Семинар №9
Spring Cloud. Микросервисная архитектура

1. **Инструментарий:**

Урок
Презентация

---

2. **Цели семинара №9:**

- Освоить основные понятия и компоненты Spring Cloud в контексте микросервисной архитектуры.
- Понять принципы работы и взаимодействия микросервисов с использованием инструментов Spring Cloud.

По итогам семинара №9 слушатель должен **знать**:

- Что такое Spring Cloud и какие проблемы микросервисной архитектуры он помогает решить.
- Основные компоненты Spring Cloud, такие как Eureka, Hystrix, Zuul и их роль в микросервисной архитектуре.

По итогам семинара №9 слушатель должен **уметь**:

- Развертывать и настраивать основные компоненты Spring Cloud для организации микросервисной архитектуры.
- Реализовывать взаимодействие и управление микросервисами с помощью инструментов Spring Cloud.

---

3. **План Содержание:**

Этап урока	Тайминг, минуты	Формат
Введение, обзор темы	20	Модерирует преподаватель
Задание 1	80	Студенты выполняют, преподаватель помогает в решении проблем

Вопросы и обсуждение	20	Модерирует преподаватель
Длительность:	**120**	

4. Блок 1.

Тайминг:
Объяснение правил – 10 минут
Работа в команде – 30 минут

Задание:
Создайте микросервисную архитектуру с использованием Spring Cloud, включающую:

1. Сервис "Пользователи": Регистрация и управление профилями пользователей.
2. Сервис "Задачи": Добавление, удаление и просмотр задач.
3. API Gateway: Точка входа для всех запросов.
4. Используйте Eureka для обнаружения сервисов и Hystrix для обработки ошибок.

Пример решения:

1. Сервис "Пользователи"

```
@SpringBootApplication
@EnableEurekaClient
public class UserServiceApplication {
   public static void main(String[] args) {
      SpringApplication.run(UserServiceApplication.class, args);
   }
}

@RestController
@RequestMapping("/users")
public class UserController {
   @PostMapping("/register")
   public ResponseEntity<String> register(@RequestBody User user) {
      // Код регистрации пользователя
      return ResponseEntity.ok("Пользователь успешно зарегистрирован");
   }
```

```
}
```

2. Сервис "Задачи"

```
@SpringBootApplication
@EnableEurekaClient
public class TaskServiceApplication {
  public static void main(String[] args) {
    SpringApplication.run(TaskServiceApplication.class, args);
  }
}

@RestController
@RequestMapping("/tasks")
public class TaskController {
  @GetMapping
  public ResponseEntity<List<Task>> getTasks() {
    // Получение списка задач
    return ResponseEntity.ok(new ArrayList<>());
  }
}
```

3. API Gateway

```
@SpringBootApplication
@EnableEurekaClient
@EnableZuulProxy
public class GatewayServiceApplication {
  public static void main(String[] args) {
    SpringApplication.run(GatewayServiceApplication.class, args);
  }
}
```

4. Eureka Server

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {
  public static void main(String[] args) {
    SpringApplication.run(EurekaServerApplication.class, args);
  }
```

}

Часто встречающиеся ошибки:

1. Ошибки конфигурации Eureka: сервисы не могут зарегистрироваться или быть обнаруженными.
2. Ошибки в конфигурации API Gateway: некорректная маршрутизация запросов или отсутствие необходимых фильтров.
3. Проблемы с безопасностью: отсутствие защиты конечных точек API или неэффективное использование авторизации и аутентификации.
4. Неадекватная обработка ошибок: отсутствие глобальных обработчиков ошибок или некорректная настройка Hystrix для обработки сбоев сервиса.
5. Проблемы с зависимостями: использование устаревших или несовместимых версий библиотек и инструментов Spring Cloud.

5. **Домашнее задание**

Разработайте микросервисную архитектуру для онлайн-магазина электроники с использованием Spring Cloud. Структура должна включать:

1. Сервис "Товары": Управление каталогом товаров (добавление, удаление, просмотр).
2. Сервис "Корзина": Добавление товаров в корзину, удаление товаров из корзины и оформление заказа.
3. Сервис "Отзывы": Добавление и просмотр отзывов на товары.
4. API Gateway: Централизованный вход для обработки всех запросов.
5. Используйте Eureka для обнаружения сервисов и Hystrix для обработки ошибок и отказоустойчивости.

Пример решения:

1. Сервис "Товары"

```
@SpringBootApplication
@EnableEurekaClient
public class ProductServiceApplication {
   public static void main(String[] args) {
      SpringApplication.run(ProductServiceApplication.class, args);
   }
```

```
}

@RestController
@RequestMapping("/products")
public class ProductController {
  @GetMapping
  public ResponseEntity<List<Product>> listProducts() {
    // Возврат списка товаров
    return ResponseEntity.ok(new ArrayList<>());
  }
}
```

2. Сервис "Корзина"

```
@SpringBootApplication
@EnableEurekaClient
public class CartServiceApplication {
  public static void main(String[] args) {
    SpringApplication.run(CartServiceApplication.class, args);
  }
}

@RestController
@RequestMapping("/cart")
public class CartController {
  @PostMapping
  public ResponseEntity<Cart> addItem(@RequestBody CartItem item) {
    // Добавление товара в корзину
    return ResponseEntity.ok(new Cart());
  }
}
```

3. Сервис "Отзывы"

```
@SpringBootApplication
@EnableEurekaClient
public class ReviewServiceApplication {
  public static void main(String[] args) {
    SpringApplication.run(ReviewServiceApplication.class, args);
  }
}

@RestController
```

```
@RequestMapping("/reviews")
public class ReviewController {
  @GetMapping("/{productId}")
  public ResponseEntity<List<Review>> getReviews(@PathVariable String
productId) {
    // Возврат отзывов для товара
    return ResponseEntity.ok(new ArrayList<>());
  }
}
```

4. API Gateway

```
@SpringBootApplication
@EnableEurekaClient
@EnableZuulProxy
public class GatewayServiceApplication {
  public static void main(String[] args) {
    SpringApplication.run(GatewayServiceApplication.class, args);
  }
}
```

Рекомендации для преподавателей по оценке задания:

1. Проверка на понимание микросервисной архитектуры: Убедитесь, что каждый сервис независим и выполняет свою специфическую функцию.
2. Функциональность: Проверьте, работает ли каждый сервис корректно. Например, можно ли добавлять и удалять товары, а также оставлять отзывы?
3. Интеграция с Eureka и Hystrix: Убедитесь, что сервисы корректно регистрируются в Eureka и что Hystrix обеспечивает отказоустойчивость при потенциальных сбоях.
4. Безопасность: Проверьте, что конечные точки API защищены, и рассмотрите вопросы авторизации и аутентификации, если они применимы к заданию.
5. Общее качество кода: Проверьте код на наличие ясной структуры, комментариев и следование принципам чистого кода.