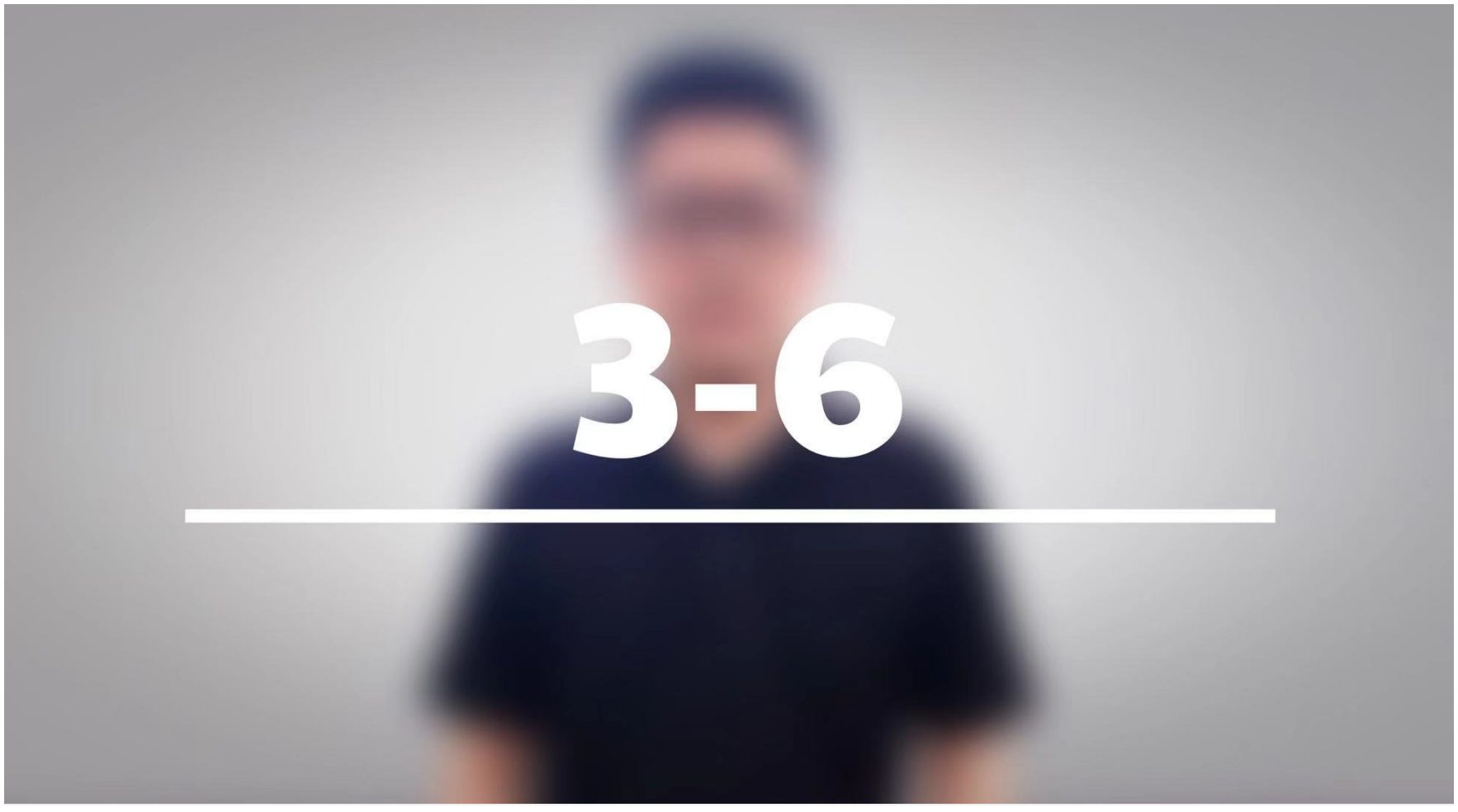


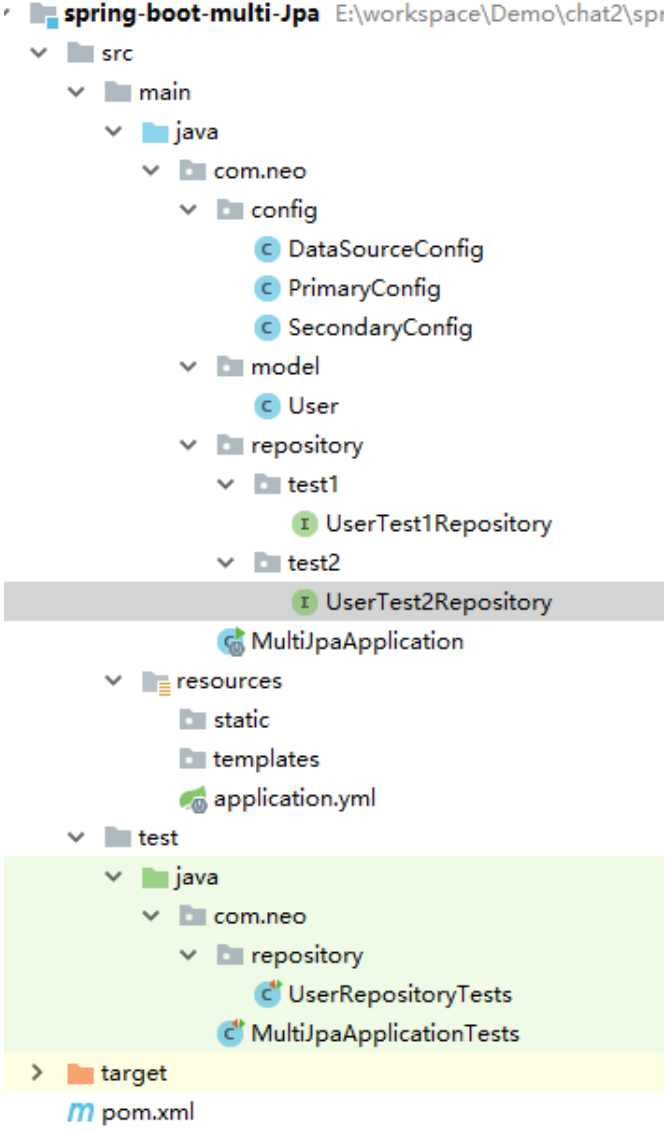
第 3-6 课：Spring Data JPA 多数据源的使用

项目中使用多个数据源在以往工作中比较常见，微服务架构中不建议一个项目使用多个数据源。在微服务架构下，一个微服务拥有自己独立的一个数据库，如果此微服务要使用其他数据库的数据，需要调用对应库的微服务接口来调用，而不是在一个项目中连接使用多个数据库，这样微服务更独立、更容易水平扩展。

虽然在微服务架构下，不提倡一个项目拥有多个数据源，但在 Spring Boot 体系中，项目实现多数据源调用却是一件很容易的事情，本节课将介绍 Spring Data JPA 多数据源的使用。

Spring Data JPA 使用多数据源的整体思路是，配置不同的数据源，在启动时分别加载多个数据源配置，并且注入到不同的 repository 中。这样不同的 repository 包就有不同的数据源，使用时注入对应包下的 repository，就会使用对应数据源的操作。

对照前两课的示例项目，本课内容将会对项目结构有所调整，如下：



其中：

- config 启动时加载、配置多数据源；
- model 存放数据操作的实体类；
- repository 目录下有两个包路径 test1 和 test2 ， 分别代表两个不同数据源下的仓库，这两个包下的 repository 可以相同也可以不同。

下面演示一下项目。

多数据源的支持

配置 Spring Data JPA 对多数据源的使用，一般分为以下几步：

- 创建数据库 test1 和 test2
- 配置多数据源
- 不同源的 repository 放入不同包路径
- 声明不同的包路径下使用不同的数据源、事务支持
- 不同的包路径下创建对应的 repository
- 测试使用

上面的一些步骤我们在前面两课中已经讲过了，这里只补充不同的内容。

配置两个数据源：

```
spring.datasource.primary.jdbc-url=jdbc:mysql://local-
host:3306/test1?serverTimezone=UTC&useUnicode=true&characterEncod-
ing=utf-8&useSSL=true
spring.datasource.primary.username=root
spring.datasource.primary.password=root
spring.datasource.primary.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.secondary.jdbc-url=jdbc:mysql://local-
host:3306/test2?serverTimezone=UTC&useUnicode=true&characterEncod-
ing=utf-8&useSSL=true
spring.datasource.secondary.username=root
spring.datasource.secondary.password=root
spring.datasource.secondary.driver-class-name=com.mysql.cj.jdbc.Dri-
ver

#SQL 输出
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.hbm2ddl.auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.My-
SQL5InnoDBDialect
#format 一下 SQL 进行输出
spring.jpa.properties.hibernate.format_sql=true
```

设置将项目中的 SQL 格式化后打印出来，方便在开发过程中调试跟踪。

创建 DataSourceConfig 添加 @Configuration 注解，在项目启动时运行初始化数据库资源。

```
@Configuration
public class DataSourceConfig {
}
```

在 DataSourceConfig 类中加载配置文件，利用 ConfigurationProperties 自动装配的特性加载两个数据源。

加载第一个数据源，数据源配置以 spring.datasource.primary 开头，注意当有多个数据源时，需要将其中一个标注为 @Primary，作为默认的数据源使用。

```
@Bean(name = "primaryDataSource")
@Primary
@ConfigurationProperties("spring.datasource.primary")
public DataSource firstDataSource() {
    return DataSourceBuilder.create().build();
}
```

加载第二个数据源，数据源配置以 spring.datasource.secondary 为开头。

```
@Bean(name = "secondaryDataSource")
@ConfigurationProperties("spring.datasource.secondary")
public DataSource secondDataSource() {
    return DataSourceBuilder.create().build();
}
```

加载 JPA 的相关配置信息，JpaProperties 是 JPA 的一些属性配置信息，构建 Lo - calEntityManagerFactoryBean 需要参数信息注入到方法中。

```
@Autowired
private JpaProperties jpaProperties;
@Autowired
private HibernateProperties hibernateProperties;

@Bean(name = "vendorProperties")
public Map<String, Object> getVendorProperties() {
    return hibernateProperties.determineHibernateProperties(jpa-
Properties.getProperties(), new HibernateSettings());
}
```

第一个数据源的加载配置过程

首先来看第一个数据源的加载配置过程，创建 PrimaryConfig 类，将上面创建好的第一个数据源注入到类中，添加 @Configuration 和 @EnableTransactionManage - ment 注解，第一个代表启动时加载，第二个注解表示启用事务，同时将第一个数据源和 JPA 配置信息注入到类中。

```
@Configuration
@EnableTransactionManagement
public class PrimaryConfig {
    @Autowired
    @Qualifier("primaryDataSource")
    private DataSource primaryDataSource;
    @Autowired
    @Qualifier("vendorProperties")
    private Map<String, Object> vendorProperties;
}
```

LocalEntityManagerFactoryBean 负责创建一个适合于仅使用 JPA 进行数据访问的环境的 EntityManager，构建的时候需要指明提示实体类的包路径、数据源和 JPA 配置信息。

```
@Bean(name = "entityManagerFactoryPrimary")
@Primary
public LocalContainerEntityManagerFactoryBean entityManagerFacto-
ryPrimary (EntityManagerFactoryBuilder builder) {
    return builder
            .dataSource(primaryDataSource)
            .properties(vendorProperties)
            .packages("com.neo.model") //设置实体类所在位置
            .persistenceUnit("primaryPersistenceUnit")
            .build();
}
```

利用上面的 entityManagerFactoryPrimary() 方法构建好最终的 EntityManager。

```
@Bean(name = "entityManagerPrimary")
@Primary
public EntityManager entityManager(EntityManagerFactory-
Builder builder) {
    return entityManagerFactoryPrimary(builder).getObject().cre-
ateEntityManager();
}
```

EntityManager 是 JPA 中用于增、删、改、查的接口，它的作用相当于一座桥梁，连接内存中的 Java 对象和数据库的数据存储。使用 EntityManager 中的相关接口对数据库实体进行操作的时候， EntityManager 会跟踪实体对象的状态，并决定在特定时刻将对实体的操作映射到数据库操作上面。

同时给数据源添加上 JPA 事务。

```
@Bean(name = "transactionManagerPrimary")
@Primary
PlatformTransactionManager transactionManagerPrimary(EntityManager-
FactoryBuilder builder) {
    return new JpaTransactionManager(entityManagerFactoryPrima-
ry(builder).getObject());
}
```

最后一步最为关键，将我们在类中配置好的 EntityManager 和事务信息注入到对应数据源的 repository 目录下，这样此目录下的 repository 就会拥有对应数据源和事务的信息。

```
@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(
        entityManagerFactoryRef="entityManagerFactoryPrimary",
        transactionManagerRef="transactionManagerPrimary",
        basePackages= { "com.neo.repository.test1" })//设置dao（repo
）所在位置
    public class PrimaryConfig {}
```

其中，basePackages 支持设置多个包路径，例如，`basePackages= { "com.-neo.repository.test1","com.neo.repository.test3" }`

到此第一个数据源配置完成了。

第二个数据源的加载配置过程

第二个数据源配置和第一个数据源配置类似，只是方法上去掉了注解：@Primary，第二个数据源数据源加载配置类 SecondaryConfig 完整代码如下：

```
@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(
        entityManagerFactoryRef="entityManagerFactorySecondary",
        transactionManagerRef="transactionManagerSecondary",
        basePackages= { "com.neo.repository.test2" })
public class SecondaryConfig {
    @Autowired
    @Qualifier("secondaryDataSource")
    private DataSource secondaryDataSource;

    @Autowired
    @Qualifier("vendorProperties")
    private Map<String, Object> vendorProperties;

    @Bean(name = "entityManagerFactorySecondary")
    public LocalContainerEntityManagerFactoryBean entityManagerFac-
torySecondary (EntityManagerFactoryBuilder builder) {
        return builder
                .dataSource(secondaryDataSource)
                .properties(vendorProperties)
                .packages("com.neo.model")
                .persistenceUnit("secondaryPersistenceUnit")
                .build();
    }

    @Bean(name = "entityManagerSecondary")
    public EntityManager entityManager(EntityManagerFactory-
Builder builder) {
        return entityManagerFactorySecondary(builder).getOb-
ject().createEntityManager();
    }

    @Bean(name = "transactionManagerSecondary")
    PlatformTransactionManager transactionManagerSecondary(Entity-
ManagerFactoryBuilder builder) {
        return new JpaTransactionManager(entityManagerFactorySecon-
dary(builder).getObject());
    }
}
```

到此多数据源的配置就完成了，项目中使用哪个数据源的操作，就注入对应包下的 repository 进行操作即可，接下来我们对上面配置好的数据源进行测试。

创建 UserRepositoryTests 测试类，将两个包下的 repository 都注入到测试类中：

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserRepositoryTests {
    @Resource
    private UserTest1Repository userTest1Repository;
    @Resource
    private UserTest2Repository userTest2Repository;
}
```

首先测试两个数据库中都存入数据，数据源1插入 2 条用户信息，数据源2插入 1 条用户信息。

```
@Test
public void testSave() throws Exception {
    Date date = new Date();
    DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFor-
mat.LONG, DateFormat.LONG);
    String formattedDate = dateFormat.format(date);

    userTest1Repository.save(new User("aa", "aa123456","aa@126.com"
, "aa",  formattedDate));
    userTest1Repository.save(new User("bb", "bb123456","bb@126.com"
, "bb",  formattedDate));
    userTest2Repository.save(new User("cc", "cc123456","cc@126.com"
, "cc",  formattedDate));
}
```

执行完测试用例后查看数据库，发现 test1 库有两条数据，test2 有一条，证明两个数据源均保存数据正常。下面继续测试删除功能，使用两个数据源的 repository 将用户信息全部删除。

```
@Test
public void testDelete() throws Exception {
    userTest1Repository.deleteAll();
    userTest2Repository.deleteAll();
}
```

执行完测试用例后，发现 test1 库和 test2 库用户表的信息已经被清空，证明多数据源删除成功。

总结

Spring Data JPA 通过在启动时加载不同的数据源，并将不同的数据源注入到不同的 repository 包下，从而实现项目多数据源操作，在项目中使用多数据源时，需要用到哪个数据源，只需要将对应包下的 repository 注入操作即可。本课示例中以两个数据源作为演示，但其实三个或者更多数据源配置、操作，都可以按照上面方法进行配置使用。

> 点击这里下载源码 (https://github.com/ityouknow/spring-boot-lean - ing/tree/gitbook_column2.0)。

(/gitchat/column/5b86228ce15aa17d68b5b55a/topic/5bf4c6eefae086212ccb20e4)(/gitcha