springboot-mybatis 多数据源

作者：yechuan
原文链接：https://ld246.com/article/1589171390493
来源网站：链滴

springboot-mybatis多数据源

多数据源的目的

- 随着业务的发展，单库的的形式已经无法承受高并发所带来的压力，一个项目使用多个数据库显得外重要，比如读写分离、主从复制

mybatis配置多数据源的方式

分包的方式

- 将不同数据源的mapper文件分开
- 读数据源

```
 @Configuration
 @MapperScan(basePackages="com.yechuan.dao.read",sqlSessionFactoryRef="readSqlSessi
nFactory")
 public class ReadDataSourceConfig {
   @Primary
   @Bean(name = "readDataSource")
   @ConfigurationProperties("datasources.read")
       public DataSource readDataSource() {
       return new DruidDataSource();
   }

   @Primary
   @Bean(name = "readDataSourceTransactionManager")
   public DataSourceTransactionManager dataSourceTransactionManager(DataSource dataSo
```

```
rce) {
        return new DataSourceTransactionManager(dataSource);
    }

    @Primary
    @Bean(name = "readSqlSessionFactory")
    public SqlSessionFactory sqlSessionFactory(DataSource dataSource) throws Exception {
        SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();
        sqlSessionFactoryBean.setDataSource(dataSource);
        return sqlSessionFactoryBean.getObject();
    }

    @Primary
    @Bean(name = "readSqlSessionTemplate")
    public SqlSessionTemplate sqlSessionTemplate(SqlSessionFactory sqlSessionFactory) {
        return new SqlSessionTemplate(sqlSessionFactory);
    }

   }
```

- 写数据源

```
@Configuration
@MapperScan(basePackages = "com.yechuan.dao.write", sqlSessionFactoryRef = "writeSqlSe
sionFactory")
public class WriteDataSourceConfig {

    @Bean(name = "writeDataSource")
    @ConfigurationProperties("datasources.write")
    public DataSource writeDataSource() {
        return new DruidDataSource();

    }

    @Bean(name = "writeDataSourceTransactionManager")
    public DataSourceTransactionManager dataSourceTransactionManager(@Qualifier("writeD
taSource") DataSource dataSource) {
        return new DataSourceTransactionManager(dataSource);
    }

    @Bean(name = "writeSqlSessionFactory")
    public SqlSessionFactory sqlSessionFactory(@Qualifier("writeDataSource") DataSource dat
Source) throws Exception {
        SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();
        sqlSessionFactoryBean.setDataSource(dataSource);
        return sqlSessionFactoryBean.getObject();
    }

    @Bean(name = "writeSqlSessionTemplate")
    public SqlSessionTemplate sqlSessionTemplate(@Qualifier("writeSqlSessionFactory") SqlSe
sionFactory sqlSessionFactory) {
        return new SqlSessionTemplate(sqlSessionFactory);
    }
```

```
}
```

AOP

- 利用aop原理，在访问数据库之前，替换数据库实例
- 在访问数据库的时候，会访问 **AbstractRoutingDataSource的determineCurrentLookupKey**法来获取数据库的实例key

1. 定义一个枚举类来说明一下当前数据源实例key有哪些

```
public enum DataSourceKey {
    READ("read"),
    WRITE("write");

    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    private DataSourceKey(String name) {
        this.name = name;
    }
}
```

2. 创建一个工具类来实现，存储与获取数据源key

```
public class DynamicDataSourceContextHolder {
    /**
    *
    * 保证其线程安全
    */
    private static ThreadLocal<Object> CONTEXT_HOLDER = ThreadLocal.withInitial(() -> Data
ourceKey.READ.getName());
    /**
    *
    * 存储
    */
    public static void setDataSourceKey(String key){
        CONTEXT_HOLDER.set(key);
    }
    /**
    *
    * 获取
    */
    public static Object getDataSourceKey(){
        return CONTEXT_HOLDER.get();
    }
    /**
    *
```

```
    * 删除
    */
    public static void clearDataSourceKey(){
        CONTEXT_HOLDER.remove();
    }

}
```

3. 重写AbstractRoutingDataSource

```
@Slf4j
public class DynamicRoutingDataSource extends AbstractRoutingDataSource {

    @Override
    protected Object determineCurrentLookupKey() {
        log.info("当前使用的数据源是 : {}",DynamicDataSourceContextHolder.getDataSourceKey());
        return DynamicDataSourceContextHolder.getDataSourceKey();
    }

}
```

4. 配置数据源

```
@Configuration
public class DataSourceConfig {
  @Primary
  @Bean(name = "readDataSource")
  @ConfigurationProperties("datasources.read")
  public DataSource readDataSource() {
      return new DruidDataSource();
  }

  @Bean(name = "writeDataSource")
  @ConfigurationProperties("datasources.write")
  public DataSource writeDataSource() {
        return new DruidDataSource();

  }
  @Bean(name = "dynamicDataSource")
  public DataSource dynamicDataSource(@Qualifier(value = "readDataSource")DataSource re
dDataSource,@Qualifier(value = "writeDataSource")DataSource writeDataSource){
        DynamicRoutingDataSource dynamicRoutingDataSource = new DynamicRoutingDataSo
rce();

        Map<Object, Object> dataSourceMap = new HashMap<Object, Object>(2);
        dataSourceMap.put(DataSourceKey.READ.getName(),readDataSource);
        dataSourceMap.put(DataSourceKey.WRITE.getName(),writeDataSource);

        dynamicRoutingDataSource.setDefaultTargetDataSource(readDataSource);
        dynamicRoutingDataSource.setTargetDataSources(dataSourceMap);

        DynamicDataSourceContextHolder.dataSourceKeys.addAll(dataSourceMap.keySet());

        return dynamicRoutingDataSource;
  }
```

```
  @Bean
  public DataSourceTransactionManager dataSourceTransactionManager(@Qualifier(value = "
ynamicDataSource") DataSource dataSource) {
      return new DataSourceTransactionManager(dataSource);
  }

  @Bean
  public SqlSessionFactory sqlSessionFactory(@Qualifier(value = "dynamicDataSource") DataS
urce dataSource) throws Exception {
      SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();
      sqlSessionFactoryBean.setDataSource(dataSource);
      return sqlSessionFactoryBean.getObject();
  }

  @Bean
  public SqlSessionTemplate sqlSessionTemplate(SqlSessionFactory sqlSessionFactory) {
      return new SqlSessionTemplate(sqlSessionFactory);
  }
}
```

5. 自定义注解

```
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface UserDataSource {
    DataSourceKey value() default DataSourceKey.READ;
}
```

6. AOP动态切换数据源

```
@Slf4j
@Aspect
@Component
public class DynamicDataSourceAspect {


    @Before("@annotation(userDataSource))")
    public void switchDataSource(JoinPoint joinPoint,UserDataSource userDataSource){
    DynamicDataSourceContextHolder.setDataSourceKey(targetDataSource.value().getName());
        log.info("在方法： [{}] 内切换数据源为： [{}]",
        joinPoint.getSignature(),DynamicDataSourceContextHolder.getDataSourceKey());

    }

    @After("@annotation(userDataSource))")
    public void restoreDataSource(JoinPoint joinPoint,UserDataSource userDataSource){
        DynamicDataSourceContextHolder.clearDataSourceKey();
        log.info("在方法 [{}] 执行后，恢复数据源 [{}]",
          joinPoint.getSignature(),DynamicDataSourceContextHolder.getDataSourceKey());
    }

}
```

原文链接：springboot-mybatis 多数据源

7. 在调用的时候使用注解即可

使用mybatis-puls与dynamic

- 鲁迅先生曾经说过我们不要自己造轮子，要使用别人的轮子

1. 导入相关依赖

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>dynamic-datasource-spring-boot-starter</artifactId>
    <version>2.5.4</version>
</dependency>
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.3.1.tmp</version>
</dependency>
```

2. yml配置

```
spring:
  datasource:
    dynamic:
      datasource:
        read:
          password: root
          username: root
          url: jdbc:mysql://localhost:3306/sharding_sphere_read
          driverClassName: com.mysql.jdbc.Driver
        write:
          password: root
          username: root
          url: jdbc:mysql://localhost:3306/sharding_sphere_write
          driverClassName: com.mysql.jdbc.Driver
      primary: read
      strict: true
```

3. 使用

```
    @DS("write")
    @Transactional(rollbackFor = Exception.class)
    public void addUser( UserDto userDto) {
        userDao.insert(userDto);
    }
```