Тема: Конфигуратор мультимодульных программных систем

**Содержание**

Введение 3

1. Анализ мультимодульной архитектуры проектирования ПО 4

1.1. Сравнительный анализ монолитной и мултимодульной архитектуры проектирования 4

1.2. Описание и состав классического модуля системы 6

1.3. Описание методов межмодульного взаимодействия 7

1.4. Конфигурация модуля 8

2. Адаптация мультимодульной программной системы 9

3. Анализ процессов управления конфигурациями 11

Введение

Начиная с 80-х годов прошлого столетия началось активное использование языков программирования для разработки промышленных приложений. Очень долгое время при их создании использовали монолитную архитектуру проектирования, согласно которой вся система являлась большим набором взаимосвязанных элементов. Но в современном мире, когда такие системы должны выполнять все больше и больше различных функций, такой подход сильно усложняет их, в следствие чего появляется все больше багов и время на разработку новых требований так же увеличивается.

С недавнего времени для решения данной проблемы многие предприятия стали использовать мультимодульный принцип проектирования программных систем. В его основу заложено то, что каждая логически обособленная часть функционала выносится в отдельный независимый модуль. В итоге вся система строится на множестве независимых модулей, что делает ее легко расширяемой и легко управляемой.

В рамках выпускной квалификационной работы будет рассмотрен принцип мультимодульного проектирование программных систем, а затем, на основе применения конфигурационного моделирования, будет продемонстрирован способ управления настройками (конфигурациями) каждого модуля.

Итогом данной работы будет являться приложение «Конфигуратор» с помощью которого будет происходить управление модулями системы.

1. Анализ мультимодульной архитектуры проектирования ПО

1.1. Сравнительный анализ монолитной и мултимодульной архитектуры проектирования

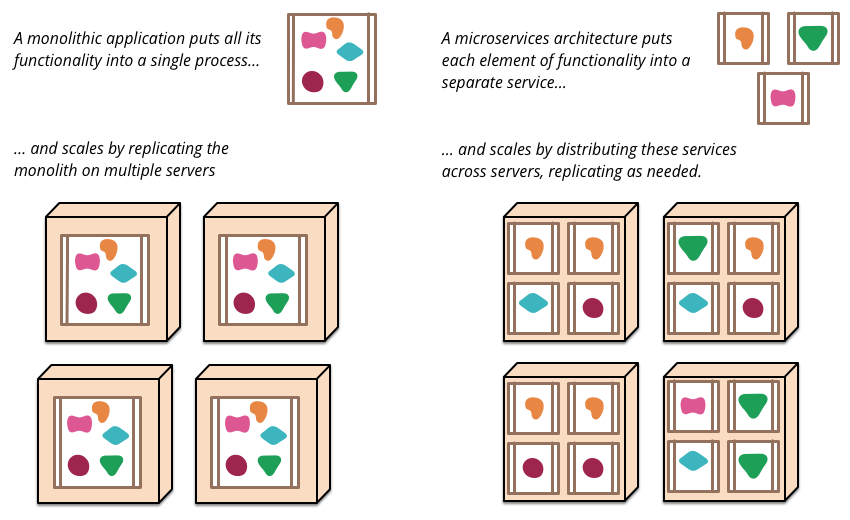
«Мультимодульная» архитектура проектирования, которую так же называют «микросервисной», является относительно новым термином в области разработки программного обеспечения. К сожалению, довольно часто люди, использующие его, не имеют конечного понимания того, что же данная архитектура из себя представляет.

Таким образом, в первую очередь определим, какие особенности имеет мультимодульная архитектура и сравним ее с классическим «монолитом», т.е. системы, работающей как единое целое.

При использовании монолитной архитектуры разработчики сталкиваются со следующими проблемами:

* необходимо ориентироваться в большом объеме кода;
* при добавлении нового функционала, его декомпозиция сильно ограничена, что приводит к большим временным, и, следовательно, финансовым затратам со стороны заказчика;
* при выходе из стоя какого либо компонента большой системы, вся система выходит из строя, что является неприемлемым для многих предприятий;
* сильная ограниченность в используемых технологиях;
* масштабирование может быть осуществлено только посредством копирования системы на отдельный сервер. Данная проблема так же обусловлена тем, что если необходимо увеличить производительность только одного элемента системы, т.к. остальные вычислительные мощности расходуются впустую.

На рисунке 1.1 представлена иллюстрация системы, имеющую монолитную архитектуру.

Рисунок 1.1 - Иллюстрация «монолитной» системы

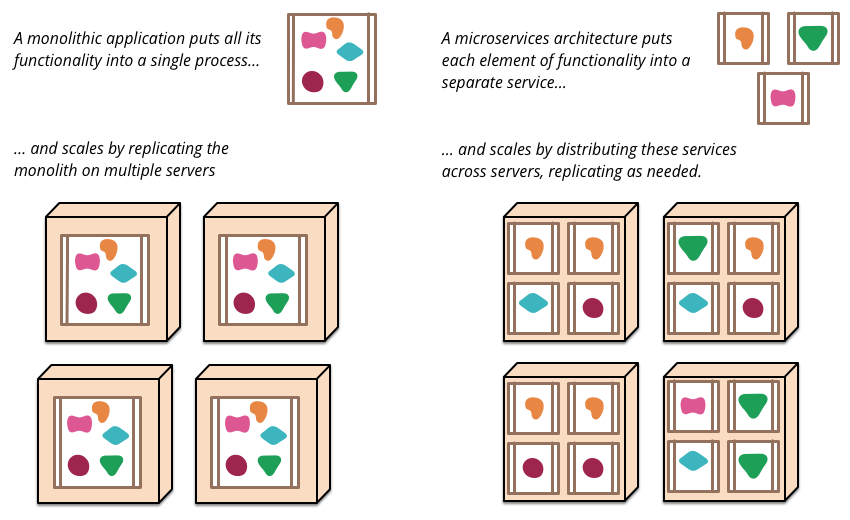
Проектирование системы на основе независимых модулей по сути является решением вышеперечисленных проблем. Рассмотрим иллюстрацию построения системы с помощью мультимодульной архитектуры (рисунок 1.2).

Рисунок 1.2 - Иллюстрация «монолитной» системы

На ней видно, что каждый компонент системы обособлен и не зависит от всех остальных компонентов. Так же при необходимости увеличения производительности можно делать копию только того модуля, для которого это требуется.

Еще одним преимуществом такой архитектуры является простая декомпозиция задач при реализации новых требований, т.к. каждый модуль может разрабатываться не зависимо от готовности других модулей.

Таким образом, можно сделать вывод, что проектирование систем на основе модулей является предпочтительным решением.

1.2. Описание и состав классического модуля системы

Рассмотрим более подробно принцип построения мультимодульных программных систем. Как говорилось ранее, в основе их архитектуры лежит понятие модуля, а так же его взаимодействия с остальными модулями системы.

Каждый модуль должен выполнять определенную, свойственную только ему, логику, реализация которой представлена набором определенных методов. Для удобства использования данных методов, их объединяют в единый «сервис».

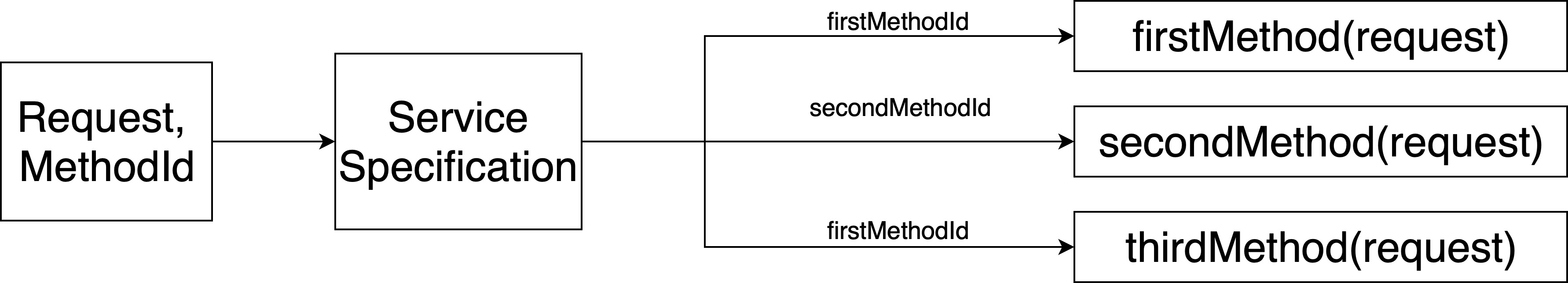
Сервис представляет собой совокупность методов, а так же интерфейса для работы с ними (рисунок 1.3). Данный интерфейс по сути является спецификацией сервиса, в которой указан идентификатор самого сервиса, а так же идентификатор каждого его метода.

Рисунок 1.3 - Схема классического сервиса

Проектирование модуля на основе сервисов дает возможность другим модулям вызывать его методы, используя лишь вышеуказанные идентификаторы. Данный подход позволяет инкапсулировать логику работы модуля от внешних модулей, что является хорошим тоном в кругах разработки.

Таким образом мы плавно подошли к вопросу межмодульного взаимодействия.

1.3. Описание методов межмодульного взаимодействия

Полноценная работоспособность системы невозможна без модульного взаимодействия. Рассмотрим, каким образом можно обмениваться данными между модулями.

Первым, самым простым способом является включение сервисного модуля в состав другого модуля (включение его артефакта в программный код) и вызов метода сервиса напрямую (рис. 1.4). Минусом данного подходя является то, что нарушается правило инкапсуляции, а так же сформированный артефакт исходного модуля будет сильно расширен за счет включения в него всего кода вызываемого им модуля.

Второй способ взаимодействия состоит в том, что бы вызывать метод сервиса через выставленный интерфейс (рис. 1.5). Такой подход решает проблему инкапсуляции, но при этом сервисный модуль так же необходимо включить в состав исходного модуля.

Третьим и наиболее предпочтительным способом является организация сетевого взаимодействия между модулями (рис. 1.6.). Для этого в каждый модуль включается http/protobuf сервер и выставляются соответствующие интерфейсы сервисов. На первый взгляд может показаться, что включение сервера в модуль усложнит его архитектуру, но на самом деле это заметно повысит удобство использования данного модуля. При таком подходе решаются все проблемы, описанные выше, а так же появляется возможность реализации модулей с помощью разных языков программирования и технологий. Но для такого взаимодействия необходимо определить единый формат передачи данных или выбрать единую структуру данных.

Рисунок 1.4 - Схема вызова метода сервиса напрямую

Рисунок 1.5 - Схема вызова метода сервиса через его интерфейс

Рисунок 1.6 - Схема вызова метода сервиса по сети

1.4. Конфигурация модуля

В процессе работы сервис модуля может использовать параметры, которые можно отнести к настраиваемым параметрам, значения которых могут изменяться в зависимости от условий эксплуатации модуля. Примерами таких параметров могут являться различные «флаги», отвечающие за активацию/деактивацию какой-либо части функционала, ip адреса внешних модулей и систем, ip адрес балансировщика, а так же любые другие параметры, значение которых будет необходимо изменить в ходе работы модуля без дополнительный сборки его артефакта.

Совокупность таких параметров определяет конфигурацию модуля, которая по факту является его основной характеристикой.

Стоит отметить, что конфигурация не задаёт поведение работы модуля, а только лишь управляет значениями параметров, которыми оперирует программная логика модуля, но при этом с ее помощью можно переключаться между режимами работы модуля (активация / деактивация).

Рассмотрим, каким образом можно задать модулю его конфигурацию. В основном выделяют три способа:

* непосредственно в коде (дефолтная конфигурация);
* в ресурсном файле (файловая конфигурация)
* с помощью внешнего запроса (удаленная конфигурация)

Последний способ представляет особый интерес, так как таким образом можно менять значения параметров в режиме реального времени, соответственно не будет оказано никакого влияния на пользователей системы. Но для его реализации модуль необходимо адаптировать, используя подход конфигурационного моделирования.

2. Адаптация мультимодульной программной системы

Тут будет то, что вы писали в статье (кстати, ее все же опубликовали, нов elibrary я ее не нашел)) потом могу скинуть сборник)

Загляни в книжку Растригина (<http://www.mtas.ru/Library/uploads/1189332881.pdf>). Там есть что подобрать сюда. Также пни Макса, чтобы скинул тебе что-нибудь про его «решатель». Он тож с Нечаем делал на эту тему статьи.

Конфигурационное моделирование программных систем

Тут напишу, что для адаптированной системы можно применить конфигуратор и тд (там один из пунктов в ваших статьях)

Описание механизма управления конфигурациями (относится к предыдущему пункту)

Рассмотрим концептуальную модель конфигурируемой программной системы, состоящей из множества модулей.

где

1. M – множество модулей
2. SM – множество состояний модулей
3. CM – множество конфигураций модулей
4. DM – взаимосвязи модулей
5. B – балансировщик запросов между модулями
6. C – конфигуратор

Основная реализация механизма управления конфигурациями ложится на плечи конфигуратора и балансировщика и осуществляется посредством асинхронного обновления параметров на стороне конфигуратора с последующим сигналом на запрос обновленной конфигурации ко всем модулям. Схематично данный процесс можно отобразить следующим образом:

Здесь покажи схему последовательных операций с её описанием ([https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0\_%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8](https://ru.wikipedia.org/wiki/%25D0%2594%25D0%25B8%25D0%25B0%25D0%25B3%25D1%2580%25D0%25B0%25D0%25BC%25D0%25BC%25D0%25B0_%25D0%25BF%25D0%25BE%25D1%2581%25D0%25BB%25D0%25B5%25D0%25B4%25D0%25BE%25D0%25B2%25D0%25B0%25D1%2582%25D0%25B5%25D0%25BB%25D1%258C%25D0%25BD%25D0%25BE%25D1%2581%25D1%2582%25D0%25B8))

3. Анализ процессов управления конфигурациями

Рассмотрим систему, состоящую из определенного количества модулей: 5 и взаимосвязанных систем: 2, а затем распишем на её примере основные бизнес-процессы, в которых участвуют операции по управлению конфигураций.

Определим инфраструктурную модель системы:

Здесь схема кластера. Покажи кластер с несколькими профилями (blue, green, standalone).

Детализируем основной набор процессов обслуживания и конфигурирования, применимый к данной системе:

1. Обновление профиля
   1. Гашение модулей
   2. Обновление модулей
   3. Обновление конфигурации профиля
   4. Поднятие модулей
   5. Получение конфигурации
2. Обновление приложения (blue-green схема)
3. Обновление конфигурации модуля в рантайме
4. Деактивация сервиса и метода
5. Перевод интеграционных модулей в режим заглушки

И описываешь процессы

Функциональные требования программной системы

Какие требования конкретно конфигуратор должен выполнять, короче, что он конкретно должен уметь.

Распиши здесь списком требования. После по каждому требованию сценарий использования с указаниями ролей (где админ, где модуль). Отдельно покажи сценарий по изменению конфигурации активации сервисов, то есть сценарий, где мы деактивируем сервис/метод.

Разработка программного продукта «Конфигуратор мультимодульных программных систем»

Здесь общее описание того, что конфигуратор как кусок кода будет делать.

По сути схемка с человечком-админом, который загружает JSON, а тот уже летит в конфигуратор, в роксу, в модули и описание этой схемки.

Здесь уже можно оперировать понятиями JSON, GRPC, HTTP и так далее.

Проектирование и дизайн пользовательского интерфейса

Здесь основные формы, действия, переходы по интерфейсу и скрины

Разработка пользовательского интерфейса

Здесь расскажи про технологию, про запросы, примеры запросов и описание входных и выходных данных

Разработка серверной части

Инфраструктурная схемка. Модули, конфигуратор, бд-шенька, какие где протоколы и всё такое. Расскажи про принцип работы ADK протобаф модулей (клиент, сервак). Почему выбрали протобаф. Его принцип работы. Про сервисную модель, про Flow (GRPC -> Spec -> Service -> Proxy -> GRPC).

Проектирование базы данных (RocksDB)

Концептуальная схема сущностей, параметры их. Расскажи про Value, что он из себя представляет, какие имеет формы. И дополнительно про типы данных конфигурации. По сути основные возможности конфигурации как типа данных. Также добавь про плюшки роксы.

Обеспечение безопасности передачи данных и управления модулями

Здесь расскажи про HTTPS SSL взаимодействие с конфигуратором и GRPC SSL между конфигуратором и модулями

Функциональное тестирование программной системы

Список основных тестов-спек. В идеале запилить на котспеках или грувиспоках и сюда скрины.

Нагрузочное тестирование программной системы

Здесь расскажи про гатлинг (<https://gatling.io/>). Он умеет и в GRPC тоже. Ну типо какую дал нагрузку, какие результаты получил. Пишется быстро, в принципе. Если долго/сложно, то сделай проще – Postman. Пошли лям запросов, здесь покажи стату из графаны по тому, как там что изменилось.

Анализ производительности системы

Расскажи здесь про мониторинг. Про графану, прометея, заббикс (Zabbix). Про то, как с их помощью можно смотреть на нагрузку. Дополнительно напиши, какие мы знаем способы оптимизации нагрузки. И можешь попробовать составить какую-нить формулу прогноза нагрузки. То есть, сколько будет выдерживать конфигуратор при наличии таких-то модулей с частотой обращения такой-то. Расскажи про кеш модулей здесь.

Заключение

Список литературы