原 Spring+Mybatis读写分离

mahengyang 发表于 3周前 阅读 276 收藏 23 点赞 2 评论 0

已收藏

HOT

摘要: 自定义mybatis插件，实现数据库读写分离，通过扩展spring的AbstractRoutingDataSource实现动态切换数据源

说明

mybatis插件+扩展Spring数据源是最省力的方式，不需要改动任何原代码，如果是数据库事务，则插件不生效，不会对数据库事务造成影响

spring配置

```
<bean id="abstractDataSource" abstract="true" class="com.alibaba.druid.pool.DruidDataSource"
    init-method="init" destroy-method="close">
</bean>

<bean id="write_db" parent="abstractDataSource" >
    <property name="url" value="${write.jdbc.url}" />
    <property name="username" value="${write.jdbc.username}" />
    <property name="password" value="${write.jdbc.password}" />
</bean>

<bean id="read_db" parent="abstractDataSource">
    <property name="url" value="${read.jdbc.url}" />
    <property name="username" value="${read.jdbc.username}" />
    <property name="password" value="${read.jdbc.password}" />
</bean>

<!-- 数据源切换 -->
<bean id="routingDataSource" class="com.main.common.mybatis.MyRoutingDataSource">
    <property name="read" ref="read_db" />
    <property name="write" ref="write_db" />
</bean>

<!-- MyBatis配置 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="routingDataSource" />
    <!-- 自动扫描domain目录, 省掉Configuration.xml里的手工配置 -->
    <property name="typeAliasesPackage" value="com.main.common.domain" />
    <property name="mapperLocations" value="classpath:/mybatis/*Mapper.xml" />
    <property name="configLocation" value="classpath:mybatis-config.xml" />
</bean>

<!-- mybatis增强Mapper 通用CRUD -->
<bean class="tk.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
    <property name="basePackage" value="com.main.common.repository"/>
    <property name="markerInterface" value="tk.mybatis.mapper.common.Mapper"/>
</bean>

<!-- 使用自定义的事务管理器 -->
<bean id="transactionManager" class="com.main.common.mybatis.MyDynamicDataSourceTransactionManager" >
    <property name="dataSource" ref="routingDataSource" />
</bean>

<tx:annotation-driven transaction-manager="transactionManager" order="1"/>
```

在mybatis配置文件中配置插件

```
<configuration>
    <plugins>
        <!-- 分页插件 -->
        <plugin interceptor="com.github.pagehelper.PageInterceptor">
            <property name="helperDialect" value="mysql"/>
            <property name="reasonable" value="true"/>
```

```
            <property name="supportMethodsArguments" value="true"/>
            <property name="autoRuntimeDialect" value="true"/>
        </plugin>

        <!-- 读写分离插件 -->
        <plugin interceptor="com.main.common.mybatis.MyDynamicDataSourcePlugin">
        </plugin>
    </plugins>
</configuration>
```

mybatis读写分离插件源码MyDynamicDataSourcePlugin.java

```
package com.main.common.mybatis;

import org.apache.ibatis.executor.Executor;
import org.apache.ibatis.executor.keygen.SelectKeyGenerator;
import org.apache.ibatis.mapping.BoundSql;
import org.apache.ibatis.mapping.MappedStatement;
import org.apache.ibatis.mapping.SqlCommandType;
import org.apache.ibatis.plugin.*;
import org.apache.ibatis.session.ResultHandler;
import org.apache.ibatis.session.RowBounds;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.transaction.support.TransactionSynchronizationManager;

import java.util.Locale;
import java.util.Map;
import java.util.Properties;
import java.util.concurrent.ConcurrentHashMap;

/**
 * mybatis插件，根据sql类型判断应该使用哪种数据源（读/写），如果是数据库事务，则不处理
 * Created by mahengyang on 2017/7/21.
 */
@Intercepts({
        @Signature(type = Executor.class, method = "update", args = {MappedStatement.class, Object.class}),
        @Signature(type = Executor.class, method = "query", args = {MappedStatement.class, Object.class, RowBoun
})
public class MyDynamicDataSourcePlugin implements Interceptor {
    protected static final Logger log = LoggerFactory.getLogger(MyDynamicDataSourcePlugin.class);
    private static final String REGEX = ".*insert\\u0020.*|.*delete\\u0020.*|.*update\\u0020.*";

    private static final Map<String, MyDynamicDataSourceHolder.MyDataSource> cacheMap = new ConcurrentHashMap<>(

    [@Override](https://my.oschina.net/u/1162528)
    public Object intercept(Invocation invocation) throws Throwable {
        // 判断是否是数据库事务
        boolean synchronizationActive = TransactionSynchronizationManager.isSynchronizationActive();
        log.debug("动态数据源插件 是否是事务:{}", synchronizationActive);
        if (!synchronizationActive) {
            Object[] objects = invocation.getArgs();
            MappedStatement ms = (MappedStatement) objects[0];
            String statementId = ms.getId();
            MyDynamicDataSourceHolder.MyDataSource dataSource = cacheMap.get(statementId);
            log.debug("动态数据源插件 非事务 数据源:{}", dataSource);
            if (dataSource == null) {
                //读方法
                if (ms.getSqlCommandType().equals(SqlCommandType.SELECT)) {
                    //!selectKey 为自增id查询主键(SELECT LAST_INSERT_ID() )方法，使用主库
                    if (statementId.contains(SelectKeyGenerator.SELECT_KEY_SUFFIX)) {
                        dataSource = MyDynamicDataSourceHolder.MyDataSource.WRITE;
                    } else {
                        BoundSql boundSql = ms.getSqlSource().getBoundSql(objects[1]);
                        String sql = boundSql.getSql().toLowerCase(Locale.CHINA).replaceAll("[\\t\\n\\r]", " ");
                        if (sql.matches(REGEX)) {
                            dataSource = MyDynamicDataSourceHolder.MyDataSource.WRITE;
                        } else {
                            dataSource = MyDynamicDataSourceHolder.MyDataSource.READ;
                        }
                    }
                } else {
                    // 写方法
                    dataSource = MyDynamicDataSourceHolder.MyDataSource.WRITE;
                }
                log.debug("读写分离插件 设置方法:{} 使用:{} 数据源 SqlCommandType [{}]", statementId, dataSource
                cacheMap.put(statementId, dataSource);
            }
            MyDynamicDataSourceHolder.putDataSource(dataSource);
        }
        return invocation.proceed();
    }

    [@Override](https://my.oschina.net/u/1162528)
    public Object plugin(Object target) {
```

```
        if (target instanceof Executor) {
            return Plugin.wrap(target, this);
        } else {
            return target;
        }
    }

    @Override
    public void setProperties(Properties properties) {

    }
}
```

自定义数据源切换类MyRoutingDataSource.java

```
package com.main.common.mybatis;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;

import java.util.HashMap;
import java.util.Map;

/**
 * 根据MyDynamicDataSourceHolder中的线程局部变量，选择数据源（读/写）
 * Created by mahengyang on 2017/7/21.
 */
public class MyRoutingDataSource extends AbstractRoutingDataSource {
    protected static final Logger log = LoggerFactory.getLogger(MyRoutingDataSource.class);

    private Object read;
    private Object write;

    @Override
    protected Object determineCurrentLookupKey() {
        MyDynamicDataSourceHolder.MyDataSource dataSource = MyDynamicDataSourceHolder.getDataSource();
        if (dataSource == null || dataSource == MyDynamicDataSourceHolder.MyDataSource.WRITE) {
            log.debug("动态路由数据源 本次选择使用 写 数据源");
            return MyDynamicDataSourceHolder.MyDataSource.WRITE.name();
        }
        log.debug("动态路由数据源 本次选择使用 读 数据源");
        return MyDynamicDataSourceHolder.MyDataSource.READ.name();
    }

    /**
     * 把数据源配置到spring的管理器中，因为spring的数据源是用Map<Object,Object>存储的，所以这里以枚举的名称作为k
     */
    @Override
    public void afterPropertiesSet() {
        if (this.write == null) {
            throw new IllegalArgumentException("Property 'write' is required");
        }
        setDefaultTargetDataSource(write);
        Map<Object, Object> targetDataSources = new HashMap<>();
        targetDataSources.put(MyDynamicDataSourceHolder.MyDataSource.WRITE.name(), write);
        if (read != null) {
            targetDataSources.put(MyDynamicDataSourceHolder.MyDataSource.READ.name(), read);
        }
        setTargetDataSources(targetDataSources);
        super.afterPropertiesSet();
    }

    public void setRead(Object read) {
        this.read = read;
    }

    public void setWrite(Object write) {
        this.write = write;
    }
}
```

自定义数据库事务处理类MyDynamicDataSourceTransactionManager

```
package com.main.common.mybatis;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.transaction.TransactionDefinition;

/**
```

```
 * 自定义事务管理器，继承自spring的事务管理器，只做了数据源的切换功能，其他全部继承父类
 * Created by mahengyang on 2017/7/24.
 */
public class MyDynamicDataSourceTransactionManager extends DataSourceTransactionManager {
    protected static final Logger log = LoggerFactory.getLogger(MyDynamicDataSourceTransactionManager.class);

    @Override
    protected void doBegin(Object transaction, TransactionDefinition definition) {
        boolean readOnly = definition.isReadOnly();
        if (readOnly) {
            log.debug("数据库事务管理器 读");
            MyDynamicDataSourceHolder.putDataSource(MyDynamicDataSourceHolder.MyDataSource.READ);
        } else {
            log.debug("数据库事务管理器 写");
            MyDynamicDataSourceHolder.putDataSource(MyDynamicDataSourceHolder.MyDataSource.WRITE);
        }
        super.doBegin(transaction, definition);
    }

    @Override
    protected void doCleanupAfterCompletion(Object transaction) {
        super.doCleanupAfterCompletion(transaction);
        MyDynamicDataSourceHolder.clearDataSource();
    }
}
```

自定义数据源（读/写）

```
package com.main.common.mybatis;

/**
 * 把数据源作为线程局部变量，线程使用完清除
 * Created by mahengyang on 2017/7/24.
 */
public class MyDynamicDataSourceHolder {
    private static final ThreadLocal<MyDataSource> holder = new ThreadLocal<MyDataSource>();

    private MyDynamicDataSourceHolder() {
    }

    public static void putDataSource(MyDataSource dataSource){
        holder.set(dataSource);
    }

    public static MyDataSource getDataSource(){
        return holder.get();
    }

    public static void clearDataSource() {
        holder.remove();
    }

    public enum MyDataSource {
        READ, WRITE;
    }
}
```

© 著作权归作者所有

分类：工作日志　字数：1059

标签：mybatis　读写分离　spring

打赏　点赞　已收藏　分享

mahengyang

程序员 苏州

粉丝 48 | 博文 46 | 码字总数 32487

+ 关注

相关博客

spring jpa 读写分离

hangge111

100 0

spring boot 进行读写分离

蔡少东

690 0

spring 集成mybatis动态设置数据源，实现读写分离

四月李

47 0

评论 (0)

Ctrl+Enter 发表评论