spring实现数据库的自动切换

我们要实现的是多个数据库之间的自动切换，因此面临的主要问题就是：

1. 数据源是否有效的检测
2. 程序中切换使用的数据库

数据源的有效检测方法：

单开一条线程，间断的发送一条命令执行，如果执行成功则认为数据库有效，不成功则认为该数据源无效。

数据库的切换：

Spring提供了一个抽象类 `AbstractRoutingDataSource` ，实类中有一个抽象方法

`protected Object determineCurrentLookupKey()`

实现这个方法返回要使用的数据源的key就能够使用对应的数据源做数据库的操作。

因此两个基本的问题就都解决了，接下来就是代码实现了。

这里省去基本的spring工程的各类配置步骤，以及数据库的互备和数据的导入步骤。

下面示例工程使用的是spring+mybatis+druid连接池作为数据库操作的框架。两个数据库主主互备，并且将工程使用的数据已经导入数据库中。在此基础上编写我们切换数据库的代码。

首先按照传统的方式配置数据源，配置方式和以前spring工程配置数据源的方式相同。

如下图所示，我使用druid配置了两个数据源。分别是*dataSourceTest1*和*dataSourceTest2*

```
    <bean id="dataSourceTest1" class="com.alibaba.druid.pool.DruidDataSource"
destroy-method="close" init-method="init"
        lazy-init="true">
        <property name="driverClassName" value="${jdbc.driver}" />
        <property name="url" value="${jdbc.test1.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>

    <bean id="dataSourceTest2" class="com.alibaba.druid.pool.DruidDataSource"
destroy-method="close" init-method="init"
        lazy-init="true">
        <property name="driverClassName" value="${jdbc.driver}" />
        <property name="url" value="${jdbc.test2.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>
```

Jdbc配置属性如下：

```
jdbc.driver=com.mysql.cj.jdbc.Driver
jdbc.test1.url=jdbc:mysql://127.0.0.1:3307/gwpms?
useSSL=false&useUnicode=true&characterEncoding=utf-
8&serverTimezone=PRC&autoReconnect=true
jdbc.test2.url=jdbc:mysql://127.0.0.1:3308/gwpms?
useSSL=false&useUnicode=true&characterEncoding=utf-
8&serverTimezone=PRC&autoReconnect=true
jdbc.username=root
jdbc.password=123456
```

可以看到两个数据源分别使用两个不同的数据库。

然后就是使用 AbstractRoutingDataSource 的实现类，配置一个数据源提供给程序使用。

```
    <bean id="dataSourceTest"
class="com.starnet.gwpms.web.MultipleDataSourceToChoose" lazy-init="true">
        <description>数据源</description>
        <property name="targetDataSources">
            <map key-type="java.lang.String" value-type="javax.sql.DataSource">
                <entry key="dataSourceTest1" value-ref="dataSourceTest1" />
                <entry key="dataSourceTest2" value-ref="dataSourceTest2" />
            </map>
        </property>
        <!-- 设置默认的目标数据源 -->
        <property name="defaultTargetDataSource" ref="dataSourceTest1" />
    </bean>
```

MultipleDataSourceToChoose 是我定义的AbstractRoutingDataSource 的实现类，观察配置可以看见配置了一个*targetDataSources*，当中配置的就是我们之前配置的数据源bean。使用map key-value的方式配置。这里的key就是我们要在determineCurrentLookupKey()方法中返回的值，返回key后，连接会切换到对应的数据源。

MultipleDataSourceToChoose 的实现如下：

```
package com.starnet.gwpms.web;
import org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;
public class MultipleDataSourceToChoose extends AbstractRoutingDataSource {

    private static String dataSourceKey;

    public static void setDataSourceKey(String dataSource) {
        dataSourceKey = dataSource;
    }

    /**
     * @desction: 根据Key获取数据源的信息，上层抽象函数的钩子
     * @param:
     * @return:
     */
```

```
    @Override
    protected Object determineCurrentLookupKey() {
        return dataSourceKey;
    }
}
```

这里还提供了一个静态方法设置使用的数据源。

之后再配置一个bean，同样的把所有的数据源注入进去，方便在当中检查数据源的有效性。

```
    <bean id="dataSourceContextHolder"
class="com.starnet.gwpms.web.DataSourceContextHolder">
        <description>数据源</description>
        <property name="targetDataSources">
            <map key-type="java.lang.String" value-type="javax.sql.DataSource">
                <entry key="dataSourceTest1" value-ref="dataSourceTest1" />
                <entry key="dataSourceTest2" value-ref="dataSourceTest2" />
            </map>
        </property>
        <!-- 设置默认的目标数据源 -->
        <property name="defaultTargetDataSource" ref="dataSourceTest1" />
    </bean>
```

*DataSourceContextHolder*的代码实现。主要做了以下工作：

- 通过配置文件将所有数据源注入成员变量当中。
- 单开一条线程间断的查询数据库，通过查询结果判断数据库的有效性。使用的数据库无效则切换数据库。

代码如下：

```
package com.starnet.gwpms.web;

import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

import javax.sql.DataSource;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

public class DataSourceContextHolder implements InitializingBean {
```

```
    private static Logger logger =
LoggerFactory.getLogger(DataSourceContextHolder.class);

    //配置中注入所有的数据源
    private Map<String, DataSource> targetDataSources;

    private DataSource defaultTargetDataSource;

    private Map<String, Boolean> validDataSources = new HashMap<String, Boolean>();

    private String useDataSourceKey;


    public void setTargetDataSources(Map<String, DataSource> targetDataSources) {
        this.targetDataSources = targetDataSources;
    }

    public void setDefaultTargetDataSource(DataSource defaultTargetDataSource) {
        this.defaultTargetDataSource = defaultTargetDataSource;
    }

    @Override
    public void afterPropertiesSet() throws Exception {
        for (Entry<String, DataSource> dataSource: targetDataSources.entrySet()) {
            validDataSources.put(dataSource.getKey(), true);
            if (defaultTargetDataSource == dataSource.getValue()) {
                useDataSourceKey = dataSource.getKey();
            }
        }
        if (useDataSourceKey == null) {
            logger.error("default datasource key is null.");
        }
        new Thread(new DataSourceValidate(), "DATASOURCE-VALIDATE-THREAD").start();
    }

    //数据源有效性检测线程，每隔1秒检测一次
    public class DataSourceValidate implements Runnable {

        @Override
        public void run() {
            while (true) {
                for (Entry<String, DataSource> dataSource:
targetDataSources.entrySet()) {
                    if (checkDataSource(dataSource.getValue())) {
                        validDataSources.put(dataSource.getKey(), true);
                    } else {
                        validDataSources.put(dataSource.getKey(), false);
                    }
                }
                //如果正在使用的数据源不可用，自动切换到可用的数据源上
                if (!validDataSources.get(useDataSourceKey)) {
                    for (Entry<String, Boolean> validDataSource:
validDataSources.entrySet()) {
```

```
                        if (validDataSource.getValue()) {
                            useDataSourceKey = validDataSource.getKey();
                            logger.debug("change datasource, {}", useDataSourceKey);

MultipleDataSourceToChoose.setDataSourceKey(validDataSource.getKey());
                            break;
                        }
                    }
                }
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    logger.debug(e.getMessage(), e);
                }
            }
        }

        public boolean checkDataSource(DataSource ds) {
            try (
                Connection conn = ds.getConnection();
                Statement stmt = conn.createStatement();
            ) {
                stmt.execute("select 1");
                return true;
            } catch (SQLException e) {
                logger.debug(e.getMessage(), e);
                return false;
            }
        }
    }

}
```