芋道源码 2022-04-02

多数据源（读写分离）

`yudao-spring-boot-starter-mybatis` 技术组件，除了提供 MyBatis 数据库操作，还提供了如下 2 种功能：

- 数据连接池：基于 Alibaba Druid 实现，额外提供监控的能力。
- 多数据源（读写分离）：基于 Dynamic Datasource 实现，支持 Druid 连接池，可集成 Seata 实现分布式事务。

1. 数据连接池

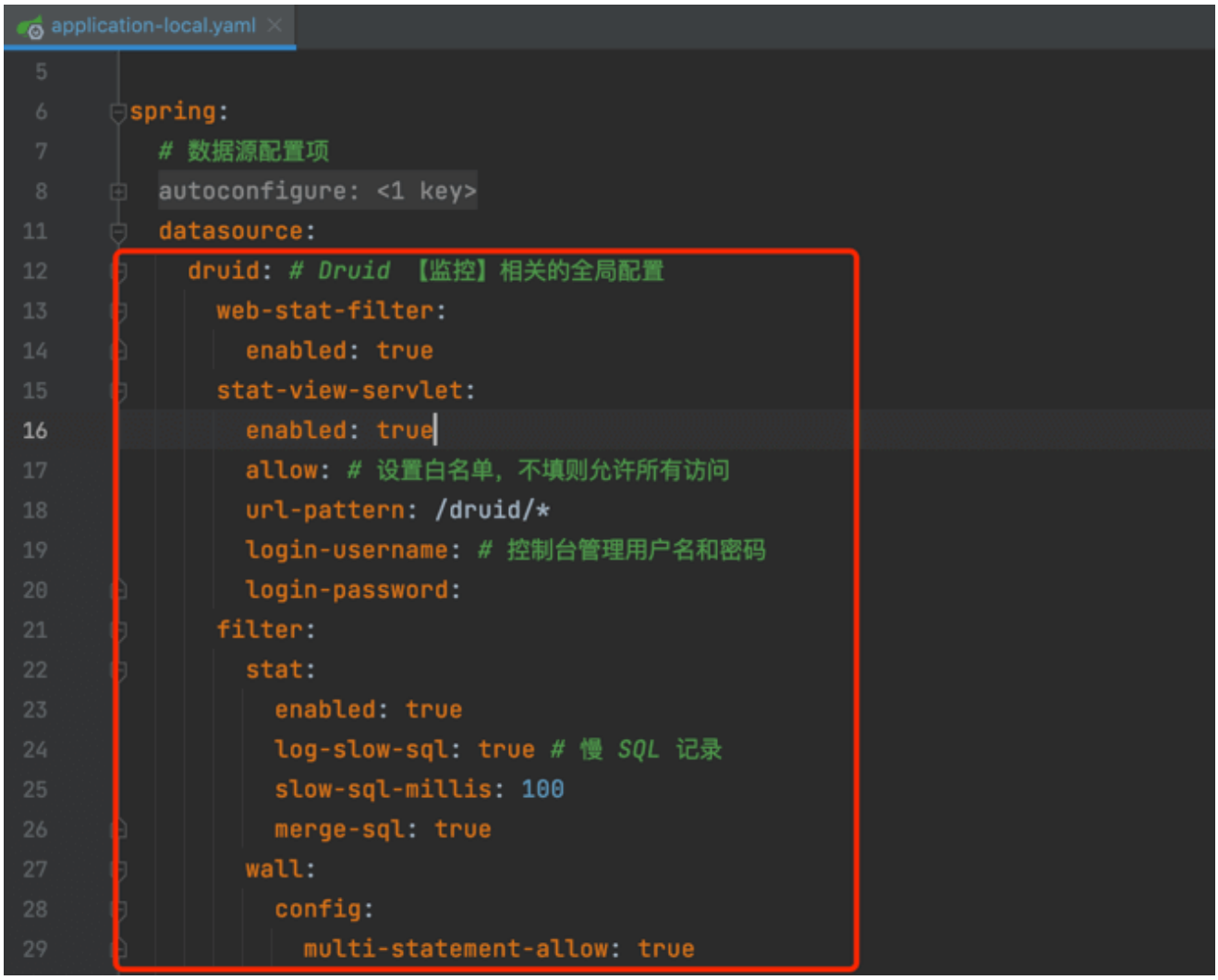
> **友情提示：**
> 如果你未学习过 Druid 数据库连接池，可以后续阅读《芋道 Spring Boot 数据库连接池入门》文章。

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

1.1 Druid 监控配置

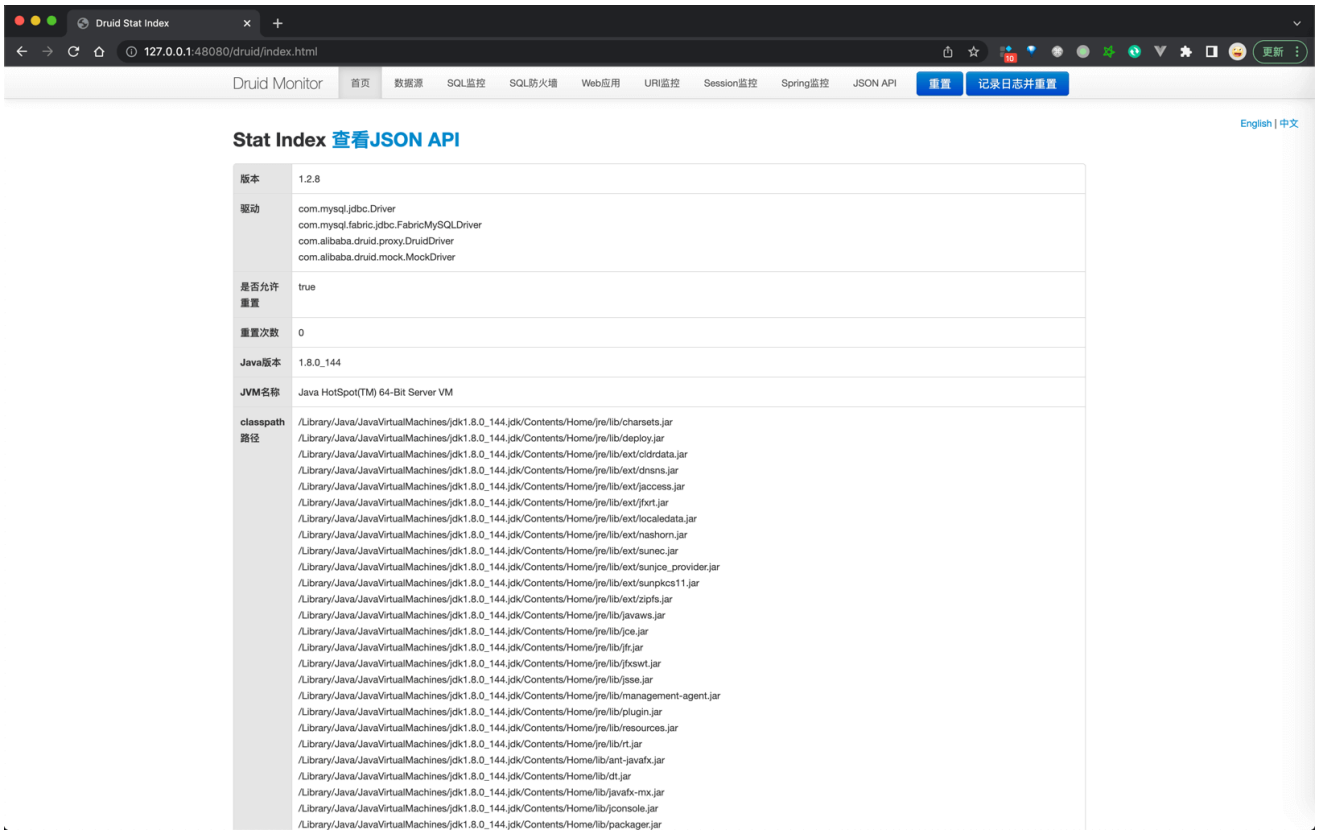
> **友情提示：以 yudao-module-system 服务为例子。**

在 `application-local.yaml` 配置文件中，通过 `spring.datasource.druid` 配置项，仅仅设置了 Druid **监控**相关的配置项目，具体数据库的设置需要使用 Dynamic Datasource 的配置项。如下图所示：

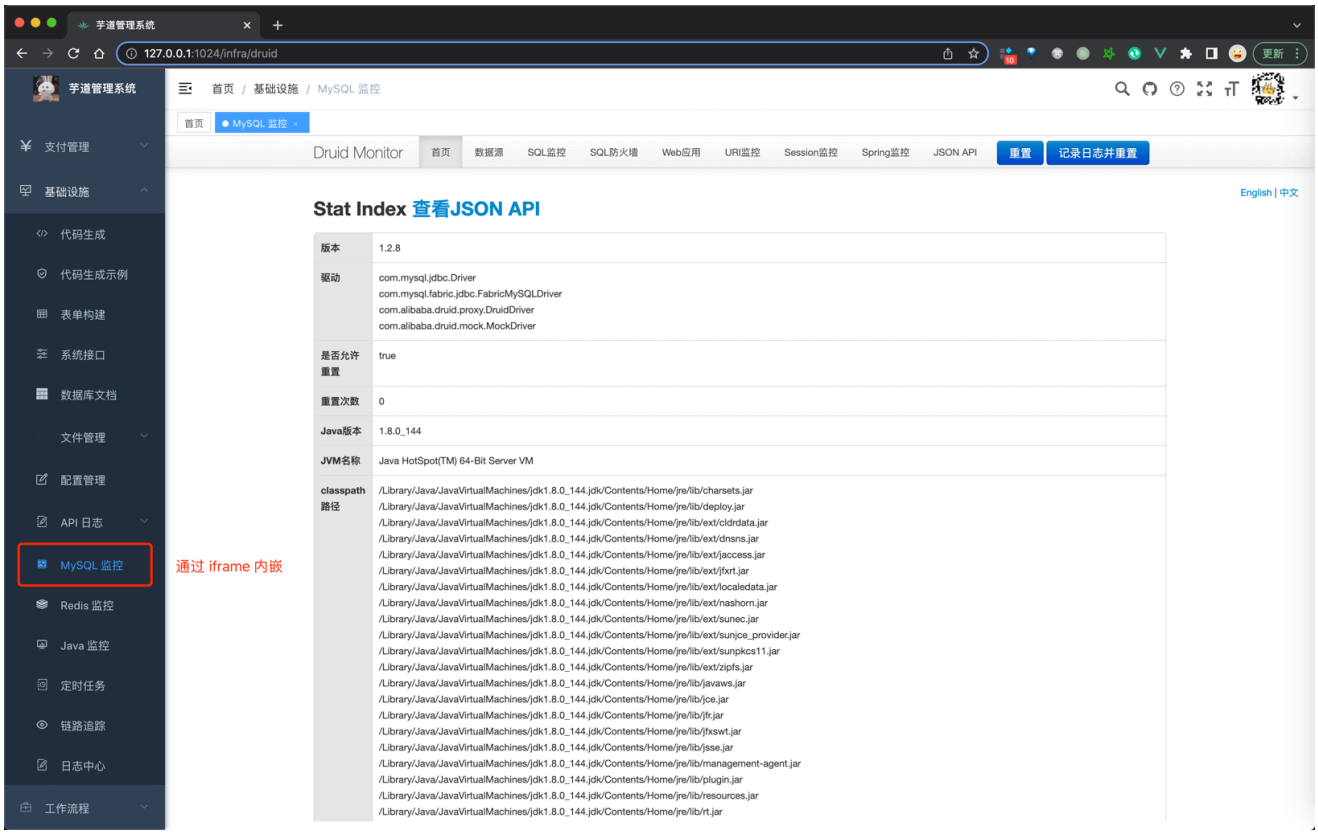


1.2 Druid 监控界面

① 访问后端的 `/druid/index.html` 路径，例如说本地的 `http://127.0.0.1:48080/druid/index.html` 地址，可以查看到 Druid 监控界面。如下图所示：



② 访问前端的 [基础设施 -> MySQL 监控] 菜单，也可以查看到 Druid 监控界面。如下图所示：



> **补充说明：**
>
> 前端 [基础设施 -> MySQL 监控] 菜单，通过 iframe 内嵌后端的 `/druid/index.html` 路径。

> 如果你想自定义地址，可以前往 [基础设置 -> 配置管理] 菜单，设置 key 为 `url.druid` 配置项。

2. 多数据源

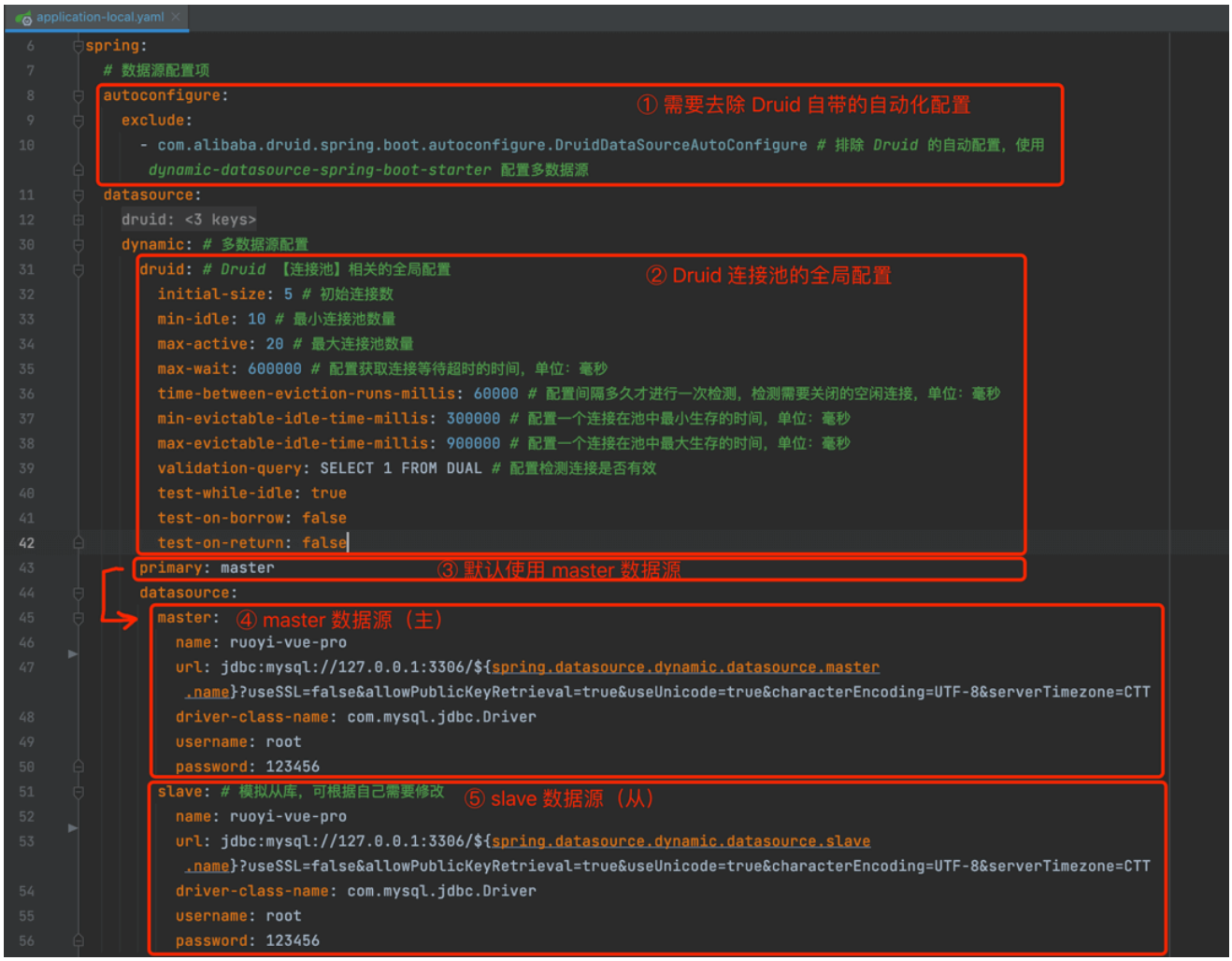
> **友情提示：**
>
> 如果你未学习过多数据源，可以后续阅读《芋道 Spring Boot 多数据源（读写分离）入门》文章。

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>dynamic-datasource-spring-boot-starter</artifactId>
</dependency>
```

2.1 多数据源配置

> **友情提示：以 yudao-module-system 服务为例子。**

在 `application-local.yaml` 配置文件中，通过 `spring.datasource.dynamic` 配置项，配置了 Master-Slave 主从两个数据源。如下图所示：

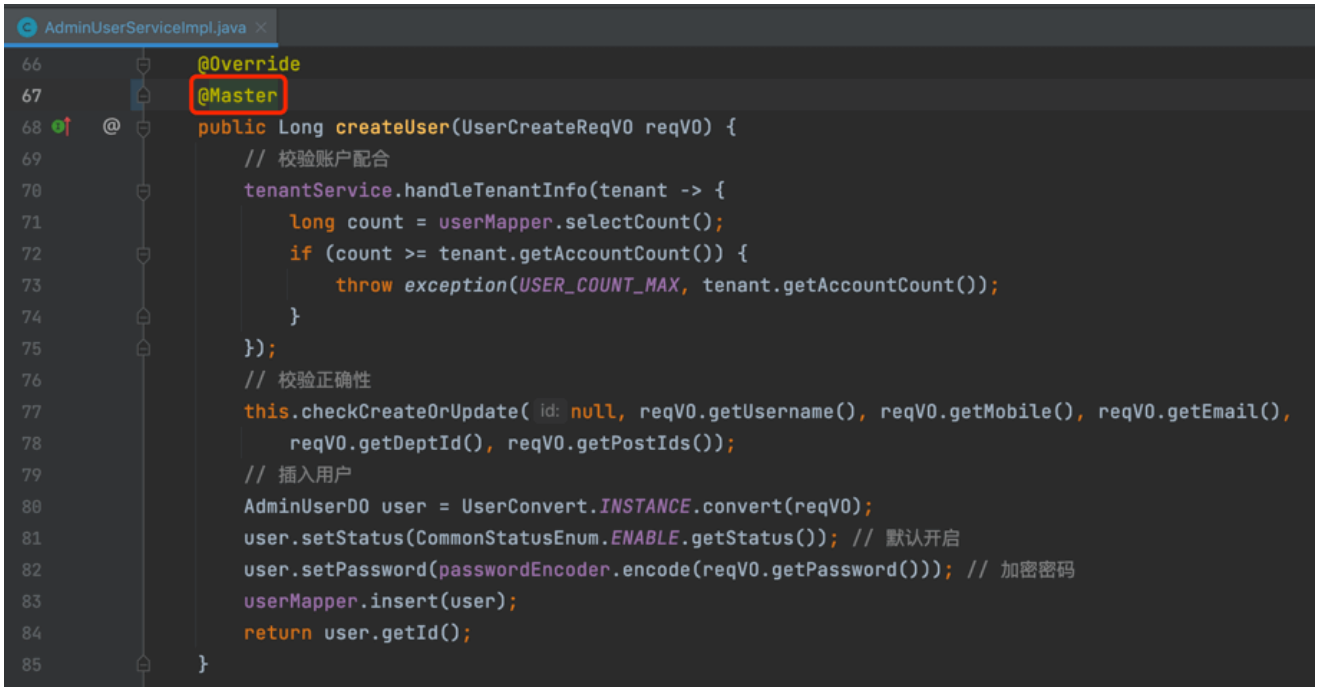



```yaml
spring:
  # 数据源配置项
  autoconfigure:
    exclude:
      - com.alibaba.druid.spring.boot.autoconfigure.DruidDataSourceAutoConfigure # 排除 Druid 的自动配置，使用 dynamic-datasource-spring-boot-starter 配置多数据源
  datasource:
    druid: <3 keys>
    dynamic: # 多数据源配置
      druid: # Druid 【连接池】相关的全局配置
        initial-size: 5 # 初始连接数
        min-idle: 10 # 最小连接池数量
        max-active: 20 # 最大连接池数量
        max-wait: 600000 # 配置获取连接等待超时的时间，单位：毫秒
        time-between-eviction-runs-millis: 60000 # 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位：毫秒
        min-evictable-idle-time-millis: 300000 # 配置一个连接在池中最小生存的时间，单位：毫秒
        max-evictable-idle-time-millis: 900000 # 配置一个连接在池中最大生存的时间，单位：毫秒
        validation-query: SELECT 1 FROM DUAL # 配置检测连接是否有效
        test-while-idle: true
        test-on-borrow: false
        test-on-return: false
      primary: master
      datasource:
        master:
          name: ruoyi-vue-pro
          url: jdbc:mysql://127.0.0.1:3306/${spring.datasource.dynamic.datasource.master.name}?useSSL=false&allowPublicKeyRetrieval=true&useUnicode=true&characterEncoding=UTF-8&serverTimezone=CTT
          driver-class-name: com.mysql.jdbc.Driver
          username: root
          password: 123456
        slave: # 模拟从库，可根据自己需要修改
          name: ruoyi-vue-pro
          url: jdbc:mysql://127.0.0.1:3306/${spring.datasource.dynamic.datasource.slave.name}?useSSL=false&allowPublicKeyRetrieval=true&useUnicode=true&characterEncoding=UTF-8&serverTimezone=CTT
          driver-class-name: com.mysql.jdbc.Driver
          username: root
          password: 123456
```

2.2 数据源切换

2.2.1 @Master 注解

在方法上添加 `@Master` 注解，使用名字为 `master` 的数据源，即使用【主】库，一般适合【写】场景。示例如下图：




```java
@Override
@Master
public Long createUser(UserCreateReqVO reqVO) {
    // 校验账户配合
    tenantService.handleTenantInfo(tenant -> {
        long count = userMapper.selectCount();
        if (count >= tenant.getAccountCount()) {
            throw exception(USER_COUNT_MAX, tenant.getAccountCount());
        }
    });
    // 校验正确性
    this.checkCreateOrUpdate( id: null, reqVO.getUsername(), reqVO.getMobile(), reqVO.getEmail(),
        reqVO.getDeptId(), reqVO.getPostIds());
    // 插入用户
    AdminUserDO user = UserConvert.INSTANCE.convert(reqVO);
    user.setStatus(CommonStatusEnum.ENABLE.getStatus()); // 默认开启
    user.setPassword(passwordEncoder.encode(reqVO.getPassword())); // 加密密码
    userMapper.insert(user);
    return user.getId();
}
```

由于项目的 `spring.datasource.dynamic.primary` 为 `master` ，默认使用【主】库，所以无需手动添加 `@Master` 注解。

2.2.2 @Slave 注解

在方法上添加 `@Slave` 注解，使用名字为 `slave` 的数据源，即使用【从】库，一般适合【读】场景。示例如下图：

2.2.3 @DS 注解

在方法上添加 `@DS` 注解，使用指定名字的数据源，适合多数据源的情况。示例如下图：

2.3 分布式事务

在使用 Spring `@Transactional` 声明的事务中，无法进行数据源的切换，此时有 3 种解决方案：

① 拆分成多个 Spring 事务，每个事务对应一个数据源。如果是【写】场景，可能会存在多数据源的事务不一致的问题。

② 引入 Seata 框架，提供完整的分布式事务的解决方案，可学习《芋道 Seata 极简入门》文章。

③ 使用 Dynamic Datasource 提供的 `@DSTransactional` 注解，支持多数据源的切换，不提供绝对可靠的多数据源的事务一致性（强于 ① 弱于 ②），可学习《DSTransactional 实现源码分析》文章。

3. 分库分表

建议采用 ShardingSphere 的子项目 Sharding-JDBC 完成分库分表的功能，可阅读《芋道 Spring Boot 分库分表入门》文章，学习如何整合进项目。

← MyBatis 联表&分页查询

Redis 缓存→