Университет ИТМО

Факультет программной инженерии и компьютерной техники

Сервис-ориентированная архитектура

Лабораторная работа №4

Вариант 3007

Группа: P34122

Студент: Данков Антон Игоревич

Преподаватель: Усков Иван Владимирович

Санкт-Петербург

2021

Оглавление

Задание:

Переработать сервисы из лабораторной работы #3 следующим образом:

- Второй ("вызывающий") сервис переписать в соответствии с требованиями протокола SOAP.
- Развернуть переработанный сервис на сервере приложений по собственному выбору.
- Оставшийся сервис не модифицировать, не менять его API, протокол и используемый сервер приложений.
- Установить и сконфигурировать на сервере Helios программное обеспечение Mule ESB.
- Настроить интеграцию двух сервисов с использованием установленного программного обеспечения.
- Реализовать дополнительную REST-"прослойку", обеспечивающую возможность доступа к переработанному сервису клиентского приложения без необходимости его модификации. Никакой дополнительной логики, помимо вызовов SOAP-сервиса, разработанная REST-прослойка содержать не должна.

Код

https://github.com/Antondstd/Service-oriented-architecture/tree/master/Lab3

Xml схема с описанием домена

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.itmo.ru/springsoap/generated"
        targetNamespace="http://www.itmo.ru/springsoap/generated" elementFormDefault="qualified">


   <xs:element name="makeVipAndDoubleCostRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="id" type="xs:long"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="makeVipAndDoubleCostResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="ticket" type="tns:ticketDTO"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="cancelEventRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="eventId" type="xs:long"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="cancelEventResponse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="id" type="xs:long"/>
         </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>


  <xs:complexType name="ticketDTO">
    <xs:sequence>
      <xs:element name="creationDate" type="xs:string" minOccurs="0"/>
      <xs:element name="name" type="xs:string" minOccurs="0"/>
      <xs:element name="coordinates" type="tns:coordinates"/>
      <xs:element name="price" type="xs:long" minOccurs="0"/>
      <xs:element name="discount" type="xs:int" minOccurs="0"/>
      <xs:element name="comment" type="xs:string" minOccurs="0"/>
      <xs:element name="type" type="tns:ticketType"/>
      <xs:element name="event" type="tns:eventDTO"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:long" use="required"/>
  </xs:complexType>

  <xs:complexType name="coordinates">
    <xs:sequence>
      <xs:element name="x" type="xs:int"/>
      <xs:element name="y" type="xs:long"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:long" use="required"/>
  </xs:complexType>

  <xs:complexType name="eventDTO">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="date" type="xs:string"/>
      <xs:element name="minAge" type="xs:int"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:long" use="required"/>
  </xs:complexType>

  <xs:simpleType name="ticketType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="VIP"/>
      <xs:enumeration value="USUAL"/>
      <xs:enumeration value="BUDGETARY"/>
      <xs:enumeration value="CHEAP"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Описание Enpoint

```
@Endpoint
class TicketEndpoint {

  @Autowired
  lateinit var ticketService: TicketService

  @PayloadRoot(namespace = "http://www.itmo.ru/springsoap/generated", localPart = "makeVipAndDoubleCostRequest")
  @ResponsePayload
  fun saleVip(@RequestPayload request: MakeVipAndDoubleCostRequest): MakeVipAndDoubleCostResponse {
    val ticket = ticketService.makeVipAndDoubleCost(request.id)
    val response = MakeVipAndDoubleCostResponse()
    response.ticket = TicketDTO().apply { fromTicket(ticket) }
    return response
  }

  @PayloadRoot(namespace = "http://www.itmo.ru/springsoap/generated", localPart = "cancelEventRequest")
  @ResponsePayload
  fun cancelEvent(@RequestPayload request: CancelEventRequest): CancelEventResponse {
    ticketService.cancelEvent(request.eventId)
```

```
        return CancelEventResponse().apply { id = request.eventId }
    }
}
    @PayloadRoot(namespace = "http://www.itmo.ru/springsoap/generated", localPart = "cancelEventRequest")
    @ResponsePayload
    fun cancelEvent(@RequestPayload request: CancelEventRequest):CancelEventResponse{
        ticketService.cancelEvent(request.eventId)
        return CancelEventResponse().apply { id =  request.eventId}
    }
}
```

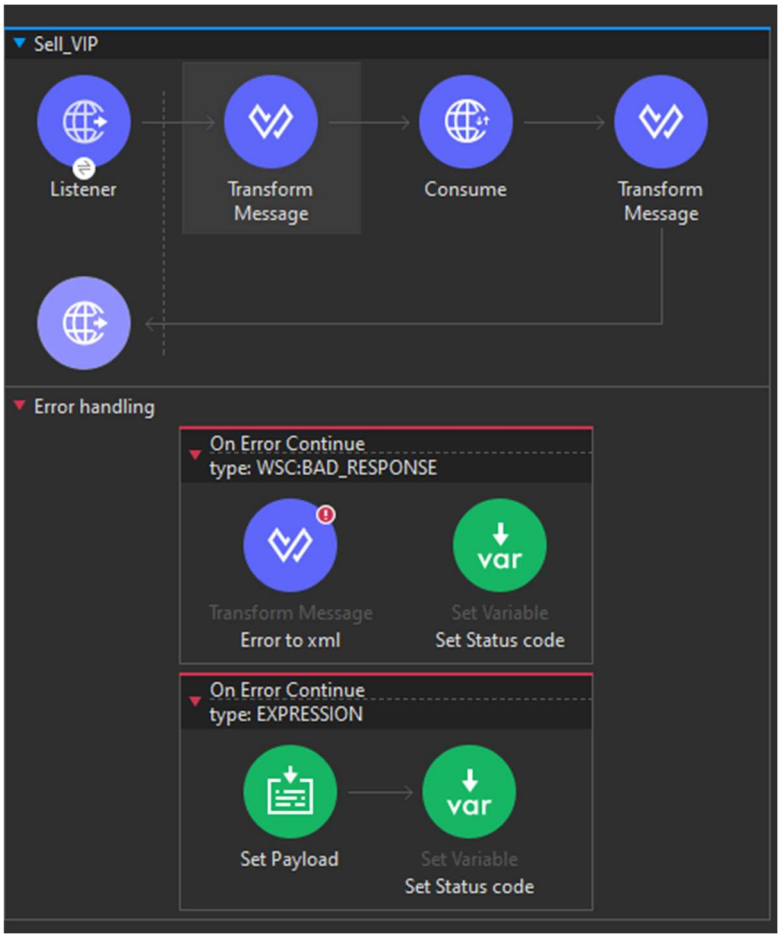
Обработчик ошибок для SOAP

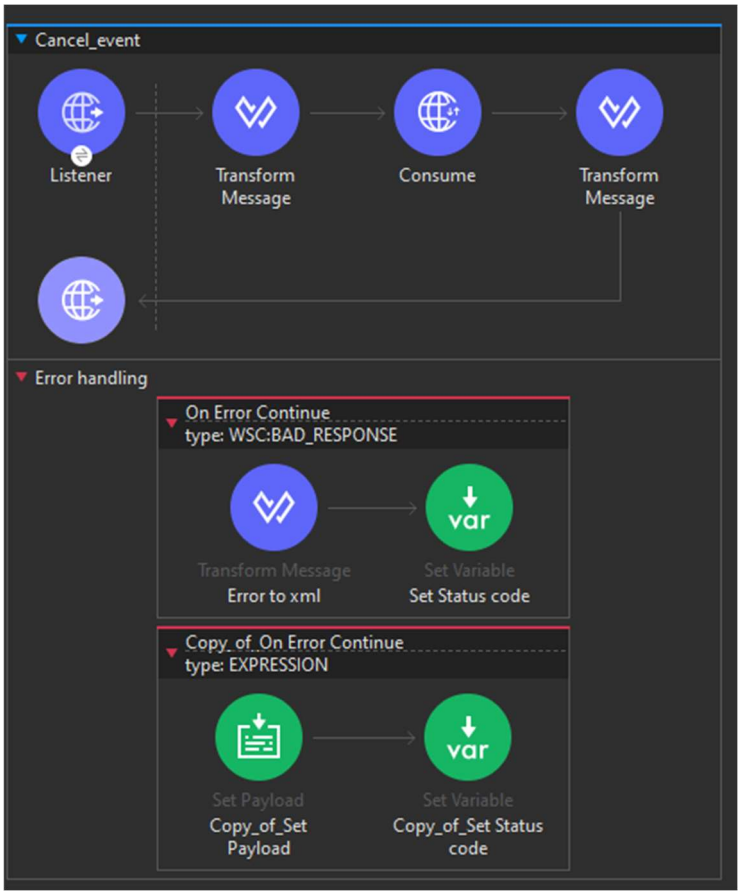
```
@Component
class CustomExceptionResolver: AbstractEndpointExceptionResolver() {

    override fun resolveExceptionInternal(
        messageContext: MessageContext,
        endpoint: Any?,
        exception: Exception?
    ): Boolean {
        if (exception is NotFoundException || exception is BadRequestException) {
            val response = messageContext.getResponse() as SoapMessage
            val soapBody = response.soapBody
            val soapFault: SoapFault = soapBody.addClientOrSenderFault(
                exception.javaClass.name,
                Locale.ENGLISH
            )
            val detail = soapFault.addFaultDetail()
            detail.addFaultDetailElement(QName("code")).addText(if (exception is NotFoundException) "404" else "400")
            detail.addFaultDetailElement(QName("message")).addText(exception.message)
            return true
        }
        return false
    }
}
```

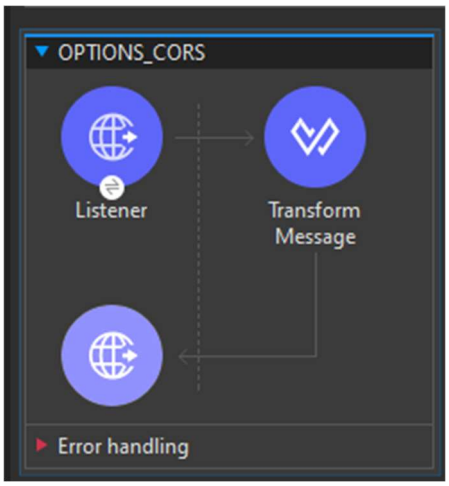
Настройка Spring-WS

```
@EnableWs
@Configuration
class WebServiceConfig : WsConfigurerAdapter() { // bean definitions
    @Bean
    fun messageDispatcherServlet(applicationContext: ApplicationContext): ServletRegistrationBean<*> {
        val servlet1: MessageDispatcherServlet = MessageDispatcherServlet()
        servlet1.setApplicationContext(applicationContext)
        servlet1.isTransformWsdlLocations = true
        return ServletRegistrationBean<MessageDispatcherServlet>(servlet1,*arrayOf("/ws/*"))
    }

    @Bean(name = ["tickets"])
    fun defaultWsdl11Definition(ticketsSchema: XsdSchema?): DefaultWsdl11Definition? {
        val wsdl11Definition = DefaultWsdl11Definition()
        wsdl11Definition.setPortTypeName("TicketsPort")
        wsdl11Definition.setLocationUri("/ws")
        wsdl11Definition.setTargetNamespace("http://www.itmo.ru/springsoap/generated")
        wsdl11Definition.setSchema(ticketsSchema)
        return wsdl11Definition
    }

    @Bean
    fun ticketsSchema(): XsdSchema? {
        return SimpleXsdSchema(ClassPathResource("tickets.xsd"))
    }
}
```

Настройка ESB Mule







Вывод:

В Spring-WS используется подход contact-first, изначально создается xml файл, по которму после генерируются классы и пишет endpoint-ы, из-за чего переписывать уже готовый REST сервис на SOAP не оптимальный вариант. При этом подходе предполагается что архитектура изначально написана в xml и уже после пишется код. Также такой подход лучше позволяется сохранить консистентность wsdl (изменение в коде не приведет к изменению описания сервиса).

В ESB Mule была проблема с получением дополнительной информации из объекта ошибки. Лишь через дебаг удалось найти поле, в котором оно содержится, сама ide его не подсказывала, и в самом коде указывает что поля exception у объекта error нету.