springboot(七)：springboot+mybatis多数据源最简解决方案

2016/11/25

说起多数据源，一般都来解决那些问题呢，主从模式或者业务比较复杂需要连接不同的分库来支持业务。我们项目是后者的模式，网上找了很多，大都是根据jpa来做多数据源解决方案，要不就是老的spring多数据源解决方案，还有的是利用aop动态切换，感觉有点小复杂，其实我只是想找一个简单的多数据支持而已，折腾了两个小时整理出来，供大家参考。

> 废话不多说直接上代码吧

配置文件

pom包就不贴了比较简单该依赖的就依赖，主要是数据库这边的配置：

```
mybatis.config-locations=classpath:mybatis/mybatis-config.xml

spring.datasource.test1.driverClassName = com.mysql.jdbc.Driver
spring.datasource.test1.url = jdbc:mysql://localhost:3306/test1?useUnicode=true&characterEncoding=utf-
spring.datasource.test1.username = root
spring.datasource.test1.password = root

spring.datasource.test2.driverClassName = com.mysql.jdbc.Driver
spring.datasource.test2.url = jdbc:mysql://localhost:3306/test2?useUnicode=true&characterEncoding=utf-
spring.datasource.test2.username = root
spring.datasource.test2.password = root
```

一个test1库和一个test2库，其中test1位主库，在使用的过程中必须指定主库，不然会报错。

数据源配置

```
@Configuration
@MapperScan(basePackages = "com.neo.mapper.test1", sqlSessionTemplateRef  = "test1SqlSessionTem
public class DataSource1Config {

    @Bean(name = "test1DataSource")
    @ConfigurationProperties(prefix = "spring.datasource.test1")
    @Primary
    public DataSource testDataSource() {
        return DataSourceBuilder.create().build();
    }

    @Bean(name = "test1SqlSessionFactory")
    @Primary
    public SqlSessionFactory testSqlSessionFactory(@Qualifier("test1DataSource") DataSource dataSou
        SqlSessionFactoryBean bean = new SqlSessionFactoryBean();
        bean.setDataSource(dataSource);
        bean.setMapperLocations(new PathMatchingResourcePatternResolver().getResources("classpath:
        return bean.getObject();
    }

    @Bean(name = "test1TransactionManager")
    @Primary
    public DataSourceTransactionManager testTransactionManager(@Qualifier("test1DataSource") Dat
        return new DataSourceTransactionManager(dataSource);
    }

    @Bean(name = "test1SqlSessionTemplate")
    @Primary
    public SqlSessionTemplate testSqlSessionTemplate(@Qualifier("test1SqlSessionFactory") SqlSessior
        return new SqlSessionTemplate(sqlSessionFactory);
    }

}
```

最关键的地方就是这块了，一层一层注入,首先创建DataSource，然后创建SqlSessionFactory再创建事务，最后包装到SqlSessionTemplate中。其中需要指定分库的mapper文件地址，以及分库dao层代码

```
@MapperScan(basePackages = "com.neo.mapper.test1", sqlSessionTemplateRef  = "test1SqlSessionTem
```

这块的注解就是指明了扫描dao层，并且给dao层注入指定的SqlSessionTemplate。所有 `@Bean` 都需要按照命名指定正确。

dao层和xml层

dao层和xml需要按照库来分在不同的目录，比如：test1库dao层在com.neo.mapper.test1包下，test2库在com.neo.mapper.test1

```
public interface User1Mapper {

	List<UserEntity> getAll();

	UserEntity getOne(Long id);

	void insert(UserEntity user);

	void update(UserEntity user);

	void delete(Long id);

}
```

xml层

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-
<mapper namespace="com.neo.mapper.test1.User1Mapper" >
    <resultMap id="BaseResultMap" type="com.neo.entity.UserEntity" >
        <id column="id" property="id" jdbcType="BIGINT" />
        <result column="userName" property="userName" jdbcType="VARCHAR" />
        <result column="passWord" property="passWord" jdbcType="VARCHAR" />
        <result column="user_sex" property="userSex" javaType="com.neo.enums.UserSexEnum"/>
        <result column="nick_name" property="nickName" jdbcType="VARCHAR" />
    </resultMap>

    <sql id="Base_Column_List" >
        id, userName, passWord, user_sex, nick_name
    </sql>

    <select id="getAll" resultMap="BaseResultMap"  >
       SELECT
       <include refid="Base_Column_List" />
           FROM users
    </select>

    <select id="getOne" parameterType="java.lang.Long" resultMap="BaseResultMap" >
        SELECT
       <include refid="Base_Column_List" />
           FROM users
           WHERE id = #{id}
    </select>

    <insert id="insert" parameterType="com.neo.entity.UserEntity" >
       INSERT INTO
                users
                (userName,passWord,user_sex)
        VALUES
                (#{userName}, #{passWord}, #{userSex})
    </insert>
    <update id="update" parameterType="com.neo.entity.UserEntity" >
       UPDATE
                users
       SET
        <if test="userName != null">userName = #{userName},</if>
        <if test="passWord != null">passWord = #{passWord},</if>
        nick_name = #{nickName}
       WHERE
                id = #{id}
    </update>

    <delete id="delete" parameterType="java.lang.Long" >
       DELETE FROM
                 users
       WHERE
                 id =#{id}
    </delete>

</mapper>
```

测试

测试可以使用SpringBootTest,也可以放到Controller中，这里只贴Controller层的使用

```
@RestController
public class UserController {

    @Autowired
    private User1Mapper user1Mapper;

        @Autowired
        private User2Mapper user2Mapper;

        @RequestMapping("/getUsers")
        public List<UserEntity> getUsers() {
                List<UserEntity> users=user1Mapper.getAll();
                return users;
        }

    @RequestMapping("/getUser")
    public UserEntity getUser(Long id) {
        UserEntity user=user2Mapper.getOne(id);
        return user;
    }

    @RequestMapping("/add")
    public void save(UserEntity user) {
        user2Mapper.insert(user);
    }

    @RequestMapping(value="update")
    public void update(UserEntity user) {
        user2Mapper.update(user);
    }

    @RequestMapping(value="/delete/{id}")
    public void delete(@PathVariable("id") Long id) {
        user1Mapper.delete(id);
    }
}
```

示例代码-github (https://github.com/ityouknow/spring-boot-examples)

示例代码-码云 (https://gitee.com/ityouknow/spring-boot-examples)

扫码关注有惊喜

Show Disqus Comments

未找到相关的 Issues (https://github.com/ityouknow/blog-comments/issues) 进行评论

请联系 @ityouknow 初始化创建

Post Directory