概要设计

一、概述

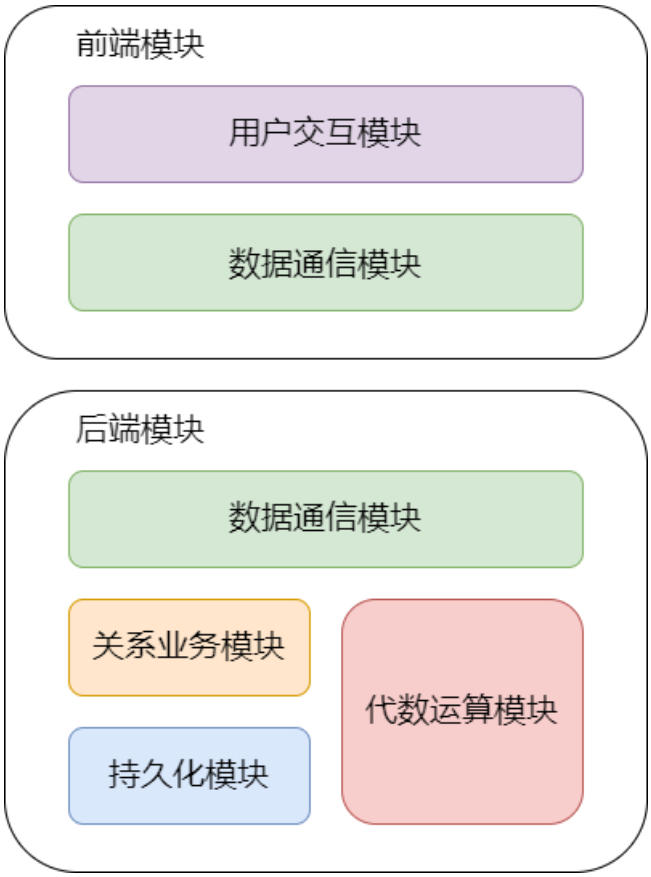
设计并实现一个具有良好交互界面的**关系代数运算系统**。系统能完成关系代数的运算逻辑，包括并、交、差、笛卡尔积、选择、投影、连接、除这8种运算。

经商议，为了给予用户良好的交互界面，该系统采取前后端分离的设计模式，

二、系统结构设计

2.1 软件总体结构和功能模块设计

软件总体结构宏观上可以分为前端模块与后端模块两个，两个模块下又各有若干个功能模块。



2.2 各项功能与程序结构的关系

程序结构由前后端两个模块构成，各个模块以及子模块的作用如下。

前端模块

- 用户交互模块：负责与用户的直接交互，用户通过该模块向系统输入，同时系统将输入处理成结果，通过该模块向用户呈现。
- 数据通信模块：负责前后端数据的通信，功能包括关系信息的传输、计算表达式的传输等。将用户输入的数据打包交由后端，将后端的响应处理后交由交互模块展现。

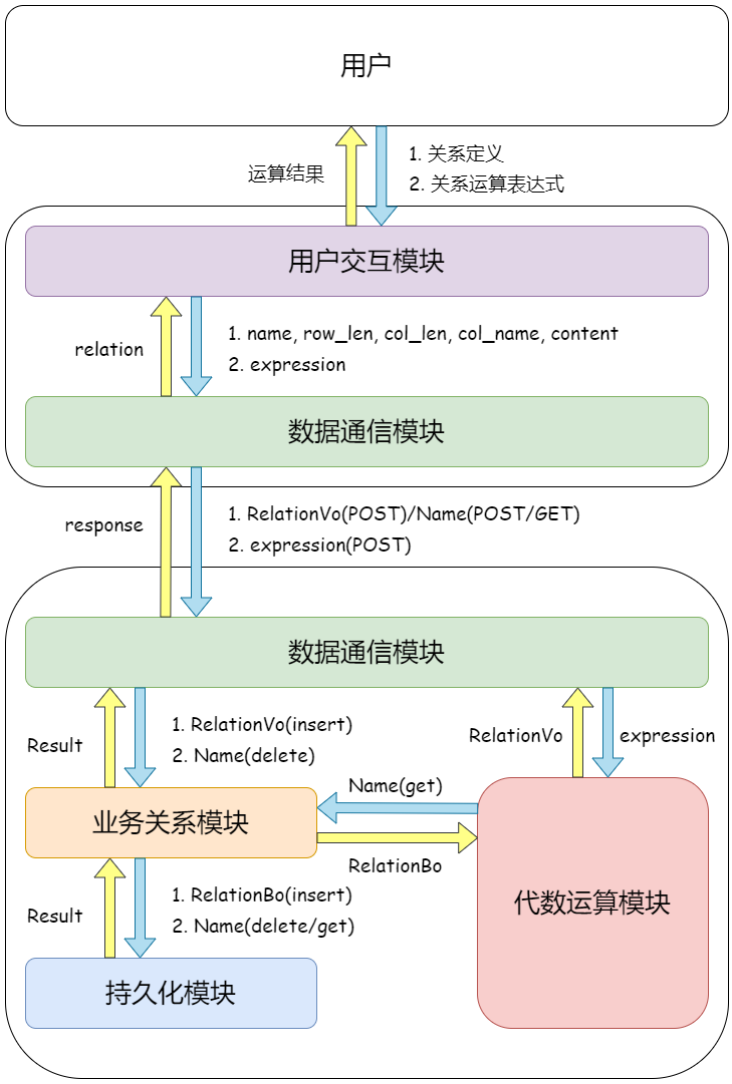
后端模块

- 数据通信模块：负责前后端数据的通信，接收前端传来的数据，并将数据交由业务模块或是运算模块，同时将下层模块的处理结果打包返回前端。
- 关系业务模式：处理关系的存储。用户在前端进行关系的新增与删除时，通过该模块与持久化模块进行通信，增加或删除对应的关系。
- 持久化模块：负责已有关系的持久化。
- 代数运算模块：接收用户输入的运算表达式并进行计算，并向数据通信模块返回结果。

2.3 模块之间的层次结构以及模块间的调用关系

用户与前端模块中的用户交互子模块直接交互，包含关系的定义、关系运算表达式输入、结果输出等。前后端之间通过数据通信模块进行交互，负责将用户的输入结果按规定格式传递至后端、将计算结果按规定格式传递至前端。

业务关系模块负责处理关系的新增/删除等操作，与数据通信模块交互以获取数据，与代数运算模块交互以提供关系信息。持久化模块则为保存用户事先定义的关系，与业务关系模块交互。代数运算模块则借助业务关系模块与数据通信模块完成表达式的计算以及返回结构。



三、接口设计

3.1 外部接口

外部接口主要为前端通过交互界面为用户提供的操作本系统的接口。

关系管理接口

关系管理接口包含关系的新增、删除以及重置。用户可以通过交互界面进行关系的新增、关系的删除以及重置已保存的所有关系。

表达式接口

表达式接口为用户呈现一个输入框，用户在此输入待运算的表达式，通过点击提交来获取系统的计算结果。系统通过该接口将运算结果向用户呈现。

3.2 内部接口

后端程序内部模块间接口

后端程序中主要涉及到几个主要接口。`ComputingService`用于表达式的计算，对外提供了以表达式字符串为入参计算表达式的接口。`RelationService`用于关系业务的处理，对外暴露的接口可以执行关系的新增、删除、重置等操作。`JudgementOfLegalityService` 用于对输入表达式进行合法性判断。

`RelationService` 接口伪代码定义

```
interface RelationService {

    /**
     * @param relationVo 前端传来的Relations
     * @throws ParamLenException 参数不对时抛出此异常
     */
    void insertRelation(RelationVo relationVo);

    /**
     * 删除关系
     */
    void deleteRelation(String name);

    /**
     * 删除所有已建立关系
     */
    void deleteAll();

    /**
     * 是否存在名为key的关系
     * @param key
     * @return
     */
    boolean contains(String key);

    /**
     * 获取name对应的Bo
     * @param name
     * @return
     */
    RelationBo get(String name);
}
```

`ComputingService` 接口定义

```java
public interface ComputingService {

    /**
     * @param expression 关系代数表达式
     * @return 计算后的结果
     */
    RelationVo compute(String expression);

}
```

`JudgementOfLegalityService` 接口定义

```java
public interface JudgmentOfLegalityService {

    /**
     * @param expression 待计算的表达式
     * @return 表达式是否合法
     */
    boolean judgeLegality(String expression);
}
```

前后端接口定义

新增关系

- 调用方式：POST
- URL：`/api/insert/`
- 入参：relation
- 出参：data为空的response
- 说明：新增关系，保存至后端。支持传入一个json格式的关系。

例子如下。

入参

```json
{
    "relation_name":"student",
    "row_len":3,
    "col_len":2,
    "col_name":"name,age,gender",
    "content":"Johnny,18,male,Jack,20,male"
}
```

出参

```json
{
    "code":200,
    "msg":"ok",
    "data":{}
}
```

删除关系

- 调用方式：POST
- URL：`/api/delete/`
- 入参：关系名
- 出参：data为空的response
- 说明：删除关系。

例子如下。

入参

```
{
    "name":"student"
}
```

出参

```
{
    "code":200,
    "msg":"ok",
    "data":{}
}
```

重置

- 调用方式：GET
- URL：`/api/delete_all/`
- 入参：无
- 出参：data为空的response
- 说明：删除已有的所有关系。

例子如下。

出参

```
{
    "code":200,
    "msg":"ok",
    "data":{}
}
```

计算关系代数

调用方式：POST

URL：`/api/compute/`

入参：表达式字符串

出参：data为计算结果（一个relation）的response

说明：计算关系代数表达式，内部出现的关系必须事先定义过。

关系运算符在表达式字符串中的表示对应关系如下：

运算符	含义	在字符串的表示	举例	说明
∪	并	#or	A #or B	
∩	交	#and	A #and B	
−	差	#diff	A #diff B	
×	笛卡尔积	#prod	A #prod B	
σ	选择	#select	#select[A,条件表达式,1]	参数为关系名、条件、固定参数
π	投影	#project	#project[A,name,age,2]	参数为关系名、投影列名、投影列数
⋈	连接	#join	A #join B	连接运算仅支持自然连接
÷	除	#div	A #div B	
(	左括号	(	-	-
)	右括号	)	-	-

共计8个运算类型：并、交、差、笛卡尔积、选择、投影、连接、除。

优先级定义：括号 > 选择 = 投影 > 连接 > 差 > 积 = 除 > 交 > 并

条件表达式中支持的运算符：>,<,=,>=,<=,!=。以及使用$and与$or来连接多个条件。

```
sage>=3|\$and|ssex=女|\$or|sage>=5|\$and|ssex=男
(|sage>=3|\$or|ssex=女|)|\$and|(|sage>=5|$or|ssex=男|)
```

例子如下。

入参

> 注意：任意两个符号（运算符、关系名或是括号）之间请添加一个英文空格。

```
{
    "expression":"( student #or class ) #and teacher",
    "original_expression":"(student∪class)∩teacher"
}
```

出参

```
{
    "code":200,
    "msg":"ok",
    "data":{
        "relation_name":"student",
        "row_len":3,
        "col_len":2,
        "col_name":"name,age,gender",
        "content":"Johnny,18,male,Jack,20,male"
    }
}
```

四、数据结构和数据库设计

本系统不涉及数据库设计，在此，主要展现一些计算过程中用到的数据结构。

4.1 后端

关系

后端存储关系主要涉及两个类，`RelationVo`以及`RelationBo`，前者主要用于与前端交互时存储从前端送来的关系/准备送至前端的关系信息，后者则主要用于后端业务处理，包括关系的存储/表达式的运算等。

鉴于二者相似性，仅详细介绍`RelationBo`的成员，伪代码如下。

```
RelationBo {

    // 关系行长
    int rowLen;

    // 关系列长
    int colLen;

    // 关系列名
    String[] colName;

    // 关系内容，通过二维数组存储
    String[][] content;

    // bo与vo相互转换的方法
    toRelationVo(RelationBo bo, String name);

    toRelationBo(RelationVo vo, String name);
}
```

常量

此外，常量类`Constant`，用于统一管理后端程序中用到的常量。

```
public class Constant {

```

```
    public static final String OR = "#or";

    public static final String AND = "#and";

    ...

    public static final Set<String> UNARY_OPERATOR;

    public static final Map<String, Integer> PRIORITY;

    public static final String TEMP_RELATION_PREFIX = "TEMP";

    ...
}
```

4.2 前后端交互

前后端交互主要用到两个数据结构。

relation 表示关系的数据结构（表）

包含五个字段：关系名、行长、列长、列名以及表的内容。

表的内容（content）使用一个字符串表示，每个元素间使用英文逗号分隔，应当满足字符串内元素的数量等于行长乘以列长。

举例，以下关系以及对应的json结构：

name	age	gender
Johnny	18	male
Jack	20	male

```
{
    "relation_name":"student",
    "row_len":3,
    "col_len":2,
    "col_name":"name,age,gender",
    "content":"Johnny,18,male,Jack,20,male"
}
```

response 响应

前端调用接口后返回的是一个response对象，包含三个字段：状态码、信息、数据体。如果是需要返回结果的请求调用，结果对象会存储在数据体中。

举例，一个请求成功，且返回数据存放在data中的response对象：

```
{
    "code":200,
    "msg":"ok",
    "data":{
        "relation_name":"student",
        "row_len":3,
        "col_len":2,
        "col_name":"name,age,gender",
        "content":"Johnny,18,male,Jack,20,male"
    }
}
```

4.3 前端

前端主要涉及两个数据结构，分别用于存储关系表及运算结果表。

关系表

关系表用于存储用户输入的关系名，行数，列数，列名，关系表内容，便于后续发送至后端。同时使用dis标识该表的提交状态，避免多次提交同一关系表。

```
domains: [{
        name: '',
        row: '',
        col: '',
        col_name: '',
        text: '',
        dis: false
      }]
```

运算结果表

运算结果表中，每行用id标识行号，id依次递加。行中每项值使用num和data两个变量标识，其中num为列名，data为该项值。

```
list: [{
        id: '',
        dataList: [{
          num: '',
          data: ''
        }]
```

五、编码规范

> 包括接口规约和命名规则等；编码规范的主要元素可参考附件五。

后端的编写语言为Java，编码规范复用了阿里巴巴的规范，详见Java开发手册（嵩山版）.pdf。在此仅列举几个较为重要的编码规范：

- 类名以驼峰式命名，变量以及方法名以首字母小写的驼峰式命名。
- 后端内部接口命名统一以Service结尾，采用Interface结构，接口实现类统一以Impl结尾。
- 后端数据通信层类名统一以Controller结尾。

前后端接口编码规范：

- 接口URL地址统一接在`/api/`下，具体的接口名定义应该使用下划线式命名。
  如`/api/insert_relation/`。
- 依据具体的业务场景，当使用http协议实现接口时，涉及到json数据传输的应该使用`POST`方法，
  不涉及到数据传输的应该使用`GET`方法。