

- 首页
- 归档
- 分类
- 标签
- 关于
- 搜索

## 文章目录 站点概览

- 1. maven依赖
- 2. 初始化数据库
- 3. 配置文件
- 4. 动态切换数据源
- 5. 配置类
- 6. 实体类
- 7. 定义DAO
- 8. 定义Service
- 9. 测试
- 10. GitHub源码
- 10. GitHub源码

10. GitHub源码

10. GitHub源码

## SpringBoot系列 - 多数据源配置

□ 发表于 2017-07-10 | □ 分类于 [spring](#)

项目中经常会出现需要同时连接两个数据源的情况，这里还是演示基于MyBatis来配置两个数据源，并演示如何切换不同的数据源。

网上的一些例子都写的有点冗余，这里我通过自定义注解+AOP的方式，来简化这种数据源的切换操作。

### maven依赖

```
1 <properties>
2   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
3   <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
4   <java.version>1.8</java.version>
5   <druid.version>1.1.2</druid.version>
6   <mysql-connector.version>8.0.7-dmr</mysql-connector.version>
7   <mybatis-plus.version>2.1.8</mybatis-plus.version>
8   <mybatisplus-spring-boot-starter.version>1.0.5</mybatisplus-spring-boot-starter.version>
9 </properties>
10
11 <dependencies>
12   <dependency>
13     <groupId>org.springframework.boot</groupId>
14     <artifactId>spring-boot-starter-aop</artifactId>
15   </dependency>
16   <dependency>
17     <groupId>mysql</groupId>
18     <artifactId>mysql-connector-java</artifactId>
19     <version>${mysql-connector.version}</version>
20     <scope>runtime</scope>
21   </dependency>
22   <dependency>
23     <groupId>com.alibaba</groupId>
24     <artifactId>druid</artifactId>
25     <version>${druid.version}</version>
26   </dependency>
27   <!-- MyBatis plus增强和springboot的集成-->
28   <dependency>
29     <groupId>com.baomidou</groupId>
30     <artifactId>mybatis-plus</artifactId>
```

63% ↑ 0.03K ↓ 0.06K

63% ↑ 0.03K ↓ 0.06K

63% ↑ 0.03K ↓ 0.06K

63% ↑ 0K ↓ 0.03K

63% ↑ 0K ↓ 0.03K

63% ↑ 0K ↓ 0.03K

63% ↑ 0K ↓ 0.03K

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

```
31     <version>${mybatis-plus.version}</version>
32 </dependency>
33 <dependency>
34     <groupId>com.baomidou</groupId>
35     <artifactId>mybatisplus-spring-boot-starter</artifactId>
36     <version>${mybatisplus-spring-boot-starter.version}</version>
37 </dependency>
38
39 <dependency>
40     <groupId>org.springframework.boot</groupId>
41     <artifactId>spring-boot-starter-test</artifactId>
42     <scope>test</scope>
43 </dependency>
44 <dependency>
45     <groupId>org.hamcrest</groupId>
46     <artifactId>hamcrest-all</artifactId>
47     <version>1.3</version>
48     <scope>test</scope>
49 </dependency>
50 </dependencies>
```

## 初始化数据库

这里我们需要创建两个数据库，初始化脚本如下：

```
1  # -----以下是pos业务库开始-----
2  CREATE DATABASE IF NOT EXISTS pos default charset utf8 COLLATE utf8_general_ci;
3  SET FOREIGN_KEY_CHECKS=0;
4  USE pos;
5
6  -- 后台管理用户表
7  DROP TABLE IF EXISTS `t_user`;
8  CREATE TABLE `t_user` (
9      `id` INT(11) PRIMARY KEY AUTO_INCREMENT COMMENT '主键ID',
10     `username` VARCHAR(32) NOT NULL COMMENT '账号',
11     `name` VARCHAR(16) DEFAULT '' COMMENT '名字',
12     `password` VARCHAR(128) DEFAULT '' COMMENT '密码',
13     `salt` VARCHAR(64) DEFAULT '' COMMENT 'md5密码盐',
14     `phone` VARCHAR(32) DEFAULT '' COMMENT '联系电话',
15     `tips` VARCHAR(255) COMMENT '备注',
16     `state` TINYINT(1) DEFAULT 1 COMMENT '状态 1:正常 2:禁用',
17     `created_time` DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
18     `updated_time` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
19 ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='后台管理用户表';
20
21 # 下面是pos数据库中的插入数据
22 INSERT INTO `t_user` VALUES (1,'admin','系统管理员','123456','www','17890908889','系统管理员',1
```

63% ↑ OK ↓ OK

63% ↑ OK ↓ OK

63% ↑ OK ↓ OK

63% ↑ 0.6K ↓ 0.8K

63% ↑ 0.6K ↓ 0.8K

63% ↑ 0.6K ↓ 0.8K

63% ↑ OK ↓ 0.1K

63% ↑ OK ↓ 0.1K

63% ↑ OK ↓ 0.1K

63% ↑ 0.03K ↓ 0.1K

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

```
23 INSERT INTO `t_user` VALUES (2,'aix','张三','123456','eee', '17859569358', '', 1, '2017-12-12 09
24
25
26 # -----以下biz业务库开始-----
27 CREATE DATABASE IF NOT EXISTS pos default charset utf8 COLLATE utf8_general_ci;
28 SET FOREIGN_KEY_CHECKS=0;
29 USE biz;
30
31 -- 后台管理用户表
32 DROP TABLE IF EXISTS `t_user`;
33 CREATE TABLE `t_user` (
34   `id` INT(11) PRIMARY KEY AUTO_INCREMENT COMMENT '主键ID',
35   `username` VARCHAR(32) NOT NULL COMMENT '账号',
36   `name` VARCHAR(16) DEFAULT '' COMMENT '名字',
37   `password` VARCHAR(128) DEFAULT '' COMMENT '密码',
38   `salt` VARCHAR(64) DEFAULT '' COMMENT 'md5密码盐',
39   `phone` VARCHAR(32) DEFAULT '' COMMENT '联系电话',
40   `tips` VARCHAR(255) COMMENT '备注',
41   `state` TINYINT(1) DEFAULT 1 COMMENT '状态 1:正常 2:禁用',
42   `created_time` DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
43   `updated_time` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COM
44 ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='后台管理用户表';
45
46
47 # 下面是biz数据库中的插入数据
48 INSERT INTO `t_user` VALUES (1,'admin1','系统管理员','123456','www', '17890908889', '系统管理员',
49 INSERT INTO `t_user` VALUES (2,'aix1','张三','123456','eee', '17859569358', '', 1, '2017-12-12 09
```

可以看到我创建了两个数据库pos和biz，同时还初始化了用户表，并分别插入两条初始数据。注意用户名数据不相同。

## 配置文件

接下来修改application.yml配置文件，如下：

```
1 ##### 自定义配置 #####
2 xncoding:
3   muti-datasource-open: true #是否开启多数据源(true/false)
4
5 ##### mybatis-plus配置 #####
6 mybatis-plus:
7   mapper-locations: classpath*:com/xncoding/pos/common/dao/repository/mapping/*.xml
8   typeAliasesPackage: >
9     com.xncoding.pos.common.dao.entity
10  global-config:
11    id-type: 0 # 0:数据库ID自增 1:用户输入id 2:全局唯一id(IdWorker) 3:全局唯一ID(uuid)
12    db-column-underline: false
```

63% ↑ 0.03K ↓ 0.1K

63% ↑ 0.03K ↓ 0.1K

63% ↑ 0.03K ↓ 0.1K

63% ↑ 0K ↓ 0K

63% ↑ 0K ↓ 0K

63% ↑ 0K ↓ 0K

62% ↑ 0K ↓ 0.2K

62% ↑ 0K ↓ 0.2K

62% ↑ 0K ↓ 0.2K

63% ↑ 0.06K ↓ 0.07K

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

```
13     refresh-mapper: true
14     configuration:
15         map-underscore-to-camel-case: true
16         cache-enabled: true #配置的缓存的全局开关
17         lazyLoadingEnabled: true #延时加载的开关
18         multipleResultSetsEnabled: true #开启的话，延时加载一个属性时会加载该对象全部属性，否则按需加载属性
19
20     #默认数据源
21     spring:
22         datasource:
23             url: jdbc:mysql://127.0.0.1:3306/pos?useSSL=false&autoReconnect=true&tinyInttisBit=false&use
24             username: root
25             password: 123456
26
27     #多数据源
28     biz:
29         datasource:
30             url: jdbc:mysql://127.0.0.1:3306/biz?useSSL=false&autoReconnect=true&tinyInttisBit=false&use
31             username: root
32             password: 123456
```

解释一下：

这里我添加了一个自定义配置项 `muti-datasource-open`，用来控制是否开启多数据源支持。这个配置项后面我会用到。接下来定义MyBatis的配置，最后定义了两个MySQL数据库的连接信息，一个是pos库，一个是biz库。

## 动态切换数据源

这里通过Spring的AOP技术实现数据源的动态切换。

多数据源的常量类：

```
1 public interface DSEnum {
2     String DATA_SOURCE_CORE = "dataSourceCore"; //核心数据源
3     String DATA_SOURCE_BIZ = "dataSourceBiz"; //其他业务的数据源
4 }
```

datasource的上下文，用来存储当前线程的数据源类型：

```
1 public class DataSourceContextHolder {
2
3     private static final ThreadLocal<String> contextHolder = new ThreadLocal<String>();
4
5     /**
6      * @param dataSourceType 数据库类型
```



10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

```
7      * @Description: 设置数据源类型
8      */
9      public static void setDataSourceType(String dataSourceType) {
10         contextHolder.set(dataSourceType);
11     }
12
13     /**
14      * @Description: 获取数据源类型
15      */
16     public static String getDataSourceType() {
17         return contextHolder.get();
18     }
19
20     /**
21      * @Description: 清除数据源类型
22      */
23     public static void clearDataSourceType() {
24         contextHolder.remove();
25     }
26 }
```

定义动态数据源，继承 `AbstractRoutingDataSource`：

```
1 public class DynamicDataSource extends AbstractRoutingDataSource {
2
3     @Override
4     protected Object determineCurrentLookupKey() {
5         return DataSourceContextHolder.getDataSourceType();
6     }
7 }
```

接下来自定义一个注解，用来在Service方法上面注解使用哪个数据源：

```
1 /**
2  * 多数据源标识
3  *
4  * @author xiongneng
5  */
6 @Inherited
7 @Retention(RetentionPolicy.RUNTIME)
8 @Target({ElementType.METHOD})
9 public @interface DataSource {
10     String name() default "";
11 }
```

最后，最核心的AOP类定义：

```
1 /**
```

63% ↑ 0.03K  
↓ 50.1K

63% ↑ 0.03K  
↓ 50.1K

63% ↑ 0K  
↓ 0.2K

63% ↑ 0K  
↓ 0.2K

63% ↑ 0K  
↓ 0.2K

63% ↑ 0.03K  
↓ 0.04K

63% ↑ 0.03K  
↓ 0.04K

63% ↑ 0.03K  
↓ 0.04K

63% ↑ 0.03K  
↓ 0.07K

63% ↑ 0.03K  
↓ 0.07K

63% ↑ 0.03K  
↓ 0.07K

63% ↑ 0.03K  
↓ 0.07K

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

```
2  * 多数据源切换的aop
3  *
4  * @author xiongeng
5  */
6  @Aspect
7  @Component
8  @ConditionalOnProperty(prefix = "xncoding", name = "muti-datasource-open", havingValue = "true")
9  public class MultiSourceExAop implements Ordered {
10
11     private Logger log = Logger.getLogger(this.getClass());
12
13     @Pointcut(value = "@annotation(com.xncoding.pos.common.annotation.DataSource)")
14     private void cut() {
15
16     }
17
18     @Around("cut()")
19     public Object around(ProceedingJoinPoint point) throws Throwable {
20
21         Signature signature = point.getSignature();
22         MethodSignature methodSignature = null;
23         if (!(signature instanceof MethodSignature)) {
24             throw new IllegalArgumentException("该注解只能用于方法");
25         }
26         methodSignature = (MethodSignature) signature;
27
28         Object target = point.getTarget();
29         Method currentMethod = target.getClass().getMethod(methodSignature.getName(), methodSignature.getParameterTypes());
30
31         DataSource datasource = currentMethod.getAnnotation(DataSource.class);
32         if (datasource != null) {
33             DataSourceContextHolder.setDataSourceType(datasource.name());
34             log.debug("设置数据源为: " + datasource.name());
35         } else {
36             DataSourceContextHolder.setDataSourceType(DSEnum.DATA_SOURCE_CORE);
37             log.debug("设置数据源为: dataSourceCore");
38         }
39         try {
40             return point.proceed();
41         } finally {
42             log.debug("清空数据源信息! ");
43             DataSourceContextHolder.clearDataSourceType();
44         }
45     }
46
47     /**
48     * aop的顺序要早于spring的事务
49     */
50
51     @Override
```

63% ↑ OK  
↓ OK

63% ↑ OK  
↓ OK

63% ↑ OK  
↓ OK

63% ↑ 0.03K  
↓ OK

63% ↑ 0.03K  
↓ OK

63% ↑ 0.03K  
↓ OK

64% ↑ OK  
↓ OK

64% ↑ OK  
↓ OK

64% ↑ OK  
↓ OK

64% ↑ OK  
↓ OK

63% ↑ OK  
↓ 0.2K

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

```
52     public int getOrder() {  
53         return 1;  
54     }  
55  
56 }
```

这里使用到了注解@ConditionalOnProperty，只有当我的配置文件中muti-datasource-open=true的时候注解才会生效。

另外还有一个要注意的地方，就是order，aop的顺序一定要早于spring的事务，这里我将它设置成1，后面你会看到我将spring事务顺序设置成2。

## 配置类

首先有两个属性类：

1. `DruidProperties` 连接池的属性类
2. `MutiDataSourceProperties` biz数据源的属性类

然后定义配置类：

```
1  @Configuration  
2  @EnableTransactionManagement(order = 2)  
3  @MapperScan(basePackages = {"com.xncoding.pos.common.dao.repository"})  
4  public class MybatisPlusConfig {  
5  
6      @Autowired  
7      DruidProperties druidProperties;  
8  
9      @Autowired  
10     MutiDataSourceProperties mutiDataSourceProperties;  
11  
12     /**  
13      * 核心数据源  
14      */  
15     private DruidDataSource coreDataSource() {  
16         DruidDataSource dataSource = new DruidDataSource();  
17         druidProperties.config(dataSource);  
18         return dataSource;  
19     }  
20  
21     /**  
22      * 另一个数据源  
23      */  
24     private DruidDataSource bizDataSource() {
```

63% ↑ OK  
↓ 0.2K

63% ↑ OK  
↓ 0.2K

63% ↑ 0.03K  
↓ OK

63% ↑ 0.03K  
↓ OK

63% ↑ 0.03K  
↓ OK

63% ↑ OK  
↓ OK

63% ↑ OK  
↓ OK

63% ↑ OK  
↓ OK

63% ↑ 0.03K  
↓ OK

63% ↑ 0.03K  
↓ OK

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

```
25     DruidDataSource dataSource = new DruidDataSource();
26     druidProperties.config(dataSource);
27     mutiDataSourceProperties.config(dataSource);
28     return dataSource;
29 }
30
31 /**
32  * 单数据源连接池配置
33  */
34 @Bean
35 @ConditionalOnProperty(prefix = "xncoding", name = "muti-datasource-open", havingValue = "fa
36 public DruidDataSource singleDataSource() {
37     return coreDataSource();
38 }
39
40 /**
41  * 多数据源连接池配置
42  */
43 @Bean
44 @ConditionalOnProperty(prefix = "xncoding", name = "muti-datasource-open", havingValue = "tr
45 public DynamicDataSource mutiDataSource() {
46
47     DruidDataSource coreDataSource = coreDataSource();
48     DruidDataSource bizDataSource = bizDataSource();
49
50     try {
51         coreDataSource.init();
52         bizDataSource.init();
53     } catch (SQLException sql) {
54         sql.printStackTrace();
55     }
56
57     DynamicDataSource dynamicDataSource = new DynamicDataSource();
58     HashMap<Object, Object> hashMap = new HashMap<>();
59     hashMap.put(DSEnum.DATA_SOURCE_CORE, coreDataSource);
60     hashMap.put(DSEnum.DATA_SOURCE_BIZ, bizDataSource);
61     dynamicDataSource.setTargetDataSources(hashMap);
62     dynamicDataSource.setDefaultTargetDataSource(coreDataSource);
63     return dynamicDataSource;
64 }
65 }
```

代码其实很好理解，我就不再多做解释了。

然后步骤跟普通的集成MyBatis是一样的，我简单的过一遍。

## 实体类

63% ↑ 0.03K ↓ OK

63% ↑ OK ↓ OK

63% ↑ OK ↓ OK

63% ↑ OK ↓ OK

64% ↑ 0.03K ↓ OK

64% ↑ 0.03K ↓ OK

64% ↑ 0.4K ↓ 0.6K

64% ↑ 0.4K ↓ 0.6K

64% ↑ 0.4K ↓ 0.6K

64% ↑ 0.5K ↓ 0.1K



[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

```
1 @TableName(value = "t_user")
2 public class User extends Model<User> {
3
4     private static final long serialVersionUID = 1L;
5
6     /**
7      * 主键ID
8      */
9     @TableId(value="id", type= IdType.AUTO)
10    private Integer id;
11    /**
12     * 账号
13     */
14    private String username;
15    /**
16     * 名字
17     */
18    private String name;
19    /**
20     * 密码
21     */
22    private String password;
23    /**
24     * md5密码盐
25     */
26    private String salt;
27    /**
28     * 联系电话
29     */
30    private String phone;
31    /**
32     * 备注
33     */
34    private String tips;
35    /**
36     * 状态 1:正常 2:禁用
37     */
38    private Integer state;
39    /**
40     * 创建时间
41     */
42    private Date createTime;
43    /**
44     * 更新时间
45     */
46    private Date updateTime;
47    // 省略getter/setter方法
48 }
```

64% ↑ 0.5K ↓ 0.1K

64% ↑ 0.5K ↓ 0.1K

64% ↑ 1K ↓ 2.4K

64% ↑ 1K ↓ 2.4K

64% ↑ 1K ↓ 2.4K

64% ↑ 1.5K ↓ 6.6K

64% ↑ 1.5K ↓ 6.6K

64% ↑ 1.5K ↓ 6.6K

64% ↑ 0.5K ↓ 0K

64% ↑ 0.5K ↓ 0K

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

[10. GitHub源码](#)

## 定义DAO

```
1 public interface UserMapper extends BaseMapper<User> {
2
3 }
```

## 定义Service

```
1 @Service
2 public class UserService {
3
4     @Resource
5     private UserMapper userMapper;
6
7     /**
8      * 通过ID查找用户
9      * @param id
10     * @return
11     */
12     public User findById(Integer id) {
13         return userMapper.selectById(id);
14     }
15
16     /**
17      * 通过ID查找用户
18      * @param id
19      * @return
20      */
21     @DataSource(name = DSEnum.DATA_SOURCE_BIZ)
22     public User findById1(Integer id) {
23         return userMapper.selectById(id);
24     }
25
26     /**
27      * 新增用户
28      * @param user
29      */
30     public void insertUser(User user) {
31         userMapper.insert(user);
32     }
33
34     /**
35      * 修改用户
36      * @param user
37      */
38     public void updateUser(User user) {
39         userMapper.updateById(user);
40     }
41 }
```

64% ↑ 0.5K ↓ 0K

64% ↑ 0K ↓ 0.2K

64% ↑ 0K ↓ 0.2K

64% ↑ 0K ↓ 0.2K

64% ↑ 0.03K ↓ 0K

64% ↑ 0.03K ↓ 0K

64% ↑ 0K ↓ 0.09K

64% ↑ 0K ↓ 0.09K

64% ↑ 0K ↓ 0.09K

64% ↑ 0.03K ↓ 0K

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

10. [GitHub源码](#)

```
40     }
41
42     /**
43      * 删除用户
44      * @param id
45      */
46     public void deleteUser(Integer id) {
47         userMapper.deleteById(id);
48     }
49
50 }
```

这里唯一要说明的就是我在方法 `findById1()` 上面增加了注解 `@DataSource(name = DSEnum.DATA_SOURCE_BIZ)`，这样这个方法就会访问biz数据库。

注意，不加注解就会访问默认数据库pos。

## 测试

最后编写一个简单的测试，我只测试 `findById()` 方法和 `findById1()` 方法，看它们是否访问的是不同的数据源。

```
1  @RunWith(SpringRunner.class)
2  @SpringBootTest
3  public class ApplicationTests {
4      private static final Logger log = LoggerFactory.getLogger(ApplicationTests.class);
5
6      @Resource
7      private UserService userService;
8
9      /**
10       * 测试增删改查
11       */
12      @Test
13      public void test() {
14          // 核心数据库中的用户id=1
15          User user = userService.findById(1);
16          assertThat(user.getUsername(), is("admin"));
17
18          // biz数据库中的用户id=1
19          User user1 = userService.findById1(1);
20          assertThat(user1.getUsername(), is("admin1"));
21      }
22  }
```

运行测试，结果显示为green bar，成功！

64% ↑ 0.03K ↓ 0.03K

64% ↑ 0.03K ↓ 0.03K

63% ↑ 0K ↓ 0.03K

63% ↑ 0K ↓ 0.03K

63% ↑ 0K ↓ 0.03K

64% ↑ 2.1K ↓ 8.5K

64% ↑ 2.1K ↓ 8.5K

64% ↑ 2.1K ↓ 8.5K

63% ↑ 0.04K ↓ 0.1K

63% ↑ 0.04K ↓ 0.1K

[10. GitHub源码](#)

[10. GitHub源码](#)

## GitHub源码

[springboot-multisource](#)

# [springboot](#)

▣ [SpringBoot系列 - 集成JWT实现接口权限认证](#)

[SpringBoot系列 - 定时任务](#) ▣

© 2015 - 2019 ▣ 熊能

由 [Hexo](#) 强力驱动 | 粤ICP备17025554号-1

63% ↑ 0.9K ↓ 0.7K

63% ↑ 0.1K ↓ 0.5K

