# 单个数据源

这里修改为一个数据源了。一个用到三个类，如下：这种方式在项目启动时直接将数据源和sqlSessionFactory注入到Spring容器中。

## 类一

|  |
| --- |
| package com.thunisoft.jy.exchange.datasource;  import java.util.HashMap;  import java.util.Map;  import javax.sql.DataSource;  import org.apache.ibatis.session.SqlSessionFactory;  import org.springframework.beans.factory.annotation.Qualifier;  import org.springframework.boot.context.properties.ConfigurationProperties;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.context.annotation.Primary;  import com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceBuilder;  import com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean;  @Configuration  public class MyBatiesPlusConfiguration {  @Bean(name = "db1")  @ConfigurationProperties(prefix = "spring.datasource.druid")  public DataSource db1() {  return DruidDataSourceBuilder.create().build();  }    /\*\*  \* 动态数据源配置  \* @return  \*/  @Bean  @Primary  public DataSource multipleDataSource(@Qualifier("db1") DataSource db1) {  DynamicDataSource multipleDataSource = new DynamicDataSource();  Map<Object, Object> targetDataSources = new HashMap<>();  targetDataSources.put("db1", db1);    // 添加数据源  multipleDataSource.setTargetDataSources(targetDataSources);  // 设置默认数据源  multipleDataSource.setDefaultTargetDataSource(db1);  return multipleDataSource;  }  @Bean("sqlSessionFactory")  public SqlSessionFactory sqlSessionFactory() throws Exception {  MybatisSqlSessionFactoryBean sqlSessionFactory = new MybatisSqlSessionFactoryBean();  sqlSessionFactory.setDataSource(multipleDataSource(db1()));  return sqlSessionFactory.getObject();  }  } |

## 类二

|  |
| --- |
| package com.thunisoft.jy.exchange.datasource;  import org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;  /\*\*  \*  \* @ClassName: DynamicDataSource  \* @Description: 动态数据源  \* @author: xxj  \* @date: 2018年6月11日 下午3:37:08  \*/  public class DynamicDataSource extends AbstractRoutingDataSource {  /\* (non Javadoc)  \* @Title: determineCurrentLookupKey  \* @Description: 重写方法达到动态切换数据库  \* @return  \* @see org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource#determineCurrentLookupKey()  \*/  @Override  protected Object determineCurrentLookupKey() {  return DataSourceContextHolder.getDataSourceType();  }    } |

## 类三

|  |
| --- |
| package com.thunisoft.jy.exchange.datasource;  /\*\*  \* @ClassName: DataSourceContextHolder  \* @Description: 动态切换数据源工具类  \* @author: xxj  \* @date: 2018年6月11日 下午3:38:01  \*/  public final class DataSourceContextHolder {    private DataSourceContextHolder(){}    private static final ThreadLocal<String> contextHolder = new ThreadLocal<>();  public static void setDataSourceType(String dataSourceType) {  contextHolder.set(dataSourceType);  }  public static String getDataSourceType() {  return (contextHolder.get());  }  public static void clearDataSourceType() {  contextHolder.remove();  }  } |

# 多个数据源

将上面的第一个类修改如下：

|  |
| --- |
| package com.thunisoft.jy.exchange.datasource;  import java.util.HashMap;  import java.util.Map;  import javax.sql.DataSource;  import org.apache.ibatis.session.SqlSessionFactory;  import org.springframework.beans.factory.annotation.Qualifier;  import org.springframework.boot.context.properties.ConfigurationProperties;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.context.annotation.Primary;  import com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceBuilder;  import com.baomidou.mybatisplus.extension.plugins.PerformanceInterceptor;  import com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean;  @Configuration  public class MyBatiesPlusConfiguration {  /\*\*  \* SQL执行效率插件  \*/  @Bean  public PerformanceInterceptor performanceInterceptor() {  PerformanceInterceptor performanceInterceptor = new PerformanceInterceptor();  performanceInterceptor.setMaxTime(1000);  performanceInterceptor.setFormat(true);  return performanceInterceptor;  }  @Bean(name = "db1")  @ConfigurationProperties(prefix = "spring.datasource.druid.jgpt")  public DataSource db1() {  return DruidDataSourceBuilder.create().build();  }  @Bean(name = "db2")  @ConfigurationProperties(prefix = "spring.datasource.druid.jxjs")  public DataSource db2() {  return DruidDataSourceBuilder.create().build();  }  @Bean(name = "db3")  @ConfigurationProperties(prefix = "spring.datasource.druid.bjcm")  public DataSource db3() {  return DruidDataSourceBuilder.create().build();  }  /\*\*  \* 动态数据源配置  \* @return  \*/  @Bean  @Primary  public DataSource multipleDataSource(@Qualifier("db1") DataSource db1, @Qualifier("db2") DataSource db2,  @Qualifier("db3") DataSource db3) {  DynamicDataSource multipleDataSource = new DynamicDataSource();  Map<Object, Object> targetDataSources = new HashMap<>();  targetDataSources.put(DataSourceType.JGPT.getDataSourceType(), db1);  targetDataSources.put(DataSourceType.JXJS.getDataSourceType(), db2);  targetDataSources.put(DataSourceType.BJCM.getDataSourceType(), db3);  // 添加数据源  multipleDataSource.setTargetDataSources(targetDataSources);  // 设置默认数据源  multipleDataSource.setDefaultTargetDataSource(db1);  return multipleDataSource;  }  @Bean("sqlSessionFactory")  public SqlSessionFactory sqlSessionFactory() throws Exception {  MybatisSqlSessionFactoryBean sqlSessionFactory = new MybatisSqlSessionFactoryBean();  sqlSessionFactory.setDataSource(multipleDataSource(db1(), db2(), db3()));  return sqlSessionFactory.getObject();  }  } |

这里注入sqlSessionFactory是关键所在.

# sqlSessionFactory的创建

--> ①定义Configuration对象(包括数据源，事务，mapper文件资源以及影响数据库行为属性设置的settings)

--> ②由Configuration对象创建一个SqlSessionFactoryBuilder对象

--> ③由SessionFactoryBuilder获得SqlSessionFactory实例

--> ④由SqlSessionFactory实例获得SqlSession实例，操作数据库

这里是通过SqlSessionFactoryBuilder创建SqlSessionFactory的。但是在实际中还有两种常见的方式。

## 方式一SqlSessionFactoryBuilder

## 方式二SqlSessionFactoryBean

代替SqlSessionFactoryBuilder来创建sqlSessionFactory。

### Sql

例如实体中CBjcmCode，库中C\_BJCM\_CODE。必须采用别化名方式才能进行封装：C\_BJCM\_CODE AS CBjcmCode

### 实现的接口

FactoryBean<SqlSessionFactory>

InitializingBean

ApplicationListener<ApplicationEvent>

## 方式三MybatisSqlSessionFactoryBean

代替SqlSessionFactoryBuilder来创建sqlSessionFactory。

其实在内部是通过这个MybatisSqlSessionFactoryBuilder

而这个继承SqlSessionFactoryBuilder

### Sql

能够完成sql中大小写转换，

例如实体中CBjcmCode，库中C\_BJCM\_CODE。能够自动完成封装

### 实现的接口

FactoryBean<SqlSessionFactory>

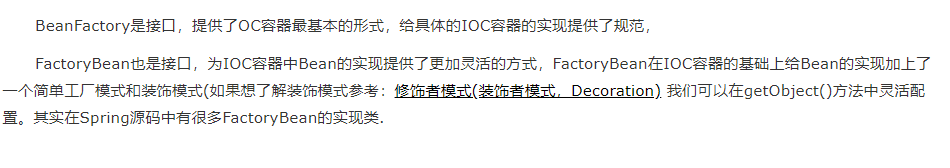
InitializingBean

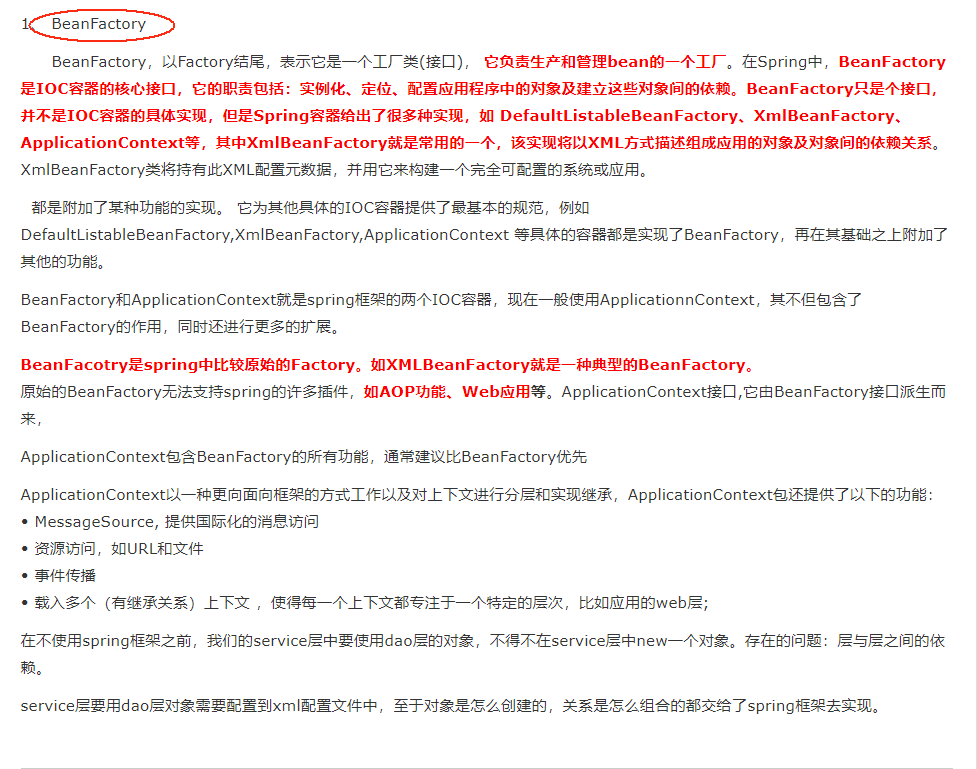
ApplicationListener<ApplicationEvent>

## 方式二方式三实现相同的接口

### FactoryBean和BeanFactory的区别

#### BeanFactory

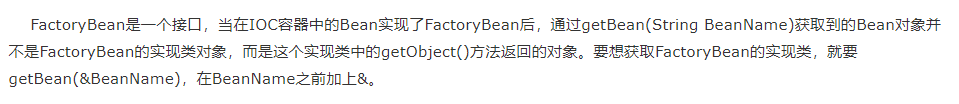






#### FactoryBean





### InitializingBean

在Spring.docx中介绍

### ApplicationListener

在Spring.docx中介绍