**Лабораторная работа №5**

**Тема:** Сервисная шина предприятия

**Задание:** Подключите реализованное приложение к сервисной шине распределённой системы «Деканат» при помощи адаптера

**Теоретическое введение**

Интеграция – процесс создания взаимосвязей между независимыми информационными системами для обеспечения эффективного, надёжного и безопасного обмена данными. Распределённая система – это набор независимых компьютеров, представляющийся их пользователям единой объединённой системой. Из определения распределённой системы следует, что её характеризуют два основных момента:

1. Все сервисы, входящие в распределённую систему, автономны.
2. Для работы пользователя распределённая система должна быть представлена как единая система.

Существует два основных подхода к построению распределённых систем: с использованием SOA и с использованием микросервисной архитектуры (рисунок 1).

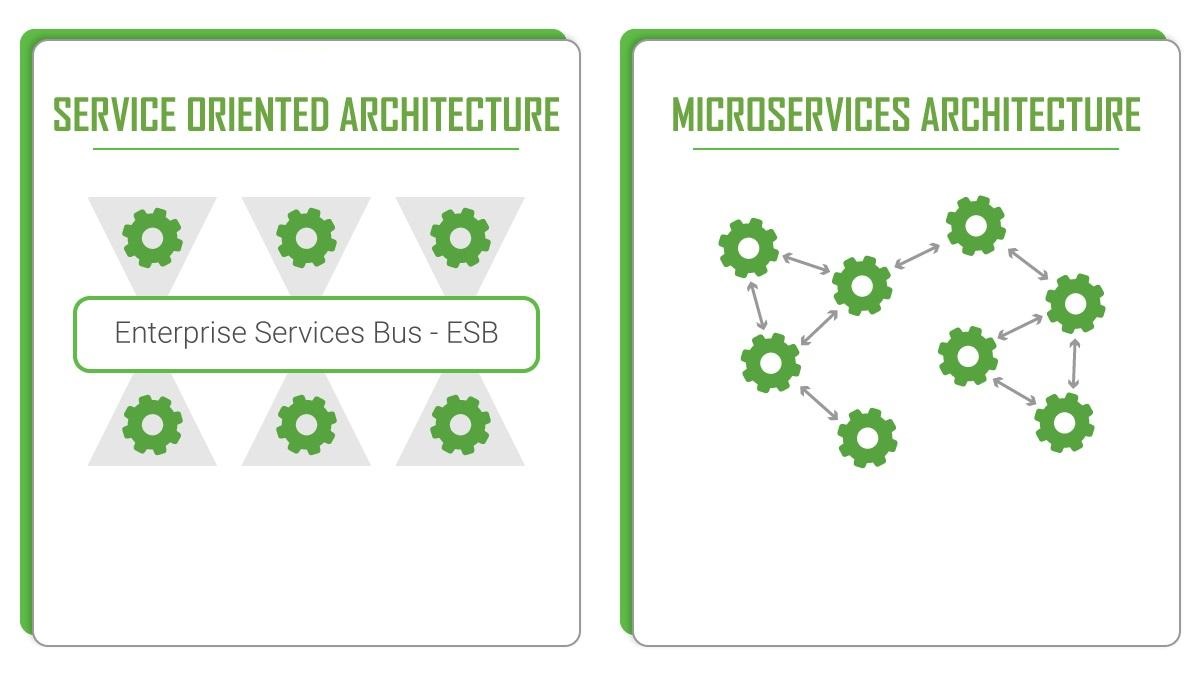


Рисунок 1. Сервис-ориентированная и микросервисная архитектура

Основным отличием этих подходов друг от друга является наличие сервисной шины предприятия (ESB). Если в сервис-ориентированной архитектуре взаимодействие компонентов распределённой информационной системы происходит только через общую шину, то в микросервисной архитектуре они самостоятельно взаимодействуют между собой. Вне зависимости от выбранного подхода наиболее важной частью создания распределённой системы является определение протоколов и механизмов взаимодействия подсистем между собой или с сервисной шиной, которую также можно рассматривать как одну из подсистем. А в дальнейшем – строгое соблюдение и поддержка выбранных протоколов на протяжении жизненного цикла как каждого компонента в отдельности, так и всей системы целиком.

**Ход работы**

1. Чтобы подключить серверное приложение к шине, необходимо установить адаптер в локальный Maven-репозиторий. Для этого откройте Sourcetree и выберите подпункт Clone / New пункта меню File.
2. В появившемся окне введите в строку Source Path / URL адрес:

https://github.com/Progruzovik/service-bus.git

1. В строке Destination Path Выберите любую подходящую папку, в которую будет склонирован репозиторий.
2. Нажмите на кнопку Advanced Options и в выпадающем списке Checkout branch выберите пункт feature/dean.
3. Нажмите на кнопку Clone.
4. Откройте склонированный проект в IntelliJ IDEA. Откройте панель Maven на правой панели, перейдите в подраздел Lifecycle раздела  
   bus-parent, выделите пункты clean и install и нажмите на зелёную кнопку запуска сборки проекта (рисунок 2).

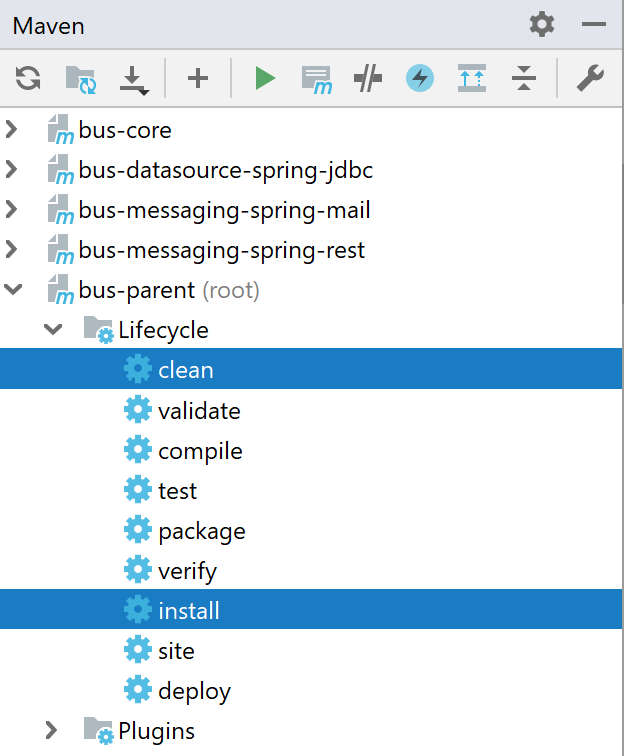


Рисунок 2

1. Дождитесь завершения сборки адаптера сервисной шины, затем откройте проект с серверным приложением, разработанным в третьей лабораторной работе, и добавьте в него адаптер в качестве зависимости. Для этого в раздел dependencies файла pom.xml добавьте:

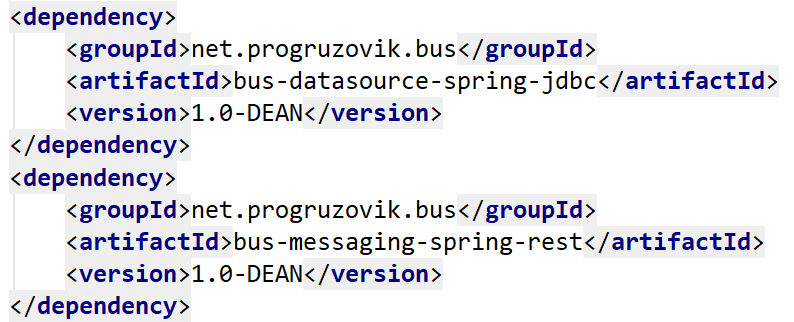


Рисунок 3. Добавление адаптера в проект

1. Откройте файл application.properties в папке resources и переключите СУБД в режим работы в оперативной памяти:

spring.datasource.url=jdbc:h2:mem:

1. Создайте файл schema.sql в папке resources. Скопируйте в него скрипт создания базы данных, подготовленный во второй лабораторной работе.
2. Добавьте в скрипт создание новой таблицы под названием student\_local. Добавьте скрипт заполнения этой таблицы теми же данными, которыми ранее заполнялась таблица student. Заполнение исходной таблицы student уберите, она должна оставаться пустой.
3. Создайте файл bus.properties в папке resources и заполните его следующими значениями:

bus.instance.address=номер\_бригады  
bus.dean.address=dean  
bus.integration.url=http://up-lab1.mirea.ru/bus

1. Добавьте в класс StudentJdbc метод для получения всех данных из таблицы student\_local (рисунок 4).

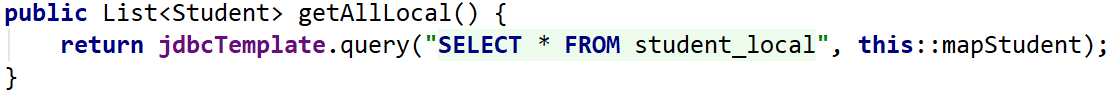


Рисунок 4

1. Создайте в проекте пакет config и добавьте в него два новых класса: BusConfig и BusInitilializer. Листинг классов представлен в приложениях 1 и 2.
2. Создайте пакет service и добавьте в него класс MessageChecker. Листинг класса представлен в приложении 3.
3. Запустите приложение.
4. Перейдите по адресу http://up-lab2.mirea.ru/student/all и убедитесь, что в списке появились данные студентов бригады. Возможна ситуация, когда новые данные не появились, а в списке уже присутствуют студенты другой бригады с аналогичными id. Это означает, что была нарушена консистентность данных, и необходимо изменить id студентов бригады на уникальные в пределах всей распределённой системы «Деканат».

Приложение 1. Листинг класса BusConfig

@Configuration  
@EnableScheduling  
@PropertySource("classpath:bus.properties")  
public class BusConfig {  
  
 @Value("${bus.instance.address}")  
 private String address;  
  
 @Value("${bus.dean.address}")  
 private String deanAddress;  
  
 @Value("${bus.integration.url}")  
 private String integrationUrl;  
  
 @Bean  
 public Replicator replicator(ObjectMapper mapper, Writer writer, InstanceDao instanceDao, EntityDao entityDao) {  
 return new ReplicationService(mapper, writer, instanceDao, entityDao);  
 }  
  
 @Bean  
 public RestReceiver restReceiver(RestTemplate restTemplate, BusHandler busHandler) {  
 return new BusRestReceiver(integrationUrl, restTemplate, busHandler);  
 }  
  
 @Bean  
 public BusHandler busHandler(ObjectMapper mapper, Writer writer, InstanceDao instanceDao, EntityDao entityDao) {  
 return new MessageHandler(mapper, writer, instanceDao, entityDao, deanAddress);  
 }  
  
 @Bean  
 public Writer writer(ObjectMapper mapper, MessageSender messageSender, InstanceDao instanceDao) {  
 return new MessageWriter(mapper, messageSender, instanceDao);  
 }  
  
 @Bean  
 public RestTemplate restTemplate() {  
 return new RestTemplate();  
 }  
  
 @Bean  
 public MessageSender messageSender(RestTemplate restTemplate) {  
 return new RestSender(address, integrationUrl, restTemplate);  
 }  
  
 @Bean  
 public EntityNameConverter entityNameConverter() {  
 return new CharBufferNameConverter();  
 }  
  
 @Bean  
 public InstanceDao instanceDao(EntityNameConverter entityNameConverter, JdbcTemplate jdbcTemplate) {  
 return new InstanceJdbc(entityNameConverter, jdbcTemplate);  
 }  
  
 @Bean  
 public EntityDao entityDao(EntityNameConverter entityNameConverter, JdbcTemplate jdbcTemplate) {  
 return new EntityJdbc(entityNameConverter, jdbcTemplate);  
 }  
}

Приложение 2. Листинг класса BusInitializer

@Component  
@DependsOn("busHandler")  
public class BusInitializer {  
  
 private final Replicator replicator;  
 private final StudentJdbc studentJdbc;  
  
 public BusInitializer(Replicator replicator, StudentJdbc studentJdbc) {  
 this.replicator = replicator;  
 this.studentJdbc = studentJdbc;  
 }  
  
 @PostConstruct  
 public void init() throws IOException {  
 initEntity("student", studentJdbc.getAllLocal());  
 }  
  
 private <T> void initEntity(String name, List<T> data) throws IOException {  
 replicator.initializeEntity(name);  
 for (T row : data) {  
 replicator.addRow(name, row);  
 }  
 }  
}

Приложение 3. Листинг класса MessageChecker

@Service  
public class MessageChecker {  
  
 private volatile Instant lastCheckTime = null;  
 private final RestReceiver restReceiver;  
  
 public MessageChecker(RestReceiver restReceiver) {  
 this.restReceiver = restReceiver;  
 }  
  
 @Scheduled(fixedDelay = 3000)  
 public synchronized void checkMessages() throws IOException {  
 lastCheckTime = restReceiver.receiveMessages(lastCheckTime);  
 }  
}