活映数据服务平台软件

使用手册

撰写人	张 鑫
日期	2024-7-5
说明	初版

目录

1. 概述

1.1. 软件名称

软件名称：活映数据服务平台软件，版本号：V1.0。简称“数据服务”，英文 DataService（下面用英文简称代替之）。

1.2. 简介

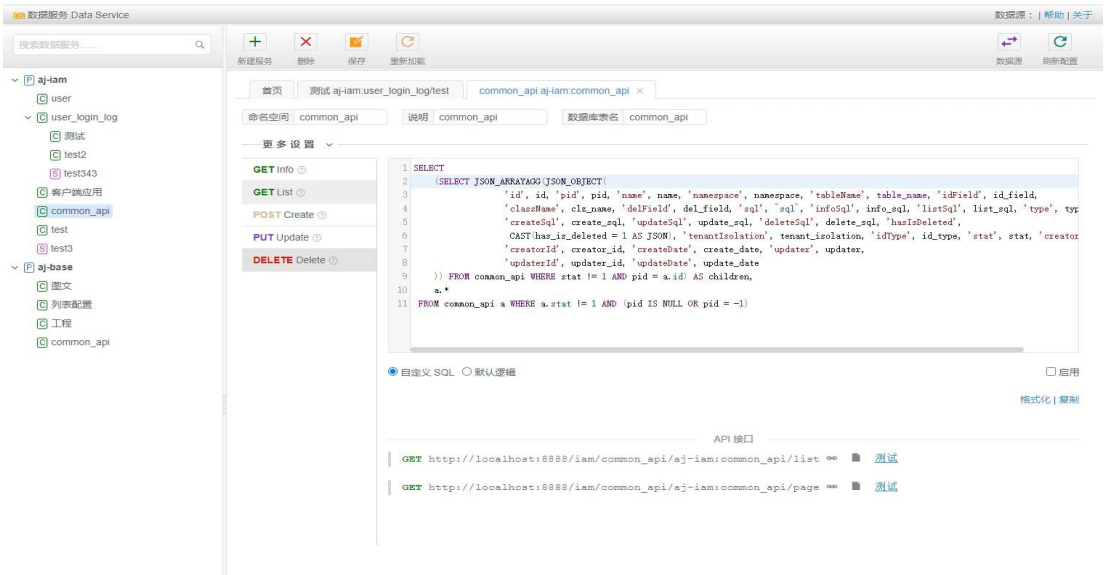
只需写 SQL 业务逻辑（甚至零代码不写！），即可快速搭建 CRUD 接口服务——最简单的方式：零代码，在页面上配置好参数，自动生成 SQL 并且直接转化成 HTTP API。Java 企业级开发中，要写 Model、DAO、Service 和 Controller 代码是一件非常繁琐的事情，里面存在着大量的重复工作，数据服务就是为了解决这个问题而生的。按照业务复杂程度的递进，可分为以下三种数据服务的应用模式：

- 大量的基础数据。这类数据表的特点是没什么或少量常见的业务逻辑，采用 DataService 创建通用的 CRUD 服务即可完成。
- 自定义的业务逻辑，并非简单的 CRUD，但通过 SQL 仍能实现，不用额外写 Java 的业务代码，尤其适用于 BI 报表、数据可视化大屏的后端接口开发。
- 复杂的业务逻辑，需 Java+SQL 协力完成。这时 DataService 仍能作为一种 ORM 机制出现，相当于一个 Data Access Object，返回 Java Bean 实体。

数据服务不是代码生成器，更直观地说它是把一切常见 CRUD 工作抽象化，然后使之可配置化的快速业务开发工具。

1.3. 软件主界面

下面为数据服务的主界面截图。



1.4. 技术框架选型

1.4.1. 前端技术

- 开发语言：JavaScript、TypeScript
- 基础框架：Vue、iView.js
- 开发软件：VS Code

1.4.2. 后端技术

- 开发语言：Java 1.8
- 数据库：MySQL Server 8
- 基础框架：Spring、AJ Framework
- 权限安全：OAuth2、JWT
- 开发软件：IDEA

2. 运行程序

2.1. 运行环境

- 硬件环境：2G MHz CPU/512MB RAM/10G HDD
- 软件环境：Linux/JDK8/MySQL Server 8

2.2. 运行步骤

2.2.1. 下载与源码

数据服务依赖于 AJ Data 库，严格来说它是该框架下的一个组件，故引入 AJ-Data 相关依赖即可。AJ 框架足够轻量级，包括依赖包在内的 JAR 包约 200 多 kb。

```
<!-- AJ-Data 库 -->
<dependency>
   <groupid>com.ajaxjs</groupid>
   <artifactid>ajaxjs-data</artifactid>
   <version>1.2.1</version>
</dependency>
```

源码：https://github.com/lightweight-component/aj-data。
JavaDoc：https://dev.ajaxjs.com/docs/javadoc/aj-data/。

2.2.2. 创建数据库

数据服务依赖两张表 ds_common_api 和 ds_project 作为配置数据表。在你的数据库中执行创建表的 DDL，或者执行 init-dataservice.sql DDL 下载：
https://github.com/lightweight-component/aj-data/blob/main/src/main/sql/init-dataservice.sql。

2.2.3. 整合到你的程序中

数据服务作为 jar 包整合到你的程序中去，自然离不开数据库连接的配置——那怎么配置呢？首先你的程序应该是 Spring 的，返回一个标准 JDBC 的数据源对象，即 DataSouce 类型的 bean。例如

```
@Bean(value = "dataSource", destroyMethod = "close")
DataSource getDs() {
   return JdbcConn.setupJdbcPool("com.mysql.cj.jdbc.Driver", url,
user, psw);
}
```

只要有了 DataSource 对象，便可以返回 Connection 对象，产生数据库连接。数据服务不关心你具体怎么配置数据库的，不关心是什么的数据连接池。

下一步是创建 jdbcReader 及 jdbcWriter 两个组件，分别代表底层的读写组件，可以配置一下数据库的模型，但此时可以先不用过多关心它们。

```
@Bean
@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public JdbcWriter jdbcWriter()  {
```

```
    JdbcWriter jdbcWriter = new JdbcWriter();
    jdbcWriter.setIdField("id");
    jdbcWriter.setIsAutoIns(true);
    jdbcWriter.setConn(JdbcConn.getConnection());

    return jdbcWriter;
}

@Bean
@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public JdbcReader jdbcReader()  {
    JdbcReader jdbcReader = new JdbcReader();
    jdbcReader.setConn(JdbcConn.getConnection());

    return jdbcReader;
}
```

接着是创建 CRUD_Service 对象，其实质是一个 DAO，也依赖上述的 jdbcReader 及 jdbcWriter 两个组件。

```
@Bean
@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
public CRUD_Service getCRUD_Service()  {
    CRUD_Service crud = new CRUD_Service();
    crud.setWriter(jdbcWriter());
    crud.setReader(jdbcReader());

    return crud;
}
```

注意，jdbcReader 及 jdbcWriter、CRUD_Service 都不是 Spring 单例 bean，而是 prototype 方式创建，主要目的是为了规避线程安全的问题。

最后一步创建控制器 Controller，@RequestMapping("/common_api") 这里的注解就是数据服务所有 API 的总前缀，一切从这里开始。我们采用“父类-继承”的模式，用户继承类库的基类便好。

```
import com.ajaxjs.data.data_service.DataServiceController;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * 数据服务接口
 */
@RestController
@RequestMapping("/common_api")
public interface CommonApiController extends
DataServiceController {
}
```

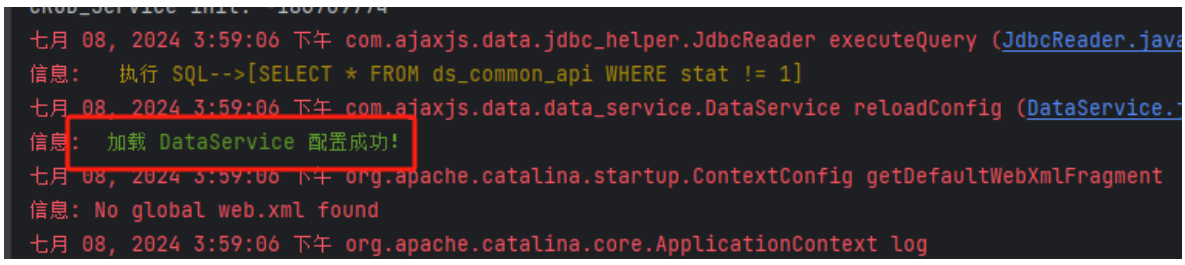
创建一个 Service，也是继承。此步还要调用初始化数据的加载。

```
import com.ajaxjs.base.controller.CommonApiController;
```

```
import com.ajaxjs.data.DataAccessObject;
import com.ajaxjs.data.crud.CRUD_Service;
import com.ajaxjs.data.data_service.DataService;
import
com.ajaxjs.framework.filter.dbconnection.DataBaseConnection;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Lazy;
import org.springframework.context.event.ContextRefreshedEvent;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Service;

import java.sql.Connection;
import java.sql.SQLException;

@Service
public class CommonApiService extends DataService implements
CommonApiController {
   @Autowired
   @Lazy
   private CRUD_Service crudService;

   @Override
   public DataAccessObject getDao() {
      return crudService;
   }

   @EventListener
   public void handleContextRefresh(ContextRefreshedEvent event)
throws SQLException {
      try (Connection conn = DataBaseConnection.initDb()) {
         setDao(crudService);

         reloadConfig();// 在 Spring 初始化完成后执行的操作
      }
   }
}
```

这里 MVC 的 C 是 Java Interface，是 SpringMVC 的新特性，参见《简化 Spring 控制器：只须写接口即可》。

最后启动你的程序，看看是否有相关的启动数据服务的日志。



2.2.4. 前端程序与用户认证

前端为独立的 npm 工程，位于：https://github.com/lightweight-component/data-service-ui。执行下面命令：

```
npm I # 安装依赖
npm run dev # 开发调试
npm run release # 打包
```

用户认证模式采用 token 认证，比较简单。

3. 功能概述

3.1. 登录与注册

4. 技术设计

4.1. 软件总体设计

企业级的开发中包含大量的表单应用，与之对应的是数据库的表。虽然也包含关联表的较复杂操作和逻辑，但总体来说存在大量的重复工作——简而化之就是 CRUD 操作。首先是后台的业务操作，必须明确，抽象一个统一的 CRUD 服务与复杂多变的业务需求并不矛盾：能抽象的就抽象，不能抽象统一的就开放，无他尔。特别是写入操作（创建 or 更新）往往是可以做到统一的，而查询（SELECT）某种程度也能抽象（如分页，等等）。这就需要我们制定一套合理、高效的应用规则，实现上述设计之目标。与前端的关系就是根据前端传入的参数来组装一条完整的 SQL，执行完毕把结果以 JSON 形式返还给前端。对于前端而言，虽然看到不同的 API 接口，但对于后端而言，只是一个 API 接口的入口，然后根据不同 URL 目录作分发而已。

前端的表单界面，实际没必要单独定制。在个性化跟统一化两边应要根据开发成本决定的。当采用统一化设计的时候，自动化生成表单界面变得很重要了。关于表单、列表 UI 生成器我们另外专题再讲。

前端表单做成统一的 CRUD 组件，字段名跟后端一致即可，提交参数为 JSON 传给后端统一的 CRUD 服务即可。

前端列表组件也是，关键在于复杂的 SELECT Query 查询，不同情况下组合的条件查询。甚至有种简单直接的办法，就是前端直接生成 SQL WHERE 语句传到后端，这并不是完整的 SQL，而是 SQL 片断，安全性更高（当然后端也要做好诸如 SQL 注入等的检查）。

退一万步讲，即使不使用统一的 CRUD 接口，也可以把数据服务作为一个 ORM 工具，或者说对 MyBatis 的进一步封装。实际上本作者也是大量这么使用的。这就是数据服务的另外一种模式：DAO 模式（Data Access Object）。DAO 后端开发人员都不陌生，在后端纯粹使用数据服务也是非常自然，无太大入侵的。最大的变化，则是之前在 MyBatis XML 写的 SQL 语句，如今却是放到数据库中保存 SQL 了，当然你不需要进入数据库修改 SQL（当然也不是不可以），而是在数据服务提供 GUI 中编辑 SQL，即时生效无需重启。有一套新的环境管理 SQL，底层仍是 MyBatis，而后端代码仍是程序员熟悉的 DAO。另外一点好处，就是既然在 Java 了，肯定是带类型的 Bean 实体 ORM，而不是 Map（当然用 Map 也是可以，看需要）。

4.2. 架构结构图

数据实体（Entity）有四种基础的操作：增删改查 CRUD。在 DataService 中它们的关系如下表所示。

	创建实体 Create	查询实体 Read	更新实体 Update	删除实体 Delete
HTTP 方法	POST	GET	PUT	DELETE
SQL 命令	INSERT INTO	SELECT	UPDATE	DELTETE/UPDATE（逻辑删除）
DataService 操作	根据提交的数据转化为 INSERT INTO 语句去操作	获取单笔详情 获取多行列表（不分页） 获取多行分页列表	根据提交的数据转化为 UPDATE SQL 语句去操作	删除逻辑或物理删除
API 入参	标准表单或 JSON 数据	Path 参数或 QueryString 参数	标准表单或 JSON 数据	Path 参数或 QueryString 参数，只须 id 参数
API 出参	新建实体的 id	单笔详情 `{}` 对象 列表 `[]` 数组	是否成功	是否成功

DataService 各项功能围绕 CRUD 展开，上述的“SQL 命令”与“DataService 操作”两部分，不仅提供默认通用的 CRUD 操作，而且如果不满足的话，还可以自定义 SQL 逻辑（下小节详述）。 不论哪种方式，均采用约定好的固定搭配请求服务，假设/common_api 为数据服务的专属 API 前缀、 foo 为命名空间，则有以下固定的请求操作。

- GET /common_api/foo/1234 获取单笔详情记录，其中 1234 是 Path 参数，即格式如 /common_api/{namespace}/{id}
- GET /common_api/foo/list 获取多行列表记录，最后是 /list 结尾的，即格式如 /common_api/{namespace}/list
- GET /common_api/foo/page 获取多行列表分页记录，最后是 /page 结尾的，即格式如 /common_api/{namespace}/page，并要传相关的分页参数