

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

```
1      <dependency>
2          <groupId>org.springframework.boot</groupId>
3          <artifactId>spring-boot-starter-
freemarker</artifactId>
4      </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

```
1 <!DOCTYPE html>
2 <html>
3 <head lang="en">
4 <meta charset="UTF-8" />
5 <title></title>
6 </head>
7 <body>
8 <div>用户名: ${user.userName}!""</div>
9 <div>手机: ${user.mobile}!""</div>
10 <div>测试: ${test}!""</div>
11 </body>
12 </html>
```

测试controller

```
1 @RequestMapping("/user/info")
2 public String index(Model model, String username) {
3     model.addAttribute("user",
4     userService.getUserByName(username));
5     model.addAttribute("test", "test");
6     return "/user/info";
7 }
```

log4j整合

引入依赖

```

1      <dependency>
2          <groupId>org.springframework.boot</groupId>
3          <artifactId>spring-boot-starter</artifactId>
4          <exclusions>
5              <exclusion>
6                  <groupId>org.springframework.boot</groupId>
7                  <artifactId>spring-boot-starter-
logging</artifactId>
8              </exclusion>
9          </exclusions>
10     </dependency>
11
12     <dependency>
13         <groupId>org.springframework.boot</groupId>
14         <artifactId>spring-boot-starter-log4j</artifactId>
15         <version>1.3.8.RELEASE</version>
16     </dependency>

```

自定义日志类，warn 和 error 日志分开输出

```

1 package com.zhuozhi.common.log4j;
2
3 import org.apache.log4j.DailyRollingFileAppender;
4 import org.apache.log4j.Priority;
5
6 /**
7  * 名称: BaseLeveDailyRollingFileAppender <br>
8  * 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>
9  *
10  */
11 public class BaseLeveDailyRollingFileAppender extends
DailyRollingFileAppender {
12     @Override
13     public boolean isAsSevereAsThreshold(Priority priority) {
14         //只判断是否相等，而不判断优先级
15         return this.getThreshold().equals(priority);
16     }
17 }

```

添加log4j.properties

```
1  #输出需要的日志
2  log4j.rootLogger=DEBUG,stdout,info,warn,error
3
4  #控制台输出
5  log4j.appender.stdout=org.apache.log4j.ConsoleAppender
6  log4j.appender.stdout.Target=System.out
7  log4j.appender.stdout.Threshold=DEBUG
8  log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
9  log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd
    HH:mm:ss SSS}|%5p|%F.%M:%L| %m%n
10
11 #INFO日志
12 log4j.logger.info=info
13 log4j.appender.info=org.apache.log4j.DailyRollingFileAppender
14 log4j.appender.info.File=./logs/info/info.log
15 log4j.appender.info.datePattern='.'yyyy-MM-dd'.log'
16 log4j.appender.info.append=true
17 log4j.appender.info.Threshold=INFO
18 log4j.appender.info.encoding=UTF-8
19 log4j.appender.info.ImmediateFlush=true
20 log4j.appender.info.layout=org.apache.log4j.PatternLayout
21 log4j.appender.info.layout.ConversionPattern=%d{yyyy-MM-dd
    HH:mm:ss SSS}|%5p|%F.%M:%L| %m%n
22 #WARN日志
23 log4j.appender.warn=com.zhuozhi.common.log4j.BaseLeveDailyRolling
    FileAppender
24 log4j.appender.warn.File=./logs/warn/warn.log
25 log4j.appender.warn.datePattern='.'yyyy-MM-dd'.log'
26 log4j.appender.warn.append=true
27 log4j.appender.warn.Threshold=WARN
28 log4j.appender.warn.encoding=UTF-8
29 log4j.appender.warn.ImmediateFlush=true
30 log4j.appender.warn.layout=org.apache.log4j.PatternLayout
31 log4j.appender.warn.layout.ConversionPattern=%d{yyyy-MM-dd
    HH:mm:ss SSS}|%5p|%F.%M:%L| %m%n
32 #ERROR日志
```

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

测试类

```
1 //导入包
2 import org.slf4j.Logger;
3 import org.slf4j.LoggerFactory;
4
5 //获取logger
6 private Logger logger = LoggerFactory.getLogger(this.getClass());
7
8 //再方法中
9 logger.debug("XXXX");
10 logger.info("XXXX");
11 //等测试即可
```

也可以用lombok方式

```
1 //在类上面加上注释
2 @Slf4j
3
4 //方法可以直接
5 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状态码；
- error：错误原因；
- exception：异常的类型；
- 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

```
1 package com.zhuozhi.result;
2
3 /**
4  * <p>
5  * <b>Result</b> is 包裹结果的基类
6  * </p>
7  *
8  */
9 public class Result<T> {
10
11     private String code;
12
13     private String msg;
14
15     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

```
1 package com.zhuozhi.exception;
2
3 public class BusinessException extends RuntimeException {
4     private static final long serialVersionUID =
5     8862339523905913528L;
6
7     public BusinessException(String message) {
8         super(message);
9         this.message = message;
10    }
11
12    public BusinessException(String code, String message) {
13        super(message);
14        this.code = code;
15        this.message = message;
16    }
17
18    public BusinessException(String message, Throwable cause) {
19        super(message, cause);
20        this.message = message;
21    }
22
23    /**
24     * 错误代码
25     */
26    private String code;
27
28    /**
29     * 错误信息
30     */
31    private String message;
32
33    /**
34     * 获取 错误代码
35     */
36    public String getCode() {
37        return code;
38    }
39
40    /**
41     * 设置 错误代码
```



```

41     */
42     public void setCode(String code) {
43         this.code = code;
44     }
45
46     /**
47      * 获取 错误信息
48      */
49     public String getMessage() {
50         return message;
51     }
52
53     /**
54      * 设置 错误信息
55      */
56     public void setMessage(String message) {
57         this.message = message;
58     }
59
60 }

```

全局异常拦截类，要确保这个类被扫描到

```

1  package com.zhuozhi.controller;
2
3  import org.springframework.validation.BindException;
4  import org.springframework.validation.ObjectError;
5  import
6  org.springframework.web.bind.MethodArgumentNotValidException;
7  import org.springframework.web.bind.annotation.ControllerAdvice;
8  import org.springframework.web.bind.annotation.ExceptionHandler;
9  import org.springframework.web.bind.annotation.ResponseBody;
10
11 import com.zhuozhi.exception.BusinessException;
12 import com.zhuozhi.result.Result;
13
14 import lombok.extern.slf4j.Slf4j;
15
16 /**
17  * <p>

```

```
17  * <b>GlobalExceptionHandler</b> is 自定义异常处理
18  * </p>
19  */
20
21 @ControllerAdvice
22 @Slf4j
23 public class GlobalExceptionHandler {
24
25     /**
26      * 业务异常处理
27      * @param e
28      * @return
29      */
30     @ExceptionHandler({ BusinessException.class })
31     @ResponseBody
32     public Result<Object>
33     handleBusinessException(BusinessException e) {
34         log.warn("业务异常", e);
35         Result<Object> result = new Result<Object>(e.getCode(),
36         e.getMessage());
37         return result;
38     }
39
40     /**
41      * @ModelAttribute 注解验证对象 出错所抛出的异常处理
42      * @param e
43      * @return
44      */
45     @ExceptionHandler({ BindException.class })
46     @ResponseBody
47     public Result<Object> handleBindException(BindException e) {
48         log.warn("参数格式错误", e);
49         ObjectError err = e.getAllErrors().get(0);
50         Result<Object> result = new Result<Object>("Bxxx",
51         err.getDefaultMessage());
52         return result;
53     }
54
55     /**
56      * @RequestBody 注解验证对象 出错所抛出的异常处理
57      * @param e
58      * @return
59      */
60 }
```

```

56     */
57     @ExceptionHandler({ MethodArgumentNotValidException.class })
58     @ResponseBody
59     public Result<Object>
handleMethodArgumentNotValidException(MethodArgumentNotValidExcep
tion e) {
60         log.warn("参数格式错误", e);
61         ObjectError err =
e.getBindingResult().getAllErrors().get(0);
62         Result<Object> result = new Result<Object>("Mxxx",
err.getDefaultMessage());
63         return result;
64     }
65
66     /**
67      * 系统异常处理
68      * @param e
69      * @return
70      */
71     @ExceptionHandler({ RuntimeException.class })
72     @ResponseBody
73     public Result<Object> handleRuntimeException(RuntimeException
e) {
74         log.error("系统内部异常", e);
75         Result<Object> result = new Result<Object>("Rxxxx",
"Exxxx");
76         return result;
77     }
78 }

```

整合mybatis

相关依赖

```

1     <dependency>
2         <groupId>org.mybatis.spring.boot</groupId>
3         <artifactId>mybatis-spring-boot-starter</artifactId>
4         <version>1.3.2</version>

```

```

5         </dependency>
6
7         <dependency>
8             <groupId>mysql</groupId>
9             <artifactId>mysql-connector-java</artifactId>
10            <scope>runtime</scope>
11        </dependency>

```

/src/main/resources/application.yml配置

```

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

```

Dao代码

```

1 package com.zhuzhi.dao;
2
3 import org.apache.ibatis.annotations.Param;
4 import org.apache.ibatis.annotations.Select;
5
6 import com.zhuzhi.entity.User;
7
8 public interface UserDao {
9     @Select("SELECT * FROM USERS WHERE USERNAME = #{username}")
10    User findByName(@Param("username") String username);
11 }

```

启动类添加@MapperScan

```

1 package com.zhuzhi;

```

```

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

```

```

9  import lombok.Data;
10
11  @Data
12  //假如数据库名称不一致需要指定表名
13  @Table(name = "users")
14  public class User {
15      @Id
16      @GeneratedValue(strategy = GenerationType.IDENTITY)
17      private Integer id;
18
19      //指定列名
20      @Column(name = "username")
21      private String userName;
22
23      private String mobile;
24
25      private String nickName;
26  }

```

dao类继承

```

1  public interface UserDao extends Mapper<User> {
2  }

```

引用包要改成

```

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

```

整合druid

添加依赖

```
1      <dependency>
2          <groupId>com.alibaba</groupId>
3          <artifactId>druid</artifactId>
4          <version>1.1.12</version>
5      </dependency>
```

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

```
1  spring:
2      datasource:
3          url: jdbc:mysql://localhost:3306/test?
serverTimezone=CTT&useUnicode=true&characterEncoding=utf-
8&allowMultiQueries=true
4          username: zero
5          password: 123456
6
7      #druid配置
8      type: com.alibaba.druid.pool.DruidDataSource
9      connectionProperties:
druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000
10     initialSize: 5
11     minIdle: 5
12     maxActive: 20
13     maxWait: 60000
14     poolPreparedStatements: true
15     filters: stat,wall,log4j
16     maxPoolPreparedStatementPerConnectionSize: 20
17     testOnBorrow: false
18     testWhileIdle: true
19     minEvictableIdleTimeMillis: 300000
20     timeBetweenEvictionRunsMillis: 60000
21     testOnReturn: false
22     validationQuery: SELECT 1 FROM DUAL
```

定义DataSource bean 覆盖原来的

```
1 package com.zhuozhi.config;
2
3 import java.sql.SQLException;
4
5 import javax.sql.DataSource;
6
7 import org.springframework.beans.factory.annotation.Value;
8 import
    org.springframework.boot.web.servlet.FilterRegistrationBean;
9 import
    org.springframework.boot.web.servlet.ServletRegistrationBean;
10 import org.springframework.context.annotation.Bean;
11 import org.springframework.context.annotation.Configuration;
12 import org.springframework.context.annotation.Primary;
13
14 import com.alibaba.druid.pool.DruidDataSource;
15 import com.alibaba.druid.support.http.StatViewServlet;
16 import com.alibaba.druid.support.http.WebStatFilter;
17
18 import lombok.extern.slf4j.Slf4j;
19
20 @Configuration
21 @Slf4j
22 public class DruidConfiguration {
23     @Value("${spring.datasource.url}")
24     private String dbUrl;
25
26     @Value("${spring.datasource.username}")
27     private String username;
28
29     @Value("${spring.datasource.password}")
30     private String password;
31
32     @Value("${spring.datasource.initialSize}")
33     private int initialSize;
34
35     @Value("${spring.datasource.minIdle}")
36     private int minIdle;
37
38     @Value("${spring.datasource.maxActive}")
39     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("${spring.datasource.maxPoolPreparedStatementPerConnectionSize}")
66 private int maxPoolPreparedStatementPerConnectionSize;
67
68 @Value("${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 DruidStatViewServlet() {
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",
118         "127.0.0.1");
119     //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.
120     //servletRegistrationBean.addInitParameter("deny",
121         "192.168.0.114");
122     //登录查看信息的账号密码.
123     servletRegistrationBean.addInitParameter("loginUsername",
124         "admin");
125     servletRegistrationBean.addInitParameter("loginPassword",
126         "admin");
127     //是否能够重置数据.
128     servletRegistrationBean.addInitParameter("resetEnable",
129         "false");
130     return servletRegistrationBean;
131 }
132
133 /**
134  * 注册一个: filterRegistrationBean
135  * @return
136  */
137 @Bean
138 public FilterRegistrationBean druidStatFilter() {
139     FilterRegistrationBean filterRegistrationBean = new
140     FilterRegistrationBean(new WebStatFilter());
141     //添加过滤规则.
142     filterRegistrationBean.addUrlPatterns("/");
143     //添加不需要忽略的格式信息.
144     filterRegistrationBean.addInitParameter("exclusions",
145         "*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*");
146     return filterRegistrationBean;
147 }
148 }

```

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

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4     http://maven.apache.org/xsd/maven-4.0.0.xsd">
5
6     <modelVersion>4.0.0</modelVersion>
7
8     <groupId>com.zhuozhi</groupId>
9     <artifactId>dbrouter-spring-boot-starter</artifactId>
10    <version>1.0</version>
11    <packaging>jar</packaging>
12
13    <name>dbrouter-spring-boot-starter</name>
14    <description>广州焯智教育-数据源路由</description>
15
16    <properties>
17        <project.build.sourceEncoding>UTF-
18        8</project.build.sourceEncoding>
19        <project.reporting.outputEncoding>UTF-
20        8</project.reporting.outputEncoding>
21        <java.version>1.8</java.version>
22
23        <druid-spring-boot-starter.version>1.1.10</druid-spring-
24        boot-starter.version>
25        <spring-boot.version>2.1.0.RELEASE</spring-boot.version>
26    </properties>
27
28    <dependencyManagement>
29        <dependencies>
30            <dependency>
31                <!-- Import dependency management from Spring
32                Boot,not include plugin
33                management as the parent import style -->
34                <groupId>org.springframework.boot</groupId>
35                <artifactId>spring-boot-dependencies</artifactId>
36                <version>${spring-boot.version}</version>
37                <type>pom</type>
38                <scope>import</scope>
39            </dependency>
40        </dependencies>
41    </dependencyManagement>
42</project>
```

```
35     </dependencyManagement>
36
37     <dependencies>
38         <dependency>
39             <groupId>com.alibaba</groupId>
40             <artifactId>druid-spring-boot-starter</artifactId>
41             <version>${druid-spring-boot-starter.version}</version>
42         </dependency>
43
44         <dependency>
45             <groupId>org.springframework.boot</groupId>
46             <artifactId>spring-boot-starter-jdbc</artifactId>
47         </dependency>
48
49         <dependency>
50             <groupId>org.springframework.boot</groupId>
51             <artifactId>spring-boot-starter-aop</artifactId>
52         </dependency>
53
54         <dependency>
55             <groupId>org.projectlombok</groupId>
56             <artifactId>lombok</artifactId>
57             <optional>true</optional>
58         </dependency>
59
60         <dependency>
61             <groupId>org.springframework.boot</groupId>
62             <artifactId>spring-boot-configuration-processor</artifactId>
63             <optional>true</optional>
64         </dependency>
65     </dependencies>
66
67     <build>
68         <plugins>
69             <plugin>
70                 <groupId>org.apache.maven.plugins</groupId>
71                 <artifactId>maven-compiler-plugin</artifactId>
72                 <configuration>
73                     <source>1.8</source>
74                     <target>1.8</target>
```

```

75         </configuration>
76     </plugin>
77 </plugins>
78 </build>
79 </project>

```

2、定义数据源切换控制器

```

1  package com.zhuozhi.dbrouter;
2
3  import java.util.HashSet;
4  import java.util.Set;
5
6  import lombok.extern.slf4j.Slf4j;
7
8  @Slf4j
9  public class DynamicDataSourceContextHolder {
10
11      /*
12       * 使用ThreadLocal维护变量，ThreadLocal为每个使用该变量的线程提
13       * 供独立的变量副本，
14       * 所以每一个线程都可以独立地改变自己的副本，而不会影响到其它线程所对
15       * 应的副本。
16       */
17      private static final ThreadLocal<String> CONTEXT_HOLDER = new
18      ThreadLocal<String>();
19
20      /*
21       * 管理所有的数据源id,用于数据源的判断
22       */
23      public static Set<String> datasourceNameSet = new
24      HashSet<String>();
25
26      /**
27       * @Title: setDateSoureName
28       * @Description: 设置数据源的变量
29       * @param dateSoureName
30       * @return void
31       * @throws
32       */
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```
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

```
1 package com.zhuozhi.dbrouter;
2
3 import java.util.LinkedHashMap;
4
5 import javax.sql.DataSource;
6
7 import
    org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;
8
9 import lombok.extern.slf4j.Slf4j;
10
11 @Slf4j
12 public class DynamicDataSource extends AbstractRoutingDataSource
13 {
14     /**
15      * 默认数据源
16      */
17     private DataSource defaultTargetDataSource;
18
19     /**
20      * 数据源列表
21      */
22     private LinkedHashMap<Object, Object> targetDataSources;
23
24     public DynamicDataSource(DataSource defaultTargetDataSource,
25         LinkedHashMap<Object, Object> targetDataSources) {
26         this.defaultTargetDataSource = defaultTargetDataSource;
27         this.targetDataSources = targetDataSources;
28     }
29
30     @Override
31     protected Object determineCurrentLookupKey() {
32         String dataSource =
33             DynamicDataSourceContextHolder.getDateSoureName();
34         log.info("DynamicDataSource current db name: {}", dataSource);
35         return dataSource;
36     }
37 }
```



```

33     }
34
35     @Override
36     public void afterPropertiesSet() {
37         super.setDefaultTargetDataSource(defaultTargetDataSource);
38         super.setTargetDataSources(targetDataSources);
39         super.afterPropertiesSet();
40     }
41
42 }

```

4、定义starter 入口类 DataSourceConfiguration

```

1  package com.zhuozhi.dbrouter;
2
3  import java.sql.SQLException;
4  import java.util.LinkedHashMap;
5
6  import org.springframework.beans.factory.annotation.Autowired;
7  import
8      org.springframework.boot.autoconfigure.AutoConfigureBefore;
9  import org.springframework.boot.context.properties.bind.Bindable;
10 import org.springframework.boot.context.properties.bind.Binder;
11 import org.springframework.context.annotation.Bean;
12 import org.springframework.context.annotation.Configuration;
13 import org.springframework.context.annotation.Primary;
14 import org.springframework.core.env.Environment;
15
16 import com.alibaba.druid.pool.DruidDataSource;
17 import
18     com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceAutoCo
19         nfigure;
20
21 @Configuration
22 //注意需要优先于DruidDataSourceAutoConfigure 和
23     DataSourceAutoConfiguration, 不然会bean冲突
24 @AutoConfigureBefore(DruidDataSourceAutoConfigure.class)
25 public class DataSourceConfiguration {
26
27     @Autowired

```

```

24 private Environment environment;
25
26 /**
27  * 配置多数据源
28  * @return
29  * @throws SQLException
30  */
31 @Bean(name = "dataSource")
32 @Primary
33 public DynamicDataSource dataSource() throws SQLException {
34     Binder binder = Binder.get(environment);
35     DruidDataSource master =
36         binder.bind("spring.datasource.master",
37             Bindable.of(DruidDataSource.class)).get();
38     DruidDataSource slave = binder.bind("spring.datasource.slave",
39         Bindable.of(DruidDataSource.class)).get();
40
41     //预加载
42     master.init();
43     slave.init();
44
45     LinkedHashMap<Object, Object> targetDatasources = new
46     LinkedHashMap<>();
47     targetDatasources.put(master.getName(), master);
48     targetDatasources.put(slave.getName(), slave);
49
50     DynamicDataSourceContextHolder.datasourceNameSet.add(master.get
51     Name());
52     DynamicDataSourceContextHolder.datasourceNameSet.add(slave.getN
53     ame());
54
55     //设置默认使用数据源为slave库
56     return new DynamicDataSource(slave, targetDatasources);
57 }
58 }
59 }

```

5、定义事务控制器DynamicDataSourceTransactionManager

```
1 package com.zhuozhi.dbrouter;
2
3 import javax.sql.DataSource;
4
5 import
    org.springframework.jdbc.datasource.DataSourceTransactionManager;
6 import org.springframework.transaction.TransactionDefinition;
7
8 public class DynamicDataSourceTransactionManager extends
    DataSourceTransactionManager {
9
10     private static final long serialVersionUID =
        1715868041601265926L;
11
12     public DynamicDataSourceTransactionManager(DataSource
        dataSource) {
13         super(dataSource);
14     }
15
16     /**
17      * 重写doBegin, 切换数据源
18      * 若有 @Transaction 注解, 事务只读, 设置为从; 否则主库
19      */
20     @Override
21     protected void doBegin(Object transaction, TransactionDefinition
        definition) {
22         if (definition.isReadOnly()) {
23             DynamicDataSourceContextHolder.setDateSoureName("slave");
24         } else {
25             DynamicDataSourceContextHolder.setDateSoureName("master");
26         }
27         super.doBegin(transaction, definition);
28     }
29
30     /**
31      * 结束事务清理刚刚设置的数据源
32      */
33     @Override
34     protected void doCleanupAfterCompletion(Object transaction) {
35         DynamicDataSourceContextHolder.clearDateSoureName();
36         super.doCleanupAfterCompletion(transaction);
37     }
38 }
```

```
37 }  
38  
39 }
```

6、自定义事务注册到spring

```
1 package com.zhuozhi.dbrouter;  
2  
3 import org.springframework.beans.factory.annotation.Autowired;  
4 import org.springframework.context.annotation.Bean;  
5 import org.springframework.context.annotation.Configuration;  
6 import  
    org.springframework.transaction.PlatformTransactionManager;  
7 import  
    org.springframework.transaction.annotation.EnableTransactionManag  
        ement;  
8 import  
    org.springframework.transaction.annotation.TransactionManagementC  
        onfigurer;  
9  
10 @Configuration  
11 @EnableTransactionManagement(proxyTargetClass = true)  
12 public class TransactionManagerConfiguration implements  
    TransactionManagementConfigurer {  
13  
14     @Autowired  
15     private DynamicDataSource dataSource;  
16  
17     @Override  
18     @Bean  
19     public PlatformTransactionManager  
        annotationDrivenTransactionManager() {  
20         return new DynamicDataSourceTransactionManager(dataSource);  
21     }  
22 }
```

7、自定义注解 用于切换service层方法的数据源

```

1 package com.zhuozhi.dbrouter;
2
3
4 import java.lang.annotation.Documented;
5 import java.lang.annotation.ElementType;
6 import java.lang.annotation.Retention;
7 import java.lang.annotation.RetentionPolicy;
8 import java.lang.annotation.Target;
9
10 /**
11  * 切换数据源，作用在service层的方法中
12  * 假如有@Transactional,只会使用master数据源
13  *
14  */
15 @Target({ ElementType.METHOD })
16 @Retention(RetentionPolicy.RUNTIME)
17 @Documented
18 public @interface DS {
19
20     /**
21      * @return 数据源名称
22      */
23     String value();
24 }

```

8、自定义注解 注册到spring

```

1 package com.zhuozhi.dbrouter;
2
3 import java.lang.reflect.Method;
4
5 import javax.annotation.PostConstruct;
6
7 import org.aopalliance.aop.Advice;
8 import org.aopalliance.intercept.MethodInterceptor;
9 import org.aopalliance.intercept.MethodInvocation;
10 import org.springframework.aop.Pointcut;
11 import org.springframework.aop.support.AbstractPointcutAdvisor;

```

```
12 import
    org.springframework.aop.support.annotation.AnnotationMatchingPoin
    tcut;
13 import org.springframework.context.annotation.Configuration;
14
15 import lombok.extern.slf4j.Slf4j;
16
17 /**
18  * 自定义注解注册到spring
19  * @author ming
20  *
21  */
22 @Slf4j
23 @Configuration
24 public class DSConfiguration extends AbstractPointcutAdvisor {
25
26     private static final long serialVersionUID =
27     4091225171686825015L;
28
29     private Pointcut pointcut;
30
31     private Advice advice;
32     @PostConstruct
33     public void init() {
34         log.info("init LogAutoConfiguration start");
35         this.pointcut =
36         AnnotationMatchingPointcut.forMethodAnnotation(DS.class);
37         this.advice = new MethodInterceptor() {
38             @Override
39             public Object invoke(MethodInvocation invocation) throws
40             Throwable {
41                 try {
42                     Method method = invocation.getMethod();
43                     DS annotation = method.getAnnotation(DS.class);
44
45                     if
46                     (!DynamicDataSourceContextHolder.existDateSoure(annotation.value(
47                     ))) {
48                         log.error("Cannot find datasource {}", annotation.value());
49                         throw new Throwable("Cannot find datasource");
50                     }
51                 }
52             }
53         };
54     }
55 }
```

```

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

```
1  spring:
2      datasource:
3          master:
4              name: master
5              url: jdbc:mysql://localhost:3306/test
6              connectionProperties:
serverTimezone=CTT;useUnicode=true;characterEncoding=utf-
8;allowMultiQueries=true;druid.stat.mergeSql=true;druid.stat.slow
SqlMillis=5000
7          username: zero
8          password: 123456
9          #配置初始化大小、最小、最大
10         initialSize: 5
11         minIdle: 5
12         maxActive: 20
13         #配置从连接池获取连接等待超时的时间
14         maxWait: 10000
15         # 配置一个连接在池中最大空闲时间，单位是毫秒
16         minEvictableIdleTimeMillis: 300000
17         #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。
18         #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
19         timeBetweenEvictionRunsMillis: 600000
20         #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
21         testOnBorrow: false
22         #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
23         testOnReturn: false
24         #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
25         testWhileIdle: true
26         #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
27         poolPreparedStatements: true
28         maxPoolPreparedStatementPerConnectionSize: 20
29         #检测语句
30         validationQuery: SELECT 1 FROM DUAL
```



```
31     filters: stat,wall,log4j
32     slave:
33         name: slave
34         url: jdbc:mysql://localhost:3306/test2
35         connectionProperties:
serverTimezone=CTT;useUnicode=true;characterEncoding=utf-
8;allowMultiQueries=true;druid.stat.mergeSql=true;druid.stat.slow
SqlMillis=5000
36         username: zero
37         password: 123456
38         #配置初始化大小、最小、最大
39         initialSize: 5
40         minIdle: 5
41         maxActive: 20
42         #配置从连接池获取连接等待超时的时间
43         maxWait: 10000
44         # 配置一个连接在池中最大空闲时间，单位是毫秒
45         minEvictableIdleTimeMillis: 300000
46         #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。
47         #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
48         timeBetweenEvictionRunsMillis: 600000
49         #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
50         testOnBorrow: false
51         #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
52         testOnReturn: false
53         #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
54         testWhileIdle: true
55         #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
56         poolPreparedStatements: true
57         maxPoolPreparedStatementPerConnectionSize: 20
58         #检测语句
59         validationQuery: SELECT 1 FROM DUAL
60         filters: stat,wall,log4j
61     druid:
```

```
62      #监控页面设置
63      stat-view-servlet:
64          enabled: true
65          loginUsername: admin
66          loginPassword: admin
67      web-stat-filter:
68          enabled: true
69
70 mybatis:
71     configuration:
72         map-underscore-to-camel-case: true
73
```

多数据源整合

创建2个测试数据库

```
1 create database test;
2 create database test2;
```

分别在2个库创建用户表

```
1 CREATE TABLE `users` (
2     `id` int(11) NOT NULL AUTO_INCREMENT,
3     `username` char(20) NOT NULL COMMENT '用户名',
4     `mobile` char(11) NOT NULL COMMENT '手机号码',
5     `nick_name` varchar(30) DEFAULT NULL COMMENT '昵称',
6     `sex` int(1) DEFAULT '1' COMMENT '性别',
7     PRIMARY KEY (`id`)
8 ) 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

```

1 INSERT INTO `test`.`users` (`username`, `mobile`, `nick_name`,
  `sex`) VALUES ('zero22', '131234', 'master', NULL);
2 INSERT INTO `test`.`users` (`username`, `mobile`, `nick_name`,
  `sex`) VALUES ('kkk22', '321112', 'master', NULL);
3
4 INSERT INTO `test2`.`users` (`username`, `mobile`, `nick_name`,
  `sex`) VALUES ('zero22', '131234', 'slave', NULL);
5 INSERT INTO `test2`.`users` (`username`, `mobile`, `nick_name`,
  `sex`) VALUES ('kkk22', '321112', 'slave', NULL);

```

项目pom

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>com.zhuozhi</groupId>
8     <artifactId>zhuozhi</artifactId>
9     <version>0.0.1-SNAPSHOT</version>
10    <packaging>jar</packaging>
11
12    <name>广州焯智教育</name>
13    <description>广州焯智教育-springboot例子</description>
14
15    <parent>
16        <groupId>org.springframework.boot</groupId>
17        <artifactId>spring-boot-starter-parent</artifactId>
18        <version>2.1.0.RELEASE</version>
19        <relativePath /> <!-- lookup parent from repository -->
20    </parent>
21
22    <properties>
23        <project.build.sourceEncoding>UTF-
24    </project.build.sourceEncoding>

```

```
24     <project.reporting.outputEncoding>UTF-  
8</project.reporting.outputEncoding>  
25     <java.version>1.8</java.version>  
26 </properties>  
27  
28 <dependencies>  
29  
30     <dependency>  
31         <groupId>com.zhuozhi</groupId>  
32         <artifactId>dbrouter-spring-boot-  
starter</artifactId>  
33         <version>1.0</version>  
34     </dependency>  
35  
36     <dependency>  
37         <groupId>org.springframework.boot</groupId>  
38         <artifactId>spring-boot-starter-web</artifactId>  
39     </dependency>  
40     <dependency>  
41         <groupId>org.mybatis.spring.boot</groupId>  
42         <artifactId>mybatis-spring-boot-starter</artifactId>  
43         <version>1.3.2</version>  
44     </dependency>  
45  
46     <dependency>  
47         <groupId>mysql</groupId>  
48         <artifactId>mysql-connector-java</artifactId>  
49         <scope>runtime</scope>  
50     </dependency>  
51     <dependency>  
52         <groupId>org.projectlombok</groupId>  
53         <artifactId>lombok</artifactId>  
54         <optional>true</optional>  
55     </dependency>  
56     <dependency>  
57         <groupId>org.springframework.boot</groupId>  
58         <artifactId>spring-boot-starter-test</artifactId>  
59         <scope>test</scope>  
60     </dependency>  
61  
62     <dependency>  
63         <groupId>org.springframework.boot</groupId>
```

```
64         <artifactId>spring-boot-starter-  
freemarker</artifactId>  
65     </dependency>  
66  
67     <dependency>  
68         <groupId>org.springframework.boot</groupId>  
69         <artifactId>spring-boot-starter</artifactId>  
70         <exclusions>  
71             <exclusion>  
72                 <groupId>org.springframework.boot</groupId>  
73                 <artifactId>spring-boot-starter-  
logging</artifactId>  
74             </exclusion>  
75         </exclusions>  
76     </dependency>  
77  
78     <dependency>  
79         <groupId>org.springframework.boot</groupId>  
80         <artifactId>spring-boot-starter-log4j</artifactId>  
81         <version>1.3.8.RELEASE</version>  
82     </dependency>  
83  
84     <dependency>  
85         <groupId>tk.mybatis</groupId>  
86         <artifactId>mapper-spring-boot-starter</artifactId>  
87         <version>2.0.4</version>  
88     </dependency>  
89  
90 </dependencies>  
91  
92 <build>  
93     <plugins>  
94         <plugin>  
95             <groupId>org.springframework.boot</groupId>  
96             <artifactId>spring-boot-maven-  
plugin</artifactId>  
97         </plugin>  
98     </plugins>  
99 </build>  
100  
101 </project>
```

yml配置

```
1  spring:
2    datasource:
3      master:
4        name: master
5        url: jdbc:mysql://localhost:3306/test
6        connectionProperties:
7          serverTimezone=CTT;useUnicode=true;characterEncoding=utf-
8            8;allowMultiQueries=true;druid.stat.mergeSql=true;druid.stat.slow
9            SqlMillis=5000
10         username: zero
11         password: 123456
12         #配置初始化大小、最小、最大
13         initialSize: 5
14         minIdle: 5
15         maxActive: 20
16         #配置从连接池获取连接等待超时的时间
17         maxWait: 10000
18         # 配置一个连接在池中最大空闲时间，单位是毫秒
19         minEvictableIdleTimeMillis: 300000
20         #配置间隔多久启动一次DestroyThread，对连接池内的连接才进行一次
21         检测，单位是毫秒。
22         #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
23         过minEvictableIdleTimeMillis设置的值则直接物理关闭。2.在minIdle以内
24         的不处理
25         timeBetweenEvictionRunsMillis: 600000
26         #申请连接时执行validationQuery检测连接是否有效，做了这个配置会
27         降低性能
28         testOnBorrow: false
29         #归还连接时执行validationQuery检测连接是否有效，做了这个配置会
30         降低性能
31         testOnReturn: false
32         #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
33         时间超过minEvictableIdleTimeMillis进行检查，否则不检查;false时，不检
34         查
35         testWhileIdle: true
36         #是否缓存preparedStatement，也就是PSCache。PSCache对支持游标的
37         数据库性能提升巨大，比如说oracle。在mysql5.5（不含）以下建议关闭
38         poolPreparedStatements: true
39         maxPoolPreparedStatementPerConnectionSize: 20
```

```
29      #检测语句
30      validationQuery: SELECT 1 FROM DUAL
31      filters: stat,wall,log4j
32      slave:
33          name: slave
34          url: jdbc:mysql://localhost:3306/test2
35          connectionProperties:
serverTimezone=CTT;useUnicode=true;characterEncoding=utf-
8;allowMultiQueries=true;druid.stat.mergeSql=true;druid.stat.slow
SqlMillis=5000
36          username: zero
37          password: 123456
38      #配置初始化大小、最小、最大
39      initialSize: 5
40      minIdle: 5
41      maxActive: 20
42      #配置从连接池获取连接等待超时的时间
43      maxWait: 10000
44      # 配置一个连接在池中最大空闲时间，单位是毫秒
45      minEvictableIdleTimeMillis: 300000
46      #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。
47      #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
48      timeBetweenEvictionRunsMillis: 600000
49      #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
50      testOnBorrow: false
51      #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
52      testOnReturn: false
53      #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
54      testWhileIdle: true
55      #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
56      poolPreparedStatements: true
57      maxPoolPreparedStatementPerConnectionSize: 20
58      #检测语句
59      validationQuery: SELECT 1 FROM DUAL
```

```
60     filters: stat,wall,log4j
61     druid:
62         #监控页面设置
63     stat-view-servlet:
64         enabled: true
65         loginUsername: admin
66         loginPassword: admin
67     web-stat-filter:
68         enabled: true
69
70 mybatis:
71     configuration:
72         map-underscore-to-camel-case: true
```

entity实体

```
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
9  import lombok.Data;
10
11 @Data
12 @Table(name = "users")
13 public class User {
14     @Id
15     @GeneratedValue(strategy = GenerationType.IDENTITY)
16     private Integer id;
17     @Column(name = "username")
18     private String userName;
19     private String mobile;
20     private String nickName;
21     private Integer sex;
22
23 }
24
```


dao类

```
1 package com.zhuozhi.dao;
2
3 import com.zhuozhi.entity.User;
4
5 import tk.mybatis.mapper.common.Mapper;
6
7 public interface UserDao extends Mapper<User> {
8
9 }
```

service类

```
1 package com.zhuozhi.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import org.springframework.transaction.annotation.Transactional;
6
7 import com.zhuozhi.dao.UserDao;
8 import com.zhuozhi.dbrouter.DS;
9 import com.zhuozhi.entity.User;
10
11 @Service
12 public class UserService {
13     @Autowired
14     UserDao userDao;
15
16     @DS(value = "master")
17     public User getUserByIdDS1(Integer id) {
18         return userDao.selectByPrimaryKey(id);
19     }
20
21     @DS(value = "slave")
22     public User getUserByIdDS2(Integer id) {
```

```

23     return userDao.selectByPrimaryKey(id);
24 }
25
26 @Transactional
27 public User saveUser(String userName, String mobile) {
28     User user = new User();
29     user.setUserName(userName);
30     user.setMobile(mobile);
31     userDao.insert(user);
32     return user;
33 }
34
35 }

```

controller类

```

1  package com.zhuozhi.controller;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Controller;
5  import org.springframework.web.bind.annotation.RequestMapping;
6  import org.springframework.web.bind.annotation.ResponseBody;
7
8  import com.zhuozhi.entity.User;
9  import com.zhuozhi.service.UserService;
10
11 import lombok.extern.slf4j.Slf4j;
12
13 @Controller
14 @Slf4j
15 public class UserController {
16
17     @Autowired
18     UserService userService;
19
20
21     /**
22      * master数据源 获取数据测试
23      * @param id
24      * @return

```

```

25     */
26     @RequestMapping("/user/getUserByIdDS1")
27     @ResponseBody
28     public User getUserByIdDS1(Integer id) {
29         log.debug("参数id:{", id);
30         return userService.getUserByIdDS1(id);
31     }
32
33     /**
34      * slave数据源 获取数据测试
35      * @param id
36      * @return
37      */
38     @RequestMapping("/user/getUserByIdDS2")
39     @ResponseBody
40     public User getUserByIdDS2(Integer id) {
41         log.debug("参数id:{", id);
42         return userService.getUserByIdDS2(id);
43     }
44
45     /**
46      * 带事务的数据源测试
47      * @param userName
48      * @param mobile
49      * @return
50      */
51     @RequestMapping("/user/save")
52     @ResponseBody
53     public User save(String userName, String mobile) {
54         return userService.saveUser(userName, mobile);
55     }
56 }
57

```

启动类

```

1 package com.zhuozhi;
2
3 import org.springframework.boot.SpringApplication;

```

```
4 import
   org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 import tk.mybatis.spring.annotation.MapperScan;
7
8 @SpringBootApplication
9 @MapperScan("com.zhuozhi.dao")
10 public class TablerApplication {
11
12     public static void main(String[] args) {
13         SpringApplication.run(TablerApplication.class, args);
14     }
15 }
16
```

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>
