Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ____ИУ7____
(Индекс)
_____ И. В. Рудаков_____
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

ЗАДАНИЕ
на выполнение курсового проекта

по дисциплине _____Распределённые системы обработки информации_____

Студент группы ИУ7И-22М_____

_____Ли Хань_____
(Фамилия, имя, отчество)

Тема курсового проекта _____Система торгового центра_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____учебный_____
Источник тематики (кафедра, предприятие, НИР) _____Кафедра_____

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание 1. Разработайте распределенную систему покупок, включающую несколько микросервисов: мик росервис пользователя, микросервис платежей, микросервис транзакций, микросервис продуктов и микро сервис корзины покупок.
2.Используйте центр регистрации Alibaba Nacos для управления микросервисами.
3.Используйте OpenFeign для реализации связи между микросервисами.
4. Используйте JWT для проверки входа пользователя.
5. Обеспечьте надежность микросервисов с помощью ограничения тока запроса, автоматического выклю чателя обслуживания и других решений.

Оформление курсового проекта:

Расчетно-пояснительная записка на 20-30 листах формата А4.
Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую, конструкторскую, технологическую части, заключение и список литературы.

Дата выдачи задания « __ » _____ 2024 г.

Руководитель курсового проекта _____ А.А.Ступников
(Подпись, дата) (И.О.Фамилия)

Студент _____ Х.Ли
(Подпись, дата) (И.О.Фамилия)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 2
1.Архитектура программного обеспечения 3
1.1 Монолитная архитектура 3
1.2 Микросервисная архитектура 3
1.3Весеннее облако 4
2. Сервис удаленного вызова 5
3. Регистрация и обнаружение службы. 7
3.1 ринцип действия регистрационного центра 7
3.2 Центр регистрации Nacos 9
3.3 Регистрация услуги 10
3.4 Обнаружение службы 10
4. Маршрутизация шлюза и проверка входа в систему 11
4.1 Маршрутизация шлюза 11
4.2 Проверка входа в систему 13
5. Защита сервиса 17
5.1 План защиты сервиса 17
5.1.1 Запрос ограничения тока 17
5.1.2 Сервисный автоматический выключатель 18
5.2 Запрос ограничения тока 18
5.3 Обслуживание автоматического выключателя 19
ЗАКЛЮЧЕНИЕ 22
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23

ВВЕДЕНИЕ

В связи с быстрым ростом Интернета и широкой популярностью онлайн-покупок потребители уделяют все больше и больше внимания опыту покупок на платформе при совершении покупок в Интернете. Торговые платформы сталкиваются с все большим количеством одновременных посещений, что, несомненно, предъявляет более высокие требования к ним. Требования к системе онлайн-покупок. В условиях монолитной архитектуры расширение бизнеса обычно использует распределенный и кластерный подход для создания и развертывания систем, способных справиться с высоким уровнем одновременного доступа. Однако из-за стремительного роста онлайн-потребления традиционные методы расширения систем столкнулись с узкими местами и не могут справиться с огромным потреблением. нуждаться. В отличие от монолитной архитектуры, микросервисная архитектура разделяет одно приложение на несколько микросервисов. Сервисы выполняют свои собственные обязанности и взаимодействуют друг с другом, чтобы предоставить пользователям конечную ценность. Каждый микросервис разделяется в соответствии с бизнес-логикой, разрабатывается и развертывается независимо. чтобы лучше соответствовать принципу единой ответственности и принципам проектирования высокой связанности и низкой связанности. Имеет более высокую гибкость и масштабируемость. Чтобы облегчить нагрузку, связанную с высоким уровнем одновременного доступа к системе, в этой статье реализована система покупок в торговом центре на основе микросервисной архитектуры.

Прежде всего, в этой статье ядро системы разбито на пять микросервисов: микросервис продукта, микросервис пользователя, микросервис корзины покупок, микросервис заказов и микросервис платежей. Во-вторых, Nacos представлен как центр конфигурации и центр регистрации, а JWT используется для аутентификации входа. Затем микросервисы защищаются с помощью ограничения тока запроса и сервисного автоматического выключателя.

1.Архитектура программного обеспечения

1.1 Монолитная архитектура

Монолитная структура: как следует из названия, все функциональные модули всего проекта разрабатываются в одном проекте; все модули необходимо скомпилировать и упаковать вместе при развертывании проекта. Архитектурный проект и модель разработки очень просты.

Когда масштаб проекта небольшой, эту модель можно быстро приступить к работе, а также она очень удобна для развертывания, эксплуатации и обслуживания. Поэтому на заре многие небольшие проекты использовали эту модель.

Однако по мере того, как бизнес-масштаб проекта становится все больше и больше, а количество команд разработчиков продолжает увеличиваться, монолитная архитектура представляет все больше проблем:

А) Стоимость совместной работы в команде высока: представьте, что десятки людей в вашей команде одновременно работают над одним и тем же проектом. Поскольку все модули находятся в одном проекте, физические границы между кодами разных модулей становятся все более размытыми. . В конечном итоге объединение функций в ветку определенно приведет вас к разрешению конфликтов.

Б) Эффективность выпуска системы низкая: любое изменение модуля требует выпуска всей системы, а процесс выпуска системы требует множества ограничений между несколькими модулями, и необходимо сравнивать различные файлы. Любая проблема в любом месте приведет к сбою выпуска. , и зачастую он выпускается за один раз. Это занимает десятки минут, а то и часов.

В) Плохая доступность системы. Каждый функциональный модуль монолитной архитектуры развертывается как служба и влияет друг на друга. Некоторые «горячие» функции истощают системные ресурсы, что приводит к низкой доступности других служб.

1.2 Микросервисная архитектура

Микросервисная архитектура, прежде всего, — это сервитизация, то есть отделение функциональных модулей монолитной архитектуры от монолитного приложения и их независимое развертывание в виде нескольких сервисов. При этом должны соблюдаться следующие характеристики:

А) Единая ответственность: микросервис отвечает за часть бизнес-функций, и его основные данные не зависят от других модулей.

Б) Автономия команды: каждый микросервис имеет свой независимый персонал по разработке, тестированию, выпуску, эксплуатации и техническому обслуживанию, а размер команды не превышает 10 человек.

В)Автономность службы. Каждый микросервис упаковывается и развертывается независимо и имеет доступ к собственной независимой базе данных. Изолировать сервисы необходимо, чтобы не влиять на другие сервисы.

Например, в проекте торгового центра этого проекта мы можем разделить такие модули, как продукты, пользователи, корзины покупок и транзакции, и передать их разным командам для независимой разработки и развертывания, как показано на рисунке 1.1 ниже.

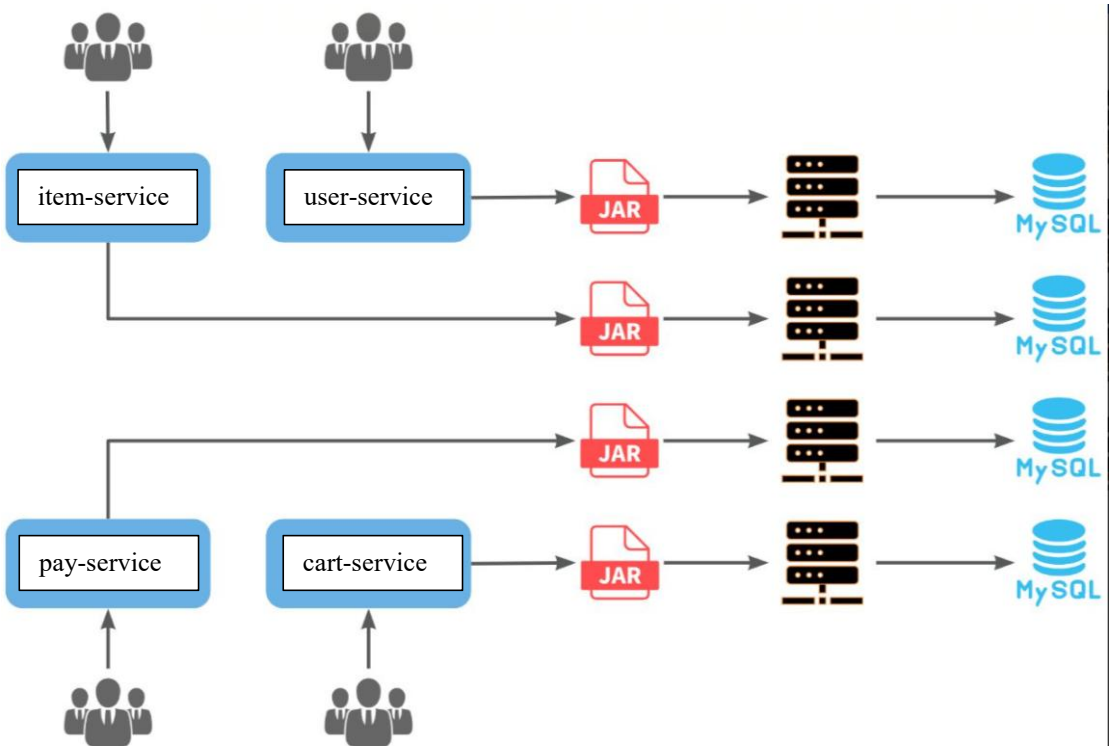

Рисунок 1.1. Микросервис торгового центра

1.3Весеннее облако

Различные проблемы, возникающие после разделения микросервисов, имеют соответствующие решения и компоненты микросервисов, а среду SpringCloud можно назвать наиболее полной коллекцией компонентов микросервисов в области Java.

Более того, SpringCloud использует возможности автоматической сборки SpringBoot, что значительно снижает стоимость разработки проекта и использования компонентов. Для малых и средних предприятий, которые не

имеют возможности самостоятельно разрабатывать компоненты микросервисов, можно сказать, что использование SpringCloud Family Bucket для реализации разработки микросервисов является наиболее подходящим выбором.

В этом проекте мы используем следующие версии: Spring Cloud 2021.0.3 и Spring Boot 2.7.12. Как показано на рисунке 1.2 ниже.

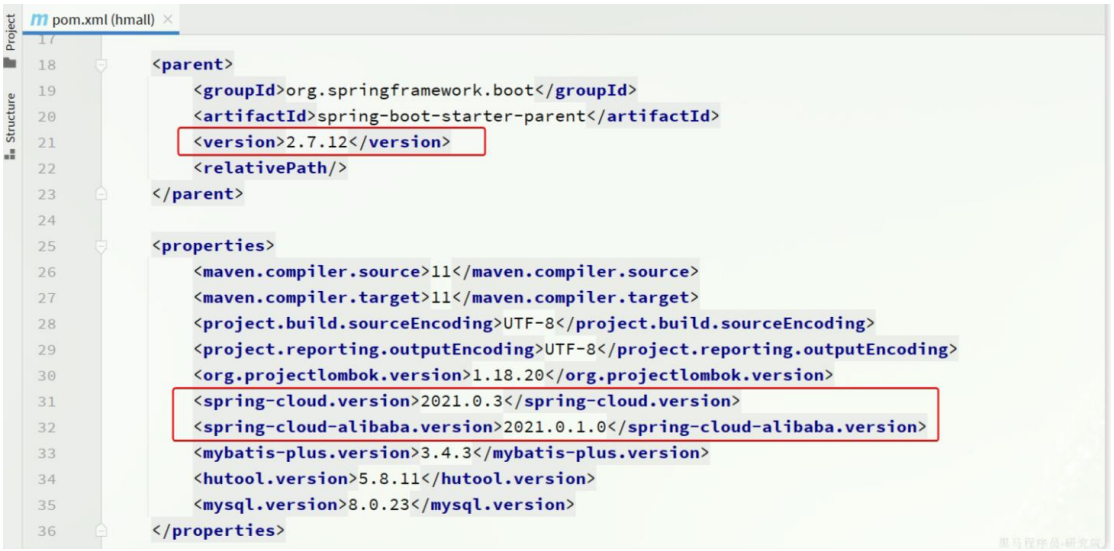

Рисунок 1.2 Схема конфигурации версии Spring Cloud

2. Сервис удаленного вызова

При развертывании проекта в виде нескольких микросервисов мы обнаружили проблему: информацию о продукте необходимо запрашивать в корзине покупок, но вся логика запроса информации о продукте была перенесена в службу обслуживания товаров, из-за чего мы не смогли выполнить запрос. Конечным результатом является то, что запрошенные данные корзины покупок являются неполными, поэтому, чтобы решить эту проблему, мы должны изменить код и преобразовать исходный вызов локального метода в удаленный вызов через микросервисы (RPC, Remote Produce Call). Таким образом, процесс запроса списка корзины покупок теперь выглядит так, как показано на рисунке 2.1 ниже.

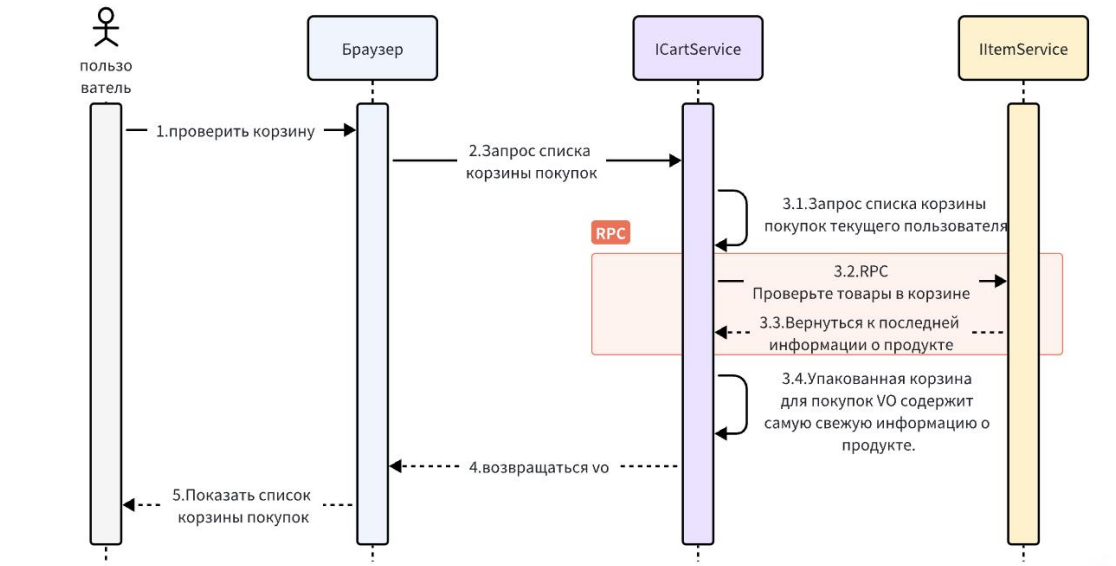

Рисунок 2.1. Схема последовательности запроса корзины покупок

Существует множество способов реализации RPC, например: на основе протокола HTTP, на основе протокола Dubbo. В этом проекте используется метод HTTP. Этот метод не учитывает конкретную техническую реализацию поставщика услуг. Ему необходимо только предоставить интерфейс HTTP внешнему миру, что больше соответствует потребностям микросервисов.

Spring предоставляет нам API RestTemplate, который может легко отправлять Http-запросы. Он предоставляет большое количество методов, упрощающих отправку HTTP-запросов. Вы можете видеть, что поддерживаются общие запросы Get, Post, Put и Delete. Если параметры запроса сложны, вы также можете использовать метод обмена для создания. запрос.

Мы определяем класс конфигурации в сервисе Cart-Service, как показано на рисунке 2.2 ниже.

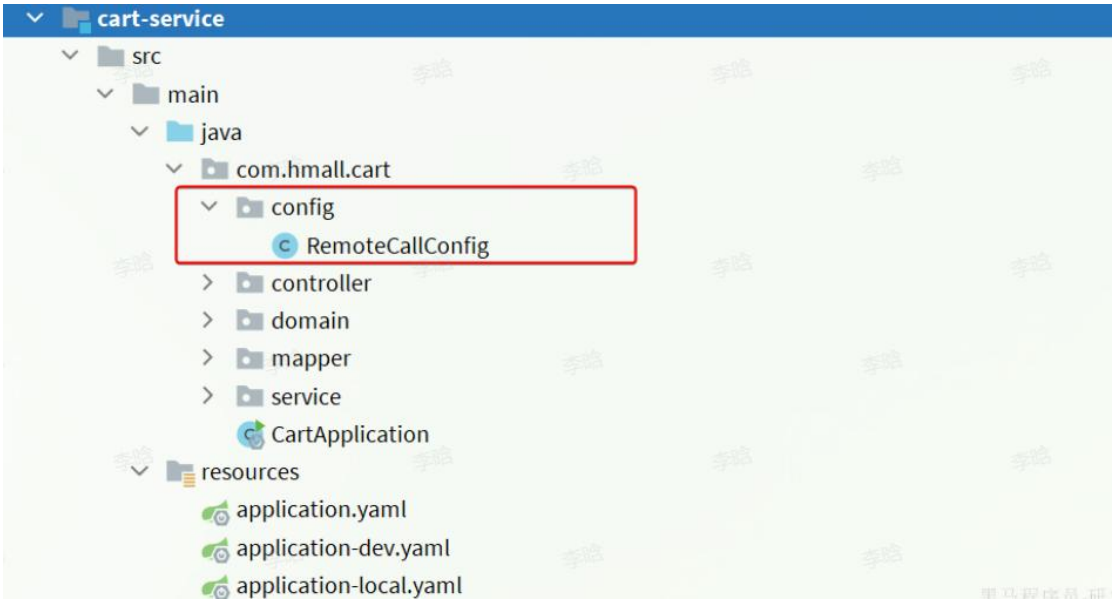

Рисунок 2.2 Класс конфигурации удаленного вызова

Затем мы модифицируем метод handleCartItems класса CartServiceImpl в сервисе cart и отправляем http-запрос в сервис item, как показано на рисунке 2.3 ниже.

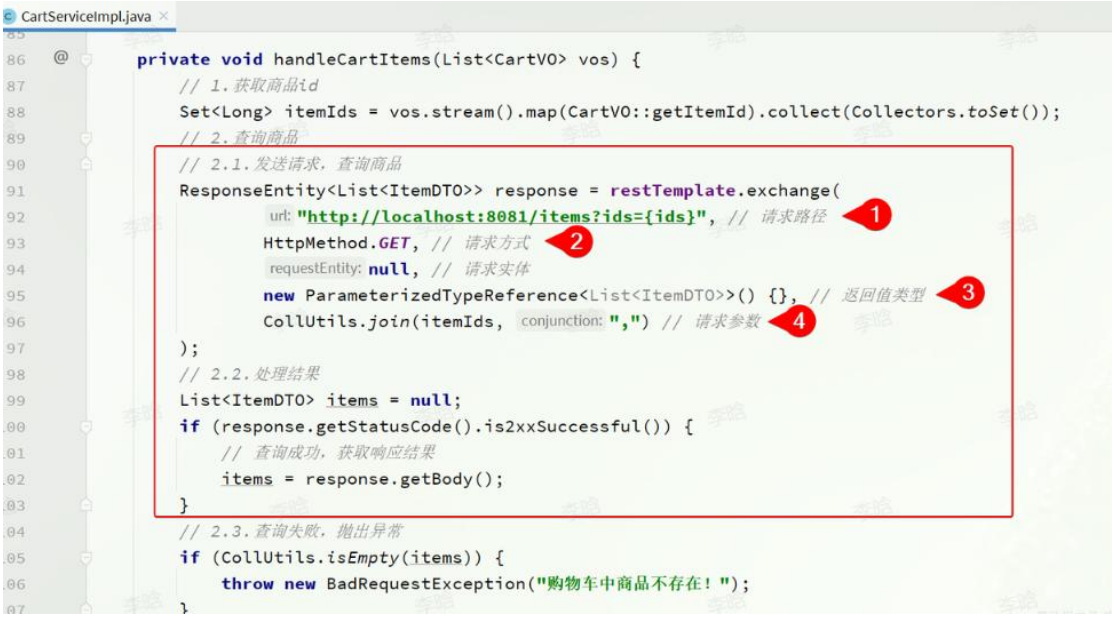

Рисунок 2.3. Изменение метода handleCartItems.

Наконец, перезапустите службу корзины и протестируйте интерфейс для запроса списка моей корзины. Вы можете обнаружить, что все данные, связанные с товаром, были запрошены. В этом процессе служба item предоставляет интерфейс запросов, а служба корзины использует HTTP-запросы для вызова этого интерфейса. Поэтому услугу-товар можно назвать поставщиком услуги, а услугу-корзину — потребителем или вызывающей услугу услугой.

3. Регистрация и обнаружение службы.

3.1 ринцип действия регистрационного центра

Мы реализуем удаленные вызовы между микросервисами посредством HTTP-запросов. Однако при использовании этого метода ручной отправки HTTP-запросов возникают некоторые проблемы. Представьте себе, если микросервисы продукта вызываются много раз, чтобы справиться с более высоким параллелизмом, мы выполняем многоэкземплярное развертывание, как показано на рисунке 3.1.

Рис. 3.1. Развертывание нескольких экземпляров службы корзины

В настоящее время каждый экземпляр службы элементов имеет свой IP-адрес или порт, и возникают следующие проблемы:

A)Существует так много экземпляров сервиса товаров, откуда сервису корзины известен адрес каждого экземпляра?

B)HTTP-запрос должен записать URL-адрес. Какой экземпляр должен вызвать сервис службы корзины?

C)Что делать, если во время работы произошел сбой определенного экземпляра службы товаров, а служба корзины все еще вызывается?

D)Если параллелизм слишком высок и служба item-service временно развертывает еще N экземпляров, как сервис-cart узнает адрес нового экземпляра?

Для решения вышеперечисленных проблем необходимо создать регистрационный центр. Далее разберем принципы работы регистрационного центра.

В процессе удаленного вызова микросервисов есть две роли: поставщик услуг: предоставляет интерфейсы для доступа к другим микросервисам, например потребителям сервиса товаров: вызывает интерфейсы, предоставляемые другими микросервисами, такими как сервис корзины;

В масштабных микросервисных проектах количество поставщиков услуг будет очень большим. Для управления этими услугами вводится концепция центра регистрации. Взаимоотношения между центром регистрации, поставщиком услуг и потребителем услуг показаны на рисунке 3.2:

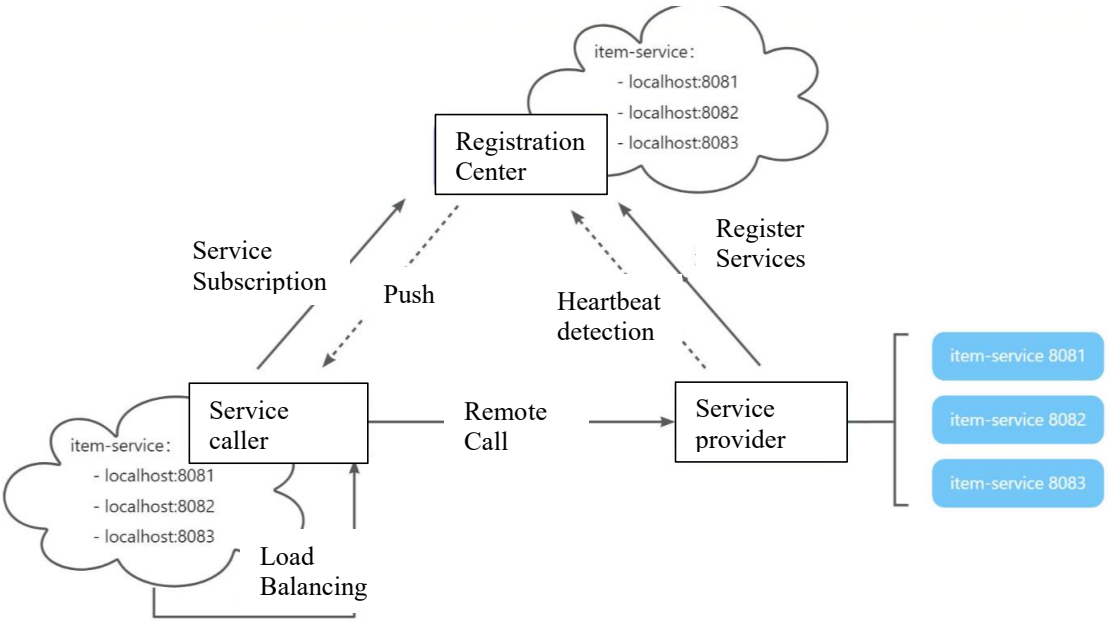

Рисунок 3.2 Концептуальная схема центра регистрации

Процесс выглядит следующим образом:

A)При запуске службы она регистрирует свою собственную информацию о службе (имя службы, IP-адрес, порт) в центре регистрации.

B)Вызывающий абонент может подписаться на желаемую услугу в центре регистрации и получить список экземпляров, соответствующий услуге (одна услуга может быть развернута с несколькими экземплярами).

C)Вызывающий загружает список экземпляров и выбирает экземпляр

D)Вызывающий инициирует удаленный вызов экземпляра.

3.2 Центр регистрации Nacos

В настоящее время существует множество инфраструктур центров регистрации с открытым исходным кодом, наиболее распространенными в Китае являются:

Eureka: создано Netflix, в настоящее время интегрировано в Spring Cloud, обычно используется для приложений Java; Nacos: создано Alibaba, в настоящее время интегрировано в Spring Cloud. Alibaba, обычно используется для приложений Java; Consul: создано HashiCorp, в настоящее время интегрировано в Spring Cloud, нет; ограничения на языки микросервисов

Все вышеперечисленные центры регистрации следуют спецификациям API в Spring Cloud, поэтому особой разницы в развитии бизнеса и использовании нет. В этом проекте мы используем Nacos в качестве центра регистрации.

3.3 Регистрация услуги

Далее мы возьмем item-service в качестве примера, чтобы зарегистрировать его в Nacos. Шаги следующие: введите зависимости и перезапустите адрес Nacos;

Добавьте зависимости в pom.xml item-service, как показано на рисунке 3.3 ниже.

```
<!--nacos 服务注册发现-->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
```

Рисунок 3.3

Добавьте конфигурацию адреса nacos в файл application.yml item-service, как показано на рисунке 3.4 ниже.

```
spring:
  application:
    name: item-service # 服务名称
  cloud:
    nacos:
      server-addr: 192.168.150.101:8848 # nacos地址
```

Рисунок 3.4

Чтобы проверить ситуацию с несколькими экземплярами службы, мы настраиваем экземпляр службы элемента, перезапускаем два экземпляра службы элемента и получаем доступ к консоли nacos. Мы можем обнаружить, что регистрация службы прошла успешно, как показано. на рисунке 3.5 ниже.

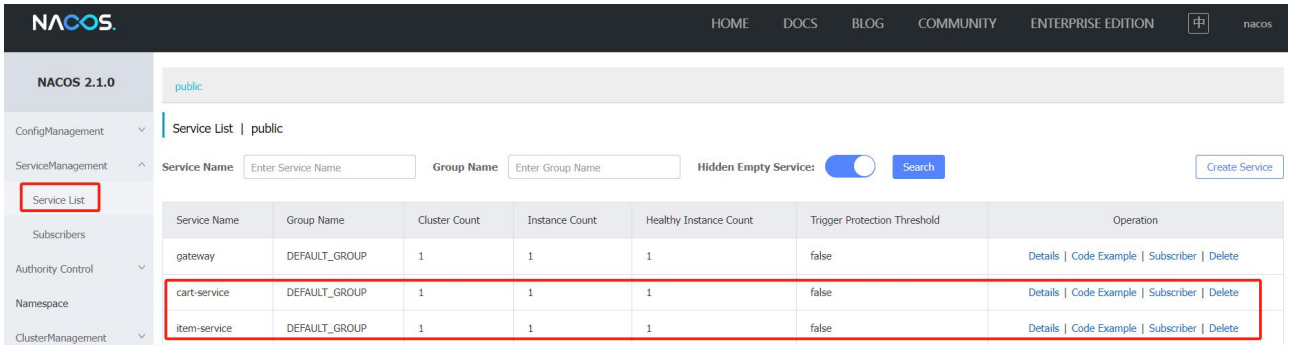

Рисунок 3.5

3.4 Обнаружение службы

Потребителю службы необходимо подписаться на службу в nacos. Этот процесс представляет собой обнаружение службы. Шаги следующие: настроить адрес Nacos и вызвать службу.

Помимо введения зависимостей nacos, обнаружение сервисов также требует балансировки нагрузки, поэтому необходимо ввести зависимость LoadBalancer, предоставляемую Spring Cloud. Мы добавляем зависимости, показанные на рисунке 3.6 ниже, в pom.xml в cart-service:

```
<!--nacos 服务注册发现-->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
```

Рисунок 3.6

Можно обнаружить, что зависимости Nacos здесь соответствуют регистрации службы. Эта зависимость включает в себя как функции регистрации службы, так и функции обнаружения. Потому что любой микросервис может вызывать другие, а также может вызываться другими, то есть он может быть вызывающим или провайдером.

Добавьте конфигурацию адреса nacos в файл application.yml службы корзины, как показано на рисунке 3.7 ниже.

```
spring:
  cloud:
    nacos:
      server-addr: 192.168.150.101:8848
```

Рисунок 3.7

Затем вызывающая служба cart-service может подписаться на услугу item-service. Однако у item-service есть несколько экземпляров, и при фактическом вызове вам нужно знать адрес только одного экземпляра. Таким образом, вызывающая служба должна использовать алгоритм балансировки нагрузки, чтобы выбрать один из нескольких экземпляров для доступа. Общие алгоритмы балансировки нагрузки включают в себя: случайный, опрос, хэш IP, последнее посещение и т. д. В этом проекте мы выбираем случайную балансировку нагрузки.

4. Маршрутизация шлюза и проверка входа в систему

4.1 Маршрутизация шлюза

Как следует из названия, шлюз — это шлюз в сеть. Данные передаются между сетями. При передаче из одной сети в другую они должны пройти через шлюз для маршрутизации и пересылки данных, а также проверки безопасности данных. В проекте микросервиса внешний запрос не может напрямую

обращаться к микросервису, но должен запрашивать шлюз: шлюз может выполнять контроль безопасности, то есть проверку личности входа, и он будет освобожден только после прохождения проверки; аутентификации, шлюз будет определять на основе запроса, какой должен быть доступ. Какой микросервис пересылает запрос.

В SpringCloud предусмотрено два решения для реализации шлюза: Netflix Zuul и SpringCloudGateway. Среди них Netflix Zuul был реализован на ранней стадии и теперь исключен, а SpringCloudGateway основан на технологии Spring WebFlux, полностью поддерживает реактивное программирование и имеет более высокие возможности пропускной способности. В этом проекте мы используем SpringCloudGateway.

Поскольку сам шлюз также является независимым микросервисом, необходимо также создать функцию разработки модулей. Примерные шаги следующие: Создайте микросервис шлюза; введите зависимости SpringCloudGateway и NacosDiscovery; напишите класс запуска;

Создайте новый файл application.yaml в каталоге ресурсов модуля шлюза, как показано на рисунке 4.1 ниже, чтобы настроить маршрутизацию.

```yaml
server:
  port: 8080
spring:
  application:
    name: gateway
  cloud:
    nacos:
      server-addr: 192.168.150.101:8848
    gateway:
      routes:
        - id: item # 路由规则id，自定义，唯一
          uri: lb://item-service # 路由的目标服务，lb代表负载均衡，会从注册中心拉取服务列表
          predicates: # 路由断言，判断当前请求是否符合当前规则，符合则路由到目标服务
            - Path=/items/**,/search/** # 这里是以请求路径作为判断规则
        - id: cart
          uri: lb://cart-service
          predicates:
            - Path=/carts/**
        - id: user
          uri: lb://user-service
          predicates:
            - Path=/users/**,/addresses/**
        - id: trade
          uri: lb://trade-service
          predicates:
            - Path=/orders/**
        - id: pay
          uri: lb://pay-service
          predicates:
```

Рисунок 4.1

4.2 Проверка входа в систему

В монолитной архитектуре нам нужно только один раз выполнить вход в систему и проверку личности, после чего мы сможем получать информацию о пользователях во всех компаниях. После разделения на микросервисы каждый микросервис развертывается независимо и больше не обменивается данными. Это означает, что каждый микросервис должен выполнять проверку входа в систему, что явно нецелесообразно.

Наш логин основан на JWT. Алгоритм проверки JWT сложен и требует использования секретного ключа. Если каждая микрослужба выполняет проверку входа, возникают две основные проблемы: каждая микрослужба должна знать секретный ключ JWT, что небезопасно; каждая микрослужба постоянно записывает код проверки входа и код проверки разрешений.

Поскольку шлюз является входом ко всем микросервисам, все запросы сначала должны проходить через шлюз. Мы можем включить функцию проверки входа в шлюз, чтобы решить упомянутую ранее проблему: нам нужно только сохранить секретный ключ в шлюзе и пользовательской службе, нам нужно только разработать функцию проверки входа в шлюз; Процесс проверки входа показан на рисунке 4.2.

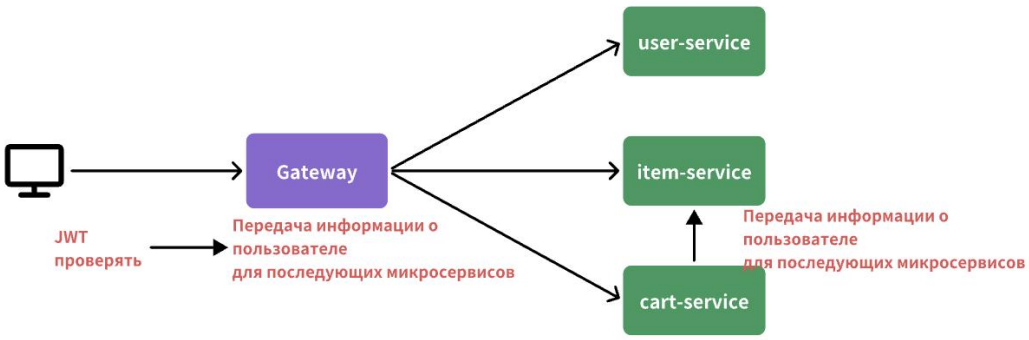

Рисунок 4.2 Блок-схема проверки входа в систему

Проверка входа должна быть выполнена до того, как запрос будет перенаправлен в микросервис, иначе он будет бессмысленным. Пересылка запросов шлюза реализуется внутренним кодом шлюза. Если вы хотите выполнить проверку входа в систему перед пересылкой запроса, вы должны понимать основные принципы внутренней работы шлюза.

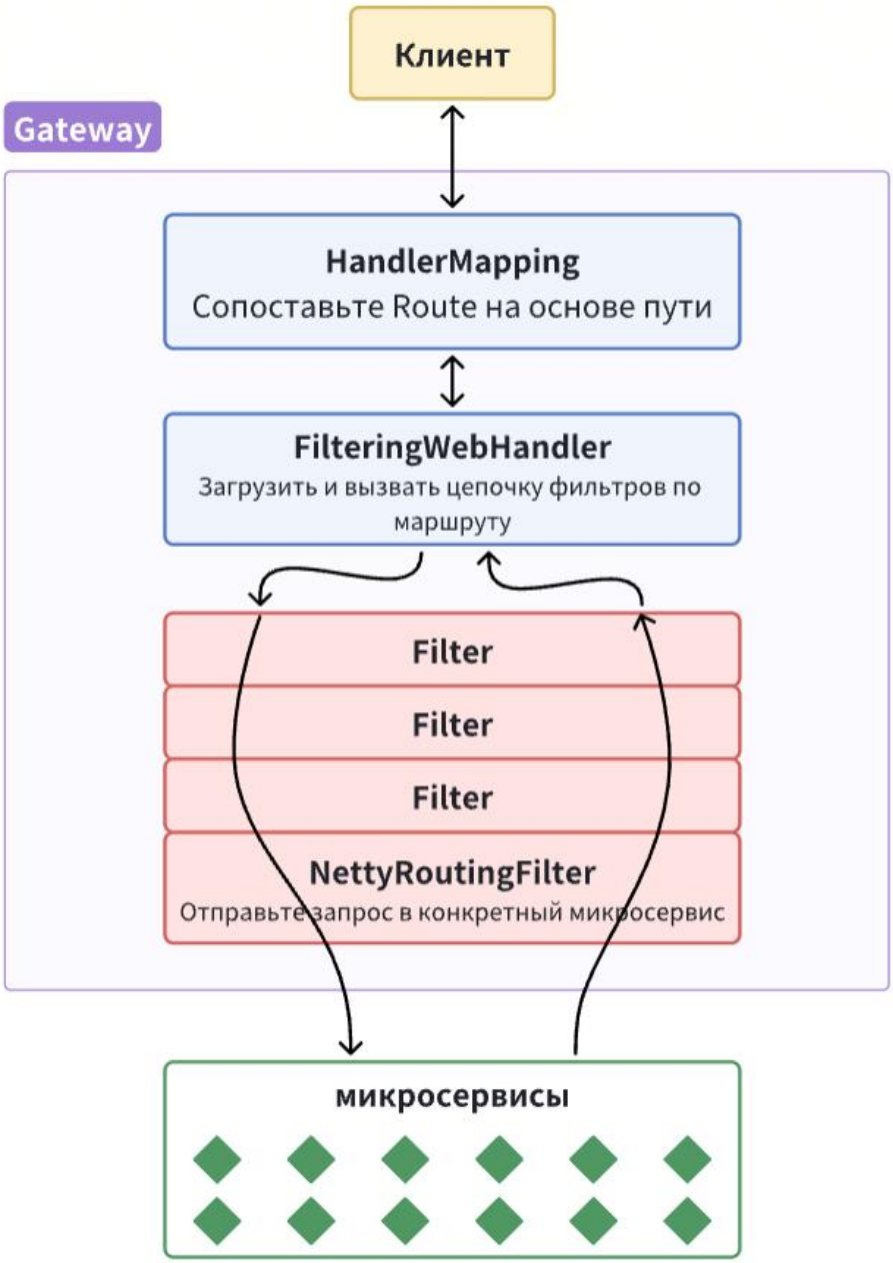

Рисунок 4.3

Как показано на рисунке 4.3:

A) После того, как запрос клиента попадет в шлюз, HandlerMapping оценит запрос, найдет правило маршрутизации (Route), соответствующее текущему запросу, а затем передаст запрос WebHandler для обработки.

B)WebHandler загрузит цепочку фильтров, которую необходимо выполнить по текущему маршруту, а затем выполнит фильтры один за другим по порядку (далее Фильтр).

C)На рисунке Фильтр разделен на левую и правую части пунктирными линиями, поскольку внутренняя логика Фильтра разделена на пред- и пост-

части, которые будут выполняться до и после маршрутизации запроса в микросервис соответственно.

D)Запрос будет перенаправлен в микросервис только после последовательного выполнения предварительной логики всех фильтров.

E)После того, как микросервис вернет результат, логика сообщения фильтра выполняется в обратном порядке.

F) Наконец, верните результат ответа.

Окончательную пересылку запроса выполняет фильтр с именем NettyRoutingFilter, и этот фильтр является последним во всей цепочке фильтров. Если мы сможем определить фильтр, реализовать в нем логику проверки входа и определить порядок выполнения фильтра перед NettyRoutingFilter, это будет соответствовать потребностям нашего проекта.

В цепочке фильтров шлюза есть два типа фильтров: GatewayFilter: фильтр маршрутизации, область действия которого относительно гибка и может быть любым указанным маршрутом. Route GlobalFilter: глобальный фильтр, областью действия которого являются все маршруты и который нельзя настроить.

В нашем проекте мы используем специальный GlobalFilter для завершения проверки входа. Основной код GlobalFilter показан на рисунке 4.4 ниже.

```java
@Override
public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
    // 1.获取Request
    ServerHttpRequest request = exchange.getRequest();
    // 2.判断是否不需要拦截
    if(isExclude(request.getPath().toString())){
        // 无需拦截，直接放行
        return chain.filter(exchange);
    }
    // 3.获取请求头中的token
    String token = null;
    List<String> headers = request.getHeaders().get("authorization");
    if (!CollUtils.isEmpty(headers)) {
        token = headers.get(0);
    }
    // 4.校验并解析token
    Long userId = null;
    try {
        userId = jwtTool.parseToken(token);
    } catch (UnauthorizedException e) {
        // 如果无效，拦截
        ServerHttpResponse response = exchange.getResponse();
        response.setRawStatusCode(401);
        return response.setComplete();
    }
```

Рисунок 4.4. Схема основного кода GlobalFilter

Затем проведите тест и получите доступ к установленному нами пути перехвата. Запрос будет перехвачен, если вы не вошли в систему, и будет возвращен код состояния 401, как показано на рисунке 4.5 ниже.

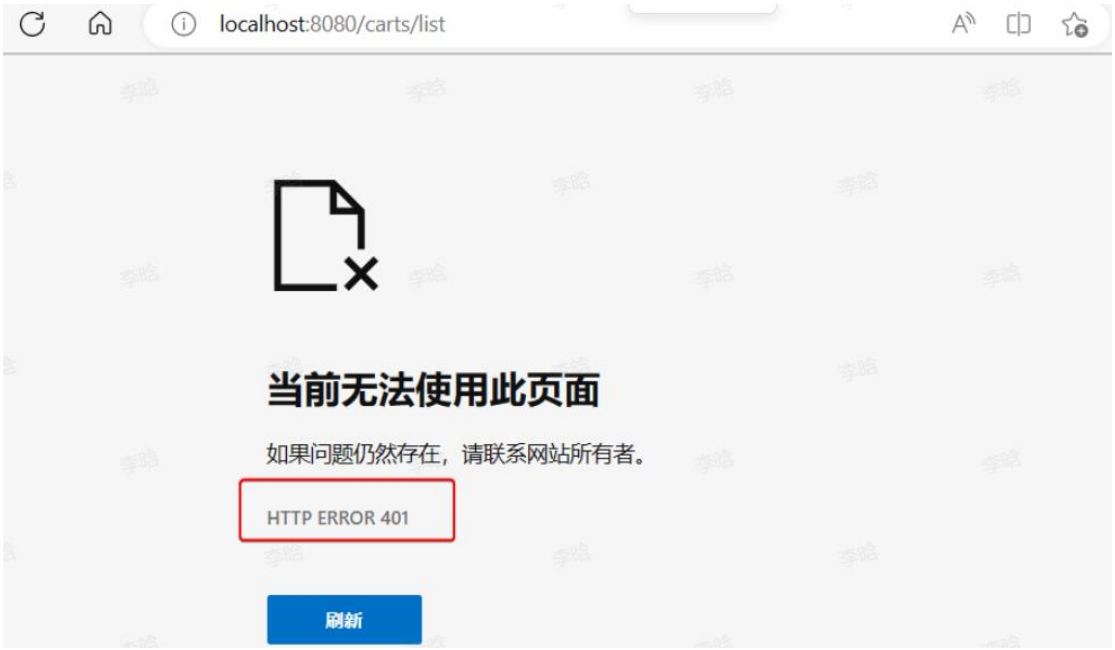

Рисунок 4.5

5. Защита сервиса

В процессе удаленного вызова микросервисов еще есть несколько проблем, которые необходимо решить.

Во-первых, это вопрос устойчивости бизнеса:

Например, в предыдущем бизнесе по запросу списка корзин покупок службе корзин покупок необходимо было запросить последнюю информацию о продукте, сравнить ее с данными корзины покупок и напомнить пользователю. С точки зрения бизнеса, чтобы улучшить взаимодействие с пользователем, даже если запрос продукта не удался, список корзины покупок должен отображаться правильно, даже если он не содержит последней информации о продукте.

Существует также проблема каскадного отказа:

Или запросите бизнес-корзину покупок, если бизнес по обслуживанию продуктов имеет высокий уровень параллелизма и требует слишком много подключений Tomcat. Это может привести к увеличению времени ответа всех интерфейсов стандартных сервисов, увеличению задержки или даже к блокировке на длительное время до тех пор, пока запрос не завершится неудачно.

В настоящее время бизнес-запрос корзины покупок должен запрашивать и ждать результатов запроса продукта, что приводит к тому, что время ответа бизнес-запроса списка корзин покупок увеличивается или даже блокируется до тех пор, пока к нему невозможно получить доступ. В настоящее время, если поступает много запросов на запрос корзины покупок, соединение Tomcat службы корзины покупок может занять больше времени, время ответа всех интерфейсов увеличится, а производительность всей службы будет низкой или даже недоступной. По аналогии, службы, связанные со службой корзины покупок, службой продуктов и т. д. во всей группе микросервисов, могут иметь проблемы, что в конечном итоге приводит к тому, что весь кластер становится недоступным.

5.1 План защиты сервиса

5.1.1 Запрос ограничения тока

Наиболее важной причиной сбоя службы является слишком высокий уровень параллелизма. Решив эту проблему, можно избежать большинства неудач. Конечно, параллельность интерфейса не всегда высока, но внезапна. Таким образом, ограничение потока запросов предназначено для ограничения

или контроля одновременного трафика, к которому осуществляется доступ через интерфейс, чтобы избежать сбоя обслуживания из-за резкого увеличения трафика.

Часто имеется ограничитель тока для ограничения тока запроса. Кривая одновременного запроса с высокими и низкими числами становится очень стабильной после прохождения ограничителя тока. Это похоже на плотину на гидроэлектростанции, которая хранит воду. Размер стока воды можно контролировать с помощью переключателя, так что поток вниз по течению всегда поддерживается на стабильном уровне.

5.1.2 Сервисный автоматический выключатель

Службы продуктов могут замедлить скорость ответа интерфейса службы корзины покупок (вызывающей службы). Более того, сбои в запросе продукта по-прежнему будут приводить к сбоям в работе функции запроса корзины покупок, а бизнес-корзина покупок также станет недоступной.

Поэтому нам нужно сделать две вещи: 1. Написать логику понижения версии службы: это логика обработки после сбоя вызова службы. В соответствии с бизнес-сценарием может быть выдано исключение или может быть возвращено дружественное приглашение или данные по умолчанию. 2. Статистика аномалий и автоматических выключателей: статистика аномального соотношения поставщиков услуг. Когда соотношение слишком велико, это указывает на то, что интерфейс будет влиять на другие услуги. Интерфейс следует отклонить и следует использовать логику перехода на более раннюю версию. напрямую.

Существует множество технологий защиты микросервисов. В этом проекте для реализации защиты микросервисов используется Sentinel — это платформа защиты сервисов с открытым исходным кодом, добавленная в SpringCloudAlibaba.

5.2 Запрос ограничения тока

Нажмите кнопку управления потоком за ссылкой на точку кластера в Sentinel, чтобы настроить ограничение потока на ней, как показано на рисунке 5.1.

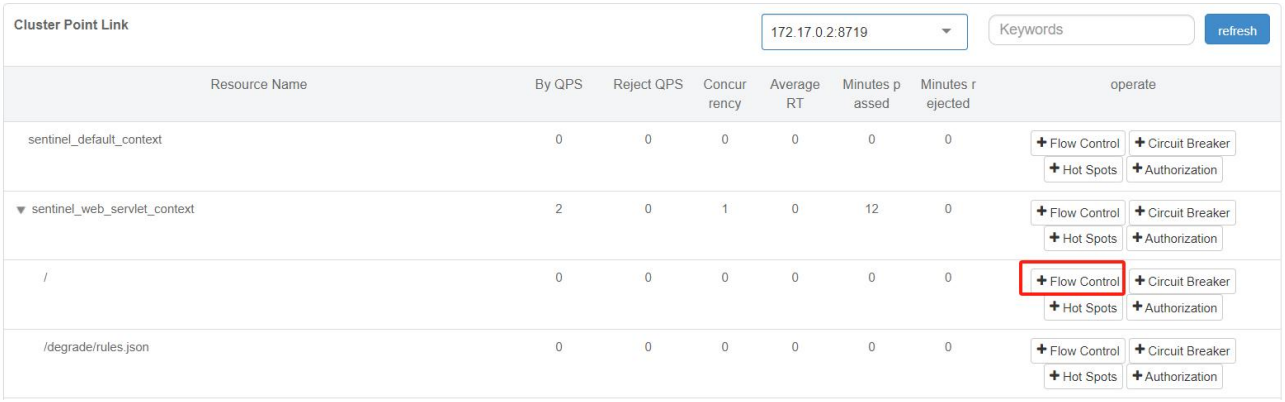

Рисунок 5.1

Вы можете установить ограничение трафика ресурса точки кластера для запроса списка корзин покупок на 6 в секунду, то есть максимальное количество QPS равно 6, как показано на рисунке 5.2.

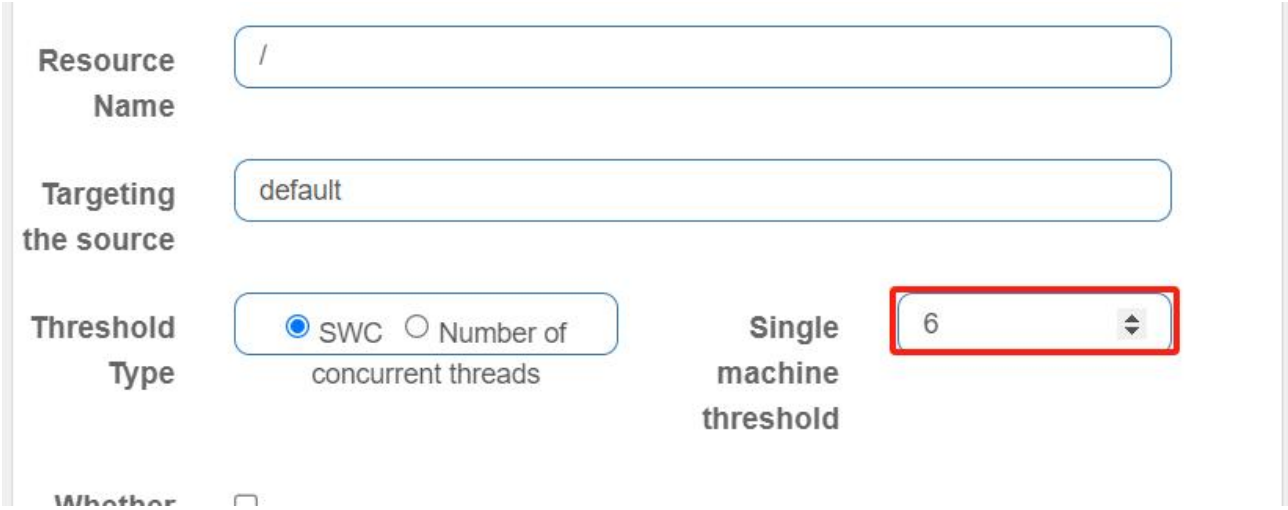

Рисунок 5.2

5.3 Обслуживание автоматического выключателя

Время ожидания для запроса продуктов велико, из-за чего время ожидания для запроса корзин покупок также становится очень долгим. Это не только замедляет работу службы корзины покупок и потребляет больше ресурсов службы корзины покупок, но и ухудшает пользовательский опыт.

Для неработоспособных интерфейсов, таких как стандартные сервисы, нам следует прекратить вызовы и сразу перейти к логике перехода на более раннюю версию, чтобы не влиять на текущий сервис. То есть интерфейс запроса продукта отключен. Когда интерфейс обслуживания продукта вернется в

нормальное состояние, вызов снова будет разрешен. Это фактически рабочий режим автоматического выключателя.

Автоматический выключатель в Sentinel может не только подсчитывать долю медленных запросов на определенном интерфейсе, но и долю аномальных запросов. Когда эти отношения превысят порог, интерфейс будет отключен, то есть все запросы на доступ к интерфейсу будут перехвачены и понижены, когда интерфейс вернется в нормальное состояние, запросы к интерфейсу будут освобождены;

Переключение рабочего состояния выключателя контролируется конечным автоматом (см. рисунок 5.3 ниже):

Рисунок 5.3

Государственный автомат включает в себя три состояния:

Close: В замкнутом состоянии автоматический выключатель отменяет все запросы и начинает подсчитывать ненормальный коэффициент и коэффициент медленного запроса. Если он превышает порог, он переключается в открытое состояние.

Open: в открытом состоянии вызов службы является автоматическим выключателем, и запрос на доступ к службе автоматического выключателя будет отклонен, быстро завершится сбоем и перейдет непосредственно к логике понижения версии. После того, как открытое состояние продлится некоторое время, оно перейдет в полуоткрытое состояние.

Half-open: полуоткрытое состояние, запрос отпускается, и следующая операция оценивается на основе результата выполнения. Если запрос успешен: он переходит в закрытое состояние, если запрос не удался: он переходит в открытое состояние;

Мы можем настроить стратегию автоматического выключателя в консоли, нажав кнопку автоматического выключателя за точкой кластера, как показано на рис. 5.4 и рис. 5.5 ниже.

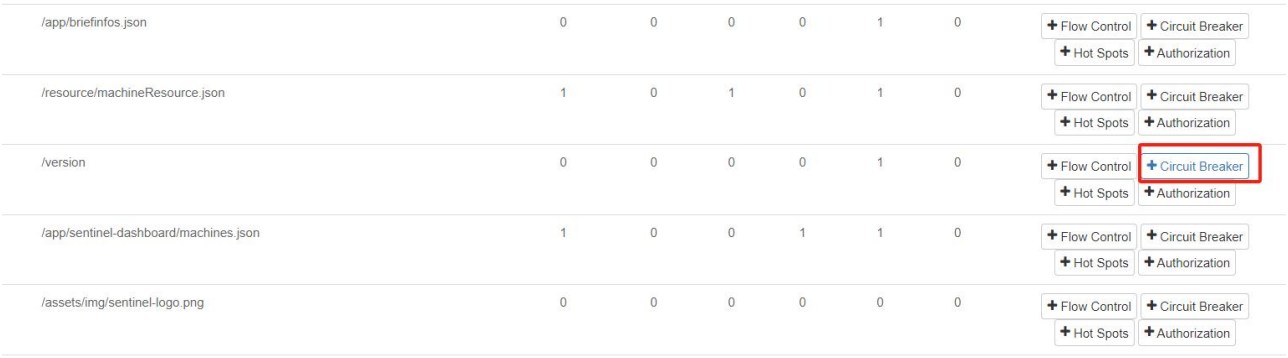

Рисунок 5.4

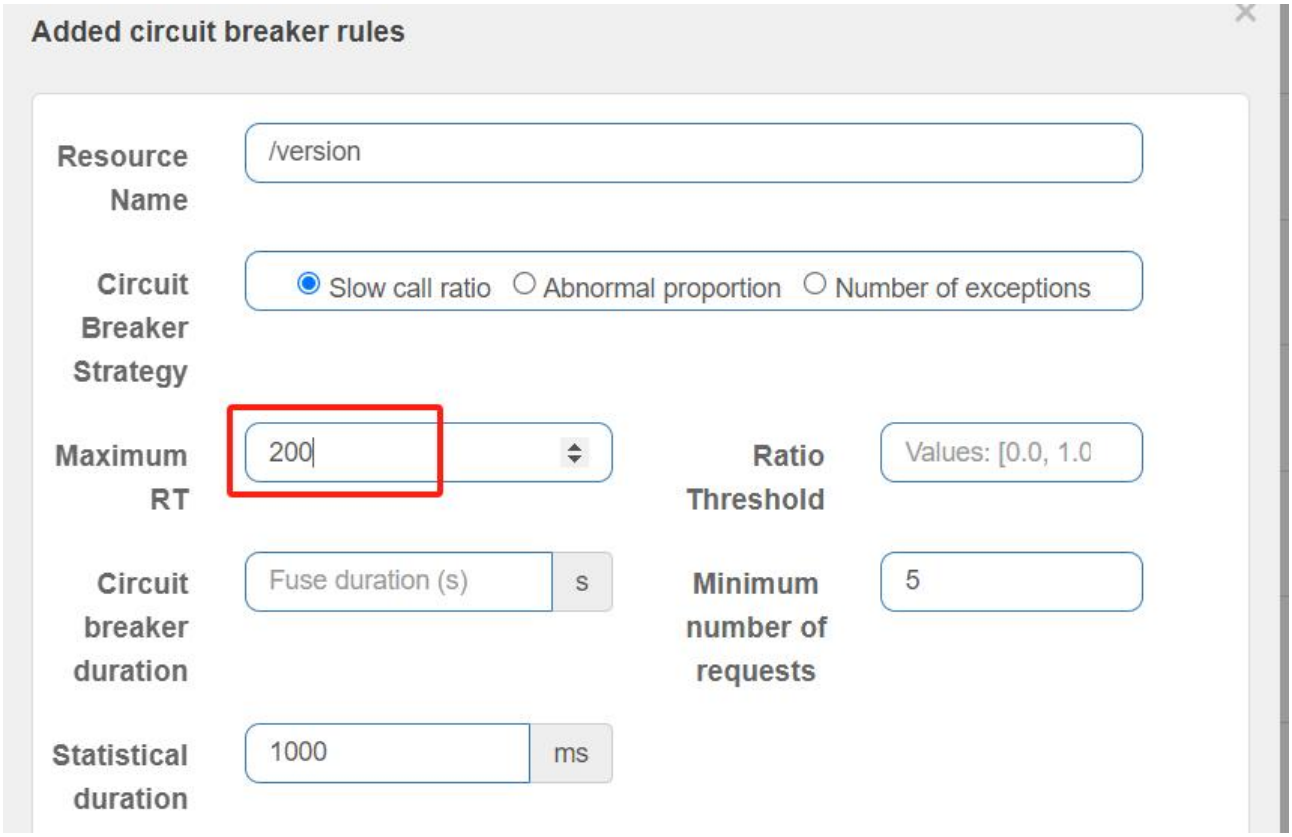

Рисунок 5.5

ЗАКЛЮЧЕНИЕ

В этой статье проводится исследование онлайн-покупок и микросервисной архитектуры, а также разрабатывается и реализуется торговая система торгового центра на основе микросервисной архитектуры. Анализируются причины быстрого развития онлайн-покупок и недостатки использования монолитной архитектуры для построения системы онлайн-покупок. Ориентируясь на популярную в настоящее время микросервисную архитектуру, проводится углубленное исследование основных компонентов микросервисов и их развитие. тенденция микросервисов. Он представляет Nacos для реализации регистрации, обнаружения и управления службами, внедрение OpenFeign решает проблему взаимных вызовов между службами, введение шлюза в качестве входного шлюза микросервисов для фильтрации и пересылки запросов к соответствующим микросервисам. и внедрение Sentinel для реализации ограничения тока запроса и автоматического выключателя для защиты микросервисов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.Чжан Суджуань, Цзэн Цян и Шихуа прокладывают путь к бизнесу [J]. Электронная коммерция Китая, 2004(11):25-29.

2.Цай Юнджин. Взлёты и падения, а также вариации трёх поколений китайской «большой тройки» Интернета [J]. Эра больших данных, 2020(07):60-76.

3.Чжу Хуйминь, Исследование стратегии развития социальной электронной коммерции[D], Шаньдунский педагогический университет, 2020 г.

4.Ху Ханг, Чен Чен, Сян Хайян и др. Трансформация межличностных отношений в эпоху микробизнеса и ее влияние на будущее микробизнеса [J]Modern Commerce and Industry 2021, 42(03):56). -58.

5. Чай Цяошань: 92 дня на рынок, Kuaishou на шаг впереди других [J].

6.SharmaA, Kumar M, AgarwalS. A complete survey on software architectural styles and pattems[9][J].Procedia Computer Science,2015,70:1628.

7.Dragoni N, Giallorenzo S, Lafuente A L, et al. Microservices: Yesterday, today, and tomorrow [G]// Present and Ulterior Software Engineering. Berlin: Springer, 2017: 195216.

8.Namiot D, Sneps-Sneppe M. On Micro-services Architecture[J]. international joural of openinformation technologies, 2014,2(9).

9.Jamshidi P, Pahl C, Mendonca N C, et al. Microservices: The Joumey So Far and ChallengesAhead[Jl.lEEE Software,2018,35(3):24-35.

10.Gray J.A conversation with Wemer Vogels{J]. Queue, 2006, 4(4):14-22

11.N.M. Josuttis, SOA in Practice: The Art of Distributed System Design[M]. O'Reily Media, Inc2007.