链滴

Spring Boot 多数据源配置

作者：umeone
原文链接：https://ld246.com/article/1551948713913
来源网站：链滴

在生产环境中，可能存在同一个项目访问多个数据源的情况，本文通过Spring Boot实现多数据源数操作。为开发示例简单，采用JPA进行数据库操作；配置文件采用yml进行配置，数据源为两个不同的MySQL数据库。同时为示例简单，配置信息我们只添加必要的配置，其他配置将省略。

一、单数据库操作示例

通过Spring Boot实现单数据源操作比较简单，只需要按照Spring Boot的约定进行配置即可，具体配如下：

```
spring:
  datasource:
    hikari:
      driver-class-name: com.mysql.cj.jdbc.Driver
      username: root
      password: root
    url: jdbc:mysql://127.0.0.1:3306/test1?serverTimezone=UTC
```

这里只配置数据库连接的必备信息，通过默认的配置方式，Spring Boot在启动时会直接加载这些必信息。

其他一些辅助测试的代码如下:

- User

```
/**
 * 用户信息
 *
 * @author zhang peng
 * @since 2019/3/6 10:12
 */
@Entity
@Table(name = "user")
public class User {

    @Id
    @GeneratedValue
    private Integer id;

    private String name;

    private int age;

    //省略getter setter
}
```

- Repository

```
/**
 * User Repository
 *
 * @author zhang peng
 * @since 2019/3/6 10:13
```

```
 */
public interface UserRepository extends JpaRepository<User, Integer> {
}
```

● Service 、Service Impl

```
/**
 * User Service Interface
 *
 * @author zhang peng
 * @since 2019/3/6 10:15
 */
public interface IUserService {

    User save(User user);

    User save(String name, Integer age);
}

/**
 * User Service Interface Impl
 *
 * @author zhang peng
 * @since 2019/3/6 10:16
 */
@Service
public class UserServiceImpl implements IUserService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public User save(User user) {
        return userRepository.save(user);
    }

    @Override
    @Transactional
    public User save(String name, Integer age) {
        User user = new User();
        user.setName(name);
        user.setAge(age);
        return save(user);
    }
}
```

● Controller

```
/**
 * User Controller
 *
 * @author zhang peng
 * @since 2019/3/6 10:18
```

```
 */
@RestController
@RequestMapping("/user")
public class UserController {

    @Autowired
    private IUserService userService;

    @RequestMapping(value = "/save")
    public Object saveUser(String name, Integer age) {
        User user = userService.save(name, age);
        return user.getId();
    }
}
```

以上就是整个从控制层到数据库存储的所有代码，在仅仅考虑简单数据库操作的情况下，利用Spring oot实现起来非常简单，这也是它作为一个快速开发框架的优势所在。

二、多数据源配置

1、自定义配置信息读取

Spring Boot以契约模式简化了项目的配置。如果我们需要配置应用参数，可以用过指定的配置文件application.yml或者application.properties进行配置。而在这里配置的信息我们可以使用spring提供默认key进行配置，这样spring在启动的时候会帮我们自动加载这些配置信息。同时，除了这些约定的配置key，我们也可以自定义自己的key进行配置，并通过配置类进行配置信息的加载。

在多数据源配置类中，我们需要配置DataSource、EntityManagerFactory、PlatformTransactionManager等信息。在单个数据源中，虽然没有明确编码配置，实际上是Spring Boot已经做了默认处理而在多数据源中，需要根据不同的数据源进行配置。本文数据源均采用MySQL，只是数据库名称不样，所以在两个数据源配置上基本一样，只是实例名称不同而已，而不同的数据库可能存在一些细节处理，这些是要注意的。

在从单数据源转到多数据源之前，我们需要了解一些Spring Boot的注解相关特性，以方便我们更好理解和运用这些特性（此处指说明注解用到的特性，其他特性参考官方文档）。

@Configuration

类注解，被该注解修饰的类Spring在加载的时候认为该类是一个配置类。

@Bean

方法注解，被该注解修饰的方法返回的bean对象被Spring接收管理。本示例会用到name属性，该属用于定义bean别名。

ConfigurationProperties

方法注解，被该注解修饰的方法可以通过注解过去配置属性。本示例会用到prefix属性，该属性用于述配置文件中自定义key的前缀信息。

@EnableJpaRepositories

类注解，该注解用于描述Jpa数据库操作信息。本示例会用到entityManagerFactoryRef、transactioManagerRef、basePackages属性，属性详细说明见下文代码注释。

此外，如果用到事务，还需要关注一个注解：@Transactional ，在多数据源配置中，由于每个数据都会配置自己的事务管理器，因为Spring管理这多个事务管理器，在操作数据库需要事务时，Spring不知道需要使用哪个事务管理器进行事务管理，所以在使用事务时，我们需要通过该注解的transactioManager属性进行事务管理器的指定。由于本文主要是讲多数据源配置，并不关心事务，所以在此简提示一下，避免按照此示例测试时出现事务配置问题。

2、示例代码：

● application.yml

```
data:
  source:
    first:
      jdbc-url: jdbc:mysql://127.0.0.1:3306/test1?serverTimezone=UTC
      driver-class-name: com.mysql.cj.jdbc.Driver
      username: root
      password: root

    second:
      jdbc-url: jdbc:mysql://127.0.0.1:3306/test2?serverTimezone=UTC
      driver-class-name: com.mysql.cj.jdbc.Driver
      username: root
      password: root
```

配置文件配置了两个数据源，分别指向不同的MySQL数据库。其中需要注意，在自定义配置信息key时候应该保持key的规律，方便后期在获取值的时候有规律可循。上述配置中，我们把data.source作前缀。

配置中由于存在两个数据源（也可以更多），所以在定义配置类时，需要多个配置类和数据源信息做应。本示例中会有两个配置类针对数据源信息，同时为了代码清晰，还会多出一个配置类专门处理数源共有的配置信息。

● 数据源一配置

```
/**
 * First DataSource Configuration
 *
 * @author zhang peng
 * @since 2019/3/6 13:36
 */
@Configuration
@EnableJpaRepositories(
        entityManagerFactoryRef = "firstEntityManagerFactory"，//EntityManagerFactory信息配

        transactionManagerRef = "firstTransactionManager", //事务管理器
        basePackages = {"com.whucke.multi.datasource.repository.first"}) //扫描包定义。同样，
方便使用，在定义包名时也需要遵循一定规律
public class FirstConfig {

    @Autowired
    private DataSource firstDataSource;
    @Autowired
    JpaVendorAdapter jpaVendorAdapter;
```

```
    /**
     * firstDataSource
     *
     * @return
     */
    @Bean(name = "firstDataSource")
    @ConfigurationProperties(prefix = "data.source.first") //定义配置信息前缀，这也是前面为什
说key的定义需要遵循一定的规律
    public DataSource firstDataSource() {
        //通过该方法调用，Spring会自动装配配置信息
        DataSource dataSource = DataSourceBuilder.create().build();
        return dataSource;
    }

    /**
     * 配置 EntityManagerFactory
     *
     * @return
     */
    @Bean("firstEntityManagerFactory")
    public EntityManagerFactory entityManagerFactory() {
        LocalContainerEntityManagerFactoryBean factoryBean = new LocalContainerEntityMana
erFactoryBean();
        factoryBean.setDataSource(firstDataSource());
        factoryBean.setJpaVendorAdapter(jpaVendorAdapter);
        factoryBean.setPackagesToScan("com.whucke.multi.datasource.domain");
        //持久层名称定义，在配置多数据源时必须配置该信息
        factoryBean.setPersistenceUnitName("first");
        factoryBean.afterPropertiesSet();
        return factoryBean.getObject();
    }

     /**
     * 事务管理器
     *
     * @return
     */
    @Bean("firstTransactionManager")
    public PlatformTransactionManager transactionManager() {
        return new JpaTransactionManager(entityManagerFactory());
    }

}
```

第二个数据源配置与第一个类似，唯一区别在于bean名称的定义和引用，在配置时需要注意，不要把ean名称配置混淆了。代码如下：

- 数据源二配置

```
/**
 * Second DataSource Configuration
 *
```

```
 * @author zhang peng
 * @since 2019/3/6 13:36
 */
@Configuration
@EnableJpaRepositories(
        entityManagerFactoryRef = "secondEntityManagerFactory",
        transactionManagerRef = "secondTransactionManager",
        basePackages = {"com.whucke.multi.datasource.repository.second"})
public class SecondConfig {

    @Autowired
    private DataSource secondDataSource;
    @Autowired
    JpaVendorAdapter jpaVendorAdapter;

    /**
     * secondDataSource
     *
     * @return
     */
    @Bean(name = "secondDataSource")
    @ConfigurationProperties(prefix = "data.source.second")
    public DataSource secondDataSource() {
        DataSource dataSource = DataSourceBuilder.create().build();
        return dataSource;
    }

    /**
     * 配置 EntityManagerFactory
     *
     * @return
     */
    @Bean("secondEntityManagerFactory")
    public EntityManagerFactory entityManagerFactory() {
        LocalContainerEntityManagerFactoryBean factoryBean = new LocalContainerEntityMana
erFactoryBean();
        factoryBean.setDataSource(secondDataSource());
        factoryBean.setJpaVendorAdapter(jpaVendorAdapter);
        factoryBean.setPackagesToScan("com.whucke.multi.datasource.domain");
        factoryBean.setPersistenceUnitName("second");
        factoryBean.afterPropertiesSet();
        return factoryBean.getObject();
    }

    @Bean("secondTransactionManager")
    public PlatformTransactionManager transactionManager() {
        return new JpaTransactionManager(entityManagerFactory());
    }

}
```

前面我们说到会有一个针对数据源的公共配置信息类存在，在本示例中公共配置信息并不多，可能在同的项目场景下配置信息还会有增加或修改，这需要根据项目的实际情况做处理。

- 数据源公共配置信息

```
/**
 * Data Source Configuration
 *
 * @author zhang peng
 * @since 2019/3/6 10:26
 */
@Configuration
public class DataSourceConfig {

    @Bean
    public JpaVendorAdapter jpaVendorAdapter() {
        HibernateJpaVendorAdapter jpaVendorAdapter = new HibernateJpaVendorAdapter();
        jpaVendorAdapter.setShowSql(true);
        jpaVendorAdapter.setGenerateDdl(true);
        jpaVendorAdapter.setDatabase(Database.MYSQL);
        return jpaVendorAdapter;
    }
}
```

此处对JpaVendorAdapter的配置是直接通过硬编码实现的。如果参数较多、不同数据源等情况，这代码也可以根据不同的数据源进行配置加载。此处只为说明问题，所以不再引申更多内容。

三、事务

最后简单提说一下上文提到的事务管理配置信息。由于真实开发环境中多数据源事务很少直接通过上配置的事务管理器去处理，所以这里只做简单的说明，需要实现真正的跨数据库事务，还需要用到其技术才行。

- 示例：

```
@Transactional(transactionManager = "firstTransactionManager")
public User save(String name, Integer age) {
    User user = new User();
    user.setName(name);
    user.setAge(age);
    return save(user);
}
```

从上述示例中可以发现，我们在添加事务注解@Transactional的时候会额外配置transactionManager性。原因是Spring容器中存在多个事务管理器配置(前面配置文件中进行配置的),在处理事务过程中，架不知道使用哪一个，所以需要开发者明确指定。

在配置中也可以看出，指定一个事务管理器，而该事务管理器只会在对应数据源起作用，所以跨数据操作事务是不会同时生效的。

以上就是Spring Boot多数据源配置的基础参考方案，需要注意的是该示例是整合的JPA。如果要整合ybatis，需要做相应的修改。

代码地址:https://gitee.com/umeone/multi-datasource.git