Электронные каталоги библиотек на мультитенантной программной платформе

О.С. Колобов

Институт сильноточной электроники СО РАН

А.А. Князева, И.Ю. Турчановский

Институт вычислительных технологий СО РАН

Аннотация. В работе рассматривается программная платформа для размещения множества электронных каталогов библиотек на основе модели «программное обеспечение как услуга». Дано описание архитектуры, основных сервисов и API. Предложенный подход имеет ряд полезных свойств для решения практических задач: параллельная обработка данных (индексирование, поиск), создание новых сервисов сети для работы с библиографическими записями и др. В качестве рабочего примера рассматривается свободное программное обеспечение – сервер IDZebra для работы с данными в форматах MARC и XML.

Введение

Известно, что системы автоматизации библиотек и автоматизированные библиотечные системы, созданные на основе формата MARC, имеют модульную организацию. С одной стороны, применение модульного подхода согласуется с хорошей практикой повторного использования программного кода, а с другой стороны такие модули имеют сильную связанность. На практике порой невозможно изменить или расширить функциональность отдельного модуля, так как большинство таких модулей имеют зависимость от проприетарного программного обеспечения, которое в свою очередь может быть частично открытым или полностью закрытым для сторонних разработчиков. Это обстоятельство определенно сдерживает развитие функции поиска в библиотеках, и поэтому библиотеки все чаще обращаются к внешним поисковым сервисам. В данной работе рассмотрен альтернативный подход к реализации функции поиска в библиотеке, который является более открытым, масштабируемым и независимым решением.

Развитие функции поиска для библиотек интересно тем, что информация, содержащаяся в записях MARC, является хорошо организованной, качественной и готовой к распространению в сети. И программные приложения, созданные на основе модели *software as service* (SaaS), потенциально также являются интересными для библиотек, так как это, прежде всего, новые возможности, новая форма для работы с читателями. Ключевой технологией в этом процессе, с нашей точки зрения, является поиск, который основан на богатом опыте библиотек в вопросах организации доступа к информации.

Создание программного приложения SaaS является процессом, который фактически не завершается. Для каждого из таких приложений всегда рассматривается ряд вопросов, которые имеют значение, как на этапе его создания, так и на этапе его эксплуатации. Один из таких вопросов – это вопрос о мультитенантности программного приложения. *Мультитенантность* (множественная аренда) – элемент архитектуры программного обеспечения, где единый экземпляр приложения, запущенного на сервере, обслуживает множество организаций-клиентов (арендаторов)[1]. Успешное решение вопроса о мультитенантности приложения, позволяет снизить прямые затраты на эксплуатацию приложения в среде облачных вычислений.

Приложение SaaS может иметь программный интерфейс, который обычно является отличной возможностью для сторонних разработчиков использовать приложение как внешний сервис.

В работе мы будем рассматривать сервис хостинга электронных каталогов библиотек. На основе этого сервиса библиотеки могут создавать множество открытых и закрытых электронных каталогов, выполнять загрузку библиографических записей и их индексирование. Каждый из созданных электронных каталогов является выделенным сервисом в сети, который имеет уникальный адрес в сети и доступен по стандартному протоколу поиска и извлечения информации.

Похожие работы

[1] По материалам статьи «Мультиарендность» из Википедии.

Система MasterKey Connect Service (MKS) [8] предоставляет простой API для поиска в большом количестве (более 10000) баз данных, онлайн журналов, электронных каталогов библиотек и др. ресурсов. Также известно, в соответствии с теоремой CAP, что от такой системы нельзя добиться одновременного соблюдения трех свойств consistency (согласованности), availability (доступности), portion tolerance (толерантность к сетевому разделению). MKS является системой, которая с одной стороны является доступной и толерантной к разбиению на сетевые разделы, а с другой допускает несогласованность данных. Решение проблемы несогласованности данных, для этой системы, решается путем определения разумного баланса между доступностью и согласованностью данных.

В качестве примера открытой программной платформы можно привести OKAPI [5], которая создается в рамках проекта FOLIO [4]. Платформа OKAPI создается на основе шаблона проектирования микросервисов [2] и представляет собой API Gateway, который включает в себя ядро базовых функций, а также механизм для расширения в виде отдельных, дополнительных модулей. Обычно, отдельный модуль – отдельный сервис, который создается на основе архитектурного стиля REST [7]. Таким образом OKAPI выполняет роль шлюза для работы с множеством различных сервисов. Это большая и объемная работа международного коллектива, первые результаты ожидаются в 2018 г. [6].

Изучение опыта создания высоконагруженных систем дает преставление о новыч технических решениях, которые можно применять на практике. Для нас значимым оказался подход к созданию сервиса хостинга репозиториев git[2], который описан в работе [3].

Рабочий пример

В качестве рабочего примера мы будем рассматривать сервер IDZebra [1], который является программным обеспечением с открытым исходным кодом и успешно применяется для решения практических задач в области поиска и извлечения информации в структурированных текстовых документах, в том числе и в библиографических записях в формате MARC. Сервер включает в себя набор программных библиотек, на основе которых можно создавать встроенные поисковые системы.

Описание подхода

Существует множество подходов к созданию приложения для хостинга электронных каталогов библиотек (далее просто приложение). Мы применяем подход, в котором каждой функции приложения соответствует отдельный сервис. Имея в задании минимальный набор функций, мы представляем их в виде сервисов, которые выполнены в едином архитектурном стиле REST. Такой подход позволяет нам использовать готовые шаблоны проектирования и делать меньше ошибок на начальном этапе. В работе мы рассматриваем минимальный набор функций приложения, который не является полным, но дает представление о функциональных возможностях приложения:

- Создать или удалить электронный каталог
- Обновить (вставить, заменить, удалить) данные электронного каталога
- Получить актуальную информацию об электронном каталоге
- Поиск и извлечение информации для электронного каталога

Из предложенного списка видно, что набор функций простой, однако задача предоставления этого набора функций для нескольких электронных каталогов одновременно нетривиальна. Для этого была создана модель и на ее основе рассмотрены пути реализации минимального набора функций приложения.

Модель. В результате нескольких итераций у нас получалась модель, согласно которой электронный каталог библиотеки помещается в *репозиторий* (кратко *репо*). Репо представляет собой автономную поисковую машину и включает в себя: данные, профиль индексирования и индексы (см. Рисунок 1).

[2] Популярная система контроля версий Git.

Рисунок 1. Структура репо

Репо может содержать множество электронных каталогов библиотеки (арендатора). Процесс индексирования данных для отдельного репо является автономным и не зависит от других репо. Это верно также и для процесса поиска. Без ущерба для нашего подхода все репо равномерно распределяются по группе серверов, и могут быть доступны по динамически создаваемым каналам связи. Для каждого из репо выполняются условия:

- Владельцем репо является тот, кто его создал – библиотека
- Библиотека может создать один и более репо
- Библиотека может создать один и более электронных каталогов в одном репо
- Репо может быть открытым или закрытым (статус репо определяет библиотека)
- Функция поиска может работать для любой комбинации электронных каталогов в одном репо

Такая модель позволяет создать различные конфигурации электронного каталога библиотеки на основе одного или нескольких репо. Однажды созданная конфигурация может динамически меняться в процессе работы. Здесь необходимо отметить, что для репо применяется один профиль индексирования данных, которой не может быть изменен, пока в репо есть хотя бы один электронный каталог.

Сервисы и ресурсы. Наше приложение можно представить, как набор различных сервисов. На основе сервисов выполняется работа по управлению ресурсами. Ресурсами в нашем случае являются:

- Repository (репозиторий)
- Database (база данных)
- Storage (хранилище)

Ресурс repository соответствует тому, что мы определили ранее как репо. Для управления ресурсом используется одноименный сервис repository, который дает нам возможность создавать, редактировать и удалять репо.

Ресурс database соответствует тому, что мы определили ранее как электронный каталог библиотеки. Для управления ресурсом используется одноименный сервис database. Помимо основных действий с ресурсом, этот сервис предоставляет возможность создавать, обновлять или удалять записи электронного каталога. Также на основе этого сервиса можно получить актуальную информацию о ресурсе (количество индексированных записей, размер базы данных, фактически поддерживаемые точки доступа).

Ресурс storage представляет собой хранилище объектов. Под объектом мы понимаем отдельный файл с данными. Для управления ресурсом используется одноименный сервис storage, который дает нам возможность загружать, хранить, извлекать и удалять объекты. Все объекты представляются как бинарные данные и снабжаются уникальным идентификатором. Структура хранилища объектов плоская и не подразумевает наличие связей между объектами.

Согласно модели, ресурс repository включает в себя ресурс database, который в свою очередь включает в себя ресурс storage. При работе с ресурсами необходимо учитывать иерархию ресурсов. Например, если удалить ресурс database, тогда будут удалены все ресурсы storage, которые связанны

с ресурсом database, а если удалить ресурс repository, то будут удалены все связанные с repository ресурсы database и storage.

Входные данные. В качестве входных данных приложения мы рассматриваем отдельную запись или коллекцию записей электронного каталога. Эти данные могут быть добавлены в ресурс database на основе двух функций update-record и update-file. Доступ к каждой из этих функций выполняется через сервис database. Для добавления в электронный каталог отдельной записи (update-record) применяется сервис database, который ожидает, что запись содержится в теле запроса. Для добавления коллекции записей в электронный каталог (update-file) применяются сервисы database и storage. Так как коллекция записей содержится в файле, а размер файла может быть большим, то разумно выполнять добавление коллекции записей в электронный каталог в два этапа. На первом этапе файл с данными загружается и размещается на основе сервиса storage. Далее загруженный файл может быть индексирован как объект ресурса storage на основе сервиса database.

Программный интерфейс (API). Для работы с ресурсами на основе сервисов применяются RESTful и REST ориентированные подходы. Подход на основе RESTful используется для управления ресурсами репо (см. описание модели), а подход на основе REST для поиска и извлечения информации. В Таблице 1 приведен список API, который соответствует минимальному набору функций приложения с указанием применяемых методов протокола HTTP.

Таблица 1. Программный интерфейс для репо

API	**Адрес (endpoint)**	**Стиль**
Repository	/repository/:*resource_id*	RESTful
Database	/repository/:*resource_id*/database/:*database_id*	RESTful
Storage	/repository/:*resource_id*/database/:*database_id*/storage/:*storage_id*	RESTful
Search	/search/:*reponame*/:*databasename*?*query_str*	REST

Repository API предназначен для управления репо. Этот API доступен только для авторизованных пользователей, которыми являются владельцы репо.

Search API предназначен для поиска и извлечения информации. Этот API доступен для анонимных пользователей. Необходимо отметить, что в качестве параметров URL используются имена ресурсов, а не их идентификаторы в базе данных. Это связано с тем, что клиент ничего не знает о внутренней организации приложения и ориентируется только на названия (имена) ресурсов, в нашем случае это имена репо и базы данных.

Платформа

Наша задача создать приложение с открытым программным интерфейсом для клиентских приложений. Такое приложение представляется нам в виде программной платформы, где библиотеки могут размещать электронные каталоги, открывать доступ к ним, применять различные правила для индексирования записей, предоставлять свои данные для создания и работы других сервисов сети.

Если у нас есть набор различных сервисов, на основе которых реализован минимальный набор функций приложения, а также есть API для работы с каждым из этих сервисов, то мы можем говорить о совокупности этих элементов как о *платформе*. Вариантов для создания такой платформы сегодня множество. Мы используем подход на основе архитектурного стиля REST, где каждому ресурсу соответствует уникальный URI. Таким образом мы можем рассматривать набор ресурсов, а также операции над ними, в форме запросов протокола HTTP.

Наша платформа является распределенной системой, детали ее внутренней организации скрыты от клиента. Вопрос об архитектуре такой платформы одновременно затрагивает вопросы о масштабировании, о нагрузке, о безопасности, об эксплуатации, а также и ряд др. вопросов. Для того чтобы четко понимать и успешно развивать платформу мы применяем известные шаблоны (паттерны проектирования):

- Microservices (микросервисы)
- API Gateway (шлюз)
- Database per service (база данных как сервис)
- Access token (маркер доступа)

Платформа на основе шаблона microservices включает в себя множество сервисов. Эти сервисы, в свою очередь, также могут включать в себя и другие сервисы. Для того чтобы не усложнять архитектуру платформы, применяется шаблон API Gateway, который инкапсулирует внутреннюю структуру сервисов платформы. Это позволяет достичь как минимум две цели. Первая – это упростить архитектуру платформы. Вторая – предоставить механизмы для определения событий отказа в обслуживании или ошибок в работе сервисов.

Применение шаблона database per service (база данных как сервис) позволяет представить электронный каталог библиотеки как отдельный сервис.

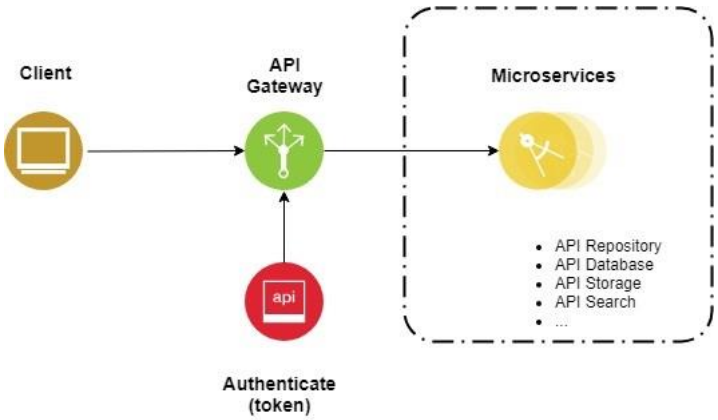

Рисунок 2. Архитектура

Применение шаблона access token (маркер доступа) предоставляет механизм для безопасного доступа к функциям платформы. Так как платформа — это набор различных сервисов, которые работают в распределенной среде, то крайне важно иметь возможность выполнять проверку прав доступа к сервису для каждого запроса. Например, если клиент успешно прошёл аутентификацию, и желает получить доступ к управлению репо, то в этом случае все запросы от клиента к сервису репо должны содержать маркер доступа. Маркер доступа создается на стороне платформы, и применяется везде для идентификации клиента. Другими словами, сервис всегда знает, кто создал запрос и как его обработать. В том случае, если маркер доступа не предоставлен, или время жизни маркера закончилось, то сервис ответит отказом в обслуживании.

Реализация

Программная платформа реализована как распределённая система и состоит из нескольких компонент. Эти компоненты могут функционировать на различных виртуальных серверах. Для понимания нашей архитектуры, и той роли, которую играет каждая из компонент, необходимо рассмотреть вопрос о том, как обрабатываются запросы к репо.

API-сервер — это сервер, который принимает HTTP-запросы клиентов. Браузеры, консольные клиенты, скрипты, роботы, и др. являются клиентами платформы, и могут делать запросы к платформе независимо и интенсивно. Поэтому для API-сервера применяется *балансировка нагрузки*, что позволяет равномерно распределять запросы клиентов между несколькими

экземплярами API-сервер. При таком подходе для нас неважно, какой экземпляр API-сервера принимает текущий запрос от клиента, так как все запросы к репо обрабатываются асинхронно на основе одного или нескольких серверов очередей.

Асинхронная обработка запроса. Так как количество поступающих запросов в единицу времени может быть велико, а некоторые запросы, по природе своей, не могут быть обработаны быстро (например, запрос на индексирование коллекции записей), то необходим подход, который обеспечит работоспособность платформы в целом при пиковых нагрузках. Для всех операций с репо мы применяем подход *асинхронной обработки запроса* на основе одного или нескольких серверов очередей. В соответствии с этим походом, запрос к репо представляется как абстрактный объект, который помещается в очередь и обрабатывается асинхронно, в тот момент, когда есть свободные вычислительные ресурсы. Процесс, который выполняет обработку запроса к репо, называется worker (работник). Процесс worker непосредственно взаимодействует с репо, и к такому процессу нет прямого доступа из вне, так как все данные для выполнения операций с репо этот процесс получает из очереди. Количество процессов worker может динамически меняться, в зависимости от интенсивности запросов к репо. Здесь необходимо отметить, что для каждого рода операций с репо создается отдельная очередь и для обработки отдельной очереди может быть назначена группа процессов worker.

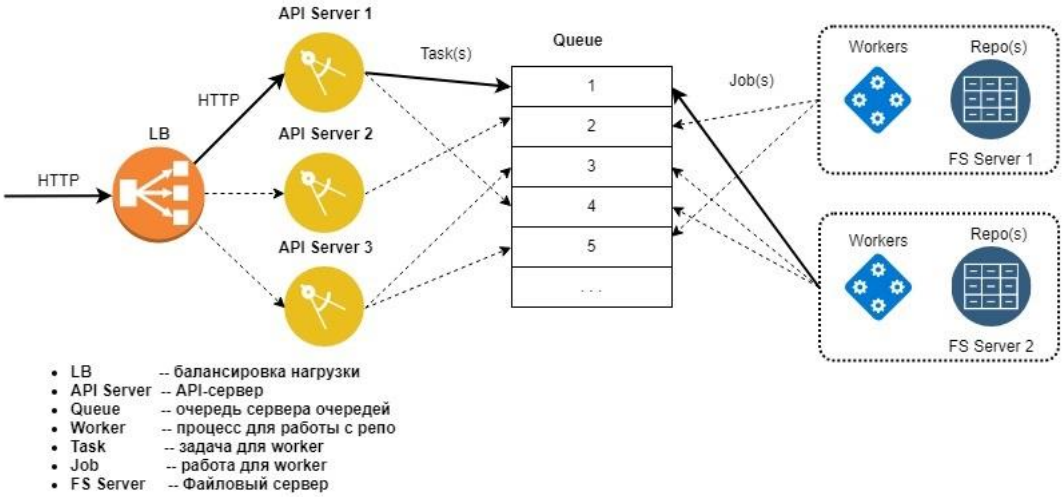

Рисунок 3. Асинхронная обработка запроса.

Асинхронная обработка запросов позволяет динамически управлять распределением нагрузки на систему в нужном месте и в нужное время. Этот подход работает тогда, когда для «тяжелых задач» задействуются мощные сервера, для остальных задач – обычные сервера. Таким образом мы можем разумно распределять вычислительные ресурсы между задачами.

Хранение данных. Для работы с файлами данных применяется хранилище с сетевым доступом по протоколам FTP, HTTP и SSH. Все файлы в хранилище являются цифровыми объектами, и у каждого такого объекта есть уникальный идентификатор и метаданные. Контент и метаданные объекта хранятся раздельно. Прямой доступ к объекту хранения выполняется на основе его идентификатора, и доступен только для авторизованных пользователей.

Репо это автономная поисковая машина, которая не имеет жесткой привязки к отдельному серверу и может мигрировать с одного сервера на другой. Как мы ранее уже указывали, репо содержит в себе все, что нужно для индексирования и поиска данных. В репо можно добавить отдельную запись или добавить коллекцию записей как объект хранилища. Репо работает с хранилищем на основе сетевых протоколов и от имени владельца репо. В основе репо лежит высокопроизводительный сервер IDZebra. Репо поддерживает широкий набор форматов данных: ИРБИС64, RUSMARC, UNIMARC, MARC21, MARCXML, для которых определены профили индексирования записей. Также в репо встроен механизм для определения дополнительных форматов данных на основе XML-ориентированных технологий.

Поиск в репо основан на стандартном протоколе поиска и извлечения информации SRU 1.2 и языке запросов CQL. Репо может выполнять операции протокола SRU как для отдельного электронного каталога в репо, так и для всего репо в целом. Последнее удобно для выполнения одновременного поиска в нескольких электронных каталогах.

Заключение

В работе мы описали наш подход к созданию сервиса хостинга электронных каталогов библиотек в форме приложения для среды облачных вычислений. Это приложение представляет собой мультитенантную программную платформу, которая предназначена для создания новых сервисов сети и новых клиентских приложений для библиотек (интегрированные поисковые системы, сводные и распределенные электронные каталоги и др.). Основные возможности платформы: раздельное индексирование библиографических записей, раздельное выполнение поиска, масштабирование, балансировка нагрузки, асинхронная обработка запросов, а также простая миграция данных и индексов между виртуальными блочными устройствами.

Литература

1. Zebra – Index Data // www.indexdata.com. – URL:https://www.indexdata.com/resources/software/zebra/ (16.05.2018).
2. Richardson, Chris. Microservice architecture pattern // microservices.io. – URL:http://microservices.io/patterns/microservices.html (19.12.2017).
3. Preston-Werner, Tom. How We Made GitHub Fast // github.com. – URL: https://github.com/blog/530-how-we-made-github-fast (19.12.2017).
4. The Open Library Foundation. FOLIO Developer // dev.folio.org. – URL: http://dev.folio.org/ (20.12.2017).
5. The Open Library Foundation. Okapi — a multitenant API Gateway // github.com. – URL: https://github.com/folio-org/okapi (20.12.2017).
6. The Open Library Foundation. App & Timelines // www.folio.org. URL: https://www.folio.org/apps-timelines/ (20.12.2017).
7. Fielding R. Architectural Styles and the Design of Network-based Software Architectures [Electronic resource] : diss. ... for the degree of Ph.D. / Roy Thomas Fielding; Univ. of California. -- Irvine, 2000.
8. MasterKey Connect // www.indexdata.com. – URL:https://www.indexdata.com/connectivity-products/masterkey-connect/ (16.05.2018)