ИСКУССТВО. ИСКУССТВОВЕДЕНИЕ

Образец ссылки на эту статью: Абрамов В.С. Module Federation: исследование способов переиспользования кода с помощью микросервисного дизайна // Бизнес и дизайн ревю. 2024. № 1 (33). С. 79-87.

УДК 004.45

MODULE FEDERATION: ИССЛЕДОВАНИЕ СПОСОБОВ ПЕРЕИСПОЛЬЗОВАНИЯ КОДА С ПОМОЩЬЮ МИКРОСЕРВИСНОГО ДИЗАЙНА

Абрамов Вадим Сергеевич

ИТМО, Санкт-Петербург, Россия (197101, Санкт-Петербург, Кронверкский проспект, д.49, литер А.), аспирант, vadimabramov322@gmail.com

Аннотация. Микросервисный дизайн долгое время применялся лишь в области серверной разработки. Однако с развитием технологий и ростом клиентских приложений, актуальность использования микросервисов на клиенте росла каждый год. Одной из актуальных технологий для реализации микросервисов на клиенте является Module Federation. Так как она является частью сборщика Webpack, то возможность создания модульного приложения существует с начала разработки. Доказывается, что загрузка модулей происходит лишь при необходимости, что позволяет сохранить изначальный уровень скорости загрузки приложения.

Ключевые слова: веб-технологии; JavaScript; Module Federation; микросервис; frontend.

MODULE FEDERATION: EXPLORING WAYS OF CODE REUSE USING MICROSERVICE DESIGN

Abramov Vadim Sergeevich

ITMO, St. Petersburg, Russia (197101, St. Petersburg, Kronverksky Prospekt, 49, liter A.), postgraduate student, vadimabramov322@gmail.com

Abstract. Microservice design has long been applied only in the field of server-side development. However, with the development of technology and the growth of client applications, the relevance of using microservices on the client has grown every year. One of the actual technologies for implementing microservices on the client is Module Federation. Since it is part of the Webpack builder, the possibility of creating a modular application exists from the beginning of development. And modules are loaded only, when necessary, which allows to keep the original level of application loading speed.

Keywords: web technologies; JavaScript; Module Federation; microservice; frontend.

Введение

При выборе дизайна системы для веб-приложения долгое время применялся монолитный подход. Он представляет собой разработку приложения как единого модуля, внутри которого находится функционал, отвечающий за работу всей программы. Это полностью автономная программа, независящее от другого ПО [1, с. 82-87]. Все части приложения имеют высокую связанность, в связи с чем растет шанс поломки всего приложения, если что-то будет работать некорректно. Монолитный подход имеет ряд плюсов[1]:

- Удобная разработка единой и малой командой;
- Упрощённая разработка и развертывание приложения;
- Отсутствие сквозных ошибок.

Однако также существует ряд недостатков, из-за которых выбирают иной подход разработки [2, с. 17-19]:

- Большой объем кода, который со временем затрудняет разработку и увеличивает время развертывание приложения;
- Очень трудно внедрять новые технологии, так как для этого потребуется переписать большую часть кодовой базы;
- Из-за высокой связанности компонентов ошибка в одном из них может остановить работу всей программы.

Вскоре на замену пришли микросервисы, которые стали широко использоваться в области бэкенд-разработки. Статистика поисковых запросов представлена на рисунке 1.

Рисунок 1 – Популярность запроса «microservices» в Интернете

Микросервис – это слабосвязанная часть приложения, разрабатываемая для решения определенной задачи[2]. Очень часто такой сервис может работать как самостоятельное приложение. Каждый микросервис может использовать

[1] Microservices vs. monolithic architecture // Atlassian - Режим доступа: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith, свободный (дата обращения 11.01.2024)

[2] Микросервисная архитектура, ее паттерны проектирования и особенности // Habr - Режим доступа: https://habr.com/ru/company/serverspace/blog/692916/, свободный (дата обращения 11.01.2024)

уникальный набор технологий, имеет небольшую кодовую базу, а их развертывание происходит независимо. Все это позволяет вести их разработку отдельно от всего приложения [3, с. 15-28].

Появляется возможность удобной работы большого количества команд, каждая из которых будет работать над своим сервисом[3]. Низкая связанность позволяет повысить надежность приложения и возможность его модернизации. Например, если сервис, отвечающий за оплату, выйдет из строя, всё остальное приложение продолжит работать, а замена одного сервиса на другой не будет проблемой, благодаря тому что микросервисы общаются между собой с помощью API, а значит необходимо лишь адаптировать новый сервис под текущий интерфейс.

Однако применение микросервисного дизайна систем в области фронтенд разработки было долгое время невостребованным и лишь последние несколько лет набирает популярность. Статистка частоты поиска о микросервисах во фронтенде представлена на рисунке 2.

Рисунок 2 – Популярность запроса «micro frontend» в Интернете

Уже существует несколько вариантов реализации микрофронтендов: использование технологии iFrame, веб-компоненты, технология Module Federation, предоставляемая сборщиком Webpack, начиная с 5 версии [4, с. 13-14].

Цель исследования: анализ технологии Module Federation и сравнение ее с аналогами.

Методы исследования: анализ литературных и электронных источников; исследование статистических данных; сравнение технологий и эксперимент.

[3] Module Federation // Webpack documentation - Режим доступа: https://webpack.js.org/concepts/module-federation/, свободный (дата обращения 11.01.2024)

Результаты исследования и их обсуждение

Module Federation — это концепция, основанная на модульности программного обеспечения[4]. Она предполагает разделение программы на более мелкие, независимые модули, которые могут быть соединены вместе для создания мощных, масштабируемых приложений. Объединение модулей развивает идею микросервисов, позволяя создавать модульные приложения, которыми можно независимо развертывать и управлять.

По своей сути, это способ структурирования приложения таким образом, чтобы его можно было разбить на более мелкие части, которые легче поддерживать и изменять [5, с. 164]. Это облегчает обновление и расширение приложения с течением времени без необходимости переписывать всю кодовую базу. Используя возможности модульности, разработчики могут создавать высоко масштабируемые веб-приложения, которые могут расти и развиваться в соответствии с потребностями бизнеса.

Главной причиной выбрать данную технологию является то, что Module Federation является частью сборщика Webpack, который уже давно стал стандартом при разработке веб-приложений[5]. Количество еженедельных загрузок превышает 27 миллионов. Это означает что если команда разработки решит перейти на микросервисный подход, то с высокой вероятностью в ядре проекта уже будет использоваться Webpack, который позволит начать разрабатывать новый сервис без лишних надстроек и использования сторонних библиотек.

Module Federation обеспечивает несколько ключевых преимуществ для разработчиков. Наиболее очевидное преимущество заключается в том, что она облегчает создание модульных приложений. Используя возможности Webpack, разработчики могут легко разбить приложение на более мелкие, независимые модули, которые могут быть соединены вместе [6, с. 69-71]. Это облегчает повторное использование кода из одного модуля в другом и быструю и простую интеграцию сторонних сервисов в приложение.

Также упростится тестирование и отладка приложения. Разбивая приложение на более мелкие, независимые модули, разработчики могут легче тестировать и отлаживать каждый компонент в отдельности. Это облегчает быстрое и эффективное выявление и устранение проблем.

Несмотря на многочисленные преимущества технологии, существует несколько проблем. Прежде всего, объединение модулей требует глубокого понимания Webpack и Module Federation [7, с. 526-528]. Без этих знаний разработчикам может быть трудно создавать действительно модульные и масштабируемые приложения.

---

[4] Практические рекомендации по разработке масштабных React-приложений. Планирование, действия, источники данных и API // Habr - Режим доступа: https://habr.com/ru/company/ruvds/blog/458496/, свободный (дата обращения 09.01.2024)

[5] Webpack Module Federation: «официальное» решение в микрофронтендах // Habr - Режим доступа: https://habr.com/ru/company/alfa/blog/668118/, свободный

Кроме того, требуется предварительное планирование и подготовка. Это связано с тем, что разработчикам необходимо тщательно спланировать и структурировать приложение, чтобы обеспечить его модульность и масштабируемость. Без такого планирования разработчики могут обнаружить, что их приложение трудно поддерживать и обновлять с течением времени[6].

Также требуется тщательно протестировать и отладить приложение. Это связано с тем, что необходимо тщательно тестировать и отлаживать каждый модуль в отдельности, чтобы убедиться, что приложение действительно масштабируемо и стабильно. Без такого тестирования разработчики могут обнаружить, что их приложение подвержено ошибкам.

Перед описанием принципа работы технологии необходимо рассмотреть следующие термины:

- Host — это бандл, который первый инициализировался во время загрузки страницы. Это корневое приложение, которое подтягивает другие части [8, с. 11-12];
- Remote — другой бандл, чьи части может импортировать host. Он запрашивает у remote необходимые компоненты;
- Omnidirectional host — это бандл, который одновременно может быть и host, и remote. Он может быть полноценным приложением или же может только раздавать общие компоненты;
- Exposed modules — модули, которые будут доступны другим приложением для импорта, например, компоненты, картинки или стили;
- Shared modules. — модули, которые могут быть общими для всего приложения, например библиотеки компонентов.

Сама технология работает следующим образом:

1. Webpack собирает бандлы для каждого из приложений (host или remote). Бандлы могут быть развернуты на разных доменах и полностью автономны;
2. Shared modules, которые используется между всеми бандлами, помещаются в отдельный файл и попадают в область общей видимости для всех приложений;
3. Когда host запросит общие модули, они попадут в область видимости host приложения. Webpack не будет знать, что эти компоненты были откуда-то загружены, и будет использовать их так, будто они всегда были в основной сборке[7].
4. Когда же host приложению потребуется отобразить части remote, эти модули загружаются обычно, с помощью динамического импорта в рантайме.

Схема работы представлена на рисунке 3.

[6] Быстрый старт с WebComponents // Medium - Режим доступа: https://medium.com/webbdev/web-1370a1426072/, свободный

[7] What are microservices // Microservice Architecture – Режим доступа: https://microservices.io/, свободный

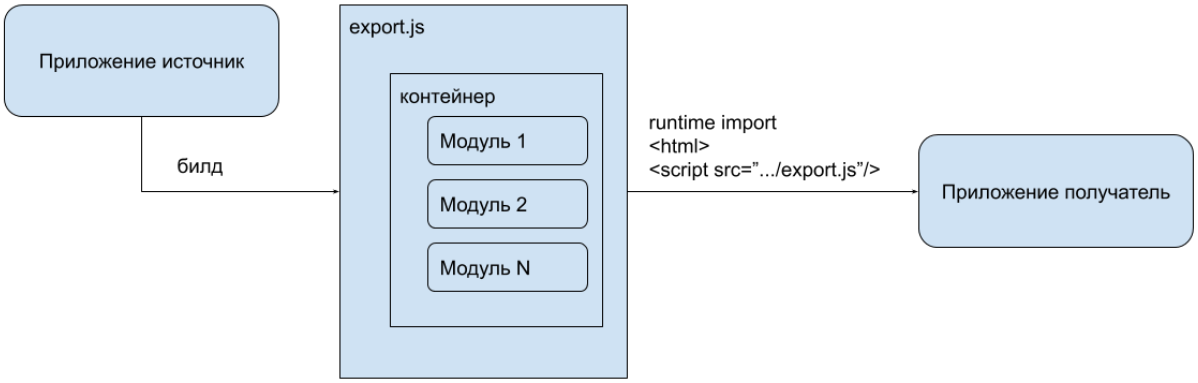

Рисунок 3 – Схема работы Module Federation

Одним из аналогов Module Federation является технология iFrame. Это HTML тег, который был создан в 1997 г. [9, с. 2214-2222]. Все основные браузеры продолжают его поддерживать, однако из-за ряда уязвимостей его следует использовать, если разработчик имеет глубокое понимание, как он работает.

Основная цель использования данного тега – встраивание одного HTML документа внутрь другого[8]. То есть на странице пользователя будут два изолированных приложения со своим состоянием, стилями и кодом. Однако приложения могут общаться между собой с помощью метода postMessage. Использование iFrame чаще всего предполагает подключение стороннего веб-приложения, к которому нет доступа у разработчика основного приложения. Из этого следует потенциальная проблема с безопасностью. Также страдает SEO оптимизацию, так как поисковые системы считают, что содержимое iFrame принадлежит другому сайту. Так как каждый фрейм по сути является отдельной страницей, то каждое подобное подключение будет увеличивать потребляемую оперативную память и вычислительные ресурсы, что сильно уменьшает количество используемый фреймов на странице.

Еще одним аналогом является использование встроенного функционала браузера – Web Components. Веб-компоненты — это набор стандартов, определяющих программные интерфейсы для организации компонентного дизайна[9]. Все они реализованы в современных версиях браузеров, т. е. не требуют подключения библиотек или транспиляторов кода. Для этого технология предоставляет следующие возможности [10, с. 11-17]:

- Custom elements – возможность регистрировать свои HTML-тэги с определенным поведением;
- Shadow DOM – создание изолированного контекста CSS;
- Slots – возможность комбинировать внешний HTML-контент с внутренним HTML компонентом[10].

[8] What are Micro Frontends // Micro Frontends – Режим доступа: https://micro-frontends.org/, свободный
[9] The Inline Frame element // Mdn Web Docs – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe, свободный
[10] Web components // JavaScript info – Режим доступа: https://javascript.info/web-components, свободный

Несмотря на подробную документацию, жизненный цикл веб-компонентов является трудным для понимания, а сама загрузка компонентов требует ручной настройки. Сравнение технологий представлено в таблице 1.

Таблица 1 – Сравнение iFrame, Web Components и Module Federation

	iFrame	Web Components	Module Federation
Изолированность сервисов	Полная	Зависит от реализации	Зависит от реализации
Общение между сервисами	postMessage	Кастомные события	Любые технологии
Безопасность технологии	Низкая	Высокая	Высокая
SEO оптимизация	Не работает	Поддерживается	Поддерживается
Производительность	Низкая	Высокая	Зависит от Webpack конфига
Поддержка SSR	Нет	Есть	Есть

Для тестирования технологии Module Federation было разработано небольшое SPA приложения, состоящее из нескольких страниц. Также было создано аналогичное приложение, части которого были вынесены в отдельный микросервис. Для замера скорости загрузки и размера загружаемого кода используется встроенный инструмент разработчика в браузере Google Chrome.

Процесс тестирования разбит на следующие шаги:

1. Замер скорости загрузки и размера загруженного кода полноценного приложения;
2. Замер скорости загрузки и размера загруженного кода приложения с микросервисом;
3. Сравнение результатов.

Итоги замеров производительности представлены в таблице 2.

Таблица 2 – Результаты тестирования

	Module Federation	Проект без Module Federation
Скорость сборки проекта	1310 мс	1530 мс
Скорость загрузки бандла	147.3 мс	161.47 мс
Размер итогового бандла	1.64 мб	1.7 мб

По итогам тестирования Module Federation показал себе немного лучше, чем проект без Module Federation. Однако отличия не настолько большие чтобы подключать технологию в каждом проекте. Разница будет заметней, если сравнивать крупные проекты. Благодаря тому что приложение разбивается на несколько независимых модулей, общее время сборки значительно ускорится из-за параллельной работы. А так как модули загружаются по мере необходимости, время загрузки и размер итогового бандла будут гораздо меньше, потому что пользователи будут загружать лишь ту часть кода, которая необходима в текущий момент.

Выводы исследования

Module Federation мощный инструмент, который позволяет создавать веб-приложение из независимых веб-компонентов или же полноценных других приложений. Благодаря встроенной поддержке импорта модулей, подключение компонентов происходит во время исполнения. А сами модули загружаются лишь в тот момент, когда они необходимы, из-за чего добавление новых модулей никак не повлияет на скорость загрузки и размер итогов бандла основного приложения.

Использование технологии накладывает некоторые обязательства на команду разработки. Необходимо заранее продумать и спланировать дизайн системы для приложения. Так как после могут возникнуть проблемы с масштабируемостью и тестирование приложения. Также необходимое глубокое понимание сборщика Webpack, так как Module Federation является его частью, а значит важно правильно сформировать конфиг сборщика, через который и будет происходить реализация модульности приложения.

Список литературы

1. Луценко Д.Ю., Полякова Л.П. Разбиение монолитного приложения на микросервисы с использованием паттерна strangler // Информационные технологии в управлении и экономике. 2021. № 3(4). С. 82-87.
2. Штефуряк А.С., Яницкая Т.С. Востребованные технологии для разработки микросервисов // Молодой ученый. 2020. № 43(333). С. 17-19.
3. Сабиров Д.А. Микросервисная архитектура на frontend // Научный журнал. 2021. № 7(62). С. 15-28.
4. Богомолов Д.Ю. Сравнительный анализ способов создания встраиваемых веб-приложений // Центральный научный вестник. 2023. № 11(76). С. 13-14.
5. Попков И.В., Курзаева Л.В. Использование webpack для сборки проекта // Международный студенческий научный вестник. 2018. № 5. С. 164.
6. Сафин А.М., Кадыров К.А. Сборщик модулей webpack // Международные научные исследования: актуальные вопросы, достижения и инновации. Пенза: Наука и просвещение, 2022. С. 69-71.
7. Аргасова В.В. Применение web-фреймворка react при разработке информационных систем // Аллея науки. 2018. № 7(23). С. 526-528.
8. Сабиров Д.А. Обзор инструмента сборки webpack // Информационные технологии как основа эффективного инновационного развития. Уфа: Аэтерна, 2021. С. 11-12.
9. Гамидов Ш.С. Использование iframe в веб-разработке // Научный аспект. 2023. Т. 17. № 6. С. 2214-2222.
10. Антошкин В.А., Мудрова А.А. Использование технологии «web components» для разработки одностраничных веб-приложений // Информатика и прикладная математика, 2023. № 29. С. 11-17.

References

1. Lutsenko D.Iu., Poliakova L.P. Razbienie monolitnogo prilozheniia na mikroservisy s ispolzovaniem patterna stranglerm, *Informatsionnye tekhnologii v upravlenii i ekonomike*, 2021, no 3(4), pp. 82-87.

2. Shtefuriak A.S., Ianitskaia T.S. Vostrebovannye tekhnologii dlia razrabotki mikroservisov, *Molodoi uchenyi*, 2020, no 43(333), pp. 17-19.

3. Sabirov D.A. Mikroservisnaia arkhitektura na frontend, *Nauchnyi zhurnal*, 2021, no 7(62), pp. 15-28.

4. Bogomolov D.Iu. Sravnitelnyi analiz sposobov sozdaniia vstraivaemykh veb-prilozhenii, *Tsentralnyi nauchnyi vestnik*, 2023, no 11(76), pp. 13-14.

5. Popkov I.V., Kurzaeva L.V. Ispolzovanie webpack dlia sborki proekta, *Mezhdunarodnyi studencheskii nauchnyi vestnik*, 2018, no 5, p. 164.

6. Safin A.M., Kadyrov K.A. Sborshchik modulei webpack, Mezhdunarodnye nauchnye issledovaniia: aktualnye voprosy, dostizheniia i innovatsii. Penza: Nauka i prosveshchenie, 2022, pp. 69-71.

7. Argasova V.V. Primenenie web-freimvorka react pri razrabotke informatsionnykh system, *Alleia nauki*, 2018, no 7(23), pp. 526-528.

8. Sabirov D.A. Obzor instrumenta sborki webpack, Informatsionnye tekhnologii kak osnova effektivnogo innovatsionnogo razvitiia. Ufa: Aeterna, 2021. pp. 11-12.

9. Gamidov Sh.S. Ispolzovanie iframe v veb-razrabotke, *Nauchnyi aspect*, 2023, Vol. 17, no 6, pp. 2214-2222.

10. Antoshkin V.A., Mudrova A.A. Ispolzovanie tekhnologii «web components» dlia razrabotki odnostranichnykh veb-prilozhenii, *Informatika i prikladnaia matematika*, 2023, no 29, pp. 11-17.

Статья поступила в редакцию 15.01.2024