Университет ИТМО

Факультет программной инженерии и компьютерной техники

Сервис-ориентированная архитектура

Лабораторная работа №3

Вариант 3007.6.03

Группа: P34122

Студент: Данков Антон Игоревич

Преподаватель: Усков Иван Владимирович

Санкт-Петербург

2021

Задание:

Переработать веб-сервисы из лабораторной работы #2 таким образом, чтобы они реализовывали основные концепции микросервисной архитектуры. Для этого внести в оба сервиса -- "вызываемый" (из лабораторной работы #1) и "вызывающий" (добавленный в лабораторной работе #2) перечисленные ниже изменения.

Изменения в "вызываемом" сервисе:

- Разделить приложение на два модуля -- веб-приложение с веб-сервисом и EJB-jar с бизнес-компонентами.
- Переместить всю логику из класса сервиса в Stateless EJB. В классе сервиса оставить только обращение к методам бизнес-интерфейса. EJB-компонент должен быть доступен удалённо (иметь Remote-интерфейс).
- Сформировать на уровне сервера приложений пул компонентов EJB настраиваемой мощности, динамически расширяемый при увеличении нагрузки.
- Установить ПО Consul и настроить Service Discovery с его помощью. Сервис должен регистрироваться в Service Discovery в момент запуска.

Изменения в "вызывающем" сервисе:

- Сконфигурировать окружение для работы сервиса на платформе Spring Boot.
- Запустить второй экземпляр сервиса на другом порту. Реализовать балансировку нагрузки между экземплярами с помощью Haproxy.

Код

https://github.com/Antondstd/Service-oriented-architecture/tree/master/Lab2

Добавление сервиса в Consul и поддержание его в активном состоянии

```
@Singleton
@Startup
open class ConsulUtil {
    private var agentClient: AgentClient? = null
    private var service_id = "1"
    private var port:Int = 8181
    private var name:String? = null
    private var ttl:Long = 3

    @PostConstruct
    fun register(){
        val classLoader = Thread.currentThread().contextClassLoader
        val input: InputStream = classLoader.getResourceAsStream("consul.properties")!!
        val properties = Properties()
        properties.load(input)
        port = properties.getProperty("consul.port").toInt()
        name = properties.getProperty("consul.name")
        service_id = properties.getProperty("consul.service_id")
        ttl = properties.getProperty("consul.ttl").toLong()

        try{
            val consul = Consul.builder().build()
            agentClient = consul!!.agentClient()
            agentClient!!.register(ImmutableRegistration.builder()
                .id(service_id)
                .name("soa-lab3-main")
                .port(8181)
```

```
            .check(Registration.RegCheck.ttl(ttl))
            .build())
        println("CONSUL REGISTERED!!!")
      }
      catch (e:Exception){
        println("Error trying to get Consul")
      }
    }
    @Schedule(hour = "*", minute = "*", second = "*/20")
    fun checkIn() {
      agentClient?.pass(service_id)
    }
}
```

Получение адреса сервиса через Consul

```
@Value("\${consul.main-app-name}")
lateinit var mainAppName: String

@Autowired
lateinit var discoveryClient: DiscoveryClient

 private var serviceInstance:ServiceInstance? = null

fun getServiceInstance(): ServiceInstance {
    return (discoveryClient!!.getInstances(mainAppName)
        .stream()
        .findFirst()).get()
}

fun urlApiService():String{
    if (serviceInstance == null){
        serviceInstance = getServiceInstance()
    }
    return "https://${serviceInstance!!.host}:${serviceInstance!!.port}/api"
}
```

Настройка пула EJB

```
<glassfish-ejb-jar>
<enterprise-beans>
    <ejb>
        <ejb-name>TicketService</ejb-name>
        <bean-pool>
            <max-pool-size>20</max-pool-size>
            <max-wait-time-in-millis>0</max-wait-time-in-millis>
            <steady-pool-size>1</steady-pool-size>
<!--            <pool-idle-timeout-in-seconds>5</pool-idle-timeout-in-seconds>-->
<!--            <disable-nonportable-jndi-names>true</disable-nonportable-jndi-names>-->
        </bean-pool>
    </ejb>
</enterprise-beans>
</glassfish-ejb-jar>
```

Вызов Remote EJB

```
class RemoteBeanUtil {
    companion object{
        fun lookupRemoteStatelessBean(): TicketServiceInterface {
            val jndiProperties = Hashtable<String, String>()
            jndiProperties[javax.naming.Context.URL_PKG_PREFIXES] = "org.jboss.ejb.client.naming"
            val contextProperties = Properties()
            contextProperties.setProperty(
                Context.INITIAL_CONTEXT_FACTORY,
                "com.sun.enterprise.naming.SerialInitContextFactory"
```

```
        )
        return try {
            val context = InitialContext(contextProperties)
            val appName = "global"
            val moduleName = "soa_lab3-remote-ejb"
            val beanName = "TicketService"
            val viewClassName: String = TicketServiceInterface::class.java.getName()
            val lookupName = "java:$appName/$moduleName/$beanName"
//              val lookupName = "java:global/soa_lab3-remote-ejb/TicketService"
            context.lookup(lookupName) as TicketServiceInterface
        } catch (e: NamingException) {
            object : TicketServiceInterface {
                override fun getTicketsSortPaging(
                    page: Int,
                    perPage: Int,
                    sortStateList: List<String>?,
                    filterMap: Map<String, MutableList<String>>
                ): ResponsePagesTickets? {
                    throw EjbNotAvailableException("Не удалось получить доступ к EJB TicketService")
                }

                override fun addTicketFromXml(xml: String?): Ticket? {
                    throw EjbNotAvailableException("Не удалось получить доступ к EJB TicketService")
                }

                override fun updateTicketFromXml(xml: String?, id: Long) {
                    throw EjbNotAvailableException("Не удалось получить доступ к EJB TicketService")
                }

                override fun deleteTicket(ticketId: Long) {
                    throw EjbNotAvailableException("Не удалось получить доступ к EJB TicketService")
                }

                override fun getGroupedByDiscount(): MutableList<ResponseGroupedDiscount>? {
                    throw EjbNotAvailableException("Не удалось получить доступ к EJB TicketService")
                }

                override fun getDistinctTypes(): MutableList<Any>? {
                    throw EjbNotAvailableException("Не удалось получить доступ к EJB TicketService")
                }

                override fun deleteSomeTicketByType(type: TicketType): Long {
                    throw EjbNotAvailableException("Не удалось получить доступ к EJB TicketService")
                }

                override fun getTicket(id: Long): Ticket? {
                    throw EjbNotAvailableException("Не удалось получить доступ к EJB TicketService")
                }
            }
        }
    }
  }
}
```

Обработка ошибок Remote EJB

```
@Provider
class EJBExceptionHandler : ExceptionMapper<EJBException> {
    override fun toResponse(exception: EJBException): Response {
        val nestedException = exception.cause!!.cause!!.cause!!
        if (nestedException is BadRequestException || nestedException is UnprocessableEntityException)
            return Response.status(400).entity(nestedException.message).type("text/plain").build()
        else
            return Response.status(404).header("Content-Type", "text/xml; charset=UTF-16LE")
                .entity(nestedException.message).type("text/plain").build()
```

```
    }
}
```

Настройка Haproxy

sudo vi /etc/haproxy/haproxy.cfg

```
frontend myfrontend
  bind 127.0.0.1:8443
  option tcplog
  mode tcp
  default_backend myservers

backend myservers
  mode tcp
  option ssl-hello-chk
  server server1 127.0.0.1:8585 check
  server server2 127.0.0.1:8584 check
```

Вывод:

В ходе выполнения данной лабораторной работы настроил регистрацию сервиса в Consul и получение из него информации другими сервисами. Переработал код и вынес функции взаимодействия в отдельный EJB, который взаимодействует с базой данных и вызывается через JNDI. Установил на WSL Ubuntu и Haproxy, сконфигурировав его на перенаправление вызыывающих сервисов через режим tcp.