Spring AOP 实现读写分离

* 发表于 *2018-03-13* | * 分类于 后端 | * 阅读数: *1307* | * 评论数: 0

一、前言

上一篇[《MySQL 实现主从复制》文章中介绍了 MySQL 主从复制的搭建，为了在项目上契合数据库的主从架构，本篇将介绍在应用层实现对数据库的读写分离。

二、原理

配置主从数据源，当接收请求时，执行具体方法之前（拦截），判断请求具体操作（读或写），最终确定从哪个数据源获取连接访问数据库。

在 JavaWeb 开发中，有 3 种方式可以对请求进行拦截：

```
filter：拦截所有请求
intercetor：拦截 handler/Action
aop 切面：依赖切入点
```

不难看出，使用 AOP 切面进行拦截最合理和灵活，因此本文将介绍使用 AOP 实现读写分离功能。

三、编码

本文只张贴关键性代码，详细代码请下载文章末尾源码进行查看。

3.1 代码

1）DynamicDataSourceHolder 确保线程安全：

```
/**
 *
 * 使用ThreadLocal技术来记录当前线程中的数据源的key
 *
 */
public class DynamicDataSourceHolder {

    //写库对应的数据源key
    private static final String MASTER = "master";

    //读库对应的数据源key
    private static final String SLAVE = "slave";

    //使用ThreadLocal记录当前线程的数据源key
    private static final ThreadLocal<String> holder = new
ThreadLocal<String>();

    /**
     * 设置数据源key
     * @param key
     */
    public static void putDataSourceKey(String key) {
        holder.set(key);
    }

    /**
     * 获取数据源key
     * @return
     */
    public static String getDataSourceKey() {
        return holder.get();
    }

    /**
     * 标记写库
     */
    public static void markMaster(){
        putDataSourceKey(MASTER);
    }
```

```
    /**
     * 标记读库
     */
    public static void markSlave(){
        putDataSourceKey(SLAVE);
    }

}
```

2）定义 AOP 切面判断当前线程的读写操作

```
/**
 * 定义数据源的AOP切面，通过该Service的方法名判断是应该走读库还是写库
 *
 */
public class DataSourceAspect {

    /**
     * 在进入Service方法之前执行
     *
     * @param point 切面对象
     */
    public void before(JoinPoint point) {
        // 获取到当前执行的方法名
        String methodName = point.getSignature().getName();
        if (isSlave(methodName)) {
            // 标记为读库
            DynamicDataSourceHolder.markSlave();
        } else {
            // 标记为写库
            DynamicDataSourceHolder.markMaster();
        }
    }

    /**
     * 判断是否为读库
     *
     * @param methodName
     * @return
     */
    private Boolean isSlave(String methodName) {
        // 方法名以query、find、get开头的方法名走从库
        return StringUtils.startsWithAny(methodName, "query", "find",
"get");
    }

}
```

3）定义动态数据源，确定最终使用的数据源：

```
/**
 * 定义动态数据源，实现通过集成Spring提供的AbstractRoutingDataSource，只需要实现determineCurrentLookupKey方法即可
 *
 * 由于DynamicDataSource是单例的，线程不安全的，所以采用ThreadLocal保证线程安全，由DynamicDataSourceHolder完成。
 *
 */
public class DynamicDataSource extends AbstractRoutingDataSource{

    @Override
    protected Object determineCurrentLookupKey() {
        // 使用DynamicDataSourceHolder保证线程安全，并且得到当前线程中的数据源key
        String dataSourceKey = DynamicDataSourceHolder.getDataSourceKey();
        System.out.println("dataSourceKey ======> "+dataSourceKey);
        return dataSourceKey;
    }

}
```

3.2 配置文件

1）jdbc.properties

```
jdbc.driver=com.mysql.jdbc.Driver

jdbc.master.url=jdbc:mysql://192.168.2.21/mysql_test?characterEncoding=utf-8&allowMultiQueries=true&serverTimezone=UTC
jdbc.master.username=root
jdbc.master.password=tiger

jdbc.slave01.url=jdbc:mysql://192.168.2.22/mysql_test?characterEncoding=utf-8&allowMultiQueries=true&serverTimezone=UTC
jdbc.slave01.username=root
jdbc.slave01.password=tiger
```

2）applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                   http://www.springframework.org/schema/beans/spring-
beans-4.0.xsd
                   http://www.springframework.org/schema/context
                   http://www.springframework.org/schema/context/spring-
context-4.0.xsd
                   http://www.springframework.org/schema/tx
                   http://www.springframework.org/schema/tx/spring-tx-
4.0.xsd
                   http://www.springframework.org/schema/aop
                   http://www.springframework.org/schema/aop/spring-aop-
4.0.xsd">

    <context:component-scan base-package="com.light.*">
        <context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>

    <context:property-placeholder location="classpath:*.properties"/>

    <!-- 数据源 -->
    <bean id="dataSource"
class="com.light.dynamicdatasource.DynamicDataSource">
        <property name="targetDataSources">
            <map key-type="java.lang.String">
                <entry key="master" value-ref="masterDataSource"></entry>
                <entry key="slave" value-ref="slave01DataSource"></entry>
            </map>
        </property>
        <!-- 默认数据源 -->
        <property name="defaultTargetDataSource" ref="masterDataSource"/>
    </bean>

    <!-- 主库数据源 -->
```

```
    <bean id="masterDataSource"
class="com.alibaba.druid.pool.DruidDataSource" destroy-method="close">
        <property name="url" value="${jdbc.master.url}"/>
        <property name="username" value="${jdbc.master.username}"/>
        <property name="password" value="${jdbc.master.password}"/>
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="initialSize" value="5"/>
        <property name="minIdle" value="5"/>
        <property name="maxActive" value="50"/>
    </bean>

    <!-- 从库数据源 -->
    <bean id="slave01DataSource"
class="com.alibaba.druid.pool.DruidDataSource" destroy-method="close">
        <property name="url" value="${jdbc.slave01.url}"/>
        <property name="username" value="${jdbc.slave01.username}"/>
        <property name="password" value="${jdbc.slave01.password}"/>
        <property name="driverClassName" value="${jdbc.driver}"/>
        <property name="initialSize" value="5"/>
        <property name="minIdle" value="5"/>
        <property name="maxActive" value="50"/>
    </bean>


    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"></property>
        <!-- 引入 mybatis 配置文件 -->
        <property name="configLocation"
value="classpath:mybatis/SqlMapConfig.xml"></property>
         <property name="typeAliasesPackage" value="com.light.domain">
</property>
        <!-- sql配置文件 -->
         <property name="mapperLocations"
value="classpath:mybatis/mapper/*.xml"></property>
    </bean>

    <!-- 扫描Mapper -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.light.mapper"></property>
        <property name="sqlSessionFactoryBeanName"
```

```
value="sqlSessionFactory"></property>
    </bean>


    <!-- 事务管理器 -->
    <bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <!-- 通知 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <!-- 传播行为 -->
            <tx:method name="save*" propagation="REQUIRED"/>
            <tx:method name="insert*" propagation="REQUIRED"/>
            <tx:method name="delete*" propagation="REQUIRED"/>
            <tx:method name="update*" propagation="REQUIRED"/>
            <tx:method name="find*" propagation="SUPPORTS" read-
only="true"/>
            <tx:method name="get*" propagation="SUPPORTS" read-
only="true"/>
            <tx:method name="query*" propagation="SUPPORTS" read-
only="true"/>
        </tx:attributes>
    </tx:advice>

    <!-- 切面 -->
    <bean id="dataSourceAspect"
class="com.light.dynamicdatasource.DataSourceAspect"></bean>

    <aop:config proxy-target-class="true">
        <aop:pointcut id="myPointcut" expression="execution(*
com.light.service.*.*(..))" />
        <!-- 事务切面 -->
        <aop:advisor advice-ref="txAdvice" pointcut-ref="myPointcut"/>
        <!-- 自定义切面 -->
        <aop:aspect ref="dataSourceAspect" order="-9999">
            <aop:before method="before" pointcut-ref="myPointcut" />
        </aop:aspect>
    </aop:config>
```

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

</beans>
```

四、测试

笔者在项目的 web 层写了 UserController 类，里边包含 get 和 delete 两个方法。

正常情况，当访问 get 方法（读操作）时，使用从库数据源，那么控制台应该打印 slave 。

正常情况，当访问 delete 方法（写操作）时，使用主库数据源，那么控制台应该打印 master 。

以下是 2 次测试结果：

get 方法：

```
 dataSourceKey ======> slave
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The driver is automatically regis
2018-03-12 16:03:42,026 [http-bio-8080-exec-1] INFO  [com.alibaba.druid.pool.DruidDataSource] - {dataSource-1} inited
 2018-03-12 16:03:42,032 [http-bio-8080-exec-1] DEBUG [org.springframework.jdbc.datasource.DataSourceUtils] - Registering transaction synchroniz
 2018-03-12 16:03:42,035 [http-bio-8080-exec-1] DEBUG [org.mybatis.spring.transaction.SpringManagedTransaction] - JDBC Connection [com.mysql.cj.
 2018-03-12 16:03:42,039 [http-bio-8080-exec-1] DEBUG [com.light.mapper.UserMapper.getById] - ==>  Preparing: select id from user where id = ?
 2018-03-12 16:03:42,059 [http-bio-8080-exec-1] DEBUG [com.light.mapper.UserMapper.getById] - ==> Parameters: 1(Integer)
 2018-03-12 16:03:42,073 [http-bio-8080-exec-1] DEBUG [com.light.mapper.UserMapper.getById] - <==      Total: 1
```

delete 方法：

```
 dataSourceKey ======> master
2018-03-12 16:04:46,925 [http-bio-8080-exec-3] INFO  [com.alibaba.druid.pool.DruidDataSource] - {dataSource-2} inited
 2018-03-12 16:04:46,925 [http-bio-8080-exec-3] DEBUG [org.springframework.jdbc.datasource.DataSourceTransactionManager] - Acquired Connection [
 2018-03-12 16:04:46,925 [http-bio-8080-exec-3] DEBUG [org.springframework.jdbc.datasource.DataSourceTransactionManager] - Switching JDBC Connec
 2018-03-12 16:04:46,926 [http-bio-8080-exec-3] DEBUG [org.mybatis.spring.SqlSessionUtils] - Creating a new SqlSession
 2018-03-12 16:04:46,926 [http-bio-8080-exec-3] DEBUG [org.mybatis.spring.SqlSessionUtils] - Registering transaction synchronization for SqlSess
 2018-03-12 16:04:46,927 [http-bio-8080-exec-3] DEBUG [org.mybatis.spring.transaction.SpringManagedTransaction] - JDBC Connection [com.mysql.cj.
 2018-03-12 16:04:46,927 [http-bio-8080-exec-3] DEBUG [com.light.mapper.UserMapper.deleteById] - ==>  Preparing: delete from user where id = ?
 2018-03-12 16:04:46,927 [http-bio-8080-exec-3] DEBUG [com.light.mapper.UserMapper.deleteById] - ==> Parameters: 1(Integer)
 2018-03-12 16:04:46,929 [http-bio-8080-exec-3] DEBUG [com.light.mapper.UserMapper.deleteById] - <==    Updates: 1
```

五、源码