. По аналогии с локальными механизмами для многопоточных программ были добавлены блокировки, которые позволяют избежать потерю информации при одновременной модификации ресурсов несколькими пользователями. Также протокол `WebDAV` позволяет сохранять и модифицировать мета-информацию для всех ресурсов, что делает структуру ресурсов на сервере похожей на обычную файловую систему. И последнее дополнение протокола – возможность копировать и перемещать ресурсы из одного URI в другой. Таким образом протокол `WebDAV` сильно расширил возможности протокола `HTTP`, это открыло возможности для создания более сложного взаимодействия между клиентами и сервером.

Сейчас можно встретить такие решения:

- Распределенных хранилищах данных: облачные хранилища, NAS хранилища.
- Приложениях для совместной работы с файлами и документами.
- Очень простой сервер, без дополнений к основному протоколу.

В первых двух случаях протокол обычно сильно заточен под конкретные задачи приложения. В последнем случае всю функциональность сервера приходится реализовывать разработчикам самостоятельно. Но есть виды задач, в которых готовые сложные решения неуместны по разным причинам:

- Сложность настройки и разворачивания решения на сервере.
- Покупка лицензии.

- Излишняя функциональность, которая только замедлит работу требуемых задач.

В качестве примера можно привести `WebDAV` сервер, который поддерживает версионирование ресурсов, то есть возможность смотреть изменения для ресурса, откатываться к прошлым версиям. Так появился протокол `DeltaV`, он добавляет к обычному протоколу `WebDAV` методы, которые позволяют реализовать версионирование. Сейчас не было найдено готовых решений, которые легко позволяют развернуть сервер с протоколом `DeltaV`, потому что он достаточно сложный и, возможно, не очень эффективный. Цель этого проекта реализовать серверное приложение, использующее протокол `WebDav`, но при этом с версионированием.

Далее будет описание требуемой функциональности сервера, затем обзор протоколов `WebDAV` и `DeltaV`, потом описание того, что уже реализовано в используемых библиотеках. После обзора следует анализ методов реализации и выбор одного из них. В конце – описание реализации и некоторых подробностей, так же анализ производительности и заключение.

2 Понятия и определения

Версионирование – механизм, предоставляющий возможность фиксировать состояние ресурса (создавать версию), и просматривать содержимое ресурса (версии), по его идентификатору.

Версия – зафиксированное состояние ресурса (снэпшот), который имеет свой собственный идентификатор.

Блокировка – действие, которое гарантирует, что доступ к ресурсу получил ровно один клиент, остальные же вынуждены ждать её освобождение или времени, когда она станет недействительной.

Dead properties, live properties – метаинформация ресурса на сервере, dead porperties предоставляет и меняет клиент (пример: какие-то кастомные тэги объекта), live properties в основном должны контролироваться сервером (пример: автор файла, время последнего изменения).

3 Требуемая функциональность сервера

Для начала нужно было определиться на каком языке программирования планируется реализовывать сервер. Обычно серверы, в основе которых лежит обмен данными по протоколу `HTTP` или похожему (в нашем случае `WebDAV` – расширение `HTTP` протокола), не

заточены под то, чтобы делать какие-то сложные вычисления. Основная задача таких серверов: уметь слушать на сокете, обрабатывать множество соединений, которые нужны для простой передачи данных по сети. Для этого не очень хорошо подходит механизм потоков, так как они нацелены не ждать, а делать какую-то работу, лучше подходят механизмы асинхронного взаимодействия. Например, механизм корутин (coroutines), вероятно, на данный момент это самый популярный способ реализации веб-серверов, работающих с `HTTP`. Поэтому выбор языка пал на `Go`, это современный простой язык, который так же предоставляет эффективный механизм асинхронной работы: горутины (*"A goroutine is a lightweight thread managed by the Go runtime"*).

Требованию к серверу:

- Сервер должен хранить ресурсы на диске, чтобы в случае намеренного или незапланированного завершения работы, он мог перезапуститься и данные не исчезли.
- Уметь обрабатывать стандартные `HTTP` методы, как заявлено в документации [5].
- Уметь обрабатывать стандартные `WebDAV` методы, как заявлено в документации [4].
- Иметь в наличии методы, который позволят получать состояние ресурса в какой-то версии, а также создавать новые версии для ресурса.

4 Обзор протоклов и библиотек

4.1 `WebDAV`

Полное описание `WebDAV` протокола можно найти на официальном сайте [4], там описаны все методы и концепции работы протокола без реализации. Реализация же есть в виде библиотеки на `Go` [8], там есть все базовые методы, но они никак не приспособлены к версионированию. Также, в процессе работы обнаружилось, что библиотека несовершенна:

- Не до конца реализована система блокировок, а именно нет разделения на *exclusive* и *shared* блокировки, если вчитаться в код, то станет понятно, что на данный момент все блокировки *exclusive*
- Система хранения метаинформации (properties), существует только in-memory, то есть пропадет, если сервер будет перезагружен.
- Система блокировок тоже существует только in-memory.

Отдельную работу можно написать по поводу улучшения этой библиотеки, но цель этой работы другая – дописать расширение, которое добавит серверу новый функционал. Описание методов, которыми `WebDAV` расширяет протокол `HTTP`.

- `PROPFIND` позволяет получить мета-информацию ресурса. Также позволяет получить информацию о коллекции (каталоге)
- `PROPPATCH` позволяет изменить мета-информацию ресурса
- `MKCOL` позволяет создать коллекцию (каталог)
- `COPY` позволяет скопировать данные из одного URI в другой
- `MOVE` позволяет переместить данные из одного URI в другой
- `LOCK` позволяет взять клиенту блокировку на ресурсе
- `UNLOCK` позволяет снять блокировку с ресурса

4.2 `DeltaV`

Есть также стандарт [2], который как раз добавляет версионирование. Он еще расширяет протокол, некоторыми методами, основные из них:

- `VERSION-CONTROL` сообщает серверу, что теперь для ресурса включается версионирование. Во время этого запроса происходит следующее: создается version history resource, который по сути является хранилищем метаинформации о версиях, также создается ресурс версии – копия оригинального ресурса, к которому обращался запрос, копируется все содержимое и dead properties. Иллюстрация 4.1 наглядно показывает это устройство

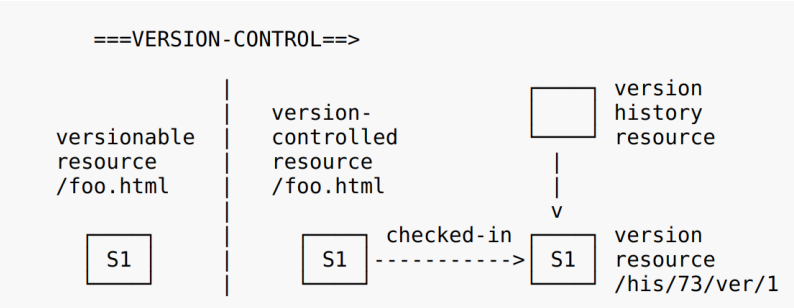

Рис. 4.1: Version-controlled resource

- `CHECKIN` создает новый ресурс версии для объекта запроса.

- `CHECKOUT` команда, которая позволяет освободить ресурс для создания новых версий. То есть после команды `CHECKIN` нельзя сразу сделать очередной `CHECKIN`, сначала требуется сделать `CHECKOUT`. Для понимания можно ознакомиться с изображением 4.2

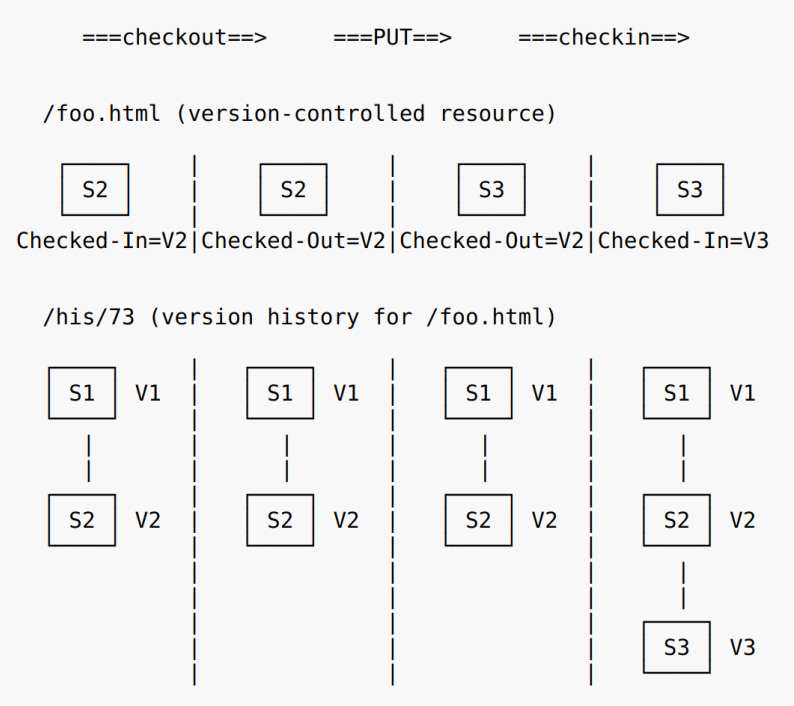

Рис. 4.2: Пример использования checkin, checkout

- `UNCHECKOUT` позволяет отменить операцию `CHECKOUT` вернуть ресурс в состояние последней версии, то есть сбросить изменения, которые еще не попали под `CHECKIN`
- `REPORT` позволяет получить информацию о версии ресурса
- Также, другие методы, которые не особо представляют интерес для этой работы: `OPTIONS`, `MKWORKSPACE`, `UPDATE`, `LABEL`, `MERGE`, `BASELINE-CONTROL`, `MKACTIVITY`

4.3 `Git`

Можно заметить, что в уже описанном протоколе 4.2 при создании новой версии создается новый ресурс. Это кажется нецелесообразным, если ресурсы достаточно большие, а изменения, наоборот, незначительные. Так появилась идея использовать какую-то систему контроля версий, самая популярная и широко используемая из которых – `git`. Он достаточно простой в использовании и при этом не требует больших накладных ресурсов для создания новых версий. Преимущество использования `git` в том, что для файлов, которые обычно хранятся на серверах (вида html, xml, json, txt), он будет хранить только изменения, а не

создавать копию всего ресурса целиком. Для каких-то бинарных файлов он создаст копию, но тут другого решения будто бы и нет. Также, он уже внутри себя содержит удобные механизмы вресионирования. Для `Go` существует клиент [1], который позволяет работать с `git`, причем он позволяет работать с файлами как на диске, так и in-memory, что присуще и библиотеке webdav [8].

5 Обзор способов реализации

Все способы предполагают использование `git`

1 Использовать разные worktree, то есть для каждого ресурса, для которого хочется поддерживать версионирование, отводить отдельную ветку и добавлять её как `git worktree add`, это удобно, потому что механизм worktree позволяет разделять работу с ресурсами одном репозитории. Также на эффктивность обычных `HTTP` запросов это не повлияет, потому что ресурсы можно все так же читать с диска.

Такой способ хорош, но его тяжело реализовать, потому что библиотека [1] не предоставляет механзма для переключениями между worktree.

2 Использовать один репозиторий, но для каждого ресурса отводить отдельную ветку. Это, вероятно, самый удобный способ, потому что здесь очевидно сопоставляются `git` операции с `DeltaV` методами:

- `VERSION-CONTROL` – `git branch -b "resource_path_hash"`
- `CHECKIN` – `git add "resource_path" git commit`
- `UNCHECKOUT` – `git reset HEAD -hard`

Но тут есть нюанс – ресурс хранится в определенной ветке, а значит на любой запрос придется делать `git checkout` на эту ветку, а потом обратно, что заметно замедлит все запросы, даже стандартные `HTTP` методы.

3 Использовать отдельный репозиторий для каждого ресурса. На первый взгляд – это самый примитивный способ, но тем не менее он хорошо работает. Стандартные `HTTP` и `WebDAV` запросы выполняются как обычно, для них нет никаких замедлений. Единственная неоптимальная часть – это создание и открытие репозитория. Создание репозитория не так уж плохо сказывается на памяти, потому что здесь у репозиториев довольно простая структура – одна ветка, все коммиты в нее. Такой репозиторий занимает 4 Кб,

что довольно мало, потому что наша задача как раз работать с большими файлами, потому что с небольшими ресурсами легче всего писалось и как раз эффективнее всего работало бы полное копирование ресурса. Открытие репозитория тоже не особо долгая операци в силу его простоты. То есть накладные расходы на работу с репозиторием растут в зависимости от его сложности, здесь же они совершенно простые, потому этот метод и выбран для реализации.

6 Реализация

Найти реализаци можно в моем github репозитории [6]

6.1 Подробности реализации

Изначальная цель была реализовать удобный и простой сервер, поэтому было принято решение не менять уже готовые библиотеки под свою задачу, а написать доработку к ним. Основная функция находится в `lib/webdavvc.go` – она возвращает сервер. При реализации было решено использовать `chi` сервер [7], он достаточно эффективный, при этом простой – позволяет настраивать фильтры для ресурсов и хэндлеров (обработчиков запросов). Также в нем можно настроить так называемые middlewares, то есть промежуточные обработки перед получением и обработкой запроса. В своей реализации я добавил подробное логирование, чтобы можно было легко следить за запросами, временами их выполнения и ошиками на сервере. Для этого добавлены `RequestID` middleware, чтобы в контексте запроса сохранять его уникальный идентификатор, и `LoggingMiddleware`, который логирует путь к ресурсу, метод, время выполнения запроса, код ответа, и размер тела ответа. Также легко можно добавить любые другие middleware, например, которые отвечают за авторизация и аутентификацию, или настроить CORS, или добавить профайлеры, или кэшировать запросы.

Стоит отметить, что сервер еще легко конфигурируется, для этого использовалась библиотека `confita` [3], она позволяет задать удобный конфиг со значениями по умолчанию, а затем загружать его из переменных окружения процесса, флагов запуска или файла с конфигом, в этом случае `config.json`.

Еще реализована функция `NewHandler`, которая создает хэндлер и инициализирует место для файловой системы сервера, создает папку для ресурсов и для репозиториев. Репозитории не хранятся вместе с ресурсами, чтобы клиент не мог их случайно испортить. Для этого используется механизм символических ссылок, то есть все ресурсы – это ссылки,

а сами файлы хранятся уже в репозиториях в месте, к котрому клиент не имеет доступа.

Релизованные методы поверх `WebDAV`:

- `VERSION-CONTROL`. Создает репозиторий, если его еще не было для объекта, и возвращает текущую версию ресурса на сервере. То есть этот метод можно выполнить всегда независимо от блокировок. Версия – просто строка, которая возвращается в заголовке *"Version"*
- `CHECKOUT`. Обязательно должен содержать заголовок *"Version"*, чтобы на неё переключиться. То есть содержимое ресурса переходит в состояние по данной версии.
- `CHECKIN`. Создает новую версию ресурса, не может быть выполнена после `CHECKOUT` без `UNCHECKOUT`. Возвращает так же версию в заголовке *"Version"*.
- `UNCHECKOUT`. Отменить `CHECKOUT` и вернуться к последней версии ресурса.

Последние 3 метода не могут быть выполнены, если объект заблокирован, потому что они предполагают необратимые изменения связанные с ресурсом.

6.2 Оптимизации

Кэшировать открытые репозитории. Чем больше размер кэша, тем больше оперативной памяти будет потреблять сервер, но при этом увеличивается скорость работы, так как все репозитории хранятся на диске, то не надо будет тратить время на их открытие, можно взять готовый указатель на нужную структуру из кэша. Размер кэша можно настроить в конфиге. Для кэширования использовался `ARCCache` [9], он учитывает и частоту использования ключа и последнее время его использования.

6.3 Тестирование

В папке `tests` репозитория [6] можно найти запускаемые бенчмарки.

```
goos: linux
goarch: amd64
pkg: jpepper_webdav/webdavvc/tests
cpu: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
BenchmarkGet-12                          3825         315318 ns/op         20842 B/op        159 allocs/op
BenchmarkVersionControlCacheMiss-12      1222         887749 ns/op         58086 B/op        519 allocs/op
BenchmarkVersionControlCacheHit-12       3012         393374 ns/op         24208 B/op        202 allocs/op
BenchmarkCheckin-12                       462        2628610 ns/op        361338 B/op       1559 allocs/op
BenchmarkCheckout-12                      739        1680523 ns/op        162929 B/op       1039 allocs/op
PASS
ok      jpepper_webdav/webdavvc/tests   10.193s
```

Рис. 6.1: Benchmarks

На результатах 6.1 видно, что кэш очень хорошо помогает, потому что запрос с методом `VERSION-CONTROL` работает почти столько же, сколько и `GET`. Как и ожидалось `CHECKIN` работает дольше всего, потому что ему приходится создавать новый коммит в репозитории. Тем не менее тестирование проводилось на небольших файлах, возможно на файлах побольше результаты были бы чуть приятнее.

7 Результаты

Получился довольно простое серверное приложение в плане использования, которое использует доработанный протокол `WebDAV`, что позволяет версионировать ресурсы. Конечно, есть еще большой простор для доработок:

- Доработать библиотеку[8] протокола `WebDAV` в `Go`
- Добавить возможность образовывать из версий древовидную структуру
- Добавить возможность просматривать версии каким-то методом, на подобии `REPORT` в `DeltaV`
- Добавить возможность получить diff для разных версиях для файлов с расширениями, которые это позволяют

Тем не мене сейчас получился работающий и достаточно эффективный сервер.

Список литературы

[1] *A highly extensible git implementation in pure Go.* URL: https://pkg.go.dev/github.com/go-git/go-git/v5.

[2] Geoffrey Clemm, Jim Amsden, Tim Ellison, Christopher Kaler и Jim Whitehead. *Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning).* URL: http://webdav.org/specs/rfc3253.html#rfc.section.1.1 (дата обр. 21.03.2002).

[3] *Confita is a library that loads configuration from multiple backends and stores it in a struct.* URL: https://pkg.go.dev/github.com/heetch/confita.

[4] L. Dusseault. *HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV).* URL: http://webdav.org/specs/rfc4918.html (дата обр. 12.05.2007).

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach и T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1.* URL: https://datatracker.ietf.org/doc/html/rfc2616 (дата обр. 05.1999).

[6] JPepperr. *WebDAV Version Control lib.* URL: https://github.com/JPepperr/webdav.

[7] *Package chi is a small, idiomatic and composable router for building HTTP services.* URL: https://pkg.go.dev/github.com/go-chi/chi.

[8] *Package webdav provides a WebDAV server implementation.* URL: https://pkg.go.dev/golang.org/x/net/webdav.

[9] *This provides the lru package which implements a fixed-size thread safe LRU cache.* URL: https://pkg.go.dev/github.com/hashicorp/golang-lru.