**SpringBoot多数据源解决方案**



[MrXionGe](https://me.csdn.net/qq_31226223) 2018-12-28 16:09:58 https://csdnimg.cn/release/blogv2/dist/pc/img/articleReadEyes.png 19726 https://csdnimg.cn/release/blogv2/dist/pc/img/tobarCollect.png 收藏 61

分类专栏： [iSpring](https://blog.csdn.net/qq_31226223/category_8569217.html) 文章标签： [spring boot](https://so.csdn.net/so/search/s.do?q=spring%20boot&t=blog&o=vip&s=&l=&f=&viparticle=) [datasource](https://www.csdn.net/gather_23/MtTaEg0sMzY3MTgtYmxvZwO0O0OO0O0O.html) [多数据源](https://www.csdn.net/gather_2d/MtTaEg0sMzM0MzMtYmxvZwO0O0OO0O0O.html)

版权

**最近在做一个数据统计的项目，这里涉及到多个数据库的数据统计，所以找到了一个不错的解决方案。**

**写一篇博客作为笔记。**

**例子介绍**

**类库：例子中使用的是MyBatis Plus和Dynamic Datasource的组合**

**开源项目地址：**[MyBatis Plus](https://mp.baomidou.com/)**&**[Dynamic Datasource](https://gitee.com/baomidou/dynamic-datasource-spring-boot-starter)

**Maven配置示例**

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-boot-starter</artifactId>

<version>3.0.6</version>

</dependency>

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>dynamic-datasource-spring-boot-starter</artifactId>

<version>2.4.2</version>

</dependency>

**配置文件**

**这里的配置使用的是yml文件，主要的配置就是datasource，别的配置都是陪衬。**

**配置文件示例：**

server:

port: 2080

logging:

level:

cn.mrxionge.bootdemo: debug

org.springframework.web: debug

org.springframework.data: debug

mybatis-plus:

executor-type: simple

spring:

datasource:

#配置hikari连接池

hikari:

minimum-idle: 4

maximum-pool-size: 16

connection-timeout: 10000

idle-timeout: 30000

connection-init-sql: set names utf8mb4

#动态数据源配置

dynamic:

#主数据源

primary: a01

datasource:

#数据源a01

a01:

driver-class-name: com.mysql.cj.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/a01?characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai

username: root

password: 123456

#数据源a02

a02:

driver-class-name: com.mysql.cj.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/a02?characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai

username: root

password: 123456

#数据源a03

a03:

driver-class-name: com.mysql.cj.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/a03?characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai

username: root

password: 123456

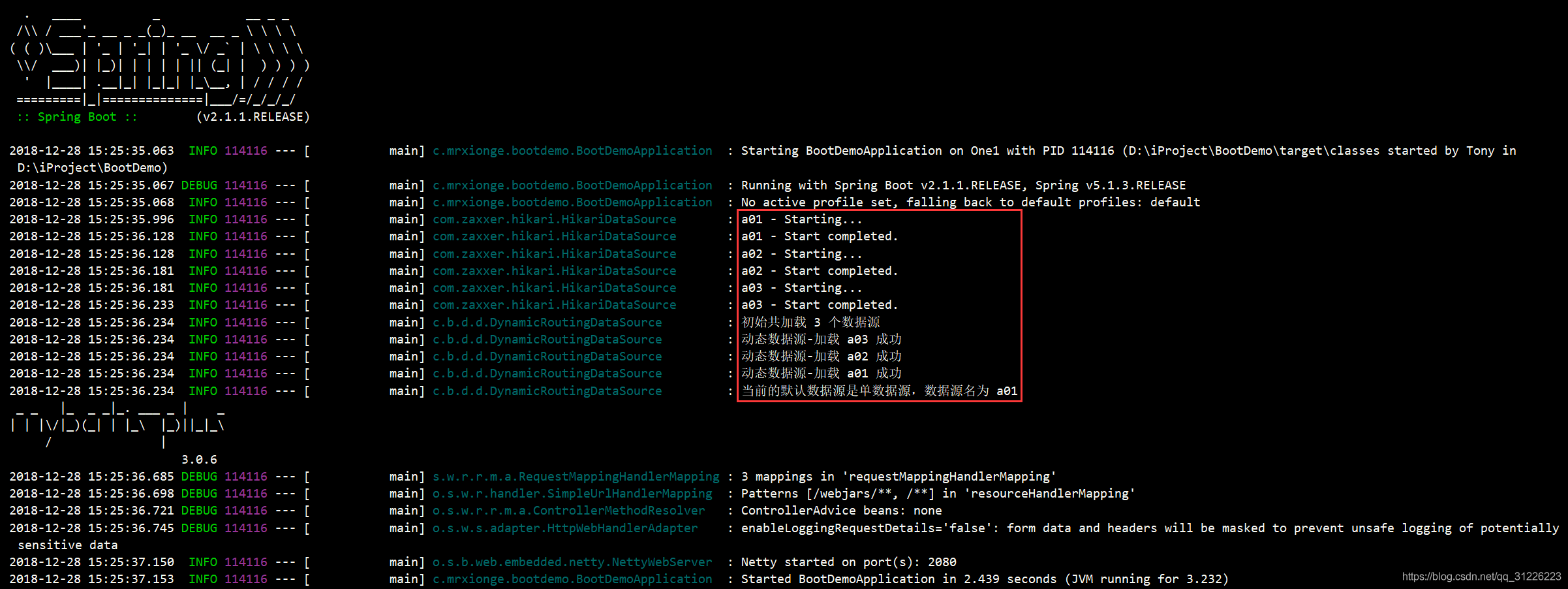
**这里需要注意一个问题，就是primary配置是必填项，这个作为全局的默认数据源。**

**每个数据源可以是不同机器上面的不同的数据库实例。例子中给的配置只是一个demo，所以是同一个MySQL实例的三个库而已，虽然简单，但是道理是一样的。**

**还有就是配置多数据源的时候，IDEA目前没有提示，自己多留意一下缩进就好了。**

**配置完成之后，我们可以通过启动程序看日志打印来查看是否配置成功。**

**日志示例：**

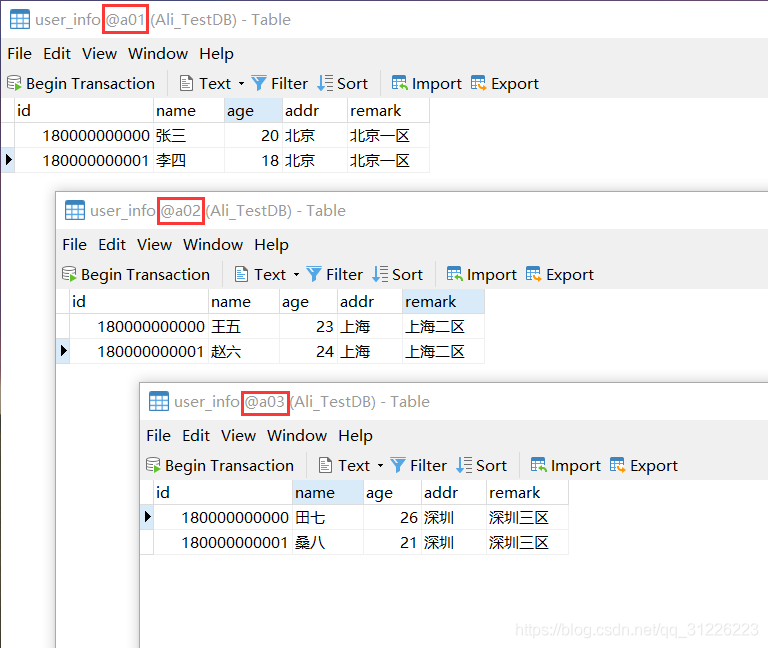


**中间红色方框的内容就是我们多数据源的加载信息。**

**添加测试数据**

**我们的示例比较简单，是三个不同的数据库中，三个同样结构的数据表，但是其中的数据不同。**

**数据库信息示例：**



**如图，我们在三个库中写入不同的数据，方便我们测试**

**动态数据源的简单实用**

**这里我们新建三个不用的Mapper文件，每个Mapper文件对应一个数据源，我们通过调用不同的Mapper文件中的方法来实现查询不同的数据库中的数据。**

**Mapper文件示例：**

@DS(value = "a01")

@Mapper

public interface MapperA01 extends BaseMapper<UserInfo> {

@Select("SELECT \* FROM user\_info")

List<UserInfo> getAllUser();

}

@DS(value = "a02")

@Mapper

public interface MapperA02 extends BaseMapper<UserInfo> {

@Select("SELECT \* FROM user\_info")

List<UserInfo> getAllUser();

}

@Mapper

public interface MapperA03 extends BaseMapper<UserInfo> {

@Select("SELECT \* FROM user\_info")

List<UserInfo> getAllUser();

}

**这里我们创建了三个Mapper，每个Mapper文件上面都有一个“@DS”注解，这个注解就是指定这个Mapper接口里面的所有的方法都使用注解里面所配置的数据源。当然，DS注解的使用方法非常灵活，这里制作一个简单的例子，给出一个思路，方便理解和入门，详细的使用，请参考他们的官方文档。**

***注意：“@DS”注解请使用在方法上面，在类上面使用会出问题，上面的例子是之前的版本，新版本在Mapper上面使用会出现异常。***

**测试阶段**

**这里我们再写一个业务层和控制层，来测试我们的多数据源查询方法。**

@Service

public class AppService {

@Autowired

private MapperA01 mapperA01;

@Autowired

private MapperA02 mapperA02;

@Autowired

private MapperA03 mapperA03;

/\*\*

\* 查询A01库中的数据

\*

\* @return 用户信息列表

\*/

List<UserInfo> getUser01() {

return mapperA01.getAllUser();

}

/\*\*

\* 查询A02库中的数据

\*

\* @return 用户信息列表

\*/

List<UserInfo> getUser02() {

return mapperA02.getAllUser();

}

/\*\*

\* 查询A03库中的数据

\*

\* @return 用户信息列表

\*/

List<UserInfo> getUser03() {

return mapperA03.getAllUser();

}

}

@RestController

public class AppHandler {

@Autowired

private AppService service;

@GetMapping(path = "/get01")

public Mono get01() {

//查询A01数据

return Mono.fromSupplier(service::getUser01);

}

@GetMapping(path = "/get02")

public Mono get02() {

//查询A02数据

return Mono.fromSupplier(service::getUser02);

}

@GetMapping(path = "/get03")

public Mono get03() {

//查询A03数据

return Mono.fromSupplier(service::getUser03);

}

}

**调用URL进行测试：**



**测试完成，是我们想要的结果。**

**结尾**

**以上的例子很简单，但并不意味着Dynamic Datasource的功能只有这些，详细的使用请大家去看这个项目的文档，结合自身的业务场景来选择一个最优的解决档案。当然，Dynamic Datasource应该是我使用中最简单最快捷最方便的多数据源解决方案**

**如果你使用的不是MyBatis Plus，而是MyBatis。嗯……答案是，没有任何问题！**

***[2020年2月18日补充]：这篇博客写了挺久了，我也没有做到保持更新，而且这个开源项目已经迭代了数个版本，增加了很多新的功能。如果这篇博客让你知道了有’dynamic-datasource’这个好东西，我觉得也足够了，因为学习的最好办法，就是去看第一手资料。那里有更多高阶且详细的使用法方，而我上面给出的例子只是最最初级操作。***

原文地址：

<https://blog.csdn.net/qq_31226223/article/details/85322561?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.channel_param&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.channel_param>

注意：

遇到的问题：

问题：启动时会提示找不到：

Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.

Reason: Failed to determine a suitable driver class

Action:

Consider the following:

If you want an embedded database (H2, HSQL or Derby), please put it on the classpath.

If you have database settings to be loaded from a particular profile you may need to activate it (no profiles are currently active).

原因：

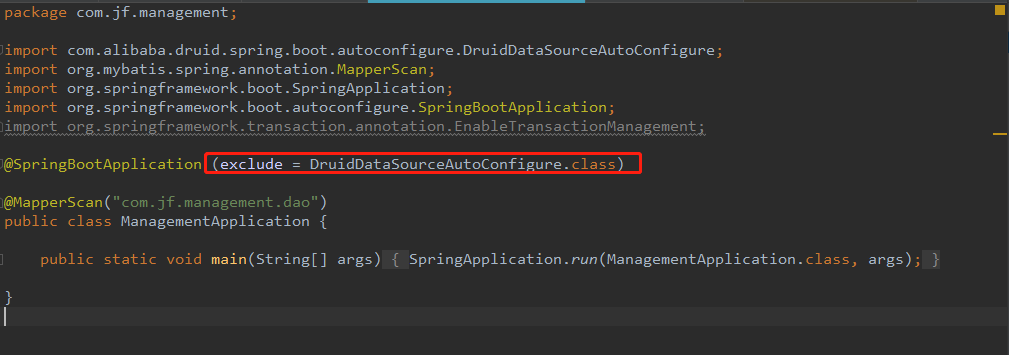
* DruidDataSourceAutoConfigure会注入一个DataSourceWrapper，其会在原生的spring.datasource下找 url, username, password 等。动态数据源 URL 等配置是在 dynamic 下，因此需要排除，否则会报错。排除方式有两种，一种是 配置文件排除，还有一种可以在项目启动类排除

解决：

1. 添加druid-spring-boot-starter依赖

<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid-spring-boot-starter</artifactId>  
 <version>1.1.20</version>  
</dependency>

在启动类跳过（或者配置文件：autoconfigure.exclude= com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceAutoConfigure # 去除druid配置）



exclude = DruidDataSourceAutoConfigure.class

问题：

可以启动但是mapper绑定不了：

Invalid bound statement (not found): com.jf.management.dao.user.UserMapper.findByUsername

解决方法

1.增加配置类，配置指定映射路径

|  |
| --- |
| package com.jf.management.config;  import org.apache.ibatis.session.SqlSessionFactory; import org.mybatis.spring.SqlSessionFactoryBean; import org.springframework.beans.factory.annotation.Qualifier; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import org.springframework.context.annotation.Primary; import org.springframework.core.io.support.PathMatchingResourcePatternResolver;  import javax.sql.DataSource; import java.io.IOException;  @Configuration public class Datasource {   */\*\*  \* 创建会话工厂。  \** ***@param*** *baseDataSource 数据源  \*  \*  \** ***@return*** *会话工厂  \*/* @Bean("sqlSessionFactory")  @Primary  public SqlSessionFactory getSqlSessionFactory(@Qualifier("dataSource") DataSource baseDataSource) {   SqlSessionFactoryBean bean = new SqlSessionFactoryBean();  bean.setDataSource(baseDataSource);  try {  //指定mapper路径  bean.setMapperLocations(new PathMatchingResourcePatternResolver().getResources("classpath:mapper/\*\*/\*.xml"));  } catch (IOException e1) {  e1.printStackTrace();  }  SqlSessionFactory sqlSessionFactory = null;  try {  sqlSessionFactory = bean.getObject();  } catch (Exception e) {  e.printStackTrace();  }  return sqlSessionFactory;  }   }  <https://ld246.com/article/1542253561209> |

参考：

<https://www.cnblogs.com/aizen-sousuke/p/11756279.html>