文章目录 站点概览

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

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
    <druid.version>1.1.2</druid.version>
    <mysql-connector.version>8.0.7-dmr</mysql-connector.version>
    <mybatis-plus.version>2.1.8</mybatis-plus.version>
    <mybatisplus-spring-boot-starter.version>1.0.5</mybatisplus-spring-boot-starter.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-aop</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>${mysql-connector.version}</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>${druid.version}</version>
    </dependency>
    <!-- MyBatis plus增强和springboot的集成-->
    <dependency>
        <groupId>com.baomidou</groupId>
        <artifactId>mybatis-plus</artifactId>
```

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

```
        <version>${mybatis-plus.version}</version>
    </dependency>
    <dependency>
        <groupId>com.baomidou</groupId>
        <artifactId>mybatisplus-spring-boot-starter</artifactId>
        <version>${mybatisplus-spring-boot-starter.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.hamcrest</groupId>
        <artifactId>hamcrest-all</artifactId>
        <version>1.3</version>
        <scope>test</scope>
    </dependency>
</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：

```
# ------------------------------------以下是pos业务库开始----------------------------------------
CREATE DATABASE IF NOT EXISTS pos default charset utf8 COLLATE utf8_general_ci;
SET FOREIGN_KEY_CHECKS=0;
USE pos;

-- 后台管理用户表
DROP TABLE IF EXISTS `t_user`;
CREATE TABLE `t_user` (
  `id`                      INT(11) PRIMARY KEY AUTO_INCREMENT COMMENT '主键ID',
  `username`                VARCHAR(32) NOT NULL COMMENT '账号',
  `name`                    VARCHAR(16) DEFAULT '' COMMENT '名字',
  `password`                VARCHAR(128) DEFAULT '' COMMENT '密码',
  `salt`                    VARCHAR(64) DEFAULT '' COMMENT 'md5密码盐',
  `phone`                   VARCHAR(32) DEFAULT '' COMMENT '联系电话',
  `tips`                    VARCHAR(255) COMMENT '备注',
  `state`                   TINYINT(1) DEFAULT 1 COMMENT '状态 1:正常 2:禁用',
  `created_time`            DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  `updated_time`            DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COM
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='后台管理用户表';

# 下面是pos数据库中的插入数据
INSERT INTO `t_user` VALUES (1,'admin','系统管理员','123456','www', '17890908889', '系统管理员', 1
```

```
INSERT INTO `t_user` VALUES (2,'aix','张三','123456','eee', '17859569358', '', 1, '2017-12-12 09

# ----------------------------------以下biz业务库开始----------------------------------------
CREATE DATABASE IF NOT EXISTS pos default charset utf8 COLLATE utf8_general_ci;
SET FOREIGN_KEY_CHECKS=0;
USE biz;

-- 后台管理用户表
DROP TABLE IF EXISTS `t_user`;
CREATE TABLE `t_user` (
  `id`                      INT(11) PRIMARY KEY AUTO_INCREMENT COMMENT '主键ID',
  `username`                VARCHAR(32) NOT NULL COMMENT '账号',
  `name`                    VARCHAR(16) DEFAULT '' COMMENT '名字',
  `password`                VARCHAR(128) DEFAULT '' COMMENT '密码',
  `salt`                    VARCHAR(64) DEFAULT '' COMMENT 'md5密码盐',
  `phone`                   VARCHAR(32) DEFAULT '' COMMENT '联系电话',
  `tips`                    VARCHAR(255) COMMENT '备注',
  `state`                   TINYINT(1) DEFAULT 1 COMMENT '状态 1:正常 2:禁用',
  `created_time`            DATETIME DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  `updated_time`            DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COM
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COMMENT='后台管理用户表';


# 下面是biz数据库中的插入数据
INSERT INTO `t_user` VALUES (1,'admin1','系统管理员','123456','www', '17890908889', '系统管理员',
INSERT INTO `t_user` VALUES (2,'aix1','张三','123456','eee', '17859569358', '', 1, '2017-12-12 0
```

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：

```
################  自定义配置  ################
xncoding:
  muti-datasource-open: true #是否开启多数据源(true/false)

################  mybatis-plus配置  ################
mybatis-plus:
  mapper-locations: classpath*:com/xncoding/pos/common/dao/repository/mapping/*.xml
  typeAliasesPackage: >
    com.xncoding.pos.common.dao.entity
  global-config:
    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)
    db-column-underline: false
```

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

```
    refresh-mapper: true
  configuration:
    map-underscore-to-camel-case: true
    cache-enabled: true #配置的缓存的全局开关
    lazyLoadingEnabled: true #延时加载的开关
    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

#默认数据源
spring:
  datasource:
    url: jdbc:mysql://127.0.0.1:3306/pos?useSSL=false&autoReconnect=true&tinyInt1isBit=false&use
    username: root
    password: 123456

#多数据源
biz:
  datasource:
    url: jdbc:mysql://127.0.0.1:3306/biz?useSSL=false&autoReconnect=true&tinyInt1isBit=false&use
    username: root
    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：

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

datasource的上下文，用来存储当前线程的数据源类型：

```
public class DataSourceContextHolder {

    private static final ThreadLocal<String> contextHolder = new ThreadLocal<String>();

    /**
     * @param dataSourceType 数据库类型
```

```
     * @Description: 设置数据源类型
     */
    public static void setDataSourceType(String dataSourceType) {
        contextHolder.set(dataSourceType);
    }

    /**
     * @Description: 获取数据源类型
     */
    public static String getDataSourceType() {
        return contextHolder.get();
    }

    /**
     * @Description: 清除数据源类型
     */
    public static void clearDataSourceType() {
        contextHolder.remove();
    }
}
```

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

```
public class DynamicDataSource extends AbstractRoutingDataSource {

    @Override
    protected Object determineCurrentLookupKey() {
        return DataSourceContextHolder.getDataSourceType();
    }
}
```

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：

```
/**
 * 多数据源标识
 *
 * @author xiongneng
 */
@Inherited
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface DataSource {
    String name() default "";
}
```

最后，最核心的AOP类定义：

```
/**
```

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

```
 * 多数据源切换的aop
 *
 * @author xiongneng
 */
@Aspect
@Component
@ConditionalOnProperty(prefix = "xncoding", name = "muti-datasource-open", havingValue = "true")
public class MultiSourceExAop implements Ordered {

    private Logger log = Logger.getLogger(this.getClass());

    @Pointcut(value = "@annotation(com.xncoding.pos.common.annotion.DataSource)")
    private void cut() {

    }

    @Around("cut()")
    public Object around(ProceedingJoinPoint point) throws Throwable {

        Signature signature = point.getSignature();
        MethodSignature methodSignature = null;
        if (!(signature instanceof MethodSignature)) {
            throw new IllegalArgumentException("该注解只能用于方法");
        }
        methodSignature = (MethodSignature) signature;

        Object target = point.getTarget();
        Method currentMethod = target.getClass().getMethod(methodSignature.getName(), methodSign

        DataSource datasource = currentMethod.getAnnotation(DataSource.class);
        if (datasource != null) {
            DataSourceContextHolder.setDataSourceType(datasource.name());
            log.debug("设置数据源为: " + datasource.name());
        } else {
            DataSourceContextHolder.setDataSourceType(DSEnum.DATA_SOURCE_CORE);
            log.debug("设置数据源为: dataSourceCore");
        }
        try {
            return point.proceed();
        } finally {
            log.debug("清空数据源信息！");
            DataSourceContextHolder.clearDataSourceType();
        }
    }


    /**
     * aop的顺序要早于spring的事务
     */
    @Override
```

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() {
```

```
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

10. GitHub源码

10. GitHub源码

10. GitHub源码

10. GitHub源码

10. GitHub源码

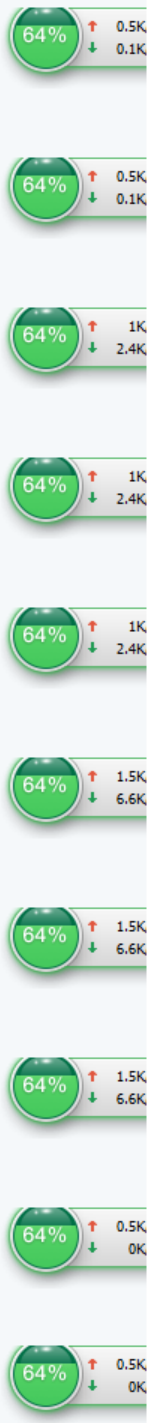
10. GitHub源码

10. GitHub源码

10. GitHub源码

10. GitHub源码

```java
@TableName(value = "t_user")
public class User extends Model<User> {

private static final long serialVersionUID = 1L;

    /**
     * 主键ID
     */
    @TableId(value="id", type= IdType.AUTO)
    private Integer id;
    /**
     * 账号
     */
    private String username;
    /**
     * 名字
     */
    private String name;
    /**
     * 密码
     */
    private String password;
    /**
     * md5密码盐
     */
    private String salt;
    /**
     * 联系电话
     */
    private String phone;
    /**
     * 备注
     */
    private String tips;
    /**
     * 状态 1:正常 2:禁用
     */
    private Integer state;
    /**
     * 创建时间
     */
    private Date createdTime;
    /**
     * 更新时间
     */
    private Date updateTime;
    // 省略getter/setter方法
}
```

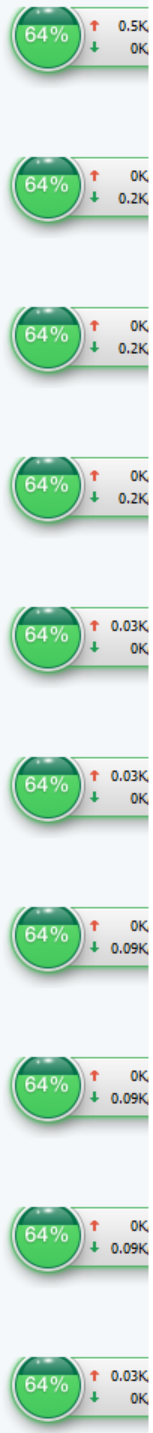


定义DAO

```java
public interface UserMapper extends BaseMapper<User> {

}
```

定义Service

```java
@Service
public class UserService {

    @Resource
    private UserMapper userMapper;

    /**
     * 通过ID查找用户
     * @param id
     * @return
     */
    public User findById(Integer id) {
        return userMapper.selectById(id);
    }

    /**
     * 通过ID查找用户
     * @param id
     * @return
     */
    @DataSource(name = DSEnum.DATA_SOURCE_BIZ)
    public User findById1(Integer id) {
        return userMapper.selectById(id);
    }

    /**
     * 新增用户
     * @param user
     */
    public void insertUser(User user) {
        userMapper.insert(user);
    }

    /**
     * 修改用户
     * @param user
     */
    public void updateUser(User user) {
        userMapper.updateById(user);
```



```
    }

    /**
     * 删除用户
     * @param id
     */
    public void deleteUser(Integer id) {
        userMapper.deleteById(id);
    }

}
```

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。

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class ApplicationTests {
    private static final Logger log = LoggerFactory.getLogger(ApplicationTests.class);

    @Resource
    private UserService userService;

    /**
     * 测试增删改查
     */
    @Test
    public void test() {
        // 核心数据库中的用户id=1
        User user = userService.findById(1);
        assertThat(user.getUsername(), is("admin"));

        // biz数据库中的用户id=1
        User user1 = userService.findById1(1);
        assertThat(user1.getUsername(), is("admin1"));
    }
}
```

运行测试，结果显示为green bar，成功！

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 □