前置环境

- jdk1.8 +
- maven3.0 +
- eclipse Version: Oxygen.3a Release (4.7.3a)
- eclipse lombok插件
- mysql5.6
- spring boot 2.0+

静态资源访问

放在/src/main/resources/static目录即可
如：创建了一个文件在/src/main/resources/static/images/test.png
访问：http://localhost:8080/images/test.png

整合lombok

简化pojo的get/set操作，log的注解等。提供一些注解简化编码
参考整合：https://blog.csdn.net/Dorothy1224/article/details/79280591/
参考文档：http://jnb.ociweb.com/jnb/jnbJan2010.html

整合Freemarker

假如依赖

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-freemarker</artifactId>
        </dependency>
```

在src/main/resources/创建一个templates文件夹，templates是默认的模板文件夹，也可以在templates下创建层级的目录。在templates目录/层级目录创建后缀为*.ftl的文件，如：
/src/main/resources/templates/user/info.ftl

```
<!DOCTYPE html>
<html>
<head lang="en">
<meta charset="UTF-8" />
<title></title>
</head>
<body>
<div>用户名：${(user.userName)!""}</div>
<div>手机：${(user.mobile)!""}</div>
<div>测试：${(test)!""}</div>
</body>
</html>
```

测试controller

```
    @RequestMapping("/user/info")
    public String index(Model model, String username) {
        model.addAttribute("user",
userService.getUserByName(username));
        model.addAttribute("test", "test");
        return "/user/info";
    }
```

log4j整合

引入依赖

```
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-log4j</artifactId>
            <version>1.3.8.RELEASE</version>
        </dependency>
```

自定义日志类，warn 和 error 日志分开输出

```
package com.zhuozhi.common.log4j;

import org.apache.log4j.DailyRollingFileAppender;
import org.apache.log4j.Priority;

/**
 * 名称：BaseLeveDailyRollingFileAppender <br>
 * 描述：自定义日志文件类，使日志记录的级别等于定义的级别，不进行继承关系<br>
 *
 */
public class BaseLeveDailyRollingFileAppender extends DailyRollingFileAppender {
    @Override
    public boolean isAsSevereAsThreshold(Priority priority) {
        //只判断是否相等，而不判断优先级
        return this.getThreshold().equals(priority);
    }
}
```

添加log4j.properties

```
#输出需要的日志
log4j.rootLogger=DEBUG,stdout,info,warn,error

#控制台输出
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.Threshold=DEBUG
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss SSS}|%5p|%F.%M:%L|%m%n

#INFO日志
log4j.logger.info=info
log4j.appender.info=org.apache.log4j.DailyRollingFileAppender
log4j.appender.info.File=./logs/info/info.log
log4j.appender.info.datePattern='.'yyyy-MM-dd'.log'
log4j.appender.info.append=true
log4j.appender.info.Threshold=INFO
log4j.appender.info.encoding=UTF-8
log4j.appender.info.ImmediateFlush=true
log4j.appender.info.layout=org.apache.log4j.PatternLayout
log4j.appender.info.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss SSS}|%5p|%F.%M:%L|%m%n
#WARN日志
log4j.appender.warn=com.zhuozhi.common.log4j.BaseLeveDailyRollingFileAppender
log4j.appender.warn.File=./logs/warn/warn.log
log4j.appender.warn.datePattern='.'yyyy-MM-dd'.log'
log4j.appender.warn.append=true
log4j.appender.warn.Threshold=WARN
log4j.appender.warn.encoding=UTF-8
log4j.appender.warn.ImmediateFlush=true
log4j.appender.warn.layout=org.apache.log4j.PatternLayout
log4j.appender.warn.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss SSS}|%5p|%F.%M:%L|%m%n
#ERROR日志
```

```
log4j.appender.error=com.zhuozhi.common.log4j.BaseLeveDailyRollingFileAppender
log4j.appender.error.File=./logs/error/error.log
log4j.appender.error.datePattern='.'yyyy-MM-dd'.log'
log4j.appender.error.append=true
log4j.appender.error.Threshold=ERROR
log4j.appender.error.encoding=UTF-8
log4j.appender.error.ImmediateFlush=true
log4j.appender.error.layout=org.apache.log4j.PatternLayout
log4j.appender.error.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss SSS}|%5p|%F.%M:%L|%m%n
```

测试类

```
//导入包
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

//获取logger
private Logger logger = LoggerFactory.getLogger(this.getClass());

//再方法中
logger.debug("XXXX");
logger.info("XXXX");
//等测试即可
```

也可以用lombok方式

```
//在类上面加上注释
@Slf4j

//方法可以直接
log.info("XXXX");
```

全局异常处理

参考：

https://blog.csdn.net/bwf_erg/article/details/53813115

https://blog.csdn.net/chwshuang/article/details/48089203

1、在resource/templates/error下添加error.html页面(error替换成对应的错误码，404、401、500等，还可以用4xx、5xx等)，springBoot会自动找到该页面作为错误页面

SpringBoot错误视图提供了以下错误属性：

- timestamp：错误发生时间；
- status：HTTP状态吗；
- error：错误原因；
- exception：异常的类名；
- message：异常消息（如果这个错误是由异常引起的）；
- errors：BindingResult异常里的各种错误（如果这个错误是由异常引起的）；
- trace：异常跟踪信息（如果这个错误是由异常引起的）；
- path：错误发生时请求的URL路径。

2、业务异常和运行时相关的异常

自定义结果类，根据需要可以添加其他属性

```
package com.zhuozhi.result;

/**
 * <p>
 * <b>Result</b> is 包裹结果的基类
 * </p>
 *
 */
public class Result<T> {

    private String code;

    private String msg;

    private T data;
```

```
16
17      public Result(String code, String msg, T data) {
18          this.code = code;
19          this.msg = msg;
20          this.data = data;
21      }
22
23      public Result(String code, String msg) {
24          this.code = code;
25          this.msg = msg;
26      }
27
28      public String getCode() {
29          return code;
30      }
31
32      public void setCode(String code) {
33          this.code = code;
34      }
35
36      public String getMsg() {
37          return msg;
38      }
39
40      public void setMsg(String msg) {
41          this.msg = msg;
42      }
43
44      public T getData() {
45          return data;
46      }
47
48      public void setData(T data) {
49          this.data = data;
50      }
51
52  }
```

自定义异常类，可分不同的业务场景

```java
package com.zhuozhi.exception;

public class BusinessException extends RuntimeException {
    private static final long serialVersionUID = 8862339523905913528L;

    public BusinessException(String message) {
        super(message);
        this.message = message;
    }

    public BusinessException(String code, String message) {
        super(message);
        this.code = code;
        this.message = message;
    }

    public BusinessException(String message, Throwable cause) {
        super(message, cause);
        this.message = message;
    }

    /**
     * 错误代码
     */
    private String code;

    /**
     * 错误信息
     */
    private String message;

    /**
     * 获取 错误代码
     */
    public String getCode() {
        return code;
    }

    /**
     * 设置 错误代码
```

```
     */
    public void setCode(String code) {
        this.code = code;
    }

    /**
     * 获取 错误信息
     */
    public String getMessage() {
        return message;
    }

    /**
     * 设置 错误信息
     */
    public void setMessage(String message) {
        this.message = message;
    }

}
```

全局异常拦截类，要确保这个类被扫描到

```
package com.zhuozhi.controller;

import org.springframework.validation.BindException;
import org.springframework.validation.ObjectError;
import
org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;

import com.zhuozhi.exception.BusinessException;
import com.zhuozhi.result.Result;

import lombok.extern.slf4j.Slf4j;

/**
 * <p>
```

```
 * <b>GlobalExceptionHandler</b> is 自定义异常处理
 * </p>
 */

@ControllerAdvice
@Slf4j
public class GlobalExceptionHandler {

    /**
     * 业务异常处理
     * @param e
     * @return
     */
    @ExceptionHandler({ BusinessException.class })
    @ResponseBody
    public Result<Object>
handleBusinessException(BusinessException e) {
        log.warn("业务异常", e);
        Result<Object> result = new Result<Object>(e.getCode(),
e.getMessage());
        return result;
    }

    /**
     * @ModelAttribute 注解验证对象 出错所抛出的异常处理
     * @param e
     * @return
     */
    @ExceptionHandler({ BindException.class })
    @ResponseBody
    public Result<Object> handleBindException(BindException e) {
        log.warn("参数格式错误", e);
        ObjectError err = e.getAllErrors().get(0);
        Result<Object> result = new Result<Object>("Bxxx",
err.getDefaultMessage());
        return result;
    }

    /**
     * @RequestBody 注解验证对象 出错所抛出的异常处理
     * @param e
     * @return
```

```
     */
    @ExceptionHandler({ MethodArgumentNotValidException.class })
    @ResponseBody
    public Result<Object>
handleMethodArgumentNotValidException(MethodArgumentNotValidExcep
tion e) {
        log.warn("参数格式错误", e);
        ObjectError err =
e.getBindingResult().getAllErrors().get(0);
        Result<Object> result = new Result<Object>("Mxxx",
err.getDefaultMessage());
        return result;
    }

    /**
     * 系统异常处理
     * @param e
     * @return
     */
    @ExceptionHandler({ RuntimeException.class })
    @ResponseBody
    public Result<Object> handleRuntimeException(RuntimeException
e) {
        log.error("系统内部异常", e);
        Result<Object> result = new Result<Object>("Rxxxx",
"Exxxx");
        return result;
    }
}
```

整合mybatis

相关依赖

```
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.3.2</version>
```

```
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
```

/src/main/resources/application.yml配置

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/test?serverTimezone=CTT&useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true
    username: zero
    password: 123456
```

Dao代码

```
package com.zhuozhi.dao;

import org.apache.ibatis.annotations.Param;
import org.apache.ibatis.annotations.Select;

import com.zhuozhi.entity.User;

public interface UserDao {
 @Select("SELECT * FROM USERS WHERE USERNAME = #{username}")
 User findByName(@Param("username") String username);
}
```

启动类添加@MapperScan

```
package com.zhuozhi;
```

```
2
3  import org.mybatis.spring.annotation.MapperScan;
4  import org.springframework.boot.SpringApplication;
5  import
   org.springframework.boot.autoconfigure.SpringBootApplication;
6
7  @SpringBootApplication
8  @MapperScan("com.zhuozhi.dao")
9  public class TablerApplication {
10
11  public static void main(String[] args) {
12   SpringApplication.run(TablerApplication.class, args);
13  }
14 }
```

整合tk.mybatis

依赖

```
1         <dependency>
2             <groupId>tk.mybatis</groupId>
3             <artifactId>mapper-spring-boot-starter</artifactId>
4             <version>2.0.4</version>
5         </dependency>
```

实体类

```
1  package com.zhuozhi.entity;
2
3  import javax.persistence.Column;
4  import javax.persistence.GeneratedValue;
5  import javax.persistence.GenerationType;
6  import javax.persistence.Id;
7  import javax.persistence.Table;
8
```

```
import lombok.Data;

@Data
//假如数据库名称不一致需要指定表名
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

         //指定列名
    @Column(name = "username")
    private String userName;

    private String mobile;

    private String nickName;
}
```

dao类继承

```
public interface UserDao extends Mapper<User> {
}
```

引用包要改成

```
//替换成tk的scan
import tk.mybatis.spring.annotation.MapperScan;
@MapperScan("com.zhuozhi.dao")
```

整合druid

添加依赖

```
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.12</version>
        </dependency>
```

配置，可以参考https://www.cnblogs.com/wuyun-blog/p/5679073.html

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/test?serverTimezone=CTT&useUnicode=true&characterEncoding=utf-8&allowMultiQueries=true
    username: zero
    password: 123456

    #druid配置
    type: com.alibaba.druid.pool.DruidDataSource
    connectionProperties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000
    initialSize: 5
    minIdle: 5
    maxActive: 20
    maxWait: 60000
    poolPreparedStatements: true
    filters: stat,wall,log4j
    maxPoolPreparedStatementPerConnectionSize: 20
    testOnBorrow: false
    testWhileIdle: true
    minEvictableIdleTimeMillis: 300000
    timeBetweenEvictionRunsMillis: 60000
    testOnReturn: false
    validationQuery: SELECT 1 FROM DUAL
```

定义DataSource bean 覆盖原来的

```
```

```java
package com.zhuozhi.config;

import java.sql.SQLException;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Value;
import
org.springframework.boot.web.servlet.FilterRegistrationBean;
import
org.springframework.boot.web.servlet.ServletRegistrationBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;

import com.alibaba.druid.pool.DruidDataSource;
import com.alibaba.druid.support.http.StatViewServlet;
import com.alibaba.druid.support.http.WebStatFilter;

import lombok.extern.slf4j.Slf4j;

@Configuration
@Slf4j
public class DruidConfiguration {
 @Value("${spring.datasource.url}")
 private String dbUrl;

 @Value("${spring.datasource.username}")
 private String username;

 @Value("${spring.datasource.password}")
 private String password;

 @Value("${spring.datasource.initialSize}")
 private int initialSize;

 @Value("${spring.datasource.minIdle}")
 private int minIdle;

 @Value("${spring.datasource.maxActive}")
 private int maxActive;
```

```
40
41  @Value("${spring.datasource.maxWait}")
42  private int maxWait;
43
44  @Value("${spring.datasource.timeBetweenEvictionRunsMillis}")
45  private int timeBetweenEvictionRunsMillis;
46
47  @Value("${spring.datasource.minEvictableIdleTimeMillis}")
48  private int minEvictableIdleTimeMillis;
49
50  @Value("${spring.datasource.validationQuery}")
51  private String validationQuery;
52
53  @Value("${spring.datasource.testWhileIdle}")
54  private boolean testWhileIdle;
55
56  @Value("${spring.datasource.testOnBorrow}")
57  private boolean testOnBorrow;
58
59  @Value("${spring.datasource.testOnReturn}")
60  private boolean testOnReturn;
61
62  @Value("${spring.datasource.poolPreparedStatements}")
63  private boolean poolPreparedStatements;
64
65  @Value("${spring.datasource.maxPoolPreparedStatementPerConnecti
onSize}")
66  private int maxPoolPreparedStatementPerConnectionSize;
67
68  @Value("${spring.datasource.filters}")
69  private String filters;
70
71  @Value("{spring.datasource.connectionProperties}")
72  private String connectionProperties;
73
74  @Bean(initMethod = "init") //声明其为Bean实例，并启动初始化
75  @Primary //在同样的DataSource中，首先使用被标注的DataSource
76  public DataSource dataSource() {
77   DruidDataSource datasource = new DruidDataSource();
78
79   datasource.setUrl(this.dbUrl);
80   datasource.setUsername(username);
```

```
81    datasource.setPassword(password);
82
83    //configuration
84    datasource.setInitialSize(initialSize);
85    datasource.setMinIdle(minIdle);
86    datasource.setMaxActive(maxActive);
87    datasource.setMaxWait(maxWait);
88    datasource.setTimeBetweenEvictionRunsMillis(timeBetweenEvictio
      nRunsMillis);
89    datasource.setMinEvictableIdleTimeMillis(minEvictableIdleTimeM
      illis);
90    datasource.setValidationQuery(validationQuery);
91    datasource.setTestWhileIdle(testWhileIdle);
92    datasource.setTestOnBorrow(testOnBorrow);
93    datasource.setTestOnReturn(testOnReturn);
94    datasource.setPoolPreparedStatements(poolPreparedStatements);
95    datasource.setMaxPoolPreparedStatementPerConnectionSize(maxPoo
      lPreparedStatementPerConnectionSize);
96    try {
97     datasource.setFilters(filters);
98    } catch (SQLException e) {
99     log.error("druid configuration initialization filter : {0}",
      e);
100   }
101   datasource.setConnectionProperties(connectionProperties);
102
103   return datasource;
104  }
105
106  /**
107   * 注册一个StatViewServlet
108   * @return
109   */
110  @Bean
111  public ServletRegistrationBean DruidStatViewServle() {
112   //org.springframework.boot.context.embedded.ServletRegistratio
      nBean提供类的进行注册.
113   ServletRegistrationBean servletRegistrationBean = new
      ServletRegistrationBean(new StatViewServlet(),
114     "/druid/*");
115   //添加初始化参数：initParams
116   //白名单：
```

```
117   servletRegistrationBean.addInitParameter("allow",
"127.0.0.1");
118   //IP黑名单 (存在共同时，deny优先于allow) : 如果满足deny的话提
示:Sorry, you are not permitted to view this page.
119   //servletRegistrationBean.addInitParameter("deny",
"192.168.0.114");
120   //登录查看信息的账号密码.
121   servletRegistrationBean.addInitParameter("loginUsername",
"admin");
122   servletRegistrationBean.addInitParameter("loginPassword",
"admin");
123   //是否能够重置数据.
124   servletRegistrationBean.addInitParameter("resetEnable",
"false");
125   return servletRegistrationBean;
126  }
127
128  /**
129   * 注册一个：filterRegistrationBean
130   * @return
131   */
132  @Bean
133  public FilterRegistrationBean druidStatFilter() {
134   FilterRegistrationBean filterRegistrationBean = new
FilterRegistrationBean(new WebStatFilter());
135   //添加过滤规则.
136   filterRegistrationBean.addUrlPatterns("/*");
137   //添加不需要忽略的格式信息.
138   filterRegistrationBean.addInitParameter("exclusions",
"*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*");
139   return filterRegistrationBean;
140  }
141 }
```

测试：http://localhost:8080/druid/login.html

自定义数据源路由starter

参考：https://gitee.com/baomidou/dynamic-datasource-spring-boot-starter

1、创建maven项目

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.zhuozhi</groupId>
    <artifactId>dbrouter-spring-boot-starter</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>

    <name>dbrouter-spring-boot-starter</name>
    <description>广州焯智教育-数据源路由</description>

    <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>

        <druid-spring-boot-starter.version>1.1.10</druid-spring-
boot-starter.version>
        <spring-boot.version>2.1.0.RELEASE</spring-boot.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <!-- Import dependency management from Spring
Boot,not include plugin
                     management as the parent import style -->
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>${spring-boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
```

```
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
            <version>${druid-spring-boot-starter.version}
</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-aop</artifactId>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-
processor</artifactId>
            <optional>true</optional>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
```

```
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

2、定义数据源切换控制器

```
package com.zhuozhi.dbrouter;

import java.util.HashSet;
import java.util.Set;

import lombok.extern.slf4j.Slf4j;

@Slf4j
public class DynamicDataSourceContextHolder {

    /*
     * 使用ThreadLocal维护变量，ThreadLocal为每个使用该变量的线程提供独立的变量副本，
     * 所以每一个线程都可以独立地改变自己的副本，而不会影响其它线程所对应的副本。
     */
    private static final ThreadLocal<String> CONTEXT_HOLDER = new ThreadLocal<String>();

    /*
     * 管理所有的数据源id,用于数据源的判断
     */
    public static Set<String> datasourceNameSet = new HashSet<String>();

    /**
     * @Title: setDateSoureName
     * @Description: 设置数据源的变量
     * @param dateSoureName
     * @return void
     * @throws
     */
```

```
29      public static void setDateSoureName(String dateSoureName) {
30          log.info("setDateSoureName {} {}",
Thread.currentThread().getName(), dateSoureName);
31          CONTEXT_HOLDER.set(dateSoureName);
32      }
33
34      /**
35       * @Title: getDateSoureName
36       * @Description: 获得数据源的变量
37       * @return String
38       * @throws
39       */
40      public static String getDateSoureName() {
41          log.info("getDateSoureName {} {}",
Thread.currentThread().getName(), CONTEXT_HOLDER.get());
42          return CONTEXT_HOLDER.get();
43      }
44
45      /**
46       * @Title: clearDateSoureNameList
47       * @Description: 清空所有的数据源变量
48       * @return void
49       * @throws
50       */
51      public static void clearDateSoureName() {
52          log.info("clearDateSoureName");
53          CONTEXT_HOLDER.remove();
54      }
55
56      /**
57       * @Title: existDateSoure
58       * @Description: 判断数据源是否已存在
59       * @param dateSoureName
60       * @return boolean
61       * @throws
62       */
63      public static boolean existDateSoure(String dateSoureName) {
64          return datasourceNameSet.contains(dateSoureName);
65      }
66
67  }
68
```

3、定义DynamicDataSource，数据源路由类

```
package com.zhuozhi.dbrouter;

import java.util.LinkedHashMap;

import javax.sql.DataSource;

import
org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSou
rce;

import lombok.extern.slf4j.Slf4j;

@Slf4j
public class DynamicDataSource extends AbstractRoutingDataSource
{
 /**
  * 默认数据源
  */
 private DataSource defaultTargetDataSource;

 /**
  * 数据源列表
  */
 private LinkedHashMap<Object, Object> targetDataSources;

 public DynamicDataSource(DataSource defaultTargetDataSource,
LinkedHashMap<Object, Object> targetDataSources) {
  this.defaultTargetDataSource = defaultTargetDataSource;
  this.targetDataSources = targetDataSources;
 }

 @Override
 protected Object determineCurrentLookupKey() {
  String dataSource =
DynamicDataSourceContextHolder.getDateSoureName();
  log.info("DynamicDataSource current db name: {}", dataSource);
  return dataSource;
```

```
 }

 @Override
 public void afterPropertiesSet() {
  super.setDefaultTargetDataSource(defaultTargetDataSource);
  super.setTargetDataSources(targetDataSources);
  super.afterPropertiesSet();
 }

}
```

4、定义starter 入口类 DataSourceConfiguration

```
package com.zhuozhi.dbrouter;

import java.sql.SQLException;
import java.util.LinkedHashMap;

import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.autoconfigure.AutoConfigureBefore;
import org.springframework.boot.context.properties.bind.Bindable;
import org.springframework.boot.context.properties.bind.Binder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;
import org.springframework.core.env.Environment;

import com.alibaba.druid.pool.DruidDataSource;
import
com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceAutoCo
nfigure;

@Configuration
//注意需要优先于DruidDataSourceAutoConfigure 和
DataSourceAutoConfiguration，不然会bean冲突
@AutoConfigureBefore(DruidDataSourceAutoConfigure.class)
public class DataSourceConfiguration {

 @Autowired
```

```
 private Environment environment;

 /**
  * 配置多数据源
  * @return
  * @throws SQLException
  */
 @Bean(name = "dataSource")
 @Primary
 public DynamicDataSource dataSource() throws SQLException {
  Binder binder = Binder.get(environment);
  DruidDataSource master =
binder.bind("spring.datasource.master",
Bindable.of(DruidDataSource.class)).get();
  DruidDataSource slave = binder.bind("spring.datasource.slave",
Bindable.of(DruidDataSource.class)).get();

  //预加载
  master.init();
  slave.init();

  LinkedHashMap<Object, Object> targetDatasources = new
LinkedHashMap<>();
  targetDatasources.put(master.getName(), master);
  targetDatasources.put(slave.getName(), slave);

  DynamicDataSourceContextHolder.datasourceNameSet.add(master.get
Name());
  DynamicDataSourceContextHolder.datasourceNameSet.add(slave.getN
ame());

  //设置默认使用数据源为slave库
  return new DynamicDataSource(slave, targetDatasources);
 }

}

```

5、定义事务控制器DynamicDataSourceTransactionManager

```java
package com.zhuozhi.dbrouter;

import javax.sql.DataSource;

import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.transaction.TransactionDefinition;

public class DynamicDataSourceTransactionManager extends DataSourceTransactionManager {

 private static final long serialVersionUID = 1715868041601265926L;

 public DynamicDataSourceTransactionManager(DataSource dataSource) {
  super(dataSource);
 }

 /**
  * 重写doBegin，切换数据源
  * 若有 @Transaction 注解，事务只读，设置为从；否则主库
  */
 @Override
 protected void doBegin(Object transaction, TransactionDefinition definition) {
  if (definition.isReadOnly()) {
   DynamicDataSourceContextHolder.setDateSoureName("slave");
  } else {
   DynamicDataSourceContextHolder.setDateSoureName("master");
  }
  super.doBegin(transaction, definition);
 }

 /**
  * 结束事务清理刚刚设置的数据源
  */
 @Override
 protected void doCleanupAfterCompletion(Object transaction) {
  DynamicDataSourceContextHolder.clearDateSoureName();
  super.doCleanupAfterCompletion(transaction);
```

```
 }

}
```

6、自定义事务注册到spring

```
package com.zhuozhi.dbrouter;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.transaction.PlatformTransactionManager;
import
org.springframework.transaction.annotation.EnableTransactionManag
ement;
import
org.springframework.transaction.annotation.TransactionManagementC
onfigurer;

@Configuration
@EnableTransactionManagement(proxyTargetClass = true)
public class TransactionManagerConfiguration implements
TransactionManagementConfigurer {

 @Autowired
 private DynamicDataSource dataSource;

 @Override
 @Bean
 public PlatformTransactionManager
annotationDrivenTransactionManager() {
  return new DynamicDataSourceTransactionManager(dataSource);
 }
}
```

7、自定义注解 用于切换service层方法的数据源

```
```

```java
package com.zhuozhi.dbrouter;


import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * 切换数据源，作用在service层的方法中
 * 假如有@Transactional,只会使用master数据源
 *
 */
@Target({ ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface DS {

    /**
     * @return 数据源名称
     */
    String value();
}
```

8、自定义注解 注册到spring

```java
package com.zhuozhi.dbrouter;

import java.lang.reflect.Method;

import javax.annotation.PostConstruct;

import org.aopalliance.aop.Advice;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
import org.springframework.aop.Pointcut;
import org.springframework.aop.support.AbstractPointcutAdvisor;
```

```
import org.springframework.aop.support.annotation.AnnotationMatchingPointcut;
import org.springframework.context.annotation.Configuration;

import lombok.extern.slf4j.Slf4j;

/**
 * 自定义注解注册到spring
 * @author ming
 *
 */
@Slf4j
@Configuration
public class DSConfiguration extends AbstractPointcutAdvisor {

 private static final long serialVersionUID = 4091225171686825015L;

 private Pointcut pointcut;

 private Advice advice;
 @PostConstruct
 public void init() {
  log.info("init LogAutoConfiguration start");
  this.pointcut = AnnotationMatchingPointcut.forMethodAnnotation(DS.class);
  this.advice = new MethodInterceptor() {
   @Override
   public Object invoke(MethodInvocation invocation) throws Throwable {
    try {
     Method method = invocation.getMethod();
     DS annotation = method.getAnnotation(DS.class);

     if (!DynamicDataSourceContextHolder.existDateSoure(annotation.value())) {
      log.error("Cannot find datasource {}", annotation.value());
      throw new Throwable("Cannot find datasource");
     }

```

```
47      //根据DS注解的值设置数据源
48
DynamicDataSourceContextHolder.setDateSoureName(annotation.value(
));
49
50      //执行service的方法并返回
51      return invocation.proceed();
52     } finally {
53      //清理数据源
54      DynamicDataSourceContextHolder.clearDateSoureName();
55     }
56    }
57   };
58   log.info("init LogAutoConfiguration end");
59  }
60
61  @Override
62  public Pointcut getPointcut() {
63   return this.pointcut;
64  }
65
66  @Override
67  public Advice getAdvice() {
68   return this.advice;
69  }
70
71 }
```

9、在/src/main/resources/下创建META-INF文件夹，然后创建spring.factories文件。另外的项目依赖该项目时，spring容器就会加载初始化DataSourceConfiguration,TransactionManagerConfiguration,DSConfiguration

```
1 org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
2 com.zhuozhi.dbrouter.DataSourceConfiguration,\
3 com.zhuozhi.dbrouter.TransactionManagerConfiguration,\
4 com.zhuozhi.dbrouter.DSConfiguration
```

10、参考配置信息

```yaml
spring:
  datasource:
    master:
      name: master
      url: jdbc:mysql://localhost:3306/test
      connectionProperties: serverTimezone=CTT;useUnicode=true;characterEncoding=utf-8;allowMultiQueries=true;druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000
      username: zero
      password: 123456
      #配置初始化大小、最小、最大
      initialSize: 5
      minIdle: 5
      maxActive: 20
      #配置从连接池获取连接等待超时的时间
      maxWait: 10000
      # 配置一个连接在池中最大空闲时间，单位是毫秒
      minEvictableIdleTimeMillis: 300000
      #配置间隔多久启动一次DestroyThread，对连接池内的连接才进行一次检测，单位是毫秒。
      #检测时:1.如果连接空闲并且超过minIdle以外的连接，如果空闲时间超过minEvictableIdleTimeMillis设置的值则直接物理关闭。2.在minIdle以内的不处理
      timeBetweenEvictionRunsMillis: 600000
      #申请连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能
      testOnBorrow: false
      #归还连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能
      testOnReturn: false
      #设置从连接池获取连接时是否检查连接有效性，true时，如果连接空闲时间超过minEvictableIdleTimeMillis进行检查，否则不检查;false时，不检查
      testWhileIdle: true
      #是否缓存preparedStatement，也就是PSCache。PSCache对支持游标的数据库性能提升巨大，比如说oracle。在mysql5.5（不含）以下建议关闭
      poolPreparedStatements: true
      maxPoolPreparedStatementPerConnectionSize: 20
      #检测语句
      validationQuery: SELECT 1 FROM DUAL
```

```
      filters: stat,wall,log4j
    slave:
      name: slave
      url: jdbc:mysql://localhost:3306/test2
      connectionProperties:
serverTimezone=CTT;useUnicode=true;characterEncoding=utf-
8;allowMultiQueries=true;druid.stat.mergeSql=true;druid.stat.slow
SqlMillis=5000
      username: zero
      password: 123456
      #配置初始化大小、最小、最大
      initialSize: 5
      minIdle: 5
      maxActive: 20
      #配置从连接池获取连接等待超时的时间
      maxWait: 10000
      # 配置一个连接在池中最大空闲时间，单位是毫秒
      minEvictableIdleTimeMillis: 300000
      #配置间隔多久启动一次DestroyThread，对连接池内的连接才进行一次
检测，单位是毫秒。
      #检测时:1.如果连接空闲并且超过minIdle以外的连接，如果空闲时间超
过minEvictableIdleTimeMillis设置的值则直接物理关闭。2.在minIdle以内
的不处理
      timeBetweenEvictionRunsMillis: 600000
      #申请连接时执行validationQuery检测连接是否有效，做了这个配置会
降低性能
      testOnBorrow: false
      #归还连接时执行validationQuery检测连接是否有效，做了这个配置会
降低性能
      testOnReturn: false
      #设置从连接池获取连接时是否检查连接有效性，true时，如果连接空闲
时间超过minEvictableIdleTimeMillis进行检查，否则不检查;false时，不检
查
      testWhileIdle: true
      #是否缓存preparedStatement，也就是PSCache。PSCache对支持游标的
数据库性能提升巨大，比如说oracle。在mysql5.5（不含）以下建议关闭
      poolPreparedStatements: true
      maxPoolPreparedStatementPerConnectionSize: 20
      #检测语句
      validationQuery: SELECT 1 FROM DUAL
      filters: stat,wall,log4j
    druid:
```

```
      #监控页面设置
      stat-view-servlet:
        enabled: true
        loginUsername: admin
        loginPassword: admin
      web-stat-filter:
        enabled: true

mybatis:
  configuration:
    map-underscore-to-camel-case: true

```

多数据源整合

创建2个测试数据库

```
create database test;
create database test2;
```

分别在2个库创建用户表

```
CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` char(20) NOT NULL COMMENT '用户名',
  `mobile` char(11) NOT NULL COMMENT '手机号码',
  `nick_name` varchar(30) DEFAULT NULL COMMENT '昵称',
  `sex` int(1) DEFAULT '1' COMMENT '性别',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=0 DEFAULT CHARSET=utf8 COMMENT='用户表';
```

分别插入一些测试数据，test 和 test2 的用户表 nick_name 区分master 和 slave ， 用于查询测试

```
INSERT INTO `test`.`users` (`username`, `mobile`, `nick_name`, `sex`) VALUES ('zero22', '131234', 'master', NULL);
INSERT INTO `test`.`users` (`username`, `mobile`, `nick_name`, `sex`) VALUES ('kkk22', '321112', 'master', NULL);

INSERT INTO `test2`.`users` (`username`, `mobile`, `nick_name`, `sex`) VALUES ('zero22', '131234', 'slave', NULL);
INSERT INTO `test2`.`users` (`username`, `mobile`, `nick_name`, `sex`) VALUES ('kkk22', '321112', 'slave', NULL);
```

项目pom

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.zhuozhi</groupId>
    <artifactId>zhuozhi</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>广州焯智教育</name>
    <description>广州焯智教育-springboot例子</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.0.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>

        <dependency>
            <groupId>com.zhuozhi</groupId>
            <artifactId>dbrouter-spring-boot-starter</artifactId>
            <version>1.0</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.3.2</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-freemarker</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-log4j</artifactId>
            <version>1.3.8.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>tk.mybatis</groupId>
            <artifactId>mapper-spring-boot-starter</artifactId>
            <version>2.0.4</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

yml配置

```yaml
spring:
  datasource:
    master:
      name: master
      url: jdbc:mysql://localhost:3306/test
      connectionProperties: serverTimezone=CTT;useUnicode=true;characterEncoding=utf-8;allowMultiQueries=true;druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000
      username: zero
      password: 123456
      #配置初始化大小、最小、最大
      initialSize: 5
      minIdle: 5
      maxActive: 20
      #配置从连接池获取连接等待超时的时间
      maxWait: 10000
      # 配置一个连接在池中最大空闲时间，单位是毫秒
      minEvictableIdleTimeMillis: 300000
      #配置间隔多久启动一次DestroyThread，对连接池内的连接才进行一次检测，单位是毫秒。
      #检测时:1.如果连接空闲并且超过minIdle以外的连接，如果空闲时间超过minEvictableIdleTimeMillis设置的值则直接物理关闭。2.在minIdle以内的不处理
      timeBetweenEvictionRunsMillis: 600000
      #申请连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能
      testOnBorrow: false
      #归还连接时执行validationQuery检测连接是否有效，做了这个配置会降低性能
      testOnReturn: false
      #设置从连接池获取连接时是否检查连接有效性，true时，如果连接空闲时间超过minEvictableIdleTimeMillis进行检查，否则不检查;false时，不检查
      testWhileIdle: true
      #是否缓存preparedStatement，也就是PSCache。PSCache对支持游标的数据库性能提升巨大，比如说oracle。在mysql5.5（不含）以下建议关闭
      poolPreparedStatements: true
      maxPoolPreparedStatementPerConnectionSize: 20
```

```
      #检测语句
      validationQuery: SELECT 1 FROM DUAL
      filters: stat,wall,log4j
    slave:
      name: slave
      url: jdbc:mysql://localhost:3306/test2
      connectionProperties:
serverTimezone=CTT;useUnicode=true;characterEncoding=utf-
8;allowMultiQueries=true;druid.stat.mergeSql=true;druid.stat.slow
SqlMillis=5000
      username: zero
      password: 123456
      #配置初始化大小、最小、最大
      initialSize: 5
      minIdle: 5
      maxActive: 20
      #配置从连接池获取连接等待超时的时间
      maxWait: 10000
      # 配置一个连接在池中最大空闲时间，单位是毫秒
      minEvictableIdleTimeMillis: 300000
      #配置间隔多久启动一次DestroyThread，对连接池内的连接才进行一次
检测，单位是毫秒。
      #检测时:1.如果连接空闲并且超过minIdle以外的连接，如果空闲时间超
过minEvictableIdleTimeMillis设置的值则直接物理关闭。2.在minIdle以内
的不处理
      timeBetweenEvictionRunsMillis: 600000
      #申请连接时执行validationQuery检测连接是否有效，做了这个配置会
降低性能
      testOnBorrow: false
      #归还连接时执行validationQuery检测连接是否有效，做了这个配置会
降低性能
      testOnReturn: false
      #设置从连接池获取连接时是否检查连接有效性，true时，如果连接空闲
时间超过minEvictableIdleTimeMillis进行检查，否则不检查;false时，不检
查
      testWhileIdle: true
      #是否缓存preparedStatement，也就是PSCache。PSCache对支持游标的
数据库性能提升巨大，比如说oracle。在mysql5.5（不含）以下建议关闭
      poolPreparedStatements: true
      maxPoolPreparedStatementPerConnectionSize: 20
      #检测语句
      validationQuery: SELECT 1 FROM DUAL
```

```
      filters: stat,wall,log4j
    druid:
      #监控页面设置
      stat-view-servlet:
        enabled: true
        loginUsername: admin
        loginPassword: admin
      web-stat-filter:
        enabled: true

mybatis:
  configuration:
    map-underscore-to-camel-case: true
```

entity实体

```
package com.zhuozhi.entity;

import javax.persistence.Column;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Data
@Table(name = "users")
public class User {
 @Id
 @GeneratedValue(strategy = GenerationType.IDENTITY)
 private Integer id;
 @Column(name = "username")
 private String userName;
 private String mobile;
 private String nickName;
 private Integer sex;

}

```

dao类

```
package com.zhuozhi.dao;

import com.zhuozhi.entity.User;

import tk.mybatis.mapper.common.Mapper;

public interface UserDao extends Mapper<User> {

}
```

service类

```
package com.zhuozhi.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.zhuozhi.dao.UserDao;
import com.zhuozhi.dbrouter.DS;
import com.zhuozhi.entity.User;

@Service
public class UserService {
 @Autowired
 UserDao userDao;

 @DS(value = "master")
 public User getUserByIdDS1(Integer id) {
  return userDao.selectByPrimaryKey(id);
 }

 @DS(value = "slave")
 public User getUserByIdDS2(Integer id) {
```

```
  return userDao.selectByPrimaryKey(id);
 }

 @Transactional
 public User saveUser(String userName, String mobile) {
  User user = new User();
  user.setUserName(userName);
  user.setMobile(mobile);
  userDao.insert(user);
  return user;
 }

}
```

controller类

```
package com.zhuozhi.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import com.zhuozhi.entity.User;
import com.zhuozhi.service.UserService;

import lombok.extern.slf4j.Slf4j;

@Controller
@Slf4j
public class UserController {

 @Autowired
 UserService userService;


 /**
  * master数据源 获取数据测试
  * @param id
  * @return
```

```
 */
@RequestMapping("/user/getUserByIdDS1")
@ResponseBody
public User getUserByIdDS1(Integer id) {
 log.debug("参数id:{}", id);
 return userService.getUserByIdDS1(id);
}

/**
 * slave数据源 获取数据测试
 * @param id
 * @return
 */
@RequestMapping("/user/getUserByIdDS2")
@ResponseBody
public User getUserByIdDS2(Integer id) {
 log.debug("参数id:{}", id);
 return userService.getUserByIdDS2(id);
}

/**
 * 带事务的数据源测试
 * @param userName
 * @param mobile
 * @return
 */
@RequestMapping("/user/save")
@ResponseBody
public User save(String userName, String mobile) {
 return userService.saveUser(userName, mobile);
}
}

```

启动类

```
package com.zhuozhi;

import org.springframework.boot.SpringApplication;

```

```
import
org.springframework.boot.autoconfigure.SpringBootApplication;

import tk.mybatis.spring.annotation.MapperScan;

@SpringBootApplication
@MapperScan("com.zhuozhi.dao")
public class TablerApplication {

 public static void main(String[] args) {
  SpringApplication.run(TablerApplication.class, args);
 }
}

```

测试

http://localhost:8080/user/getUserByIdDS1?id=1

http://localhost:8080/user/getUserByIdDS2?id=1

http://localhost:8080/user/save?userName=testuser&mobile=1352232112