Atitit 查询类接口的通用与拓展性实践总结

目录

[1. 实现目标 2](#_Toc19292)

[1.1. 扩展性 业务功能调整，代码层面尽可能少的编译部署步骤，如可以免编译免部署更好 2](#_Toc4967)

[1.2. 动态化，与表结构脱钩，字段的变动不会影响接口的编码 无需再次调整代码编译部署 2](#_Toc20978)

[1.3. 灵活性 随时调整决定要显示哪些字段 2](#_Toc17979)

[1.4. 动态化 跨表查询 接口与模型数据尽可能不绑定 2](#_Toc15823)

[1.5. 通用化 同一个技术接口，可以给很多业务功能使用 2](#_Toc27961)

[1.6. 通用化 跨项目公共化，同样的接口，放在下一个项目里面也能通用 2](#_Toc8763)

[1.7. 跨语言 接口在不同语言的实现模式(java js python php c#等）基本类似 2](#_Toc26726)

[1.8. 公有规范性，无需专门文档说明，尽可能使用现有规范 2](#_Toc6524)

[2. 方式 3](#_Toc8677)

[2.1. 适当抽象一下，与具体业务脱钩，数据标识参数化配置化，避免硬编码 3](#_Toc601)

[2.2. 如有可能优先使用动态对象 避免硬编码 3](#_Toc19941)

[2.3. 如有尽可能的避免使用硬编码模式域模型 do vo dto bo，可以使用动态化的模型json map 等 3](#_Toc22204)

[2.4. 如有可能避免在java代码里面出现表名 字段名等具体硬编码绑定。 3](#_Toc9011)

[3. 现有的选择 3](#_Toc19591)

[3.1. 数据查询语言 GraphQL （经过测试麻烦繁琐 3](#_Toc6483)

[3.2. Jpql  Java Persistence query language (JPQL 3](#_Toc14795)

[3.3. Spel ongl （不适合大量数据） 3](#_Toc24637)

[3.4. Json系列查询语言 3](#_Toc16392)

[3.5. Httpprarm转myatis参数化 3](#_Toc10041)

[4. 通用接口的实现 4](#_Toc7710)

[4.1. 核心查询 4](#_Toc13170)

[4.2. 常用接口 聚合sum 4](#_Toc18200)

[4.3. 常用接口 查询数据与翻页 5](#_Toc3121)

[5. 查询效果 5](#_Toc3063)

# 实现目标

## 扩展性 业务功能调整，代码层面尽可能少的编译部署步骤，如可以免编译免部署更好

## 动态化，与表结构脱钩，字段的变动不会影响接口的编码 无需再次调整代码编译部署

## 灵活性 随时调整决定要显示哪些字段

## 动态化 跨表查询 接口与模型数据尽可能不绑定

## 通用化 同一个技术接口，可以给很多业务功能使用

## 通用化 跨项目公共化，同样的接口，放在下一个项目里面也能通用

## 跨语言 接口在不同语言的实现模式(java js python php c#等）基本类似

## 公有规范性，无需专门文档说明，尽可能使用现有规范

# 方式

## 适当抽象一下，与具体业务脱钩，数据标识参数化配置化，避免硬编码

## 如有可能优先使用动态对象 避免硬编码

## 如有尽可能的避免使用硬编码模式域模型 do vo dto bo，可以使用动态化的模型json map 等

## 如有可能避免在java代码里面出现表名 字段名等具体硬编码绑定。

这样表与字段变化的话都不会有影响现有代码

# 现有的选择

## 数据查询语言 GraphQL （经过测试麻烦繁琐

## Jpql  Java Persistence query language (JPQL

这个比较好，但是缺乏解析器，太简单解析器。。要自己解析然乎转换sql

## Spel ongl （不适合大量数据）

几万条以下下还可，因为只能查询内存数据

## Json系列查询语言

需要预先转换为json，所以不适合大量数据(5w条以上）

## Httpprarm转myatis参数化

这个比较好的解决了转换问题，保持简单性

# 通用接口的实现

## 核心查询

主要就是把req对象接口转换为map接口对象，传入mybatismapper。。。因为mapper不能直接接受req接口，mapper接受map接口作为动态对象

@RestController

@SuppressWarnings("all")

**public** **class** ApiController {

//核心接口

@GetMapping("/query")

**public** Object query() **throws** Exception {

Map m = RequestUtil.*getMap*(req);

**return** query(m);

}

**public** Object query(Map m) {

// 简化使用性ux，增加select和limit参数的默认模式，翻页转换page等

uxEnhance(m);

logger.info(m.toString());

// 安全性检查过滤

sqlutil.*checkSafeForQuery*(m);

List r = MybatisMapper1.query(m);

**return** r;

}

Mapper接口注解如下

@Mapper

**public** **interface** MybatisMapper {

//

@Select("select ${select} from ${from} ${join} ${where} ${group} ${HAVIN} ${order} ${limit}")

List<Map> query(Map m);

## 常用接口 聚合sum

@GetMapping("/queryCount")

**public** **int** queryCount() **throws** Exception {

Map m = RequestUtil.*getMap*(req);

// m.remove("limitt");

m.put("select", " count(\*) as cnt");

List<Map> cntList = (List<Map>) query(m);

**int** cnt = Integer.*parseInt*(cntList.get(0).get("cnt").toString());

**return** cnt;

}

## 常用接口 查询数据与翻页

@GetMapping("/queryWzCount")

**public** Object queryWzCount() **throws** Exception {

Map reqM = RequestUtil.*getMap*(req);

Map m = Maps.*newLinkedHashMap*();

m.put("count", queryCount());

m.put("data", query());

m.put("page", reqM.get("page"));

m.put("pagesize", reqM.get("pagesize"));

m.put("pageCnt", Util.*getPageCount*(m.get("count"), reqM.get("pagesize")));

**return** m;

}

# 查询效果





