MyBatis-Plus-Kit

🚀 **MyBatis-Plus-Kit** 是基于 MyBatis-Plus 的增强组件，提供**零侵入、即插即用**的能力增强，聚焦**自动生成 RESTful API**、**统一分页模型**、**统一响应体封装**等常用需求，极大简化开发流程。

✨ 特性

- ✅ 动态代理生成 RESTful API，实现零Controller编码

> 在实体类上添加 `@AutoApi` 注解，即可自动生成完整 CRUD 接口，无需手写 Controller。

- ✅ 支持手动继承 BaseController 进行个性化定制

  > 如需特殊接口或权限控制，可继承 `BaseController` 自定义扩展。

- ✅ 支持一键生成所有代码

  > 包括实体类、Mapper、Service、Controller 等，并 **自动添加 `@AutoApi` 注解**，开箱即用。

- ✅ 统一分页模型

  > 提供 `PageQuery` + `PageResult` 模型，统一分页请求和响应结构。

- ✅ 自动封装响应体

  > 可配置开启或关闭，对所有 Controller 返回值进行统一包装。

- ✅ 统一异常捕获机制

  > 提供全局异常处理，支持自定义异常配置。

- ✅ 无侵入设计，可按需引入功能

🧩 模块结构

```
mybatis-plus-kit
├── kit-core        // 核心能力，动态代理，通用Service、Controller、分页、封装等
├── kit-starter     // Spring Boot Starter，自动装配、配置处理、响应/异常支持
├── kit-example     // 示例模块，演示如何使用kit-core + kit-starter
├── kit-generator     // 代码生成器，一键生成entity，service，mapper，
controller
```

📦 快速开始

1. 引入依赖（Maven Central）

```xml
<dependency>
  <groupId>io.github.l4vid4</groupId>
  <artifactId>kit-starter</artifactId>
  <version>0.2.0</version>
</dependency>
<dependency>
    <groupId>io.github.l4vid4</groupId>
    <artifactId>kit-generator</artifactId>
    <version>0.2.0</version>
</dependency>
```

2. 新建Test.java，配置代码生成器参数，运行生成所有代码

```java
public class Test {
    public static void main(String[] args) {
        // 数据库链接（必填）
        String url = "jdbc:mysql://192.168.56.10:3306/mp?
useUnicode=true&characterEncoding=UTF-
8&useSSL=false&serverTimezone=Asia/Shanghai";

        String username = "root";

        String password = "root";

        // 生成Entity，Service，Mapper
        new CodeGenerator()
                .dataSource(url, username, password)
                .packageName("io.github.l4vid4.example") //包名（必填）
                .module("kit-example") //模块名
                .tables("user") //表名，支持传多个表（必填）
                .author("l4vid4") //用户名
                .build()
                .execute();

//        // 生成Controller
//        new ControllerGenerator()
//                .dataSource(url, username, password)
//                .packageName("io.github.l4vid4.example")
//                .module("kit-example")
//                .tables("user1")
//                .author("l4vid4")
//                .build()
//                .execute();
    }
```

```
}
```

- 生成的代码示例：

  - **entity**

```
@TableName(value ="user")
@Data
@AutoApi(path = "/user")
public class User implements Serializable {
    /**
     * 用户id
     */
    @TableId(type = IdType.AUTO)
    private Long id;

    /**
     * 用户名
     */
    private String username;

    /**
     * 密码
     */
    private String password;
}
```

  - **service**

```
public interface UserService extends BaseService<User> {

}
```

  - **serviceImpl**

```java
@Service
public class UserServiceImpl extends BaseServiceImpl<UserMapper,
User>
    implements UserService {

}
```

- **mapper**

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

}
```

3. 可以在entity上加上 `@DisableApis(Api.GET_BY_ID)` 动态控制不想暴露的API

```java
@TableName(value ="user")
@Data
@DisableApis(Api.GET_BY_ID)
@AutoApi(path = "/user")
public class User implements Serializable {
    /**
     * 用户id
     */
    @TableId(type = IdType.AUTO)
    private Long id;

    /**
     * 用户名
     */
    private String username;

    /**
     * 密码
     */
    private String password;
}
```

4. 如果不想使用动态代理的功能暴露API，也可以将 `@AutoApi(path = "/user")` 加上 `mode = AutoApiMode.EXTENDS` ，然后手动继承BaseController也会实现所有方法

```java
@RestController
@RequestMapping("/user")
@DisableApis({DisableApis.Api.DELETE, DisableApis.Api.UPDATE}) //自定义不想暴露的接口
public class UserController extends BaseController<User, UserService> {

    //重写User → UserVO转换方法
    @Override
    protected Function<User, UserVO> voConvertor() {
        return user -> {
            UserVO userVO = new UserVO();
            BeanUtils.copyProperties(user, userVO);
            return userVO;
        };
    }
}
```

配置完上述所有后，自动拥有所有增删改查方法。

- 可以在entity或Controller类上加上 `@DisableApis({DisableApis.Api.DELETE, DisableApis.Api.UPDATE})` 自定义不想暴露的接口

HTTP 方法	接口路径	方法名称	功能描述	是否可禁用（通过注解）
`GET`	`/user/getById/{id}`	`getById`	根据 ID 查询单个实体	✅ `GET_BY_ID`
`GET`	`/user/list`	`list`	查询所有数据（不分页）	✅ `LIST`
`GET`	`/user/listByIds`	`listByIds`	根据 ID 集合批量查询（不分页）	✅ `LIST_BY_IDS`

HTTP 方法	接口路径	方法名称	功能描述	是否可禁用（通过注解）
`POST`	`/user/listByCondition`	`listByCondition`	根据实体字段进行条件查询（不分页）	✅ `LIST_BY_CONDITION`
`POST`	`/user/page`	`page`	分页查询实体（返回实体 T）	✅ `PAGE`
`POST`	`/user/pageVo`	`pageVo`	分页查询（返回 VO，默认无转换）	✅ `PAGE_VO`
`POST`	`/user/save`	`save`	新增数据	✅ `SAVE`
`POST`	`/user/update`	`update`	更新数据（根据 ID）	✅ `UPDATE`
`GET`	`/user/deleteById/{id}`	`deleteById`	删除单条数据（根据 ID）	✅ `DELETE_BY_ID`

HTTP 方法	接口路径	方法名称	功能描述	是否可禁用（通过注解）
`POST`	`/user/delete`	`delete`	批量删除（根据 ID 集合）	✅ `DELETE`

📜 通用分页组件

分页参数类

```java
@Data
public class PageQuery {
    private Integer pageNo; // 页码
    private Integer pageSize; // 每一页数据量
    private String sortBy; // 排序字段
    private Boolean isAsc; // 是否正序

    // 将分页条件转换为Page<T>
    public <T> Page<T> toMpPage(OrderItem ... orders){
    }

    // 默认分页条件
    public <T> Page<T> toMpPage(String defaultSortBy, boolean isAsc){
    }

    // 按照create_time字段降序
    public <T> Page<T> toMpPageDefaultSortByCreateTimeDesc() {
    }

    // 按照update_time字段降序
    public <T> Page<T> toMpPageDefaultSortByUpdateTimeDesc() {
    }
}
```

分页响应类

```java
@Data
@AllArgsConstructor
public class PageResult<T> {
    private Long total;  //总条数
    private Long pages; // 总页数
    private List<T> list; // 数据

    // 返回空分页结果
    public static <V, P> PageResult<V> empty(Page<P> p){
    }

    //将MybatisPlus分页结果转为 VO分页结果
    public static <V, P> PageResult<V> of(Page<P> p, Class<V> voClass) {
    }

    //将MybatisPlus分页结果转为 VO分页结果，允许用户自定义PO到VO的转换方式
    public static <V, P> PageResult<V> of(Page<P> p, Function<P, V>
convertor) {
    }
}
```

⚙️ 配置项说明（`application.yml`）

```yaml
mybatis-plus-kit:
  response-wrapper-enabled: true  # 是否启用统一响应封装（默认 true）
  exception-handler-enabled: true # 是否启用全局异常处理（默认 true）
  auto-api-proxy-enabled: true # 是否启用全局动态代理 （默认 true）
  base-package: io.github.l4vid4.example # 包路径
```

✅ TODO（未来计划）

- 开发代码生成插件

- 通用字段自动填充（createTime, updateTime）
- 多数据源/分表兼容支持
- 添加swagger支持

🧑‍💻 贡献指南

欢迎 PR 或 issue！如果你觉得这个项目对你有帮助，欢迎点个 ⭐！

📄 License

Apache 2.0