编码规范

一. 命名规约

1. **【强制】禁止拼音缩写，避免阅读者费劲猜测；尽量不用拼音，除非中国式业务词汇没有通用易懂的英文对应**

```
禁止： DZ[打折] / getPFByName() [评分]
尽量避免：Dazhe / DaZhePrice
```

2. **【推荐】禁止使用非标准的英文缩写**

```
反例： AbstractClass 缩写成 AbsClass; condition 缩写成 condi
```

3. **【强制】禁用其他编程语言风格的前缀和后缀**

   在其它编程语言中使用的特殊前缀或后缀，如`_name`，`name_`，`mName`，`i_name`，在Java中都不建议使用

4. **【推荐】包名全部小写点分隔符之间尽量只有一个英语单词，即使有多个单词也不使用下划线或大小写分隔**

```
正例： cn.itcast.javatool
反例： cn.itcast.java_tool, cn.itcast.javaTool
```

5. **【强制】类名与接口名使用UpperCamelCase风格，遵从驼峰形式**

   Tcp, Xml等缩写也遵循驼峰形式，可约定例外如：DTO/ VO等

```
正例：UserId / XmlService / TcpUdpDeal / UserVO
反例：UserID / XMLService / TCPUDPDeal / UserVo
```

6. **【强制】方法名、参数名、成员变量、局部变量使用lowerCamelCase风格，遵从驼峰形式**

```
正例： localValue / getHttpMessage();
```

7. **【强制】常量命名全大写，单词间用下划线隔开力求语义表达完整清楚，不要嫌名字长**

```
正例： MAX_STOCK_COUNT
反例： MAX_COUNT
```

   例外：当一个static final字段不是一个真正常量，比如不是基本类型时，不需要使用大写命名

```
private static final Logger logger = Logger.getLogger(MyClass.class);
```

8. **【推荐】枚举类名以Enum结尾; 抽象类使用Abstract或Base开头；异常类使用Exception结尾；测试类以它要测试的类名开始，以Test结尾**

```
正例：DealStatusEnum，  AbstractView，BaseView，  TimeoutException，UserServiceTest
```

9. **【推荐】实现类尽量用Impl的后缀与接口关联，除了形容能力的接口**

```
正例：CacheServiceImpl 实现 CacheService接口
正例：Foo 实现 Translatable接口
```

10. **【强制】POJO类中布尔类型的变量名，不要加is前缀，否则部分框架解析会引起序列化错误**

```
反例：Boolean isSuccess的成员变量，它的GET方法也是isSuccess()，部分框架在反射解析的时候，“以为”对应的成员变量名称是success，导致出错
```

11. **【推荐】各层命名规约**

    DAO 层方法命名规约

    1) 获取单个对象的方法用 select 做前缀

    2) 插入的方法用 insert 做前缀

    3) 删除的方法用 delete 做前缀

    4) 修改的方法用 update 做前缀

    Service 层方法命名规约

    1) 获取单个对象的方法用 get 做前缀

    2) 获取多个对象的方法用 query 做前缀

    3) 插入的方法用 create 做前缀

    4) 删除的方法用 remove 做前缀

    5) 修改的方法用 modify 做前缀

二. 常量定义

1. **【推荐】不要使用一个常量类维护所有常量，要按常量功能进行归类，分开维护 说明:大而全的常量类，杂乱无章，使用查找功能才能定位到修改的常量，不利于理解和维护**

```
正例:缓存相关常量放在类 CacheConsts 下;系统配置相关常量放在类 ConfigConsts 下
```

三. 注释规约

1. **【推荐】基本的注释要求**

   完全没有注释的大段代码对于阅读者形同天书，注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路；注释也是给继任者看的，使其能够快速接替自己的工作

   代码将被大量后续维护，注释如果对阅读者有帮助，不要吝啬在注释上花费的时间(但也综合参见规则2，3)

第一、能够准确反应设计思想和代码逻辑；第二、能够描述业务含义，使别的程序员能够迅速了解到代码背后的信息

除了特别清晰的类，都尽量编写类级别注释，说明类的目的和使用方法

除了特别清晰的方法，对外提供的公有方法，抽象类的方法，同样尽量清晰的描述：期待的输入，对应的输出，错误的处理和返回码，以及可能抛出的异常

2. **【推荐】通过更清晰的代码来避免注释**

   在编写注释前，考虑是否可以通过更好的命名，更清晰的代码结构，更好的函数和变量的抽取，让代码不言自明，此时不需要额外的注释

3. **【强制】代码修改的同时，注释也要进行相应的修改尤其是参数、返回值、异常、核心逻辑等的修改**
4. **【推荐】注释不要为了英文而英文**

   如果没有国际化要求，中文能表达得更清晰时还是用中文

5. **【推荐】TODO标记，清晰说明代办事项和处理人**
6. **【推荐】合理处理注释掉的代码**

   如果后续会恢复此段代码，在目标代码上方用`///`说明注释动机，而不是简单的注释掉代码

四. 方法设计

1. **【推荐】方法的长度度量**

   方法尽量不要超过100行

2. **【推荐】尽量减少重复的代码，抽取方法**

   超过5行以上重复的代码，都可以考虑抽取公用的方法

3. **【推荐】方法职责应尽量单一，明确具体行为**
4. **【推荐】方法参数禁止传递弱类型**，比如map。例外：有可扩展性设计预留参数
5. **【推荐】方法参数最好不超过3个，最多不超过7个**

   1）如果多个参数同属于一个对象，直接传递对象

   2）将多个参数合并为一个新创建的逻辑对象。例外: 多个参数之间毫无逻辑关联

   3）将函数拆分成多个函数，让每个函数所需的参数减少

6. **【推荐】方法内代码尽量保持统一维度**

```
方法{
  步骤1
  步骤2(多个执行细节)
  步骤3
}

步骤2(){
  细节1
  细节2
  细节3
}
```

7. 尽量减少if嵌套，善用return

```
//不推荐
if (entity != null) {
    if (entity.getStatus() == 1) {
        ...
    }
}

//建议使用
if (entity == null) {
    return;
}

if (entity.getStatus() == 1) {
        ...
}
```

8. **【推荐】不使用不稳定方法，如com.sun.*包下的类，底层类库中internal包下的类** `com.sun.*`，`sun.*`包下的类，或者底层类库中名称为internal的包下的类，都是不对外暴露的，可随时被改变的不稳定类

五. URL规约

采用Restful风格的URL

1. **常用的HTTP动词有下面五个（括号里是对应的SQL命令）**

```
GET（SELECT）：从服务器取出资源（一项或多项）
POST（CREATE）：在服务器新建一个资源
PUT（UPDATE）：在服务器更新资源（客户端提供改变后的完整资源）
DELETE（DELETE）：从服务器删除资源
```

2. **资源名尽可能使用复数形式，尽量避免在URL中使用动词，单词之间以中划线"-"分割**

```
正例：
GET /zoos：列出所有动物园
POST /zoos：新建一个动物园
GET /zoos/{id}：获取某个指定动物园的信息
GET /zoos/{id}/type/{}：多条件查询
PUT /zoos/ID：更新某个指定动物园的信息（提供该动物园的全部信息）
DELETE /zoos/ID：删除某个动物园
GET /zoos/ID/animals：列出某个指定动物园的所有动物
DELETE /zoos/ID/animals/ID：删除某个指定动物园的指定动物
```

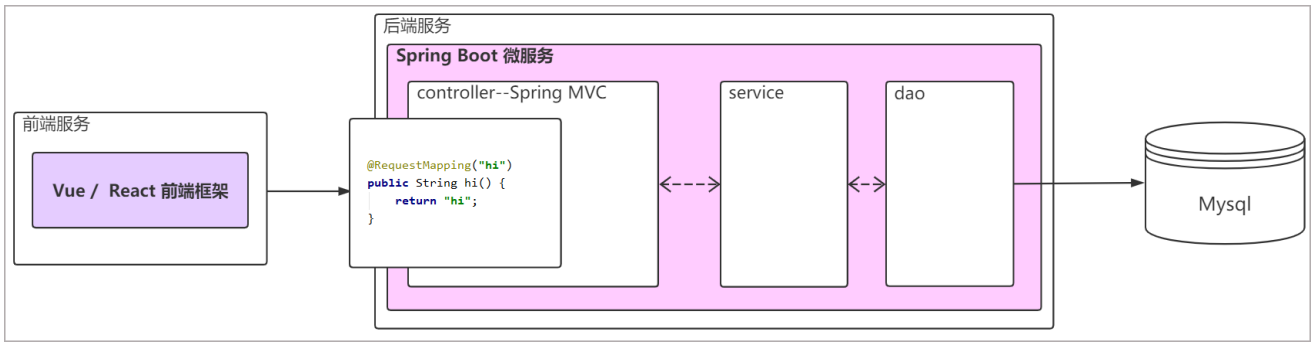
六. 接口开发规范

在微服务接口开发中，接口主要的调用对象为两类：前端调用、微服务远程调用。这两类的接口规范是有区别的，下面进行简要说明。

6.1 前端调用接口规范

在前后端分离开发中，Spring Boot对微服务软件架构的实现，其Spring MVC提供对外的 HTTP 请求地址，Spring MVC 中的 Controller 类中的每一方法都是对外提供 HTTP 访问接口。

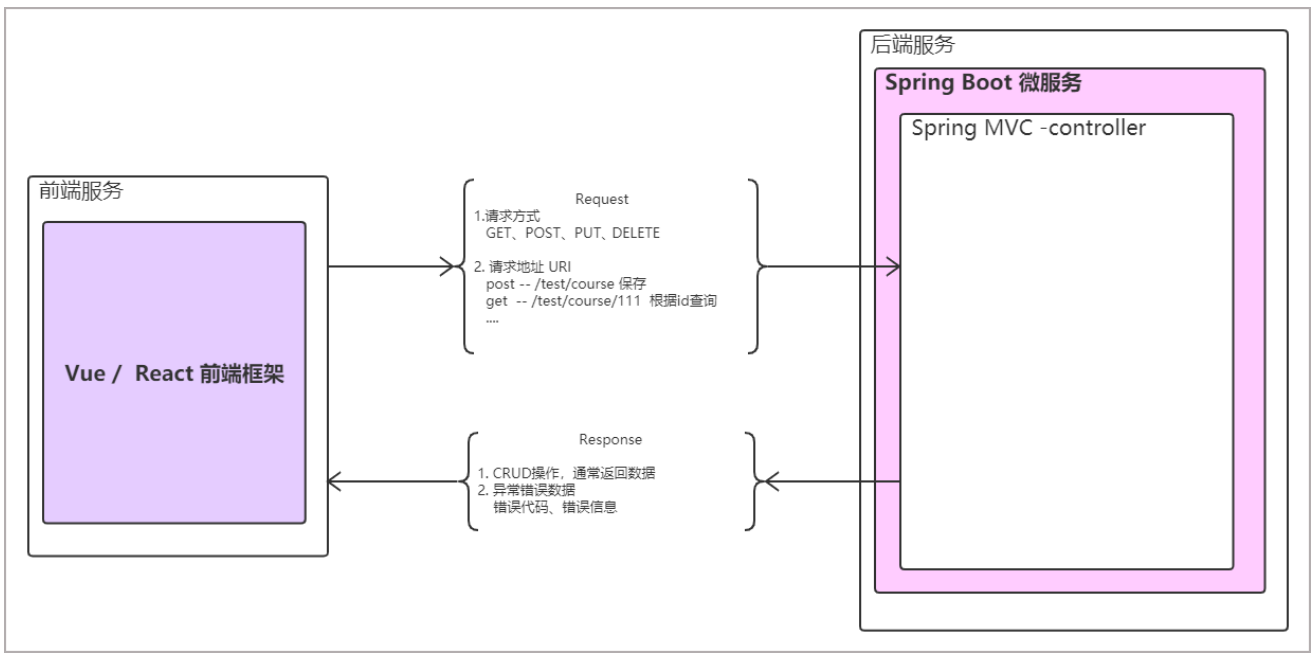
> 前端调用接口示意图



前后端分离开发中，前端会根据后端微服务所提供的 HTTP 访问接口地址来获得数据并在前端页面展示，那么前后端的数据交互的格式和规范就显得尤为重要。前端需要传入哪些参数、请求的方式是哪种、获得的数据格式和内容等等，都需要前后端进行讨论并定出接口的规范。

6.1.1 接口路径的规范说明

> 前端调用接口规范示意图



上图解释：

- HTTP 请求方式规范

  对于前端请求后端微服务，除了接口的 HTTP 请求地址不同外，后端微服务对于不同的 HTTP 请求也有相应的要求。

  - GET 请求（SELECT）：从服务器取出资源（一项或多项）
  - POST请求（CREATE）：在服务器新建一个资源
  - PUT 请求（UPDATE）：在服务器更新资源（客户端提供改变后的完整资源）

- DELETE 请求（DELETE）：从服务器删除资源

- HTTP 请求地址

  由于 HTTP 请求路径使用 Restful 风格，URI 中尽量避免使用动词，单词之间可以以中划线"-"分割。

  > URI示例

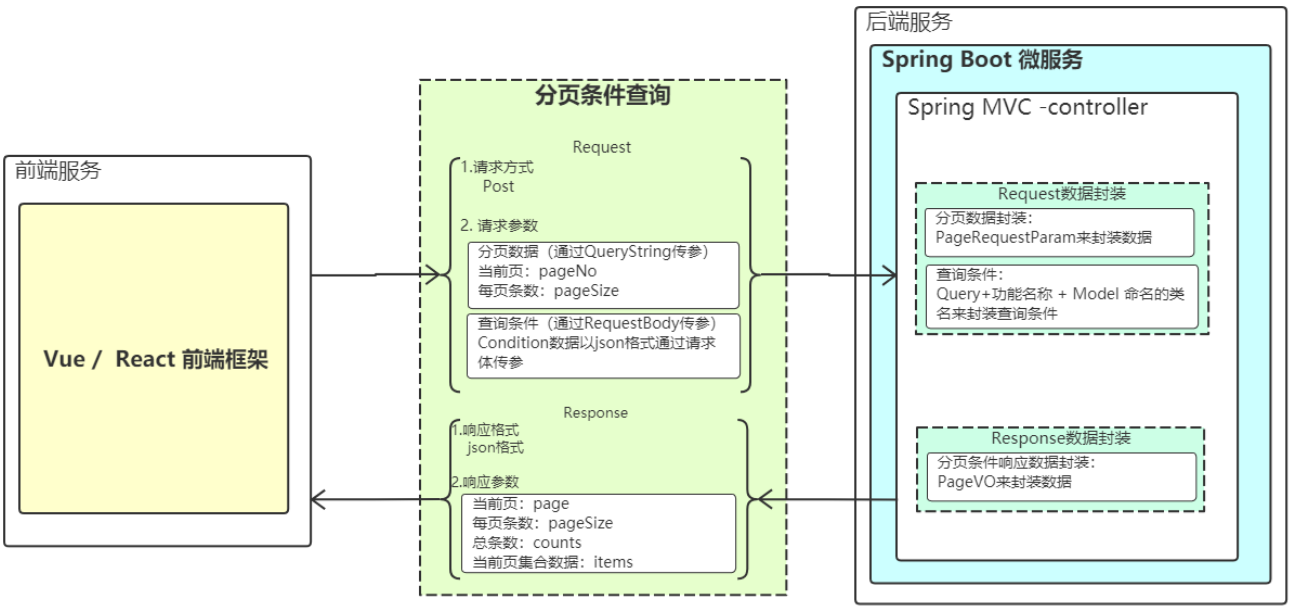
  ```
  post -- /test/course      保存课程
  get  -- /test/course/111  根据id查询课程
  del  -- /test/course/222  根据id删除课程
  ```

  对标明路径的种类，需要在路径中加上标识

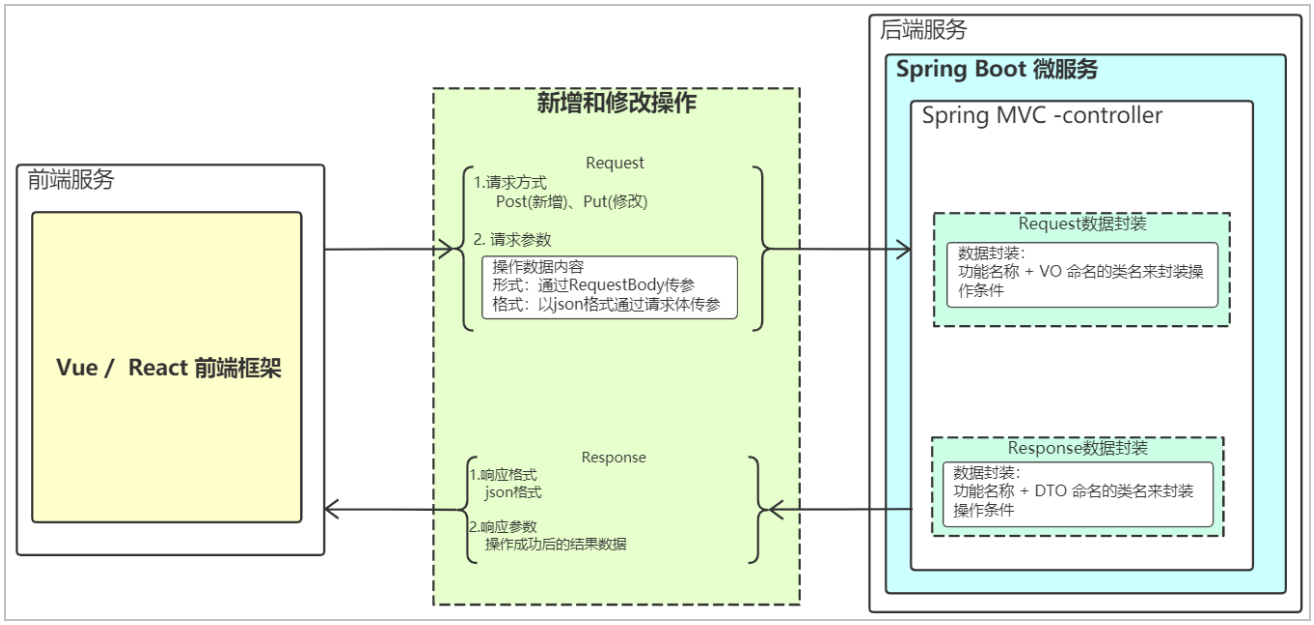
  - 运营平台管理接口地址：/项目根路径**/m/**.....
  - 学成在线内部微服务端间调用接口地址：/项目根路径**/l/**.....
  - 其他接口地址：没有特殊标识

6.1.2 接口数据的规范说明

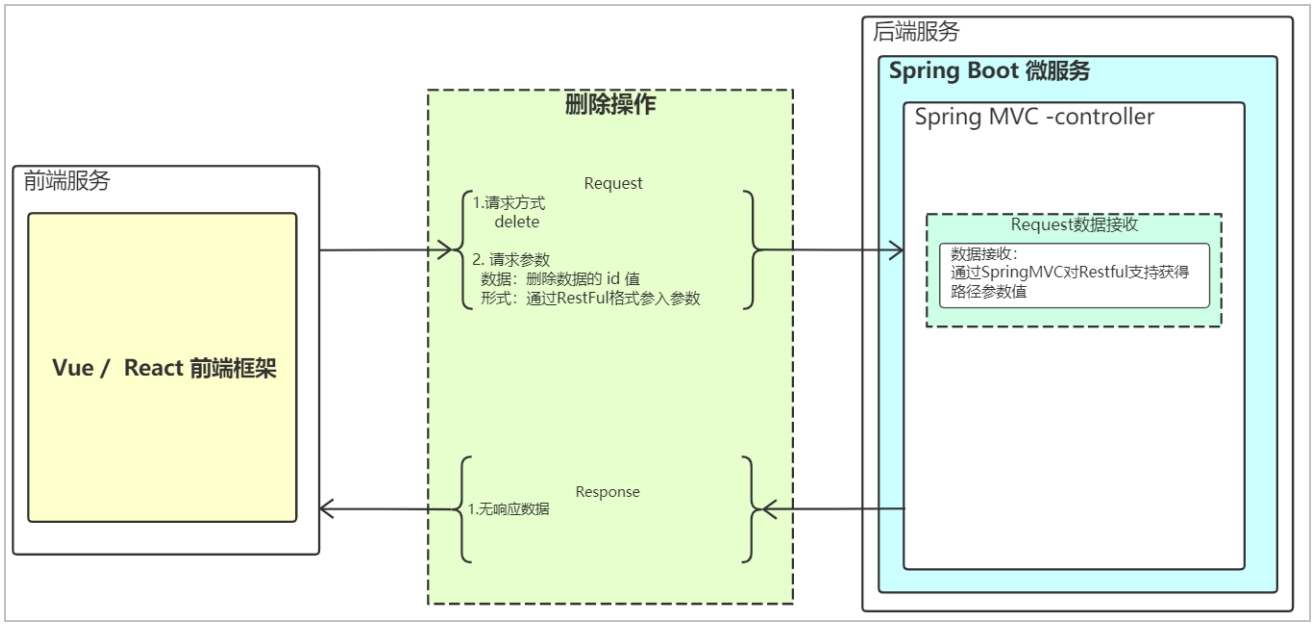
> 前端调用接口数据规范示意图--分页条件查询



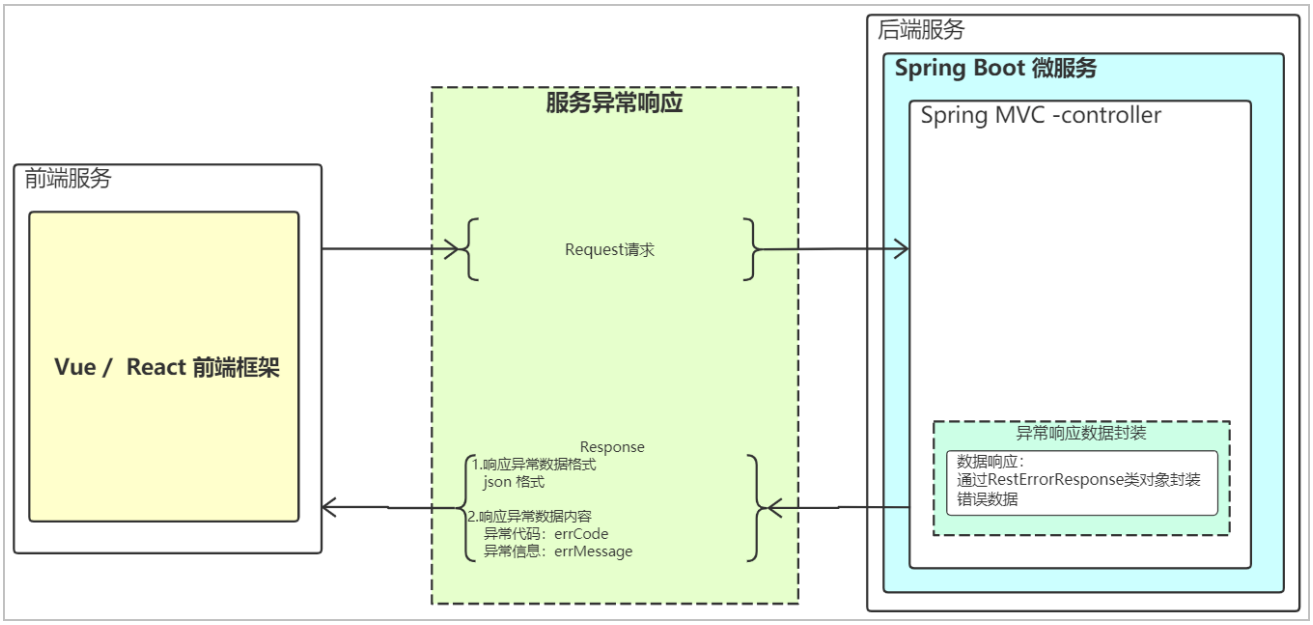
> 前端调用接口数据规范示意图--添加修改



前端调用接口数据规范示意图--删除修改



前端调用接口数据规范示意图--异常



- HTTP 服务端接收参数的封装

  Controller接收前端的参数种类：

  - 查询操作
    - 分页类和多个条件参数

      分页数据要使用 PageRequestParam 实体类，并通过 QueryString 的方式进行传入参数。

      条件查询对象封装到以 Query***Model 类中，并且条件查询数据要通过请求体以传递 json 格式来进行传递。

      请求方式有 Get 请求改为 Post请求。
  - 其他操作
    - 新增和修改

      对于服务端控制层接收前端参数，Controller 层中方法声明的传入参数建议封装到 VO 对象中。
    - 删除

      删除一般前端只需要出入删除数据的 Id 值即可。

- HTTP响应数据
  - 响应数据格式为：Json
  - 对于CRUD操作，通常返回直接数据
    - 新增和修改操作--返回保存/修改后的完整数据
    - 删除--无返回数据
    - 查询操作
      - 查询单个数据：返回单个完整数据。
      - 查询分页数据：要使用 PageVO 对象来返回 总条数、当前页、每页条数、当前页的集合数据。

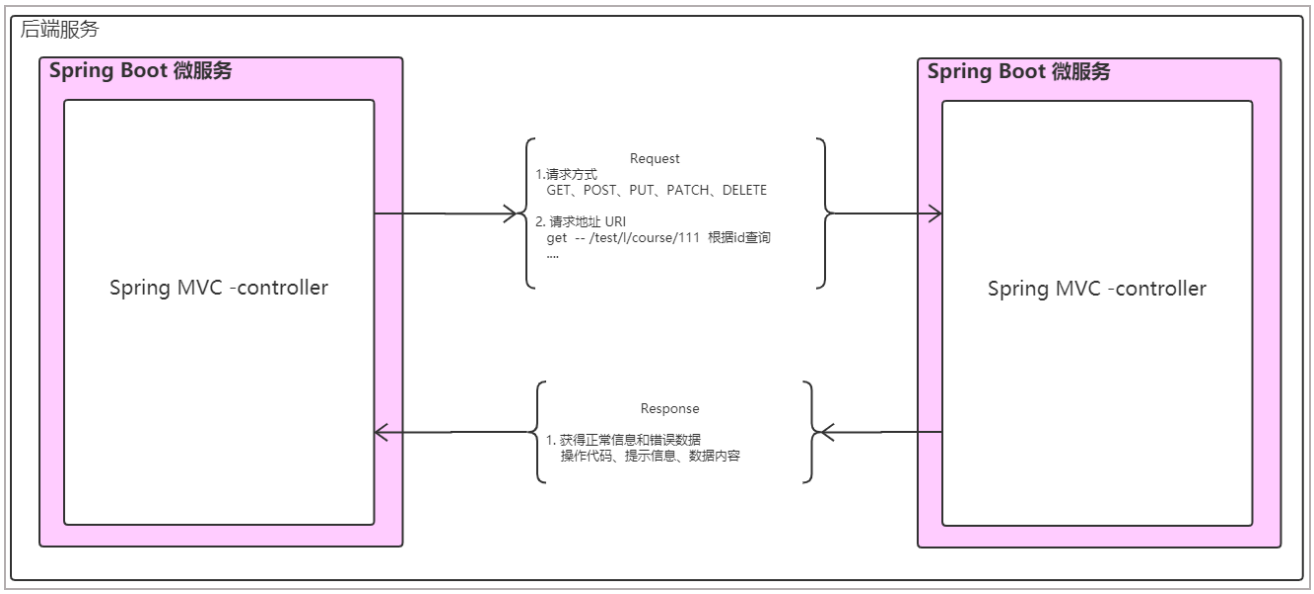
- 异常错误数据

  如果前端调用微服务端获得错误数据，错误数据需要包含：错误代码、错误信息。

6.2 微服务远程调用接口规范

对于 Spring Boot 构建的微服务，通常会有微服务远程调用其他的微服务 HTTP 接口地址，在此也有响应的规范。

> 微服务之间调用接口规范示意图



- HTTP 请求方式规范

  同 '6.4.2.1 前端调用接口规范' 中的 HTTP 请求方式规范，对此不在阐述

- HTTP 请求地址URI
  - URI 中尽量避免使用动词，单词之间不要以中划线"-"分割
  - URI 地址前缀：/服务名称/l/xxx
- HTTP响应数据

  无论是进行 CRUD 还是 异常信息返回，其中必须包含：操作代码、提示信息、数据内容。
  - 响应数据格式为：Json
    - 如果是获得正常信息
      - 操作代码固定为：0
      - 提示信息固定为：success
      - 数据内容：远程调用获得数据
    - 如果获得错误信息
      - 操作代码：错误代码（不同业务错误代码不同）
      - 提示信息：错误信息（要看具体的错误信息）
      - 数据内容：null

七. 消息队列命名规约

包括RocketMQ, RabbitMQ等消息队列产品

1. **全部使用大写，单词之间以'_'分割**
2. **生产组(Producer Group)以'PID'为前缀**
3. **消费组(Consumer Group)以'CID'为前缀**
4. **主题(Topic)以'TP'为前缀**

八. 其他规约

1. **【参考】尽量不要让魔法值（即未经定义的数字或字符串常量）直接出现在代码中**

```
//WRONG
String key = "Id#taobao_"+tradeId;
cache.put(key, value);
```

例外：-1,0,1,2,3 不认为是魔法数

2. **【推荐】变量声明尽量靠近使用的分支**

九. 项目规约

1. 所有的Entity和DTO都应使用lombok框架的@Data注解，不再自行编写getter/setter方法

```
@Data
@TableName("consumer")
public class Consumer implements Serializable {

    private static final long serialVersionUID = 1L;

    /**
     * 主键
     */
    @TableId(value = "ID", type = IdType.AUTO)
    private Long id;

    /**
     * 用户名
     */
    @TableField("USERNAME")
    private String username;
    ...
}
```

2. Entity和DTO的转换使用mapstruct(https://mapstruct.org/)组件进行转换，转换类放在各服务的convert包下

```
@Mapper
public interface CourseBaseConvert {

    CourseBaseConvert INSTANCE = Mappers.getMapper(CourseBaseConvert.class);

    @Mappings({
            @Mapping(source = "id", target = "courseBaseId"),
    })
    CourseBaseDTO entity2dto(CourseBase entity);

    @Mapping(source = "courseBaseId", target = "id")
    CourseBase dto2entity(CourseBaseDTO dto);

    List<CourseBaseDTO> entitys2dtos(List<CourseBase> list);

}
```

3. 各服务controller的接口都应定义在xc-api项目下com.xuecheng.api包中

```
package com.xuecheng.api.consumer;

public interface ConsumerApi {
  ...
}
```

4. 所有DTO都应定义在wanxinp2p-api项目下cn.itcast.wanxinp2p.api.[serviceName].model中

```
package com.xuecheng.api.consumer.model;

@Data
@ApiModel(value = "ConsumerDTO", description = "平台c端用户信息")
public class ConsumerDTO {
    ...
}
```

5. controller中所有方法返回的具体业务对象数据必须是DTO，不能直接用Entity

```
    /**
     * 获取借款人用户信息
     * @return
     */
    BorrowerDTO getBorrower(Long id);

    /**
     * 根据用户名获取用户信息
     * @param username
     * @return
     */
    ConsumerDTO getConsumerByUsername(String username);
```

6. 外界跟service打交道使用dto；entity在service内部消化，禁止暴露到sevice外部
7. service如果有业务异常可直接抛出，GlobalExceptionHandler类会进行统一拦截返回给前端

```
ExceptionCast.cast(ContentErrorCode.E_140101);
```

8. 所有的分页查询返回类型应是**PageVO**

```
    @ApiOperation("商户查询")
    @ApiImplicitParams({
            @ApiImplicitParam(name = "merchantQuery", value = "商户查询条件", dataType =
"MerchantQueryDTO", paramType = "body"),
            @ApiImplicitParam(name = "pageNo", value = "页码", required = true, dataType =
"int", paramType = "query"),
            @ApiImplicitParam(name = "pageSize", value = "每页记录数", required = true,
dataType = "int", paramType = "query")})
    @PostMapping("/m/merchants/page")
    public PageVO<MerchantDTO> queryMerchant(@RequestBody MerchantQueryDTO merchantQuery,
@RequestParam Integer pageNo, @RequestParam Integer pageSize){

        //将queryDTO转换为 MerchantDTO
        MerchantDTO merchantDTO =
MerchantDTOCovert.INSTANCE.queryDTOToMerchantDTO(merchantQuery);

        return operationAppServiceImpl.getMerchantPage(merchantDTO,pageNo,pageSize);
    }
```

9. 日志记录采用@Slf4j注解，日志记录时采用占位符来组装，避免使用"+"；异常记录时应该把堆栈信息输出

```
@Slf4j

//占位符
log.info("id为{}", id)

//堆栈输出
log.error("xxx", e)
```

10. service中的所有方法需要编写单元测试。

    测试类名：UserServiceTest，方法名：testCreate()

11. 使用Spring Cloud Feign远程调用时，使用熔断工厂类做测试
    - 引入降级工厂基类BaseFallbackFactory
    - 针对对应的Feign代理类编写熔断测试类DevAccountApiHystrix，熔断方法mock返回测试数据

```
@Slf4j
class DevAccountApiHystrix implements AccountApiAgent {

    @Override
    public RestResponse<AccountDTO> register(AccountRegisterDTO accountRegisterDTO) {
        return RestResponse.success(new AccountDTO());
    }

    @Override
    public RestResponse<Integer> checkUsername(String username) {
        return RestResponse.success(1);
    }
}
```

    - 定义熔断工厂类DepositoryAgentApiFallbackFactory：

```
@Component
class DepositoryAgentApiFallbackFactory extends
        BaseFallbackFactory<DepositoryAgentApiAgent, DepositoryAgentApiHystrix,
DevDepositoryAgentApiHystrix> {

}
```

    - 在resources/application.yml中配置是否启用测试模式

```
feign:
  fallback:
    profile: dev
```

    - Feign调用接口定义

```
@FeignClient(value = "account-service", fallbackFactory =
AccountApiFallbackFactory.class)
public interface AccountApiAgent {
}
```

十.服务启动参数列表

学成在线平台的所欲微服务的端口号约束：

- 注册与发现: 63000
- 网关：63010
- UAA：63020
- 内容管理服务：63040
- 媒资管理服务：63050
- 教学管理服务：63060
- 学习中心：63070
- 搜索服务：63080
- 订单服务：63090
- 系统管理服务：63110
- 评论服务：63120
- 用户中心：63130