文章详情开发

heima-leadnews-\*

后端工程

VERSION - 1.0.0

|  |  |  |
| --- | --- | --- |
| 文档版本说明 | | |
| 文件状态  [ ] 草稿  [ √ ] 正式发布  [ ] 正在修改 | 文件标识： |  |
| 当前版本 |  |
| 作者 | miukoo |
| 完成日期 | 2019-06-19 |

目录

[目录 2](#_Toc18938866)

[后端开发 5](#_Toc18938867)

[1.1 文章内容接口 5](#_Toc18938869)

[1.1.1 接口定义 5](#_Toc18938870)

[1.1.2 类定义 6](#_Toc18938871)

[1.1.3 Mapper实现 6](#_Toc18938872)

[1.1.4 时序说明 9](#_Toc18938873)

[1.1.5 代码实现 9](#_Toc18938874)

[1.1.6 单元测试 12](#_Toc18938875)

[1.2 文章关系接口 13](#_Toc18938876)

[1.2.1 接口定义 13](#_Toc18938877)

[1.2.2 类定义 14](#_Toc18938878)

[1.2.3 Mapper实现 14](#_Toc18938879)

[1.2.4 时序说明 26](#_Toc18938880)

[1.2.5 代码实现 26](#_Toc18938881)

[1.2.6 单元测试 29](#_Toc18938882)

[1.3 关注接口 30](#_Toc18938883)

[1.3.1 工程创建 30](#_Toc18938884)

[1.3.2 初始项目 30](#_Toc18938885)

[1.3.3 接口定义 31](#_Toc18938886)

[1.3.4 类定义 31](#_Toc18938887)

[1.3.5 Mapper实现 32](#_Toc18938888)

[1.3.6 时序说明 38](#_Toc18938889)

[1.3.7 代码实现 40](#_Toc18938890)

[1.3.8 单元测试 47](#_Toc18938891)

[1.4 点赞接口 49](#_Toc18938892)

[1.4.1 接口定义 49](#_Toc18938893)

[1.4.2 类定义 50](#_Toc18938894)

[1.4.3 Mapper实现 50](#_Toc18938895)

[1.4.4 时序说明 51](#_Toc18938896)

[1.4.5 代码实现 51](#_Toc18938897)

[1.4.6 单元测试 56](#_Toc18938898)

[1.5 不喜欢接口 56](#_Toc18938899)

[1.5.1 接口定义 56](#_Toc18938900)

[1.5.2 类定义 57](#_Toc18938901)

[1.5.3 Mapper实现 57](#_Toc18938902)

[1.5.4 时序说明 57](#_Toc18938903)

[1.5.5 代码实现 58](#_Toc18938904)

[1.6 阅读接口 62](#_Toc18938905)

[1.6.1 接口定义 62](#_Toc18938906)

[1.6.2 类定义 63](#_Toc18938907)

[1.6.3 Mapper实现 63](#_Toc18938908)

[1.6.4 时序说明 66](#_Toc18938909)

[1.6.5 代码实现 66](#_Toc18938910)

[1.6.6 单元测试 72](#_Toc18938911)

[1.7 分享接口 72](#_Toc18938912)

[1.7.1 接口定义 72](#_Toc18938913)

[1.7.2 类定义 73](#_Toc18938914)

[1.7.3 Mapper实现 73](#_Toc18938915)

[1.7.4 时序说明 75](#_Toc18938916)

[1.7.5 代码实现 75](#_Toc18938917)

[1.7.6 单元测试 79](#_Toc18938918)

[1.8 收藏接口 80](#_Toc18938919)

[1.8.1 接口定义 80](#_Toc18938920)

[1.8.2 类定义 80](#_Toc18938921)

[1.8.3 Mapper实现 81](#_Toc18938922)

[1.8.4 时序说明 82](#_Toc18938923)

[1.8.5 代码实现 82](#_Toc18938924)

[1.8.6 单元测试 88](#_Toc18938925)

[1.9 转发接口 89](#_Toc18938926)

[1.9.1 接口定义 89](#_Toc18938927)

[1.9.2 类定义 89](#_Toc18938928)

[1.9.3 Mapper实现 90](#_Toc18938929)

[1.9.4 时序说明 92](#_Toc18938930)

[1.9.5 代码实现 92](#_Toc18938931)

[1.9.6 单元测试 97](#_Toc18938932)

[1.10 后端开发总结 97](#_Toc18938933)

[1.10.1 开发原则 97](#_Toc18938934)

[1.10.2 开发步骤 97](#_Toc18938935)

[1.10.3 开发优化 97](#_Toc18938936)

# 后端开发



## 文章内容接口

### 接口定义

#### 基本定义

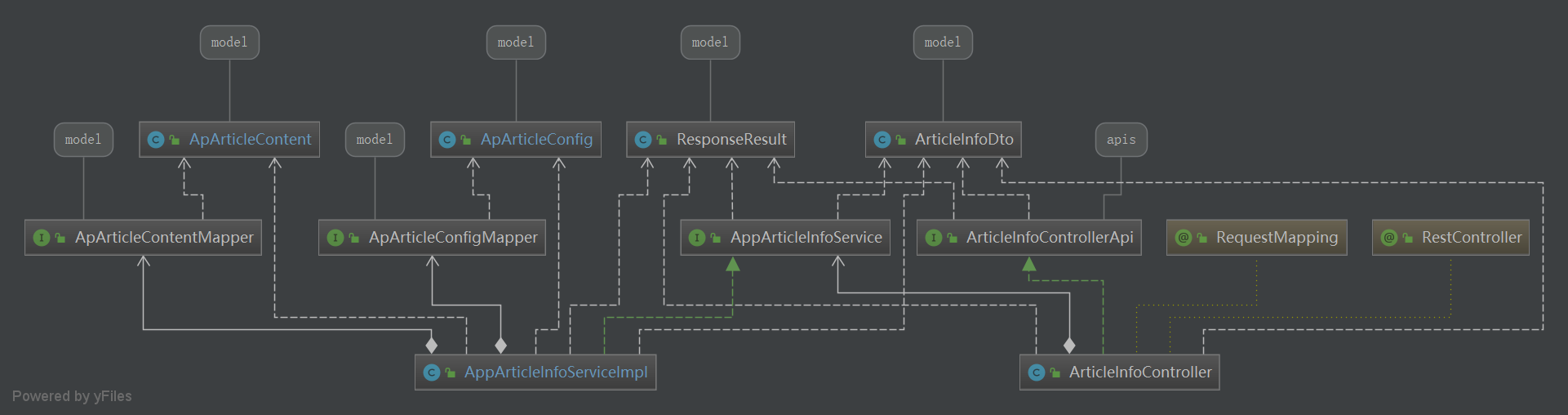
由于框架封装只对JSON反序列化自增ID，需要请求文章ID需要封装为DTO.

|  |  |
| --- | --- |
| 参考标准 | 请参考通用接口规范 |
| 接口名称 | /api/v1/article/load\_article\_info |
| 请求DTO | com.heima.article.mysql.core.model.dtos.ArticleInfoDto |
| 响应DTO | 使用Map进行封装，其中格式如下：  {  “config”:// ApArticleConfig  “content”://ApArticleContent  }  注意：如果文章已经删除，content属性将不返回 |

#### CODE定义

|  |  |
| --- | --- |
| PARAM\_INVALID | PARAM\_INVALID(501,"无效参数"), |

### 类定义



类说明：

* ApArticleContent、ApArticleConfig是对应数据表的POJO对象，放置model模块
* ArticleInfoDto封装接口请求数据，放置在model模块
* ApArticleContentMapper、ApArticleConfigMapper是MybatisMapper文件，放置在model模块
* ArticleInfoControllerApi是服务接口定义，放置在apis模块
* ApArticleInfoService、ApArticleInfoServiceImpl、ApArticleInfoController是对功能的实现，放置在article模块

### Mapper实现

#### ApArticleContent

创建类com.heima.article.mysql.core.model.pojos.app.ApArticleContent

生成的ApArticleContent注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。另外注意articleId需要增加@IdEncrypt注解，以作输出混淆。

@Data  
public class ApArticleContent {  
 private Integer id;  
 // 增加注解，JSON序列化时自动混淆加密  
 @IdEncrypt  
 private Integer articleId;  
 private String content;  
}

#### ApArticleContentMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApArticleContentMapper

定义按照文章ID查询内容方法：

public interface ApArticleContentMapper {  
 ApArticleContent selectByArticleId(Integer articleId);  
}

#### ApArticleContentMapper.xml

创建文件resources/mappers/app/ApArticleContentMapper.xml

ApArticleContent是按照article\_id字段进行分库分表，由Mycat管理自增主键，SQL如下：

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApArticleContentMapper" >  
 <resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApArticleContent" >  
 <id column="id" property="id" />  
 <result column="article\_id" property="articleId"/>  
 <result column="content" property="content" />  
 </resultMap>  
 <sql id="Base\_Column\_List" >  
 id, article\_id  
 </sql>  
 <sql id="Blob\_Column\_List" >  
 content  
 </sql>  
 <select id="selectByArticleId" resultMap="BaseResultMap" parameterType="java.lang.Integer" >  
 select  
 <include refid="Base\_Column\_List" />  
 ,  
 <include refid="Blob\_Column\_List" />  
 from ap\_article\_content  
 where article\_id = #{articleId}  
 </select>  
</mapper>

#### ApArticleConfig

创建类com.heima.article.mysql.core.model.pojos.app.ApArticleConfig

生成的ApArticleConfing注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。另外注意articleId需要增加@IdEncrypt注解，以作输出混淆。

@Data  
public class ApArticleConfig {  
 private Long id;  
 // 增加注解，JSON序列化时自动混淆加密  
 @IdEncrypt  
 private Integer articleId;  
 private Boolean isComment;  
 private Boolean isForward;  
 private Boolean isDown;  
 private Boolean isDelete;  
}

#### ApArticleConfigMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApArticleConfigMapper

定义按照文章ID查询内容方法：

public interface ApArticleConfigMapper {  
 ApArticleConfig selectByArticleId(Integer articleId);  
}

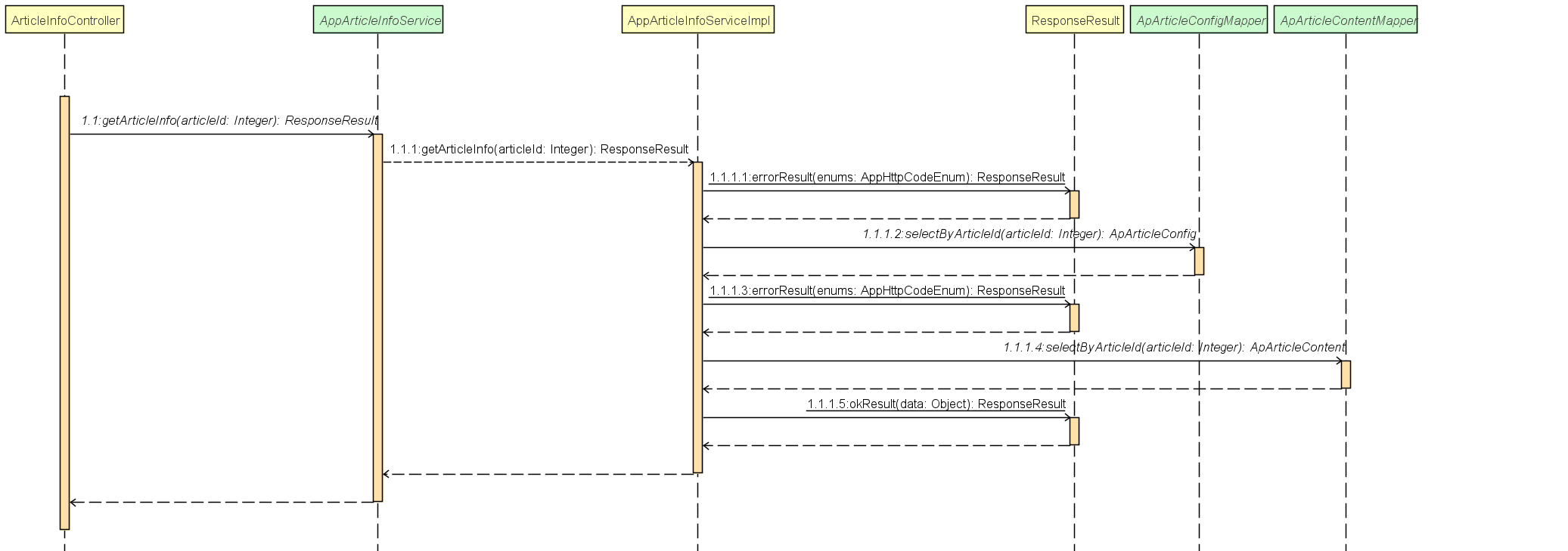
#### ApArticleConfigMapper.xml

创建文件resources/mappers/app/ApArticleConfigMapper.xml

ApArticleConfig是按照article\_id字段进行分库分表，由Mycat管理自增主键，SQL如下：

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApArticleConfigMapper" >  
 <resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApArticleConfig" >  
 <id column="id" property="id"/>  
 <result column="article\_id" property="articleId" />  
 <result column="is\_comment" property="isComment"/>  
 <result column="is\_forward" property="isForward" />  
 <result column="is\_down" property="isDown"/>  
 <result column="is\_delete" property="isDelete"/>  
 </resultMap>  
 <sql id="Base\_Column\_List" >  
 id, article\_id, is\_comment, is\_forward, is\_down, is\_delete  
 </sql>  
 <!-- 通过文章ID查询文章配置 -->  
 <select id="selectByArticleId" resultMap="BaseResultMap" parameterType="int" >  
 select <include refid="Base\_Column\_List" /> from ap\_article\_config where article\_id = #{articleId}  
 </select>  
</mapper>

### 时序说明



* 1.1.1.1判断入参articleId是否合法，不合法则返回PARAM\_INVALID错误
* 1.1.1.2查询文章对应的ApArticleConfig配置信息
* 1.1.1.3如果未查询到ApArticleConfig信息，则返回PARAM\_INVALID错误
* 1.1.1.4如果文章未被删除，则查找处理文章内容对象
* 1.1.1.5 封装响应DTO返回数据

### 代码实现

#### AppArticleInfoService

创建类：com.heima.article.service.AppArticleInfoService

定义获取文章详情接口：

public interface AppArticleInfoService {  
  
 */\*\*  
 \* 加载文章详情内容  
 \** ***@param*** *articleId  
 \** ***@return*** *\*/* ResponseResult getArticleInfo(Integer articleId);

}

#### AppArticleInfoServiceImpl

创建类：com.heima.article.service.impl.AppArticleInfoServiceImpl

@Getter

@Service  
public class AppArticleInfoServiceImpl implements AppArticleInfoService {  
  
 @Autowired  
 private ApArticleContentMapper apArticleContentMapper;  
 @Autowired  
 private ApArticleConfigMapper apArticleConfigMapper;   
  
 */\*\*  
 \* 加载文章详情内容  
 \** ***@param*** *articleId  
 \** ***@return*** *\*/* public ResponseResult getArticleInfo(Integer articleId){  
 // 参数无效  
 if(articleId==null||articleId<1){  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*PARAM\_INVALID*);  
 }  
 ApArticleConfig config = apArticleConfigMapper.selectByArticleId(articleId);  
 Map<String,Object> data = new HashMap<>();  
 // 参数无效  
 if(config==null){  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*PARAM\_INVALID*);  
 }else if(!config.getIsDelete()){  
 // 没删除的标识才返回给客户端  
 ApArticleContent content = apArticleContentMapper.selectByArticleId(articleId);  
 data.put("content",content);  
 }  
 data.put("config",config);  
 return ResponseResult.*okResult*(data);  
 }  
}

#### ArticleInfoDto

创建类：com.heima.article.mysql.core.model.dtos.ArticleInfoDto

此类在model模块中创建，定义请求入参，实现如下：

@Data  
public class ArticleInfoDto {  
 // 文章ID  
 @IdEncrypt  
 Integer articleId;   
}

#### ArticleInfoControllerApi

创建类：com.heima.article.apis.ArticleInfoControllerApi

此类在apis模块中创建，定义了相关接口，实现如下：

*/\*\*  
 \* 首頁文章  
 \*/*public interface ArticleInfoControllerApi {  
 */\*\*  
 \* 加載首頁详情  
 \** ***@param*** *dto 封装参数对象  
 \** ***@return*** *文章详情  
 \*/* ResponseResult loadArticleInfo(ArticleInfoDto dto);  
  
 */\*\*  
 \* 加载文章详情的行为内容  
 \** ***@param*** *dto  
 \** ***@return*** *\*/* ResponseResult loadArticleBehavior( ArticleInfoDto dto);  
  
}

#### ArticleInfoController

创建类：com.heima.article.controller.v1.ArticleInfoController

该类的实现较为简单，引入Service并调用即可：

@RestController  
@RequestMapping("/api/v1/article")  
public class ArticleInfoController implements ArticleInfoControllerApi {  
  
 @Autowired  
 private AppArticleInfoService appArticleInfoService;  
  
 @Override  
 @PostMapping("/load\_article\_info")  
 public ResponseResult loadArticleInfo(@RequestBody ArticleInfoDto dto) {  
 return appArticleInfoService.getArticleInfo(dto.getArticleId());  
 }

}

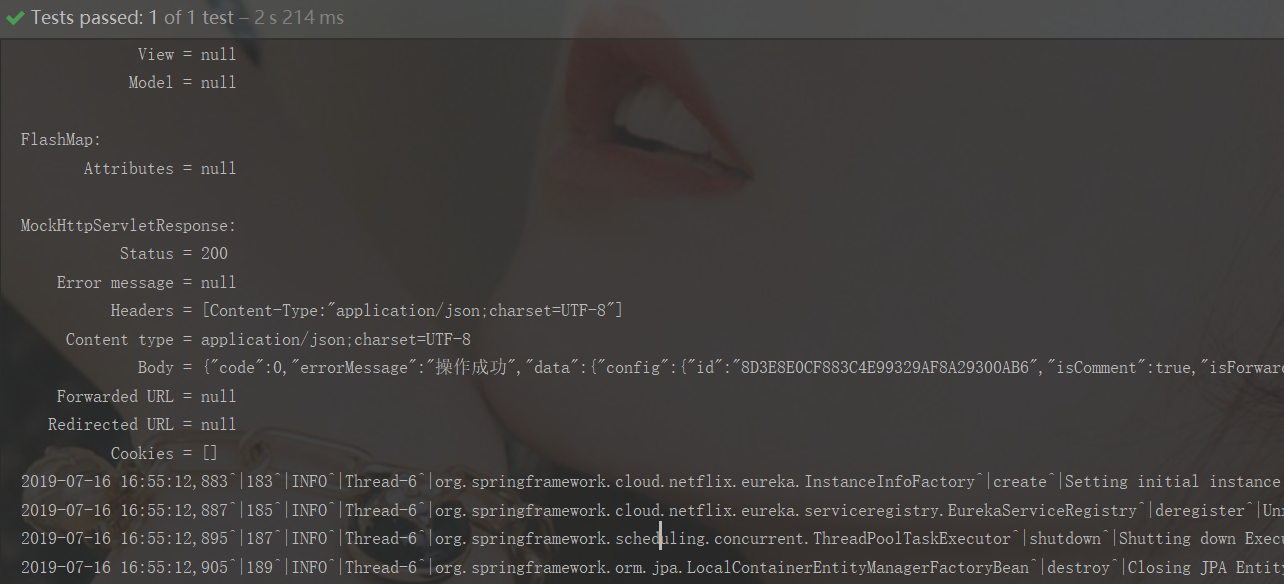
### 单元测试

创建测试类：com.heima.article.controller.v1.ArticleInfoControllerTest

使用MockMvc进行接口调用测试，代码如下：

@RunWith(SpringRunner.class)  
@SpringBootTest  
@AutoConfigureMockMvc  
public class ArticleInfoControllerTest {  
  
 @Autowired  
 MockMvc mvc;  
 @Autowired  
 ObjectMapper mapper;  
  
 @Test  
 public void testLoadArticleInfo() throws Exception{  
 ArticleInfoDto dto = new ArticleInfoDto();  
 dto.setArticleId(1);  
 MockHttpServletRequestBuilder builder = MockMvcRequestBuilders.*post*("/api/v1/article/load\_article\_info");  
 builder.contentType(MediaType.*APPLICATION\_JSON\_VALUE*)  
 .content(mapper.writeValueAsBytes(dto));  
 mvc.perform(builder).andExpect(MockMvcResultMatchers.*status*().isOk()).andDo(MockMvcResultHandlers.*print*());  
 }  
}

测试结果如下则成功：



## 文章关系接口

### 接口定义

#### 基本定义

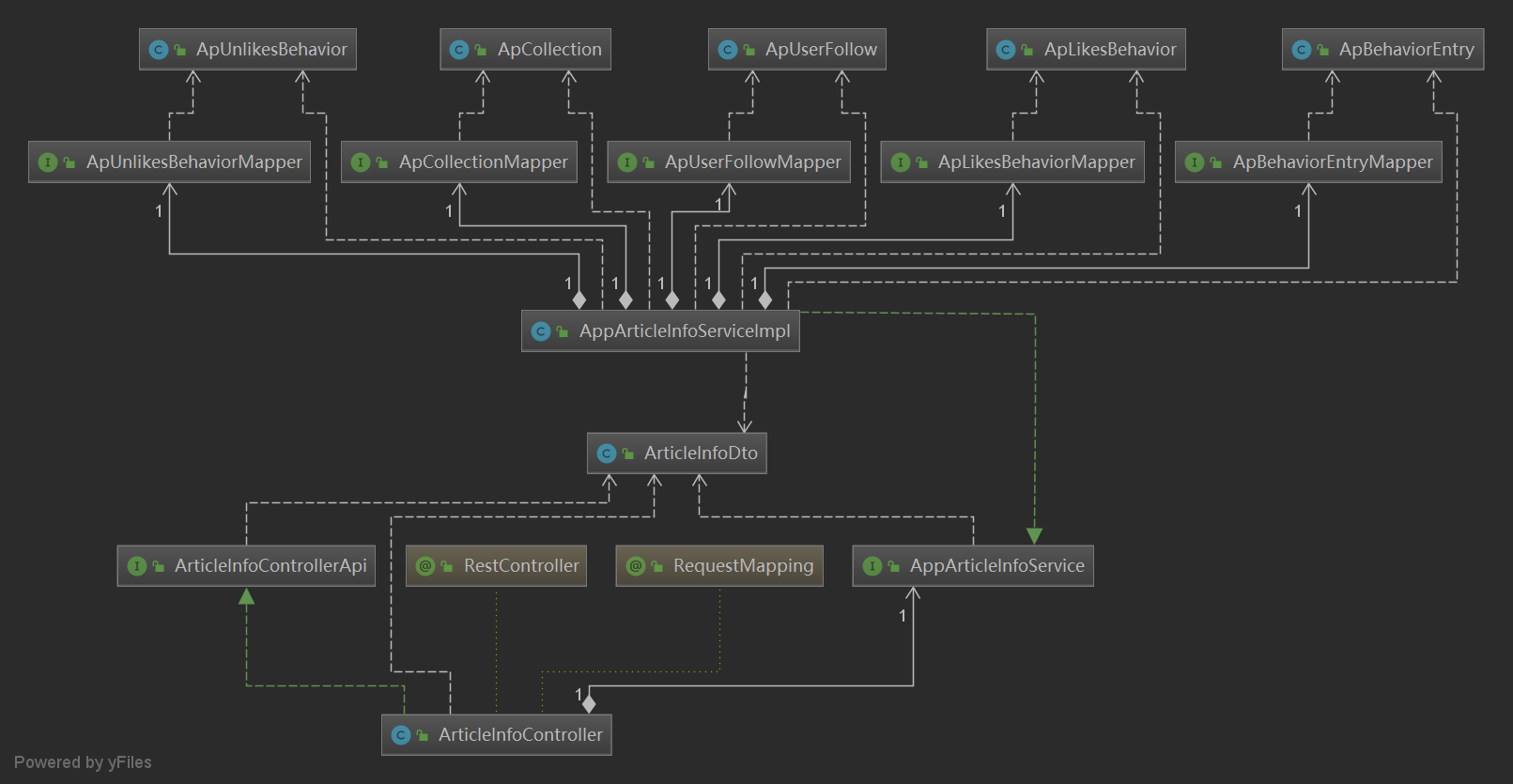
此接口用于加载当前行为实体与文章及文章作者之间的关系，比如喜欢、关注等。

|  |  |
| --- | --- |
| 参考标准 | 请参考通用接口规范 |
| 接口名称 | /api/v1/article/ load\_article\_behavior |
| 请求DTO | com.heima.article.mysql.core.model.dtos.ArticleInfoDto |
| 响应DTO | 使用Map进行封装，其中格式如下：  {  “isfollow”:// 是否关注,  “islike”:// 是否点赞,  “isunlike”:// 是否不喜欢,  “iscollection”://是否收藏  } |

#### CODE定义

|  |  |
| --- | --- |
| PARAM\_INVALID | PARAM\_INVALID(501,"无效参数") |
| PARAM\_REQUIRE | PARAM\_REQUIRE(500,"缺少参数") |

### 类定义



类说明：

* 涉及的pojo和Mapper都存储在model模块中

包括ApUnlikes\*、ApLikes\*、ApCollection\*、ApUserFollow\*、ApBehavior\*

* 请求DTO也重用ArticleInfoDto，缺少字段进行补充即可
* Service、Controller等类都进行重用，定义新的方法

### Mapper实现

相关类在model模块中实现

#### ApBehaviorEntry

创建类com.heima.article.mysql.core.model.pojos.app.ApBehaviorEntry，该类不会输出给前端，所以相关属性可不做混淆加密设置。

@Data  
public class ApBehaviorEntry {  
 private Integer id;  
 private Boolean type;  
 private Integer entryId;  
 private Date createdTime;  
 public String burst;  
}

#### ApBehaviorEntryMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApBehaviorEntryMapper

此类通过selectByUserIdOrEquipment方法查询对应行为实体数据，定义如下：

public interface ApBehaviorEntryMapper {  
 ApBehaviorEntry selectByUserIdOrEquipment(Long userId,Integer equipmentId);  
}

#### ApBehaviorEntryMapper.xml

查询行为实体优先用UserId进行查询，如未登录或提供则用设备ID进行查询，由于ApBehaviorEntry使用burst进行分片数据，因此查询时需用注解表达查询DN，实现如下：

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApBehaviorEntryMapper" >  
 <resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApBehaviorEntry" >  
 <id column="id" property="id" />  
 <result column="type" property="type"/>  
 <result column="entry\_id" property="entryId" />  
 <result column="created\_time" property="createdTime" />  
 <result column="burst" property="burst"/>  
 </resultMap>  
 <sql id="Base\_Column\_List" >  
 id, type, entry\_id, created\_time  
 </sql>  
  
 <!-- 选择用户的行为对象，优先按用户选择 -->  
 <select id="selectByUserIdOrEquipment" resultMap="BaseResultMap" >  
 <if test="userId!=null">  
 /\*!mycat:sql=select id from ap\_behavior\_entry where burst='0-${userId}'\*/  
 select \* from ap\_behavior\_entry a where a.entry\_id=#{userId} and type=1 limit 1  
 </if>  
  
 <if test="userId==null and equipmentId!=null">  
 /\*!mycat:sql=select id from ap\_behavior\_entry where burst='0-${equipmentId}'\*/  
 select \* from ap\_behavior\_entry a where a.entry\_id=#{equipmentId} and type=0 limit 1  
 </if>  
 </select>  
</mapper>

#### ApCollection

创建类com.heima.article.mysql.core.model.pojos.app.ApCollection

生成的ApCollection注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。另外注意behaviorEntryId、entryId需要增加@IdEncrypt注解，以作输出混淆，burst字段需要过滤输出。同时在此类中定义了收藏内容的枚举类型Type。

@Data  
public class ApCollection {  
 private Long id;  
 @IdEncrypt  
 private Integer behaviorEntryId;  
 @IdEncrypt  
 private Integer entryId;  
 private Short type;  
 private Date collectionTime;  
 private Date publishedTime;  
 @JsonIgnore  
 private String burst;  
 // 定义收藏内容类型的枚举  
 @Alias("ApCollectionEnumType")  
 public enum Type{  
 *ARTICLE*((short)0),*DYNAMIC*((short)1);  
 short code;  
 Type(short code){  
 this.code = code;  
 }  
 public short getCode(){  
 return this.code;  
 }  
 }  
}

#### ApCollectionMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApCollectionMapper

定义按照行为实体ID、收藏内容ID、和类型查询收藏方法：

public interface ApCollectionMapper {

/\*\*

\* 选择一个终端的收藏数据

\* @return

\*/

ApCollection selectForEntryId(String burst,Integer objectId,Integer entryId,Short type);

}

#### ApCollectionMapper.xml

创建文件resources/mappers/app/ApCollectionMapper.xml

ApCollection是按照burst字段进行分库分表，查询时注意使用Mycat注解确定路由DN，SQL如下：

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApCollectionMapper" >

<resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApCollection" >

<id column="id" property="id" />

<result column="behavior\_entry\_id" property="behaviorEntryId" />

<result column="entry\_id" property="entryId" />

<result column="type" property="type"/>

<result column="collection\_time" property="collectionTime" />

<result column="published\_time" property="publishedTime" />

<result column="burst" property="burst" />

</resultMap>

<sql id="Base\_Column\_List" >

id, behavior\_entry\_id, entry\_id, type, collection\_time, published\_time

</sql>

<select id="selectForEntryId" resultMap="BaseResultMap">

/\*!mycat:sql=select id from ap\_collection where burst='${burst}'\*/

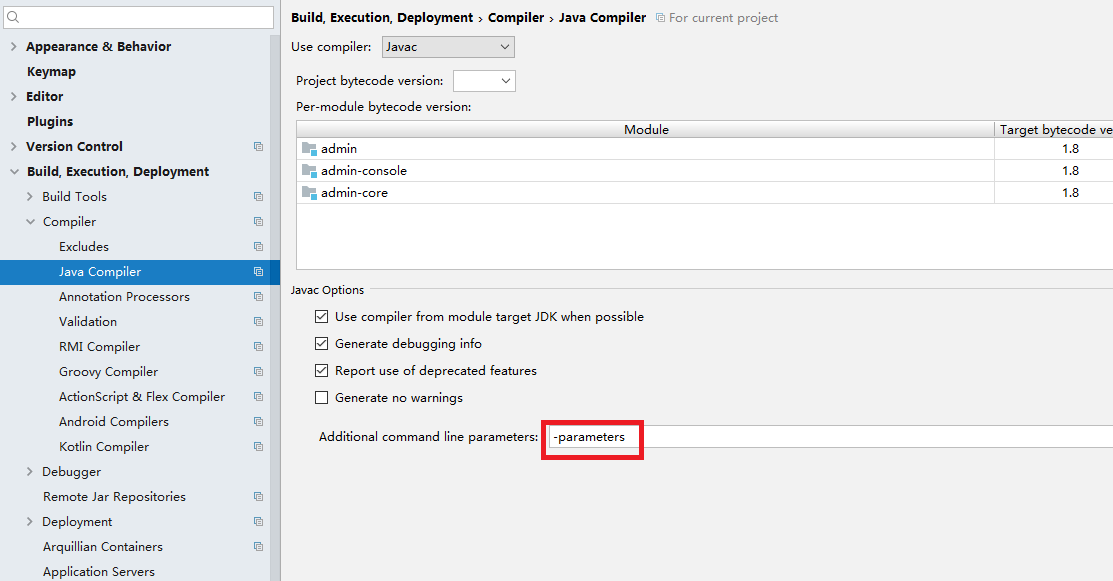
select \* from ap\_collection where behavior\_entry\_id=#{objectId} and entry\_id=#{entryId} and type=#{type}

</select>

</mapper>

注意：此处用了Mybatis的参数名称来映射SQL中的变量，要支持此方式，需要在Java编译时把方法参数名称信息也编译到class字节文件中，具体可设置一个编译参数即可：

* File->Settings->Build,Execution,Deployment->Compiler->Java Compiler
* 在 Additional command line parameters: 后面填上 -parameters，如下图



#### ApUserFollow

创建类com.heima.article.mysql.core.model.pojos.app.ApUserFollow

生成的ApUserFollow注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。另外注意userId、followId需要增加@IdEncrypt注解，以作输出混淆，burst字段需要过滤输出。

@Data  
public class ApUserFollow {  
 private Long id;  
 @IdEncrypt  
 private Long userId;  
 @IdEncrypt  
 private Integer followId;  
 private String followName;  
 private Short level;  
 private Boolean isNotice;  
 private Date createdTime;  
 @JsonIgnore  
 private String burst;  
}

#### ApUserFollowMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApUserFollowMapper

定义按照用户ID、关注用户Id查询关注信息方法：

public interface ApUserFollowMapper {

ApUserFollow selectByFollowId(String burst,Long userId,Integer followId);

}

#### ApUserFollowMapper.xml

创建文件resources/mappers/app/ApUserFollowMapper.xml

ApUserFollow是按照burst字段进行分库分表，查询时注意使用Mycat注解确定路由DN，SQL如下：

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApUserFollowMapper" >

<resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApUserFollow" >

<id column="id" property="id" />

<result column="user\_id" property="userId" />

<result column="follow\_id" property="followId" />

<result column="follow\_name" property="followName"/>

<result column="level" property="level"/>

<result column="is\_notice" property="isNotice"/>

<result column="created\_time" property="createdTime" />

<result column="burst" property="burst"/>

</resultMap>

<sql id="Base\_Column\_List" >

id, user\_id, follow\_id, follow\_name, level, is\_notice, created\_time

</sql>

<select id="selectByFollowId" resultMap="BaseResultMap" >

/\*!mycat:sql=select id from ap\_user\_follow where burst='${burst}'\*/

select \* from ap\_user\_follow where user\_id = #{userId} and follow\_id = #{followId}

</select>

</mapper>

#### ApLikesBehavior

创建类com.heima.article.mysql.core.model.pojos.app.ApLikesBehavior

生成的ApLikesBehavior注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。另外注意behaviorEntryId、entryId需要增加@IdEncrypt注解，以作输出混淆，burst字段需要过滤输出。同时在此类中定义了点赞内容的枚举类型Type、点赞操作的类型Operation。

@Data  
public class ApLikesBehavior {  
 private Long id;  
 @IdEncrypt  
 private Integer behaviorEntryId;  
 @IdEncrypt  
 private Integer entryId;  
 private Short type;  
 private Short operation;  
 private Date createdTime;  
 @JsonIgnore  
 private String burst;  
 // 定义点赞内容的类型  
 @Alias("ApLikesBehaviorEnumType")  
 public enum Type{  
 *ARTICLE*((short)0),*DYNAMIC*((short)1),*COMMENT*((short)2);  
 short code;  
 Type(short code){  
 this.code = code;  
 }  
 public short getCode(){  
 return this.code;  
 }  
 }  
 //定义点赞操作的方式，点赞还是取消点赞  
 @Alias("ApLikesBehaviorEnumOperation")  
 public enum Operation{  
 *LIKE*((short)0),*CANCEL*((short)1);  
 short code;  
 Operation(short code){  
 this.code = code;  
 }  
 public short getCode(){  
 return this.code;  
 }  
 }  
  
}

#### ApLikesBehaviorMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApLikesBehaviorMapper

定义按照行为实体、点赞内容、点赞操作方式查询点赞信息，指选择最后一条方法：

package com.heima.article.mysql.core.model.mappers.app;

public interface ApLikesBehaviorMapper {

/\*\*

\* 选择最后一条喜欢按钮

\* @return

\*/

ApLikesBehavior selectLastLike(String burst,Integer objectId,Integer entryId,Short type);

}

#### ApLikesBehaviorMapper.xml

创建文件resources/mappers/app/ApLikesBehaviorMapper.xml

ApLikesBehavior是按照burst字段进行分库分表，查询时注意使用Mycat注解确定路由DN，SQL如下：

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApLikesBehaviorMapper" >

<resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApLikesBehavior" >

<id column="id" property="id"/>

<result column="behavior\_entry\_id" property="behaviorEntryId"/>

<result column="entry\_id" property="entryId" />

<result column="type" property="type" />

<result column="operation" property="operation"/>

<result column="created\_time" property="createdTime"/>

<result column="burst" property="burst"/>

</resultMap>

<sql id="Base\_Column\_List" >

id, behavior\_entry\_id, entry\_id, type, operation, created\_time,burst

</sql>

<select id="selectLastLike" resultMap="BaseResultMap">

/\*!mycat:sql=select id from ap\_likes\_behavior where burst='${burst}'\*/

select \* from ap\_likes\_behavior where behavior\_entry\_id=#{objectId} and entry\_id=#{entryId} and type=#{type} order by created\_time desc limit 1

</select>

</mapper>

#### ApUnlikesBehavior

创建类com.heima.article.mysql.core.model.pojos.app.ApUnlikesBehavior

生成的ApUnlikesBehavior注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。另外注意articleId、entryId需要增加@IdEncrypt注解，以作输出混淆。同时在此类中定义了不喜欢操作的枚举类型Type。

@Data  
public class ApUnlikesBehavior {  
 private Long id;  
 @IdEncrypt  
 private Integer entryId;  
 @IdEncrypt  
 private Integer articleId;  
 private Short type;  
 private Date createdTime;  
 // 定义不喜欢操作的类型  
 @Alias("ApUnlikesBehaviorEnumType")  
 public enum Type{  
 *UNLIKE*((short)0),*CANCEL*((short)1);  
 short code;  
 Type(short code){  
 this.code = code;  
 }  
 public short getCode(){  
 return this.code;  
 }  
 }  
}

#### ApUnlikesBehaviorMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApUnlikesBehaviorMapper

定义按照行为实体ID、文章ID查询不喜欢最有一条信息的方法：

public interface ApUnlikesBehaviorMapper {

/\*\*

\* 选择最后一条不喜欢数据

\* @return

\*/

ApUnlikesBehavior selectLastUnLike(Integer entryId,Integer articleId);

}

#### ApUnlikesBehaviorMapper.xml

创建文件resources/mappers/app/ApUnlikesBehaviorMapper.xml

ApUnlikesBehavior是按照entry\_id字段进行分库分表，由Mycat管理自增主键，SQL如下：

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApUnlikesBehaviorMapper" >

<resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApUnlikesBehavior" >

<id column="id" property="id"/>

<result column="entry\_id" property="entryId" />

<result column="article\_id" property="articleId"/>

<result column="type" property="type"/>

<result column="created\_time" property="createdTime" />

</resultMap>

<sql id="Base\_Column\_List" >

id, entry\_id, type, created\_time

</sql>

<select id="selectLastUnLike" resultMap="BaseResultMap">

select \* from ap\_unlikes\_behavior where entry\_id=#{entryId} and article\_id=#{articleId} order by created\_time desc limit 1

</select>

</mapper>

#### ApAuthor

创建类com.heima.article.mysql.core.model.pojos.app.ApAuthor

生成的ApAuthor注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。

package com.heima.article.mysql.core.model.pojos.app;

import lombok.Data;

import java.util.Date;

@Data

public class ApAuthor {

private Integer id;

private String name;

private Boolean type;

private Integer userId;

private Date createdTime;

}

#### ApAuthorMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApAuthorMapper

定义按照作者ID查询作者信息的方法：

public interface ApAuthorMapper {  
 ApAuthor selectById(Integer id);  
}

#### ApAuthorMapper.xml

创建文件resources/mappers/app/ApAuthorMapper.xml

ApAuthor是没有分库分表，由Mycat管理自增主键，SQL如下：

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApAuthorMapper" >

<resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApAuthor" >

<result column="id" property="id" />

<result column="name" property="name"/>

<result column="type" property="type"/>

<result column="user\_id" property="userId"/>

<result column="created\_time" property="createdTime"/>

</resultMap>

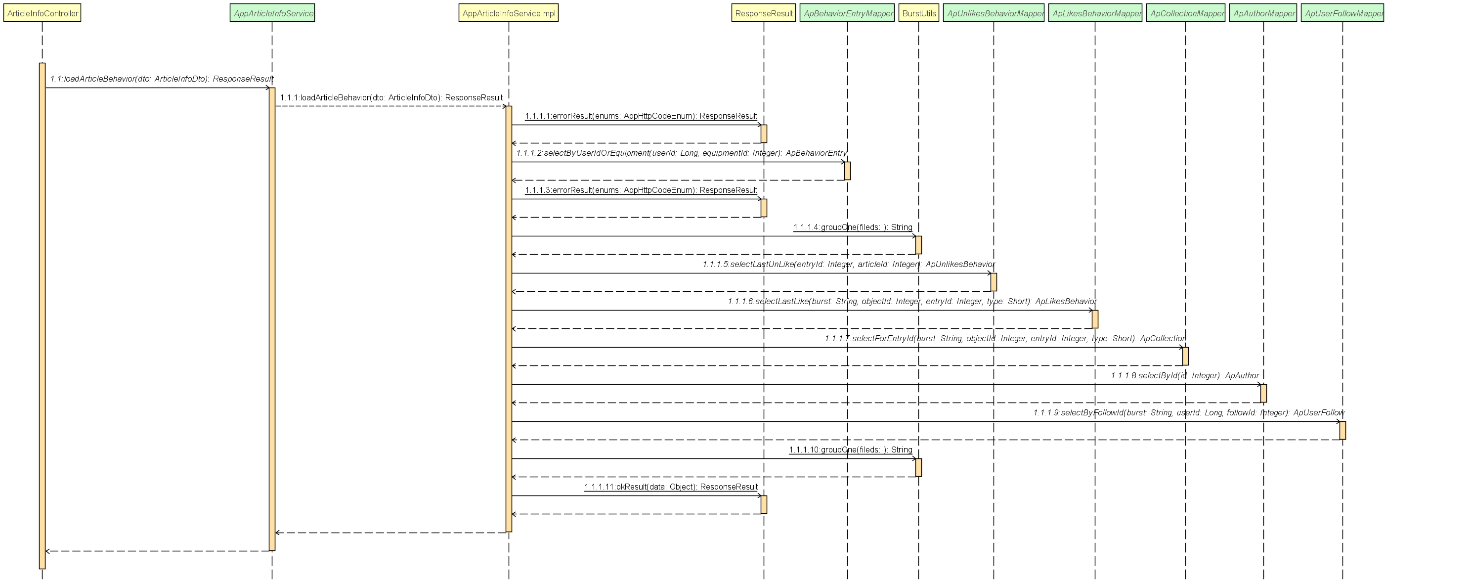
<select id="selectById" resultMap="BaseResultMap">

select \* from ap\_author where id=#{id}

</select>

</mapper>

### 时序说明



* 1.1.1.1 判断行为实体参数是否存在，如果不存在则返回PARAM\_REQUIRE错误
* 1.1.1.2 查询行为实体用户信息表
* 1.1.1.3 判断行为实体用户信息是否存在，如果不存在则返回PARAM\_INVALID错误
* 1.1.1.5 查询行为实体对当前文章是否存在记录，并判断操作类型确定当前状态
* 1.1.1.6 查询行为实体对当前文章存在的最后一天点赞操作，并判断操作类型确定当前状态
* 1.1.1.7 查询行为实体对当前文章收藏信息，并确定当前收藏的状态
* 1.1.1.8 依据文章作者ID查询作者信息，如果作者是平台用户，则进行1.1.1.9操作
* 1.1.1.9 依据当前用户ID和作者用户ID查询关注信息，并确定当前关注状态
* 1.1.1.11 封装响应DTO返回数据

### 代码实现

#### AppArticleInfoService

在com.heima.article.service.AppArticleInfoService类中定义接口：

*/\*\*  
 \* 加载文章详情的初始化配置信息，比如关注、喜欢、不喜欢、阅读位置等  
 \** ***@param*** *dto  
 \** ***@return*** *\*/*ResponseResult loadArticleBehavior(ArticleInfoDto dto);

#### AppArticleInfoServiceImpl

在com.heima.article.service.impl.AppArticleInfoServiceImpl类中实现接口：

*/\*\*  
 \* 加载文章详情的初始化配置信息，比如关注、喜欢、不喜欢、阅读位置等  
 \** ***@param*** *dto  
 \** ***@return*** *\*/*public ResponseResult loadArticleBehavior(ArticleInfoDto dto){  
 ApUser user = AppThreadLocalUtils.*getUser*();  
 // 用户和设备不能同时为空  
 if(user==null&& dto.getEquipmentId()==null){  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*PARAM\_REQUIRE*);  
 }  
 Long userId = null;  
 if(user!=null){  
 userId = user.getId();  
 }  
 ApBehaviorEntry apBehaviorEntry = getApBehaviorEntryMapper().selectByUserIdOrEquipment(userId, dto.getEquipmentId());  
 // 行为实体找以及注册了，逻辑上这里是必定有值得，除非参数错误  
 if(apBehaviorEntry==null){  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*PARAM\_INVALID*);  
 }  
 boolean isUnLike=false,isLike=false,isCollection=false,isFollow=false;  
 String burst = BurstUtils.*groudOne*(apBehaviorEntry.getId());  
  
 // 判断是否是已经不喜欢  
 ApUnlikesBehavior apUnlikesBehavior = getApUnlikesBehaviorMapper().selectLastUnLike(apBehaviorEntry.getId(),dto.getArticleId());  
 if(apUnlikesBehavior!=null&&apUnlikesBehavior.getType()==ApUnlikesBehavior.Type.*UNLIKE*.getCode()){  
 isUnLike=true;  
 }  
 // 判断是否是已经喜欢  
 ApLikesBehavior apLikesBehavior = getApLikesBehaviorMapper().selectLastLike(burst,apBehaviorEntry.getId(),dto.getArticleId(), ApCollection.Type.*ARTICLE*.getCode());  
 if(apLikesBehavior!=null&&apLikesBehavior.getOperation()==ApLikesBehavior.Operation.*LIKE*.getCode()){  
 isLike=true;  
 }  
 // 判断是否收藏  
 ApCollection apCollection = getApCollectionMapper().selectForEntryId(burst,apBehaviorEntry.getId(),dto.getArticleId(),ApCollection.Type.*ARTICLE*.getCode());  
 if(apCollection!=null){  
 isCollection=true;  
 }  
 // 判断是否关注  
 ApAuthor apAuthor = getApAuthorMapper().selectById(dto.getAuthorId());  
 if(user!=null&&apAuthor!=null&&apAuthor.getUserId()!=null) {  
 ApUserFollow apUserFollow = getApUserFollowMapper().selectByFollowId(BurstUtils.*groudOne*(user.getId()), user.getId(), apAuthor.getUserId());  
 if (apUserFollow != null) {  
 isFollow = true;  
 }  
 }  
  
 Map<String,Object> data = Maps.*newHashMap*();  
 data.put("isfollow",isFollow);  
 data.put("islike",isLike);  
 data.put("isunlike",isUnLike);  
 data.put("iscollection",isCollection);  
  
 return ResponseResult.*okResult*(data);  
}

#### ArticleInfoDto

在com.heima.article.mysql.core.model.dtos.ArticleInfoDto类中增加以下字段：

// 设备ID  
 @IdEncrypt  
 Integer equipmentId;   
 // 作者ID  
 @IdEncrypt  
 Integer authorId;

#### ArticleInfoControllerApi

在com.heima.article.apis.ArticleInfoControllerApi类中增加接口定义：

*/\*\*  
 \* 加载文章详情的行为内容  
 \** ***@param*** *dto  
 \** ***@return*** *\*/* ResponseResult loadArticleBehavior( ArticleInfoDto dto);

#### ArticleInfoController

在com.heima.article.controller.v1.ArticleInfoController类中，增加接口方法：

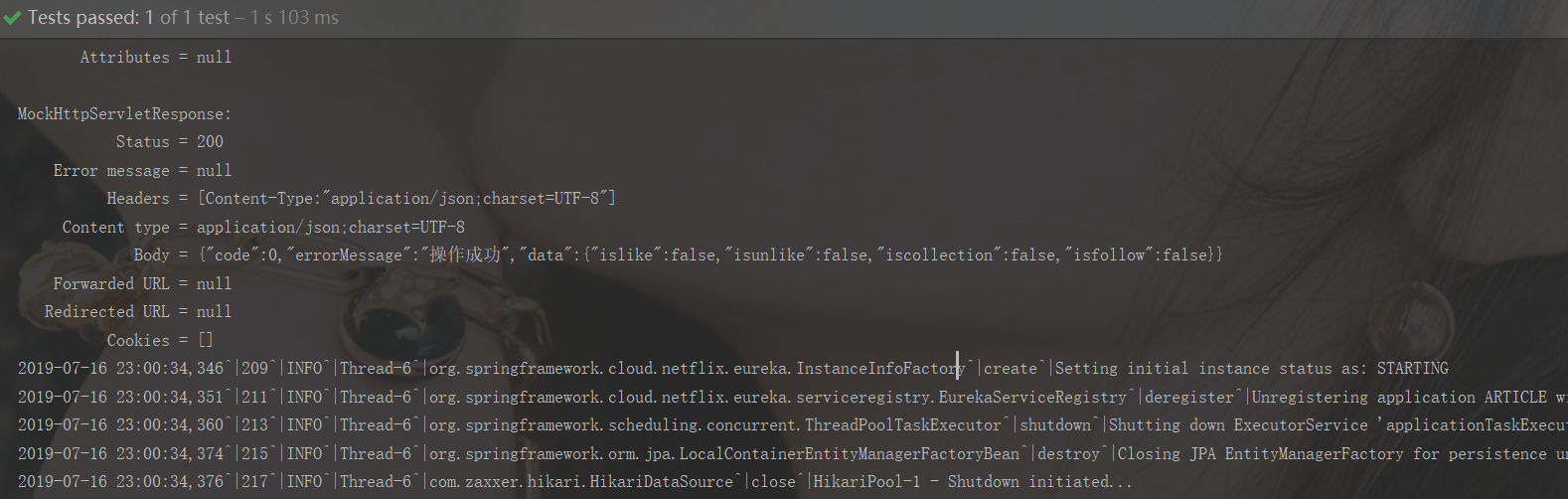
@Override  
@PostMapping("/load\_article\_behavior")  
public ResponseResult loadArticleBehavior(@RequestBody ArticleInfoDto dto) {  
 return appArticleInfoService.loadArticleBehavior(dto);  
}

### 单元测试

在com.heima.article.controller.v1.ArticleInfoControllerTest测试类中使用MockMvc进行接口调用测试，代码如下：

@Test  
public void testLoadArticleBehavior() throws Exception{  
 ArticleInfoDto dto = new ArticleInfoDto();  
 dto.setArticleId(1);  
 dto.setAuthorId(1);  
 dto.setEquipmentId(1);  
 MockHttpServletRequestBuilder builder = MockMvcRequestBuilders.*post*("/api/v1/article/load\_article\_behavior");  
 builder.contentType(MediaType.*APPLICATION\_JSON\_VALUE*)  
 .content(mapper.writeValueAsBytes(dto));  
 mvc.perform(builder).andExpect(MockMvcResultMatchers.*status*().isOk()).andDo(MockMvcResultHandlers.*print*());  
}

测试结果如下则成功：



## 关注接口

关注接口实现当前登录用户关注其他用户，属于APP用户个人中心服务的接口，因此需要先创建工程。

### 工程创建

* 在根项目下创建Maven module项目，项目名称为heima-leadnews-user，groupId:com.heima。创建后在根pom.xml中module元素中增加以下代码：

<module>heima-leadnews-user</module>

* 在heima-leadnews-user/pom.xml中增加父项目信息：

<parent>  
 <artifactId>heima-leadnews</artifactId>  
 <groupId>com.heima</groupId>  
 <version>1.0-SNAPSHOT</version>  
</parent>

* 复制heima-leadnews-article下的dev、test、prod三个环境配置文件到heima-leadnews-article下
* 复制heima-leadnews-article resources下的application.propertie、log4j2.xml文件到heima-leadnews-user/src/main/resources下
* 复制heima-leadnews-article/pom.xml中的依赖和build内容到heima-leadnews-user/pom.xml中

### 初始项目

* 创建com.heima.user.config包，并复制heima-leadnews-article项目下com.heima.article.config的InitConfig、MysqlConfig、SecurityConfig三个文件
* 创建启动类：com.heima.user.UserJarApplication

@SpringBootApplication  
@EnableDiscoveryClient  
public class UserJarApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(UserJarApplication.class, args);  
 }  
}

### 接口定义

#### 基本定义

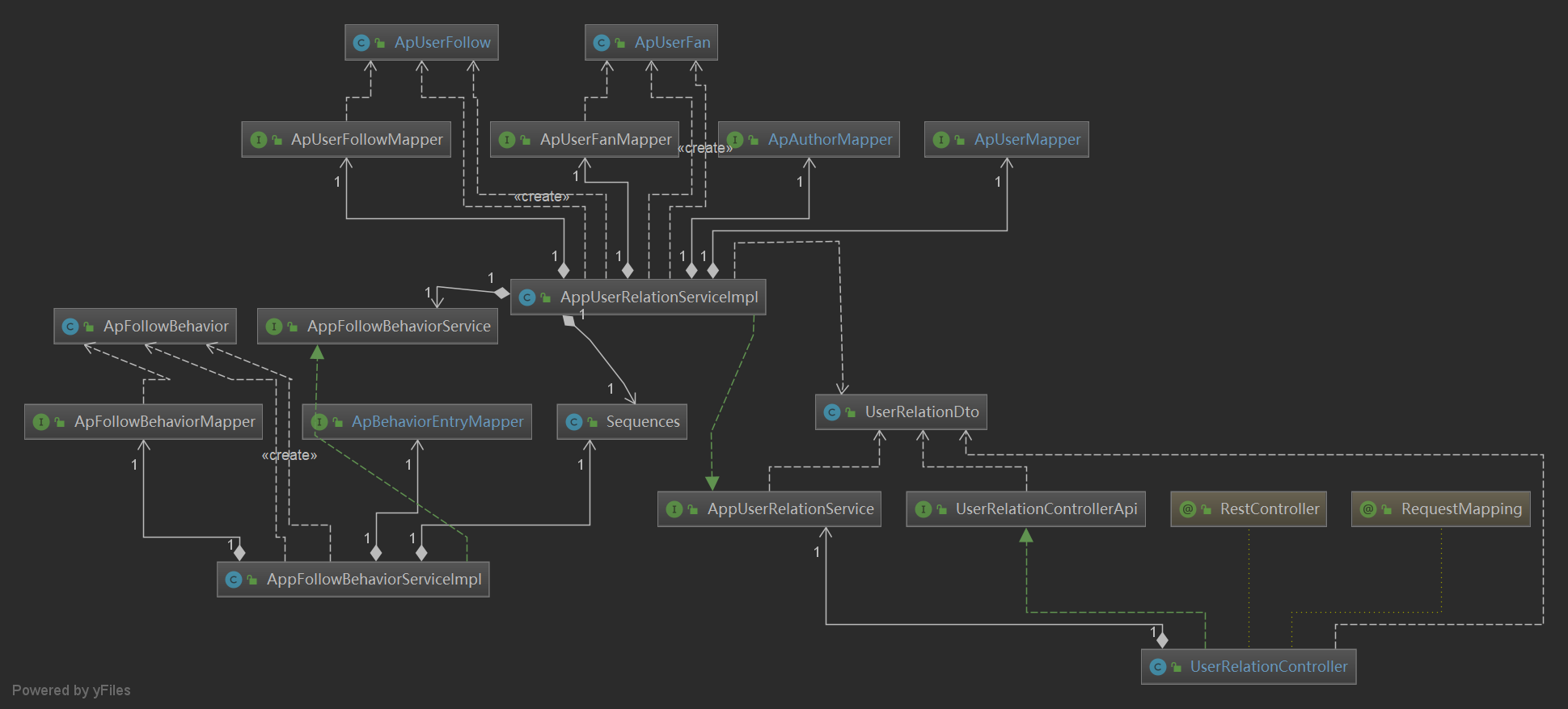
此接口用于实现当前用户关注其它用户的记录，基本定义如下：

|  |  |
| --- | --- |
| 参考标准 | 请参考通用接口规范 |
| 接口名称 | /api/v1/user/user\_follow |
| 请求DTO | com.heima.user.mysql.core.model.dtos.UserRelationDto |
| 响应DTO | 操作成功的数量 |

#### CODE定义

|  |  |
| --- | --- |
| PARAM\_INVALID | PARAM\_INVALID(501,"无效参数") |
| PARAM\_REQUIRE | PARAM\_REQUIRE(500,"缺少参数") |
| DATA\_NOT\_EXIST | DATA\_NOT\_EXIST(1000,"数据不存在") |
| NEED\_LOGIN | NEED\_LOGIN(1,"需要登录后操作") |
| DATA\_EXIST | DATA\_EXIST(1000,"数据已经存在") |

### 类定义



* Ap\*FollowBehavior\*相关类在这里主要实现关注行为数据的处理
* AppUserRelationService定义了关注方法，并在AppUserRelationServiceImpl中实现
* 相关的pojo、mapper都在model模块定义，service和controller在user模块中定义
* Sequences是Zk序列类，在之前已在Common中定义
* UserRelationControllerApi定义了follow方法，且在apis模块定义

### Mapper实现

相关类在那个模块中实现？

#### ApFollowBehavior

创建类com.heima.article.mysql.core.model.pojos.app.ApFollowBehavior

生成的ApFollowBehavior注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。

@Data  
public class ApFollowBehavior {  
 private Long id;  
 private Integer entryId;  
 private Integer articleId;  
 private Integer followId;  
 private Date createdTime;  
}

#### ApFollowBehaviorMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApFollowBehaviorMapper

定义按照关注行为插入方法：

public interface ApFollowBehaviorMapper {

int insert(ApFollowBehavior record);

}

#### ApFollowBehaviorMapper.xml

创建文件resources/mappers/app/ApFollowBehaviorMapper.xml

ApFollowBehavior是按照entry\_id字段进行分库分表，由Mycat管理自增主键，在插入时主键id值不需要设置，SQL如下：

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApFollowBehaviorMapper" >

<resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApFollowBehavior" >

<id column="id" property="id" />

<result column="entry\_id" property="entryId"/>

<result column="article\_id" property="articleId"/>

<result column="follow\_id" property="followId"/>

<result column="created\_time" property="createdTime"/>

</resultMap>

<sql id="Base\_Column\_List" >

id, entry\_id, article\_id, follow\_id, created\_time

</sql>

<insert id="insert" parameterType="com.heima.article.mysql.core.model.pojos.app.ApFollowBehavior" >

insert into ap\_follow\_behavior (entry\_id, article\_id, follow\_id, created\_time)

values ( #{entryId}, #{articleId},#{followId}, #{createdTime})

</insert>

</mapper>

#### ApUserFollowMapper

在com.heima.article.mysql.core.model.mappers.app.ApUserFollowMapper中定义关注信息插入（关注时调用）和删除（取消关注时）的方法：

int insert(ApUserFollow record);;

int deleteByFollowId(String burst,Long userId,Integer followId);

#### ApUserFollowMapper.xml

在resources/mappers/app/ApUserFollowMapper.xml中，实现对应Mapper方法，注意以下两点：

* 删除时，使用Mycat注解确定数据路由DN
* 插入时，需插入ID注解，并在程序中调用Sequences生产ZK自增ID

<delete id="deleteByFollowId">

/\*!mycat:sql=select id from ap\_user\_follow where burst='${burst}'\*/

delete from ap\_user\_follow where user\_id = #{userId} and follow\_id = #{followId}

</delete>

<insert id="insert" parameterType="com.heima.article.mysql.core.model.pojos.app.ApUserFollow" >

insert into ap\_user\_follow (id, user\_id, follow\_id,

follow\_name, level, is\_notice,

created\_time,burst)

values (#{id}, #{userId}, #{followId},

#{followName}, #{level}, #{isNotice},

#{createdTime},#{burst})

</insert>

#### ApUserFan

创建类com.heima.article.mysql.core.model.pojos.app.ApUserFan

生成的ApUserFan注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。另外注意userID、fansId需要增加@IdEncrypt注解，以作输出混淆，burst字段需要过滤输出。

@Data  
public class ApUserFan {  
 private Long id;  
 @IdEncrypt  
 private Integer userId;  
 @IdEncrypt  
 private Long fansId;  
 private String fansName;  
 private Short level;  
 private Date createdTime;  
 private Boolean isDisplay;  
 private Boolean isShieldLetter;  
 private Boolean isShieldComment;  
 @JsonIgnore  
 private String burst;  
}

#### ApUserFanMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApUserFanMapper

定义按照插入粉丝、查找粉丝、删除粉丝的方法：

public interface ApUserFanMapper {

int insert(ApUserFan record);

ApUserFan selectByFansId(String burst,Integer userId ,Long fansId);

int deleteByFansId(String burst,Integer userId ,Long fansId);

}

#### ApUserFanMapper.xml

创建文件resources/mappers/app/ApUserFanMapper.xml

ApUserFan是按照burst字段进行分库分表，查询时注意使用Mycat注解：

* 删除时，使用Mycat注解确定数据路由DN
* 插入时，需插入ID注解，并在程序中调用Sequences生产ZK自增ID
* 查询时，使用Mycat注解确定数据路由DN

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApUserFanMapper" >  
 <resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApUserFan" >  
 <id column="id" property="id"/>  
 <result column="user\_id" property="userId"/>  
 <result column="fans\_id" property="fansId" />  
 <result column="fans\_name" property="fansName"/>  
 <result column="level" property="level"/>  
 <result column="created\_time" property="createdTime" />  
 <result column="is\_display" property="isDisplay"/>  
 <result column="is\_shield\_letter" property="isShieldLetter"/>  
 <result column="is\_shield\_comment" property="isShieldComment"/>  
 <result column="burst" property="burst"/>  
 </resultMap>  
 <sql id="Base\_Column\_List" >  
 id, user\_id, fans\_id, fans\_name, level, created\_time, is\_display, is\_shield\_letter,is\_shield\_comment  
 </sql>  
 <select id="selectByFansId" resultMap="BaseResultMap">  
 /\*!mycat:sql=select id from ap\_user\_fan where burst='${burst}'\*/  
 select <include refid="Base\_Column\_List" /> from ap\_user\_fan where user\_id = #{userId} and fans\_id = #{fansId}  
 </select>  
  
 <delete id="deleteByFansId">  
 /\*!mycat:sql=select id from ap\_user\_fan where burst='${burst}'\*/  
 delete from ap\_user\_fan where user\_id = #{userId} and fans\_id = #{fansId}  
 </delete>  
  
 <insert id="insert" parameterType="com.heima.article.mysql.core.model.pojos.app.ApUserFan" >  
 insert into ap\_user\_fan (id, user\_id, fans\_id,  
 fans\_name, level, created\_time,   
 is\_display, is\_shield\_letter, is\_shield\_comment,burst  
 )  
 values (#{id}, #{userId}, #{fansId},  
 #{fansName}, #{level}, #{createdTime},  
 #{isDisplay}, #{isShieldLetter}, #{isShieldComment},{#burst}  
 )  
 </insert>  
</mapper>

#### ApUser

创建类com.heima.article.mysql.core.model.pojos.app.ApUser

生成的ApUser注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。

@Data  
public class ApUser {  
 private Long id;  
 private String salt;  
 private String name;  
 private String password;  
 private String phone;  
 private String image;  
 private Boolean sex;  
 private Boolean isCertification;  
 private Boolean isIdentityAuthentication;  
 private Boolean status;  
 private int flag;  
}

#### ApUserMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApUserMapper

定义按照用户ID查询用户信息的方法：

public interface ApUserMapper {

ApUser selectById(Integer id);

}

#### ApUserMapper.xml

创建文件resources/mappers/app/ApUserMapper.xml

ApUser是按照article\_id字段进行分库分表，由Mycat管理自增主键，SQL如下：

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApUserMapper" >

<resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApUser" >

<id column="id" property="id" />

<result column="salt" property="salt"/>

<result column="name" property="name"/>

<result column="password" property="password"/>

<result column="phone" property="phone"/>

<result column="image" property="image"/>

<result column="sex" property="sex"/>

<result column="is\_certification" property="isCertification"/>

<result column="is\_identity\_authentication" property="isIdentityAuthentication"/>

<result column="status" property="status"/>

<result column="flag" property="flag"/>

</resultMap>

<sql id="Base\_Column\_List" >

id, salt, name, password, phone, image, sex, is\_certification, is\_identity\_authentication,

status, flag

</sql>

<select id="selectById" resultMap="BaseResultMap">

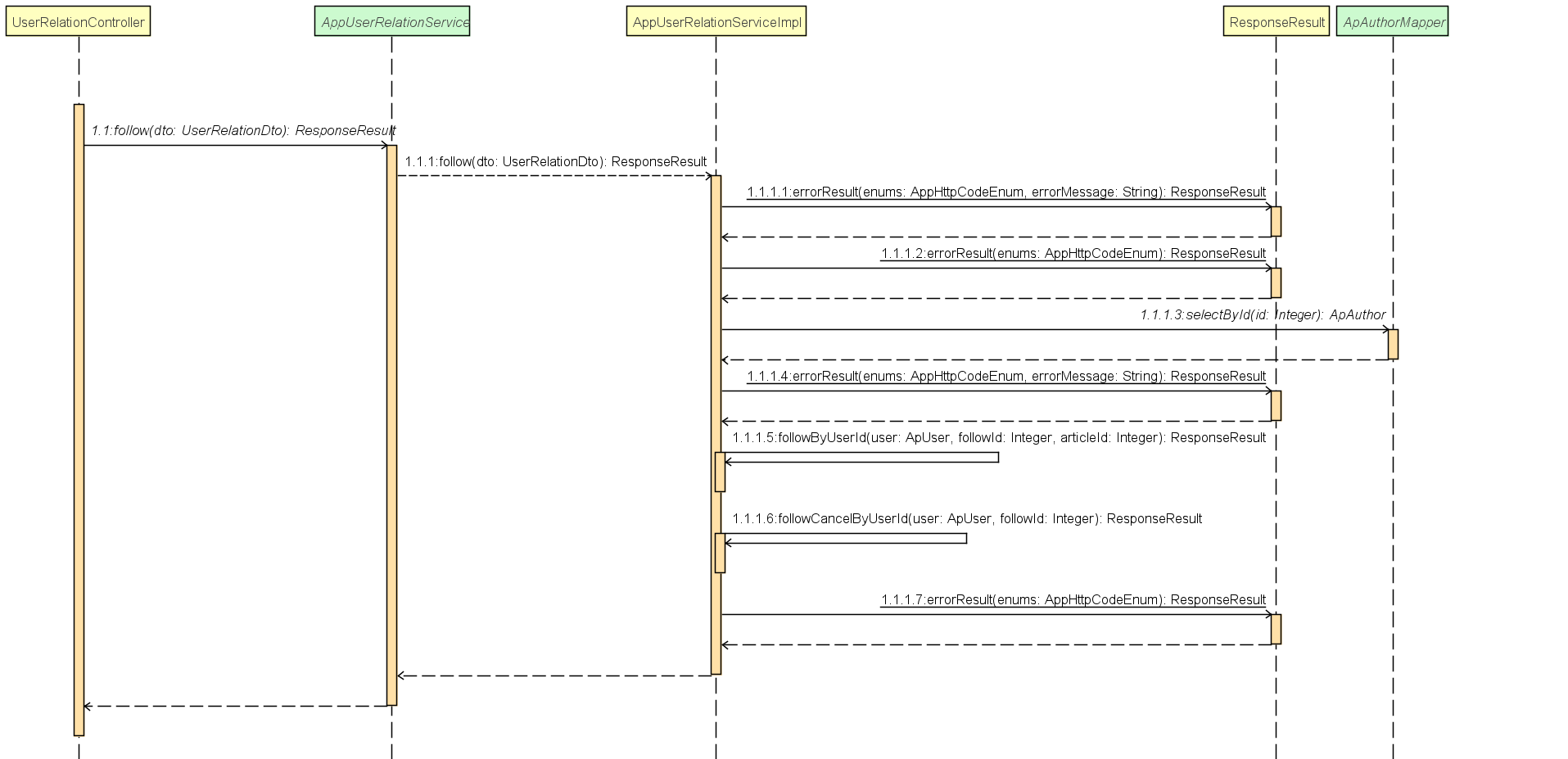
select <include refid="Base\_Column\_List" /> from ap\_user where id = #{id}

</select>

</mapper>

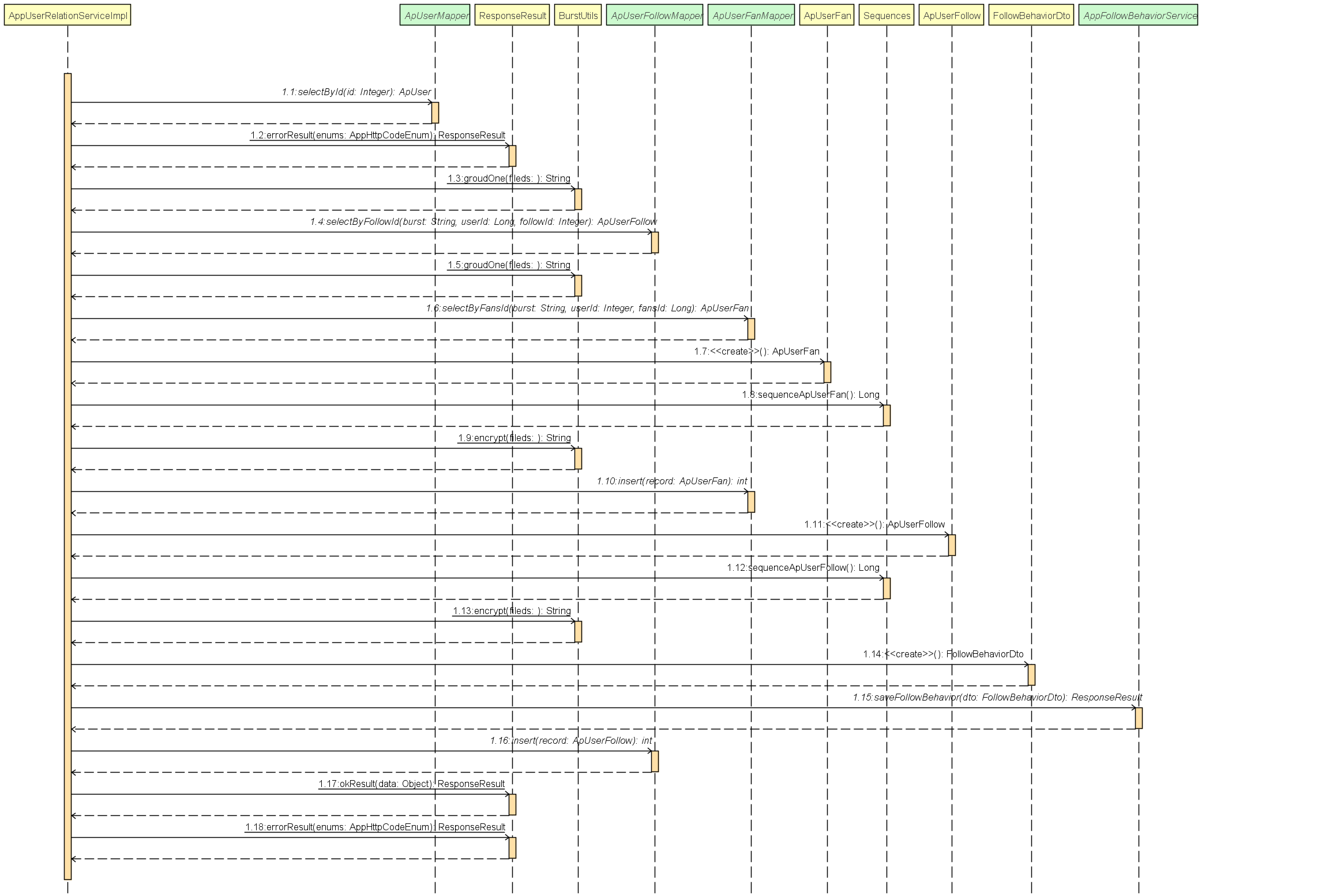
### 时序说明

#### 流程



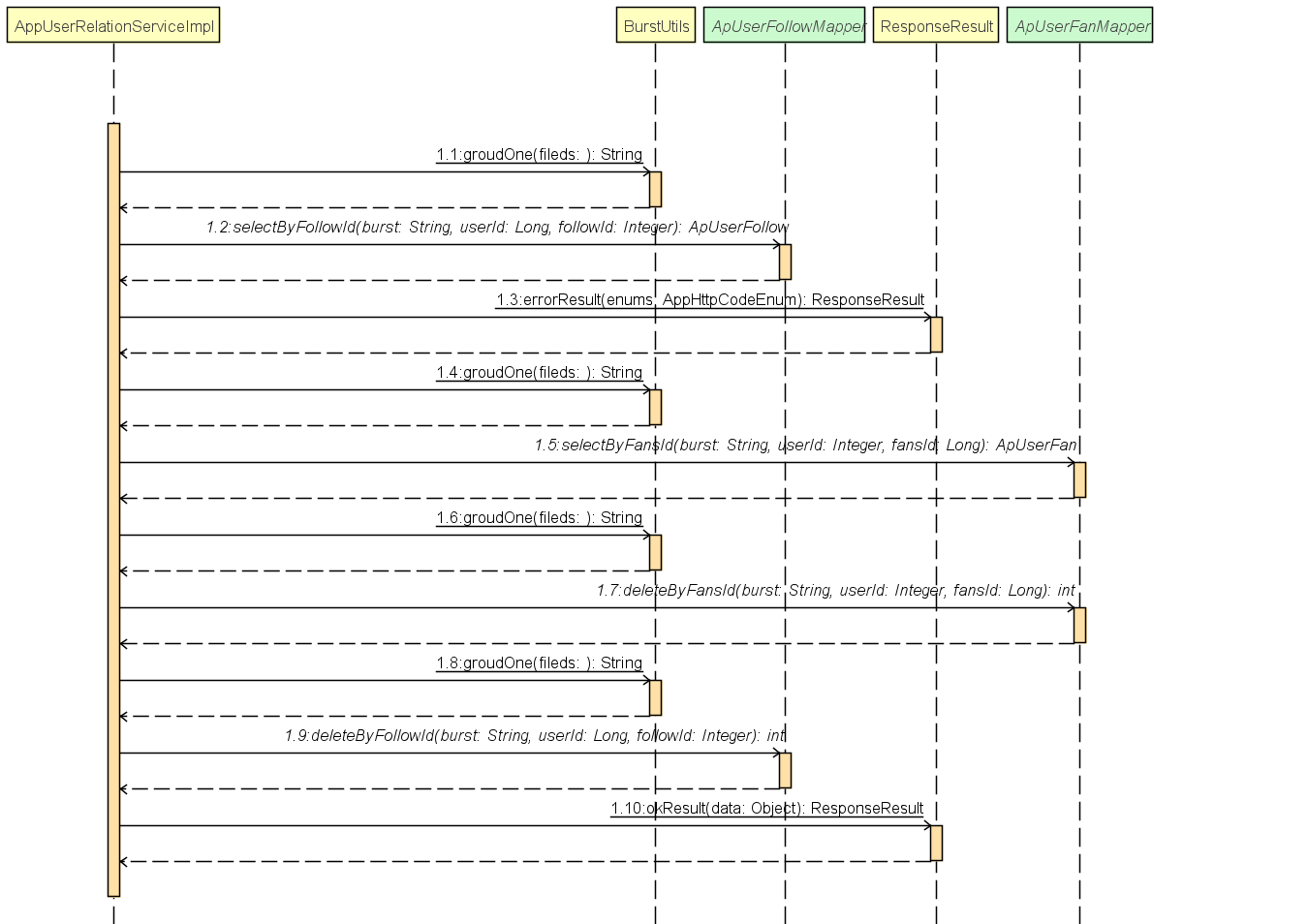
* 1.1.1.1如果operation参数不合法，则返回PARAM\_INVALID错误
* 1.1.1.2如果缺少关注者或作者ID，则返回PARAM\_REQUIRE错误
* 1.1.1.3如果传入的是作者的ID则查询作者信息
* 1.1.1.4如果作者信息为空，不能获取到关注人ID，则返回DATA\_NOT\_EXIST错误
* 1.1.1.5如果是关注操作，则调用关注方法
* 1.1.1.6如果是取消关注操作，则调用取消关注
* 1.1.1.7如果当前登录信息为null，则返回NEED\_LOGIN错误

#### 关注



* 1.2判断1.1查询的关注用户信息是否存在，如果不存在则返回DATA\_NOT\_EXIST错误
* 1.4查询当前用户与关注用户的关注信息
* 1.6查询当前用户与关注用户的粉丝信息
* 1.7如果粉丝信息不存在，则创建一条粉丝信息，并在1.10插入数据库
* 1.11创建当前用户关注行为数据，并在1.15保存关注行为数据
* 1.16插入当前关注操作的关注信息，并在1.17返回成功
* 1.18如果当前用户与关注用户已经存在数据，则返回DATA\_EXIST

#### 取消



* 1.3如果当前用户与关注用户的关注信息不存在，则返回DATA\_NOT\_EXIST错误
* 1.5查找当前用户与关注用户的粉丝信息，如果查得，则1.7进行删除
* 1.9删除当前用户与关注用户的关注信息，1.10返回成功

### 代码实现

#### AppFollowBehaviorService

创建类com.heima.user.service.AppFollowBehaviorService，并定义saveFollowBehavior方法：

public interface AppFollowBehaviorService {  
 */\*\*  
 \* 存储关注数据  
 \** ***@param*** *dto  
 \** ***@return*** *\*/* public ResponseResult saveFollowBehavior(FollowBehaviorDto dto);  
}

#### AppFollowBehaviorServiceImpl

创建类com.heima.user.service.impl.AppFollowBehaviorServiceImpl，在此需求中，存储关注行为是一个可选需求，因此采用异步存储的方式，提升关注接口的性能，并注意@Async的方法和调用方法不能存放同一个类中。

@Service  
public class AppFollowBehaviorServiceImpl implements AppFollowBehaviorService {  
Logger logger = LoggerFactory.*getLogger*(AppFollowBehaviorServiceImpl.class);  
 @Autowired  
 private ApFollowBehaviorMapper apFollowBehaviorMapper;  
 @Autowired  
 private ApBehaviorEntryMapper apBehaviorEntryMapper;  
 @Autowired  
 private Sequences sequences;  
  
 @Override  
 @Async  
 public ResponseResult saveFollowBehavior(FollowBehaviorDto dto){

logger.info("异步存储关注行为：{}",dto);  
 ApUser user = AppThreadLocalUtils.*getUser*();  
 // 用户和设备不能同时为空  
 if(user==null&& dto.getEquipmentId()==null){  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*PARAM\_REQUIRE*);  
 }  
 Long userId = null;  
 if(user!=null){  
 userId = user.getId();  
 }  
 ApBehaviorEntry apBehaviorEntry = apBehaviorEntryMapper.selectByUserIdOrEquipment(userId, dto.getEquipmentId());  
 // 行为实体找以及注册了，逻辑上这里是必定有值得，除非参数错误  
 if(apBehaviorEntry==null){  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*PARAM\_INVALID*);  
 }  
 ApFollowBehavior alb = new ApFollowBehavior();  
 alb.setEntryId(apBehaviorEntry.getId());  
 alb.setCreatedTime(new Date());  
 alb.setArticleId(dto.getArticleId());  
 alb.setFollowId(dto.getFollowId());  
 return ResponseResult.*okResult*(apFollowBehaviorMapper.insert(alb));  
 }  
}

#### ThreadPoolConfig

User模块创建类：com.heima.user.config.ThreadPoolConfig

异步方法的执行实际会使用一个SimpleThreadPoolTaskExecutor，实际项目中为监控和排除线程错误常常自定义相关线程池，代码如下：

* 注意使用@EnableAsync开启异步执行的功能
* 自定义的线程池建议都自定义前缀，便于日志排查

@Configuration  
@EnableAsync  
public class ThreadPoolConfig {  
  
 private static final int *corePoolSize* = 10; // 核心线程数（默认线程数）  
 private static final int *maxPoolSize* = 100; // 最大线程数  
 private static final int *keepAliveTime* = 10; // 允许线程空闲时间（单位：默认为秒）  
 private static final int *queueCapacity* = 500; // 缓冲队列数  
 private static final String *threadNamePrefix* = "default-async"; // 线程池名前缀  
  
 */\*\*  
 \* 默认异步线程池  
 \** ***@return*** *\*/* @Bean  
 public ThreadPoolTaskExecutor taskExecutor(){  
 ThreadPoolTaskExecutor pool = new ThreadPoolTaskExecutor();  
 pool.setThreadNamePrefix(*threadNamePrefix*);  
 pool.setCorePoolSize(*corePoolSize*);  
 pool.setMaxPoolSize(*maxPoolSize*);  
 pool.setKeepAliveSeconds(*keepAliveTime*);  
 pool.setQueueCapacity(*queueCapacity*);  
 // 直接在execute方法的调用线程中运行  
 pool.setRejectedExecutionHandler(new ThreadPoolExecutor.CallerRunsPolicy());  
 // 初始化  
 pool.initialize();  
 return pool;  
 }  
  
}

问题：在saveFollowBehavior方法中能使用AppThreadLocalUtils获取用户信息吗？为什么？

#### AppArticleInfoService

创建类com.heima.user.service.AppUserRelationService，并定义follow方法：

public interface AppUserRelationService {  
 public ResponseResult follow(UserRelationDto dto);  
}

#### AppArticleInfoServiceImpl

创建类com.heima.user.service.impl.AppUserRelationServiceImpl，并实现follow方法：

@Service  
public class AppUserRelationServiceImpl implements AppUserRelationService {  
  
 Logger logger = LoggerFactory.*getLogger*(AppUserRelationServiceImpl.class);  
  
 @Autowired  
 ApUserFollowMapper apUserFollowMapper;  
 @Autowired  
 ApUserFanMapper apUserFanMapper;  
 @Autowired  
 ApAuthorMapper apAuthorMapper;  
 @Autowired  
 ApUserMapper apUserMapper;  
 @Autowired  
 AppFollowBehaviorService appFollowBehaviorService;  
 @Autowired  
 Sequences sequences;  
  
 */\*\*  
 \* 关注/取消一个人  
 \** ***@param*** *dto  
 \** ***@return*** *\*/* public ResponseResult follow(UserRelationDto dto){  
 if(dto.getOperation()==null||dto.getOperation()<0||dto.getOperation()>1){  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*PARAM\_INVALID*,"无效的operation参数");  
 }  
 Integer followId = dto.getUserId();  
 if(followId==null&&dto.getAuthorId()==null) {  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*PARAM\_REQUIRE*,"followId或authorId不能为空");  
 }else if(followId==null) {  
 ApAuthor aa = apAuthorMapper.selectById(dto.getAuthorId());  
 if(aa!=null) {  
 followId = aa.getUserId();  
 }  
 }  
 if(followId==null){  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*DATA\_NOT\_EXIST*,"关注人不存在");  
 }else {  
 ApUser user = AppThreadLocalUtils.*getUser*();  
 if(user!=null) {  
 if(dto.getOperation()==0) {  
 return followByUserId(user, followId, dto.getArticleId());  
 }else{  
 return followCancelByUserId(user,followId);  
 }  
 }else{  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*NEED\_LOGIN*);  
 }  
 }  
 }  
  
 */\*\*  
 \* 处理关注逻辑  
 \** ***@param*** *user  
 \** ***@param*** *followId  
 \** ***@return*** *\*/* private ResponseResult followByUserId(ApUser user,Integer followId,Integer articleId){  
 // 判断用户是否存在  
 ApUser followUser = apUserMapper.selectById(followId);  
 if(followUser==null){  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*DATA\_NOT\_EXIST*,"关注用户不存在");  
 }  
 ApUserFollow auf = apUserFollowMapper.selectByFollowId(BurstUtils.*groudOne*(user.getId()),user.getId(),followId);  
 if(auf==null){   
 ApUserFan fan = apUserFanMapper.selectByFansId(BurstUtils.*groudOne*(followId), followId, user.getId());  
 if (fan == null) {  
 fan = new ApUserFan();  
 fan.setId(sequences.sequenceApUserFan());  
 fan.setUserId(followId);  
 fan.setFansId(user.getId());  
 fan.setFansName(user.getName());  
 fan.setLevel((short) 0);  
 fan.setIsDisplay(true);  
 fan.setIsShieldComment(false);  
 fan.setIsShieldLetter(false);  
 fan.setBurst(BurstUtils.*encrypt*(fan.getId(), fan.getUserId()));  
 apUserFanMapper.insert(fan);  
 }  
 auf = new ApUserFollow();  
 auf.setId(sequences.sequenceApUserFollow());  
 auf.setUserId(user.getId());  
 auf.setFollowId(followId);  
 auf.setFollowName(followUser.getName());  
 auf.setCreatedTime(new Date());  
 auf.setLevel((short) 0);  
 auf.setIsNotice(true);  
 auf.setBurst(BurstUtils.*encrypt*(auf.getId(),auf.getUserId()));  
 // 记录关注行为  
 FollowBehaviorDto dto = new FollowBehaviorDto();  
 dto.setFollowId(followId);  
 dto.setArticleId(articleId);  
 appFollowBehaviorService.saveFollowBehavior(dto);  
 return ResponseResult.*okResult*(apUserFollowMapper.insert(auf));  
 }else{  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*DATA\_EXIST*,"已关注");  
 }  
 }  
  
 */\*\*  
 \* 处理取消关注逻辑  
 \** ***@param*** *user  
 \** ***@param*** *followId  
 \** ***@return*** *\*/* private ResponseResult followCancelByUserId(ApUser user,Integer followId){  
 ApUserFollow auf = apUserFollowMapper.selectByFollowId(BurstUtils.*groudOne*(user.getId()),user.getId(),followId);  
 if(auf==null){  
 return ResponseResult.*errorResult*(AppHttpCodeEnum.*DATA\_NOT\_EXIST*,"未关注");  
 }else{  
 ApUserFan fan = apUserFanMapper.selectByFansId(BurstUtils.*groudOne*(followId), followId, user.getId());  
 if (fan != null) {  
 apUserFanMapper.deleteByFansId(BurstUtils.*groudOne*(followId), followId, user.getId());  
 }  
 return ResponseResult.*okResult*(apUserFollowMapper.deleteByFollowId(BurstUtils.*groudOne*(user.getId()),user.getId(),followId));  
 }  
 }  
}

#### UserRelationDto

在Model中创建类com.heima.user.mysql.core.model.dtos.UserRelationDto：

@Data  
public class UserRelationDto {  
  
 // 文章作者ID  
 @IdEncrypt  
 Integer authorId;  
  
 // 用户ID  
 @IdEncrypt  
 Integer userId;  
  
 // 文章  
 @IdEncrypt  
 Integer articleId;  
 */\*\*  
 \* 操作方式  
 \* 0 关注  
 \* 1 取消  
 \*/* Short operation;  
}

#### UserRelationControllerApi

在com.heima.user.apis.UserRelationControllerApi类中定义接口：

*/\*\*  
 \* 关注  
 \*/*public interface UserRelationControllerApi {  
 ResponseResult follow(UserRelationDto dto);  
}

#### UserRelationController

在com.heima.user.controller.v1.UserRelationController类中，增加接口：

@RestController  
@RequestMapping("/api/v1//user")  
public class UserRelationController implements UserRelationControllerApi {  
 @Autowired  
 private AppUserRelationService appUserRelationService;  
 @Override  
 @PostMapping("/user\_follow")  
 public ResponseResult follow(@RequestBody UserRelationDto dto){  
 return appUserRelationService.follow(dto);  
 }  
}

### 单元测试

在com.heima.user.controller.v1.UserRelationControllerTest测试类中使用MockMvc进行接口调用测试，代码如下：

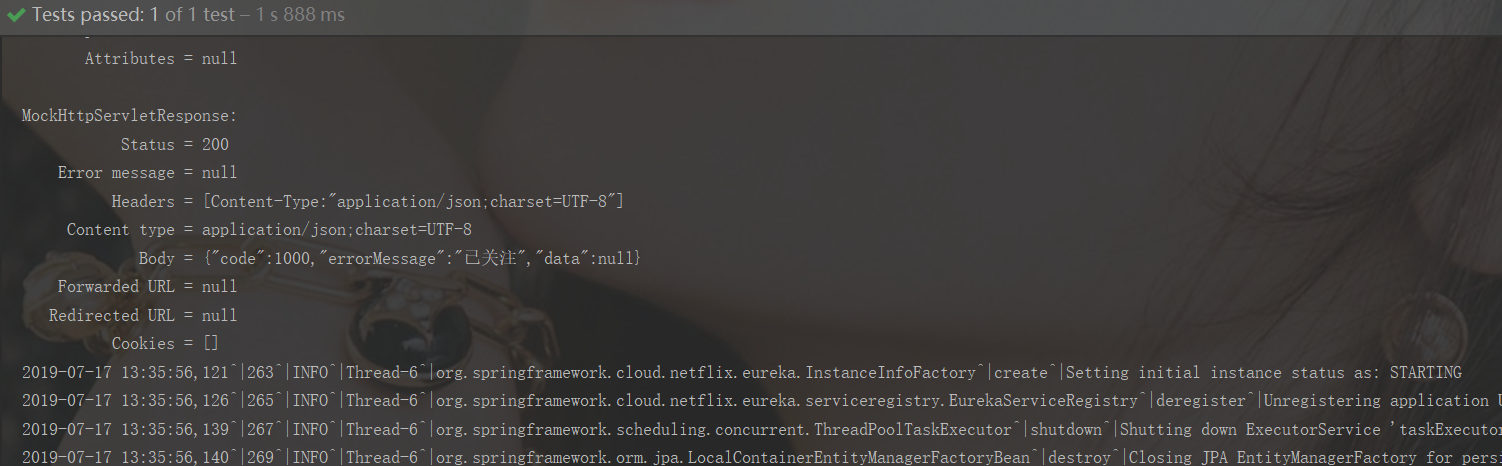
* 测试用例中不会自动加载WebFilter定义的过滤，如需要使用可手动添加，此处不演示，直接把用户信息设置在线程中，较为便捷。

@RunWith(SpringRunner.class)  
@SpringBootTest  
@AutoConfigureMockMvc  
public class UserRelationControllerTest {  
  
 @Autowired  
 MockMvc mvc;  
 @Autowired  
 ObjectMapper mapper;  
  
 // 设置线程中的用户信息  
 @Before  
 public void initUser(){  
 ApUser user = new ApUser();  
 user.setId(1l);  
 AppThreadLocalUtils.*setUser*(user);  
 }  
 // 关注  
 @Test  
 public void testFollowAdd() throws Exception{  
 UserRelationDto dto = new UserRelationDto();  
 dto.setOperation((short)0);  
 dto.setArticleId(1);  
 dto.setAuthorId(1);  
 MockHttpServletRequestBuilder builder = MockMvcRequestBuilders.*post*("/api/v1/user/user\_follow");  
 builder.contentType(MediaType.*APPLICATION\_JSON\_VALUE*)  
 .content(mapper.writeValueAsBytes(dto));  
 mvc.perform(builder).andExpect(MockMvcResultMatchers.*status*().isOk()).andDo(MockMvcResultHandlers.*print*());  
 Thread.*sleep*(10000);  
 }  
  
 // 取消关注  
 @Test  
 public void testFollowCancel() throws Exception{  
 UserRelationDto dto = new UserRelationDto();  
 dto.setOperation((short)1);  
 dto.setArticleId(1);  
 dto.setAuthorId(1);  
 MockHttpServletRequestBuilder builder = MockMvcRequestBuilders.*post*("/api/v1/user/user\_follow");  
 builder.contentType(MediaType.*APPLICATION\_JSON\_VALUE*)  
 .content(mapper.writeValueAsBytes(dto));  
 mvc.perform(builder).andExpect(MockMvcResultMatchers.*status*().isOk()).andDo(MockMvcResultHandlers.*print*());  
 }  
}

测试异步线程成功执行：



测试结果如下则成功：



## 点赞接口

此接口应在behavior模块中定义

### 接口定义

#### 基本定义

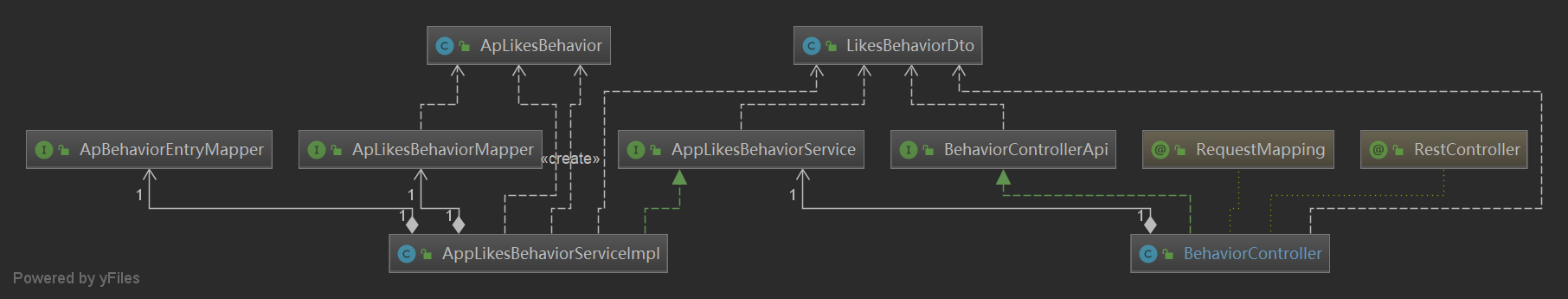
由于框架封装只对JSON反序列化自增ID，需要请求文章ID需要封装为DTO.

|  |  |
| --- | --- |
| 参考标准 | 请参考通用接口规范 |
| 接口名称 | /api/v1/behavior/like\_behavior |
| 请求DTO | com.heima.behavior.mysql.core.model.dtos.LikesBehaviorDto |
| 响应DTO | 输出插入数据条数的数量 |

#### CODE定义

|  |  |
| --- | --- |
| PARAM\_INVALID | PARAM\_INVALID(501,"无效参数"), |
| PARAM\_REQUIRE | PARAM\_REQUIRE(500,"缺少参数") |

### 类定义



Controller相关的类已创建此增量代码即可，AppLikesBehavior\*在behavior模块中定义。

### Mapper实现

#### ApLikesBehaviorMapper

在com.heima.article.mysql.core.model.mappers.app.ApLikesBehaviorMapper中定义点赞数据插入方法：

int insert(ApLikesBehavior record);

#### ApLikesBehaviorMapper.xml

在resources/mappers/app/ApLikesBehaviorMapper.xml中，实现对应Mapper的insert方法，注意插入数据需在程序中调用Sequences生产ZK自增ID：

<insert id="insert" parameterType="com.heima.article.mysql.core.model.pojos.app.ApLikesBehavior" >

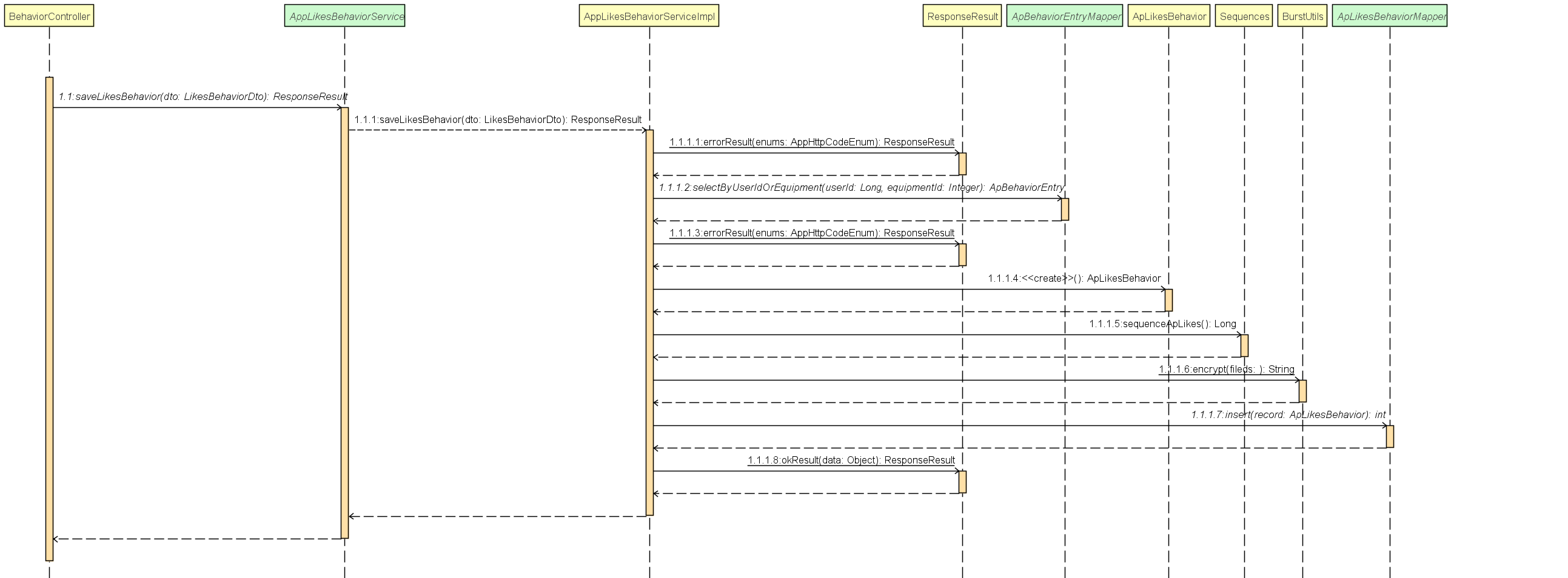
insert into ap\_likes\_behavior (id, behavior\_entry\_id, entry\_id, type, operation, created\_time,burst)

values (#{id}, #{behaviorEntryId}, #{entryId}, #{type}, #{operation},

#{createdTime},#{burst})

</insert>

### 时序说明



* 1.1.1.1如果行为实体的ID存在，则1.1.1.2查询其对象，否则返回PARAM\_REQUIRE错误
* 1.1.1.3判断行为实体是否存在，如果不存在则返回PARAM\_INVALID错误，否则1.1.1.4创建对象，并在1.1.1.7插入数据库。

### 代码实现

#### AppLikesBehaviorService

创建类：com.heima.behavior.service.AppLikesBehaviorService

定义文章关系信息接口：

public interface AppLikesBehaviorService {

/\*\*

\* 存储喜欢数据

\* @param dto

\* @return

\*/

public ResponseResult saveLikesBehavior(LikesBehaviorDto dto);

}

#### AppLikesBehaviorServiceImpl

创建类：com.heima.behavior.service.impl.AppLikesBehaviorServiceImpl，实现saveLikesBehavior方法，增量插入行为数据。

@Service

public class AppLikesBehaviorServiceImpl implements AppLikesBehaviorService {

@Autowired

private ApLikesBehaviorMapper apLikesBehaviorMapper;

@Autowired

private ApBehaviorEntryMapper apBehaviorEntryMapper;

@Autowired

private Sequences sequences;

@Override

public ResponseResult saveLikesBehavior(LikesBehaviorDto dto){

ApUser user = AppThreadLocalUtils.getUser();

// 用户和设备不能同时为空

if(user==null&& dto.getEquipmentId()==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_REQUIRE);

}

Long userId = null;

if(user!=null){

userId = user.getId();

}

ApBehaviorEntry apBehaviorEntry = apBehaviorEntryMapper.selectByUserIdOrEquipment(userId, dto.getEquipmentId());

// 行为实体找以及注册了，逻辑上这里是必定有值得，除非参数错误

if(apBehaviorEntry==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_INVALID);

}

ApLikesBehavior alb = new ApLikesBehavior();

alb.setId(sequences.sequenceApLikes());

alb.setBehaviorEntryId(apBehaviorEntry.getId());

alb.setCreatedTime(new Date());

alb.setEntryId(dto.getEntryId());

alb.setType(dto.getType());

alb.setOperation(dto.getOperation());

alb.setBurst(BurstUtils.encrypt(alb.getId(),alb.getBehaviorEntryId()));

return ResponseResult.okResult(apLikesBehaviorMapper.insert(alb));

}

}

#### LikesBehaviorDto

创建类：com.heima.behavior.mysql.core.model.dtos.LikesBehaviorDto

此类在model模块中创建，定义请求入参，实现如下，注意entryId、equipmentId属性需要序列化混淆输入。

@Data

public class LikesBehaviorDto {

// 设备ID

@IdEncrypt

Integer equipmentId;

// 文章、动态、评论等ID

@IdEncrypt

Integer entryId;

/\*\*

\* 喜欢内容类型

\* 0文章

\* 1动态

\* 2评论

\*/

Short type;

/\*\*

\* 喜欢操作方式

\* 0 点赞

\* 1 取消点赞

\*/

Short operation;

}

#### BehaviorControllerApi

在类com.heima.behavior.apis.BehaviorControllerApi中增加saveLikesBehavior方法

ResponseResult saveLikesBehavior(LikesBehaviorDto dto);

#### BehaviorController

在com.heima.behavior.controller.v1.BehaviorController类中实现saveLikesBehavior接口方法，调用对应的service接口即可。

@Autowired

private AppLikesBehaviorService appLikesBehaviorService;

@Override

@PostMapping("/like\_behavior")

public ResponseResult saveLikesBehavior(@RequestBody LikesBehaviorDto dto) {

return appLikesBehaviorService.saveLikesBehavior(dto);

}

### 单元测试

此接口就不演示单元测试，后续在集成前端后，通过界面演示。

## 不喜欢接口

### 接口定义

#### 基本定义

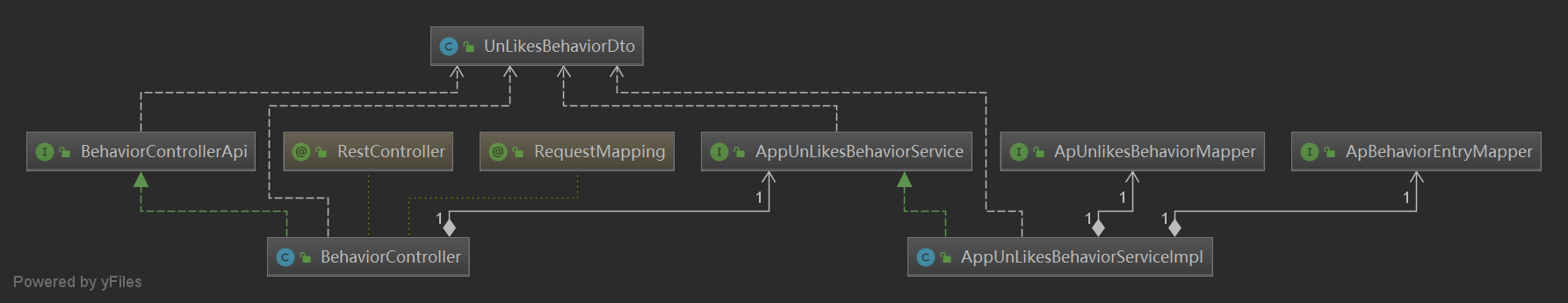
由于框架封装只对JSON反序列化自增ID，需要请求文章ID需要封装为DTO.

|  |  |
| --- | --- |
| 参考标准 | 请参考通用接口规范 |
| 接口名称 | /api/v1/behavior/unlike\_behavior |
| 请求DTO | com.heima.behavior.mysql.core.model.dtos.UnLikesBehaviorDto |
| 响应DTO | 输出插入数据条数的数量 |

#### CODE定义

|  |  |
| --- | --- |
| PARAM\_INVALID | PARAM\_INVALID(501,"无效参数"), |
| PARAM\_REQUIRE | PARAM\_REQUIRE(500,"缺少参数") |

### 类定义



Controller相关的类已创建此增量代码即可，AppUnLikesBehavior\*在behavior模块中定义。

### Mapper实现

#### ApUnlikesBehaviorMapper

在com.heima.article.mysql.core.model.mappers.app.ApUnlikesBehaviorMapper中定义不喜欢数据插入的方法：

int insert(ApUnlikesBehavior record);

#### ApUnlikesBehaviorMapper.xml

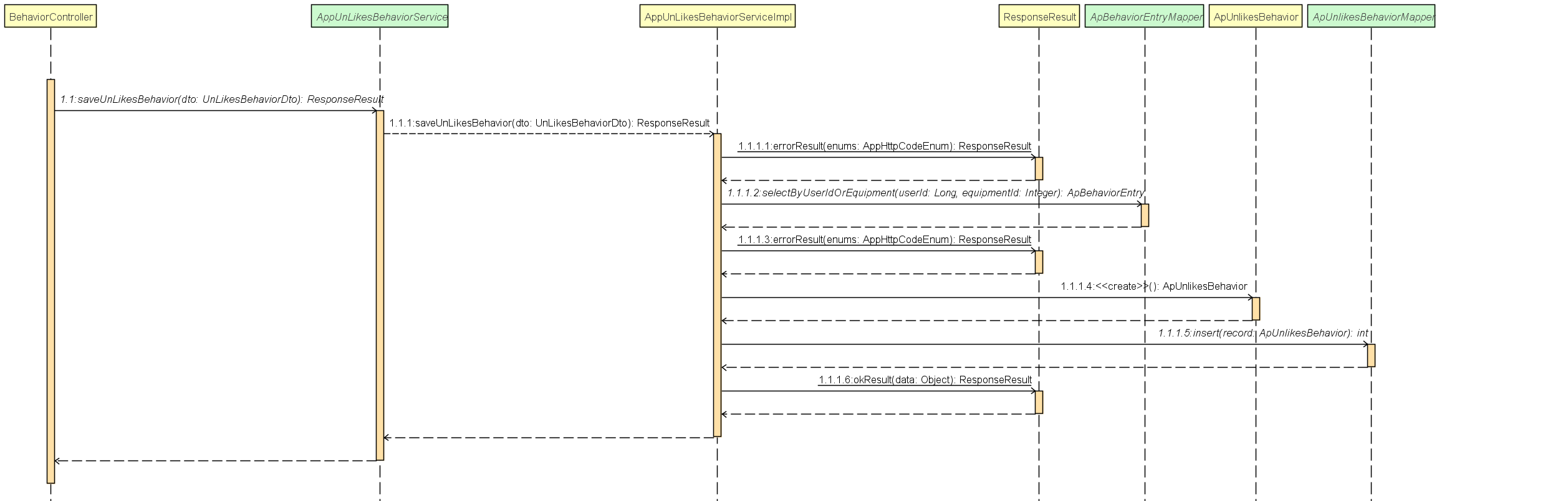
在resources/mappers/app/ApUnlikesBehaviorMapper.xml中，实现对应Mapper的insert方法，此表使用Mycat自增ID，无需插入id字段。

<insert id="insert" parameterType="com.heima.article.mysql.core.model.pojos.app.ApUnlikesBehavior" >

insert into ap\_unlikes\_behavior (entry\_id, article\_id,type, created\_time)values (#{entryId}, #{articleId},#{type,jdbcType=TINYINT}, #{createdTime})

</insert>

### 时序说明



* 1.1.1.1如果行为实体的ID存在，则1.1.1.2查询其对象，否则返回PARAM\_REQUIRE错误
* 1.1.1.3判断行为实体是否存在，如果不存在则返回PARAM\_INVALID错误，否则1.1.1.4创建对象，并在1.1.1.5插入数据库。

### 代码实现

#### AppUnLikesBehaviorService

创建类：com.heima.behavior.service.AppUnLikesBehaviorService

定义不喜欢信息保存方法：

public interface AppUnLikesBehaviorService {

/\*\*

\* 存储不喜欢数据

\* @param dto

\* @return

\*/

public ResponseResult saveUnLikesBehavior(UnLikesBehaviorDto dto);

}

#### AppUnLikesBehaviorServiceImpl

创建类：com.heima.behavior.service.impl.AppUnLikesBehaviorServiceImpl

 实现saveUnLikesBehavior方法，增量插入行为数据。

@Service

public class AppUnLikesBehaviorServiceImpl implements AppUnLikesBehaviorService {

@Autowired

private ApUnlikesBehaviorMapper apUnLikesBehaviorMapper;

@Autowired

private ApBehaviorEntryMapper apBehaviorEntryMapper;

@Override

public ResponseResult saveUnLikesBehavior(UnLikesBehaviorDto dto){

ApUser user = AppThreadLocalUtils.getUser();

// 用户和设备不能同时为空

if(user==null&& dto.getEquipmentId()==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_REQUIRE);

}

Long userId = null;

if(user!=null){

userId = user.getId();

}

ApBehaviorEntry apBehaviorEntry = apBehaviorEntryMapper.selectByUserIdOrEquipment(userId, dto.getEquipmentId());

// 行为实体找以及注册了，逻辑上这里是必定有值得，除非参数错误

if(apBehaviorEntry==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_INVALID);

}

ApUnlikesBehavior alb = new ApUnlikesBehavior();

alb.setEntryId(apBehaviorEntry.getId());

alb.setCreatedTime(new Date());

alb.setArticleId(dto.getArticleId());

alb.setType(dto.getType());

return ResponseResult.okResult(apUnLikesBehaviorMapper.insert(alb));

}

}

#### UnLikesBehaviorDto

创建类：com.heima.behavior.mysql.core.model.dtos.UnLikesBehaviorDto

此类在model模块中创建，定义请求入参，实现如下：

@Data

public class UnLikesBehaviorDto {

// 设备ID

@IdEncrypt

Integer equipmentId;

// 文章ID

@IdEncrypt

Integer articleId;

/\*\*

\* 不喜欢操作方式

\* 0 不喜欢

\* 1 取消不喜欢

\*/

Short type;

}

#### BehaviorControllerApi

在类com.heima.behavior.apis.BehaviorControllerApi中增加saveUnLikesBehavior方法：

ResponseResult saveUnLikesBehavior( UnLikesBehaviorDto dto) ;

#### BehaviorController

在com.heima.behavior.controller.v1.BehaviorController类中实现saveUnLikesBehavior接口方法，调用对应的service接口即可。

private AppUnLikesBehaviorService appUnLikesBehaviorService;

@Override

@PostMapping("/unlike\_behavior")

public ResponseResult saveUnLikesBehavior(@RequestBody UnLikesBehaviorDto dto) {

return appUnLikesBehaviorService.saveUnLikesBehavior(dto);

}

## 阅读接口

通过阅读时间、阅读百分比等数据能够进一步计算出用户对文章的喜爱程度，主要数据由前端收集，后端进行相关的存储。

在实际业务中，用户如果再次点开文章阅读，此数据还可以用来恢复上次用户阅读位置，提升产品体验，此场景功能的实现了解一下，本项目中不进行实现。

### 接口定义

#### 基本定义

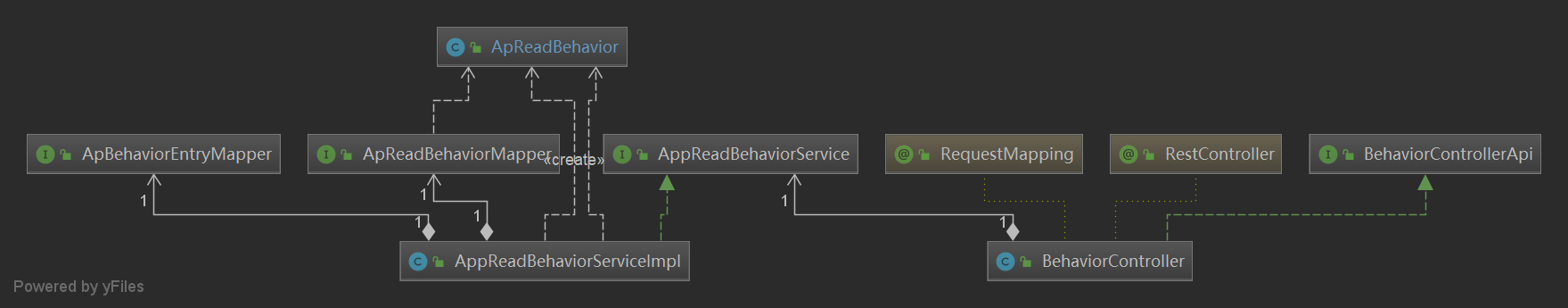
由于框架封装只对JSON反序列化自增ID，需要请求文章ID需要封装为DTO.

|  |  |
| --- | --- |
| 参考标准 | 请参考通用接口规范 |
| 接口名称 | /api/v1/behavior/read\_behavior |
| 请求DTO | com.heima.behavior.mysql.core.model.dtos.ReadBehaviorDto |
| 响应DTO | 输出插入数据条数的数量 |

#### CODE定义

|  |  |
| --- | --- |
| PARAM\_INVALID | PARAM\_INVALID(501,"无效参数"), |
| PARAM\_REQUIRE | PARAM\_REQUIRE(500,"缺少参数") |

### 类定义



Controller相关的类已创建此增量代码即可，AppReadBehavior\*在behavior模块中定义。

### Mapper实现

#### ApReadBehavior

创建类com.heima.article.mysql.core.model.pojos.app.ApReadBehavior

生成的ApReadBehavior注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。另外注意articleId、entryId需要增加@IdEncrypt注解，以作输出混淆，burst字段需要过滤输出。

@Data  
public class ApReadBehavior {  
 private Long id;  
 @IdEncrypt  
 private Integer entryId;  
 @IdEncrypt  
 private Integer articleId;  
 private Short count;  
 private Integer readDuration;  
 private Short percentage;  
 private Short loadDuration;  
 private Date createdTime;  
 private Date updatedTime;  
 @JsonIgnore  
 private String burst;  
}

#### ApReadBehaviorMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApReadBehaviorMapper

定义插入、更新、查询三个方法：

public interface ApReadBehaviorMapper {

int insert(ApReadBehavior record);

int update(ApReadBehavior record);

ApReadBehavior selectByEntryId(String burst,Integer entryId,Integer articleId);

}

#### ApReadBehaviorMapper.xml

创建文件resources/mappers/app/ApReadBehaviorMapper.xml

ApReadBehavior是按照burst字段进行分库分表，查询、更新时注意使用Mycat注解确定路由DN，插入时需在程序中生成ID，SQL如下：

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApReadBehaviorMapper" >

<resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApReadBehavior" >

<id column="id" property="id" />

<result column="entry\_id" property="entryId" />

<result column="article\_id" property="articleId" />

<result column="count" property="count"/>

<result column="read\_duration" property="readDuration" />

<result column="percentage" property="percentage"/>

<result column="load\_duration" property="loadDuration"/>

<result column="created\_time" property="createdTime" />

<result column="updated\_time" property="updatedTime" />

<result column="burst" property="burst" />

</resultMap>

<sql id="Base\_Column\_List" >

id, entry\_id, article\_id, count, read\_duration, percentage, load\_duration, created\_time,updated\_time,burst

</sql>

<!-- 使用注解方式指明SQL执行的DN -->

<select id="selectByEntryId" resultMap="BaseResultMap">

/\*!mycat:sql=select id from ap\_read\_behavior where burst='${burst}'\*/

select <include refid="Base\_Column\_List" /> from ap\_read\_behavior where entry\_id = #{entryId} and article\_id=#{articleId}

</select>

<insert id="insert" parameterType="com.heima.article.mysql.core.model.pojos.app.ApReadBehavior" >

insert into ap\_read\_behavior (id, entry\_id, article\_id, count, read\_duration, percentage, load\_duration, created\_time,updated\_time,burst)

values (#{id}, #{entryId}, #{articleId},

#{count,jdbcType=TINYINT}, #{readDuration}, #{percentage,jdbcType=TINYINT},

#{loadDuration,jdbcType=TINYINT}, #{createdTime}, #{updatedTime},#{burst})

</insert>

<update id="update" parameterType="com.heima.article.mysql.core.model.pojos.app.ApReadBehavior" >

/\*!mycat:sql=select id from ap\_read\_behavior where burst='${burst}'\*/

update ap\_read\_behavior

<set >

<if test="readDuration != null" >

read\_duration = read\_duration + #{readDuration},

</if>

<if test="percentage != null" >

percentage = GREATEST(percentage,#{percentage}),

</if>

<if test="loadDuration != null" >

load\_duration = #{loadDuration},

</if>

<if test="count != null" >

count = count+1,

</if>

<if test="updatedTime != null" >

updated\_time = #{updatedTime},

</if>

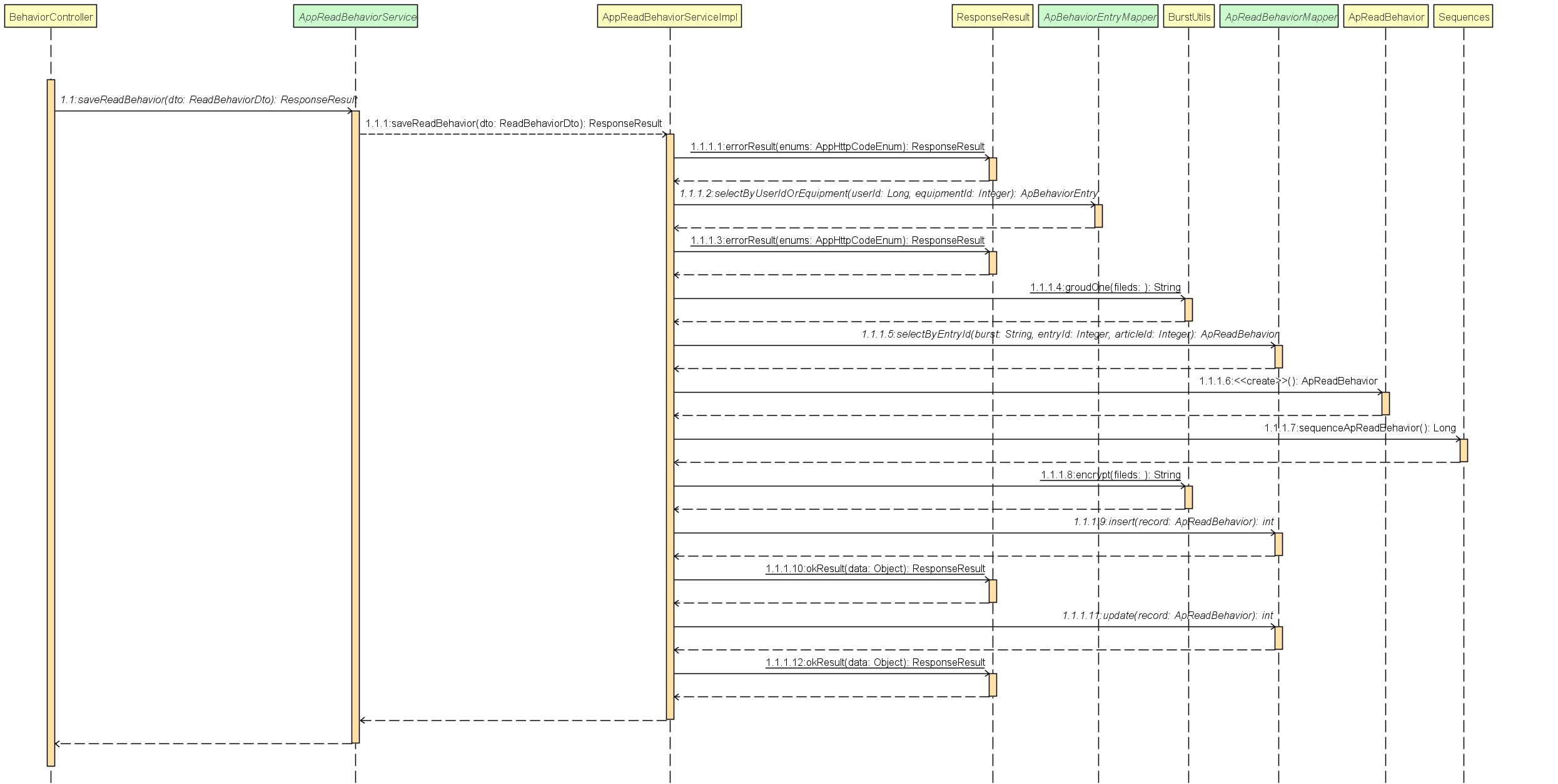
</set>

where entry\_id = #{entryId} and article\_id = #{articleId} and burst=#{burst}

</update>

</mapper>

### 时序说明



* 1.1.1.1如果行为实体的ID存在，则1.1.1.2查询其对象，否则返回PARAM\_REQUIRE错误
* 1.1.1.3判断行为实体是否存在，如果不存在则返回PARAM\_INVALID错误
* 1.1.1.5查询是否已经存在阅读行为，如果不存在则创建并插入1.1.1.9，并返回插入成功数量；如果存在则1.1.1.11更新阅读数据，返回更新成功的数量

### 代码实现

#### AppReadBehaviorService

创建类：com.heima.behavior.service.AppReadBehaviorService定义接口saveReadBehavior方法：

public interface AppReadBehaviorService {

/\*\*

\* 存储阅读数据

\* @param dto

\* @return

\*/

public ResponseResult saveReadBehavior(ReadBehaviorDto dto);

}

#### AppReadBehaviorServiceImpl

创建类：com.heima.behavior.service.impl.AppReadBehaviorServiceImpl，并实现

 saveReadBehavior方法。

阅读行为可能发生多次，第一次是新增数据，第二次是更新阅读时间，最大阅读比例，阅读次数等信息。

@Service

public class AppReadBehaviorServiceImpl implements AppReadBehaviorService {

@Autowired

private ApReadBehaviorMapper apReadBehaviorMapper;

@Autowired

private ApBehaviorEntryMapper apBehaviorEntryMapper;

@Autowired

private Sequences sequences;

@Override

public ResponseResult saveReadBehavior(ReadBehaviorDto dto){

ApUser user = AppThreadLocalUtils.getUser();

// 用户和设备不能同时为空

if(user==null&& dto.getEquipmentId()==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_REQUIRE);

}

Long userId = null;

if(user!=null){

userId = user.getId();

}

ApBehaviorEntry apBehaviorEntry = apBehaviorEntryMapper.selectByUserIdOrEquipment(userId, dto.getEquipmentId());

// 行为实体找以及注册了，逻辑上这里是必定有值得，除非参数错误

if(apBehaviorEntry==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_INVALID);

}

ApReadBehavior alb = apReadBehaviorMapper.selectByEntryId(BurstUtils.groudOne(apBehaviorEntry.getId()),apBehaviorEntry.getId(),dto.getArticleId());

boolean isInsert = false;

if(alb==null){

alb = new ApReadBehavior();

alb.setId(sequences.sequenceApReadBehavior());

isInsert = true;

}

alb.setEntryId(apBehaviorEntry.getId());

alb.setCount(dto.getCount());

alb.setPercentage(dto.getPercentage());

alb.setArticleId(dto.getArticleId());

alb.setLoadDuration(dto.getLoadDuration());

alb.setReadDuration(dto.getReadDuration());

alb.setCreatedTime(new Date());

alb.setUpdatedTime(new Date());

alb.setBurst(BurstUtils.encrypt(alb.getId(),alb.getEntryId()));

// 插入

if(isInsert){

return ResponseResult.okResult(apReadBehaviorMapper.insert(alb));

}else {

// 更新

return ResponseResult.okResult(apReadBehaviorMapper.update(alb));

}

}

}

#### ReadBehaviorDto

创建类：com.heima.behavior.mysql.core.model.dtos.ReadBehaviorDto

此类在model模块中创建，定义请求入参，实现如下：

@Data

public class ReadBehaviorDto {

// 设备ID

@IdEncrypt

Integer equipmentId;

// 文章、动态、评论等ID

@IdEncrypt

Integer articleId;

/\*\*

\* 阅读次数

\*/

Short count;

/\*\*

\* 阅读时长（S)

\*/

Integer readDuration;

/\*\*

\* 阅读百分比，0~100

\*/

Short percentage;

/\*\*

\* 加载时间

\*/

Short loadDuration;

}

#### BehaviorControllerApi

在类com.heima.behavior.apis.BehaviorControllerApi中增加saveReadBehavior方法

ResponseResult saveReadBehavior( ReadBehaviorDto dto);

#### BehaviorController

在com.heima.behavior.controller.v1.BehaviorController类中实现saveReadBehavior接口方法，调用对应的service接口即可。

@Autowired

private AppReadBehaviorService appReadBehaviorService;

@Override

@PostMapping("/read\_behavior")

public ResponseResult saveReadBehavior(@RequestBody ReadBehaviorDto dto) {

return appReadBehaviorService.saveReadBehavior(dto);

}

### 单元测试

此接口就不演示单元测试，后续在集成前端后，通过界面演示。

## 分享接口

文章分享可分享到微信、微信朋友圈、QQ等通道，且一个通道可多次分享，因此分享行为数据是一个增量。在正常的流程中分享行为应该是在分享成功的回调函数中调用，在此项目中不实际对接三方分享通道，因此提供接口给前端直接调用生成数据，为后期文章推荐算法提供数据支持。

### 接口定义

#### 基本定义

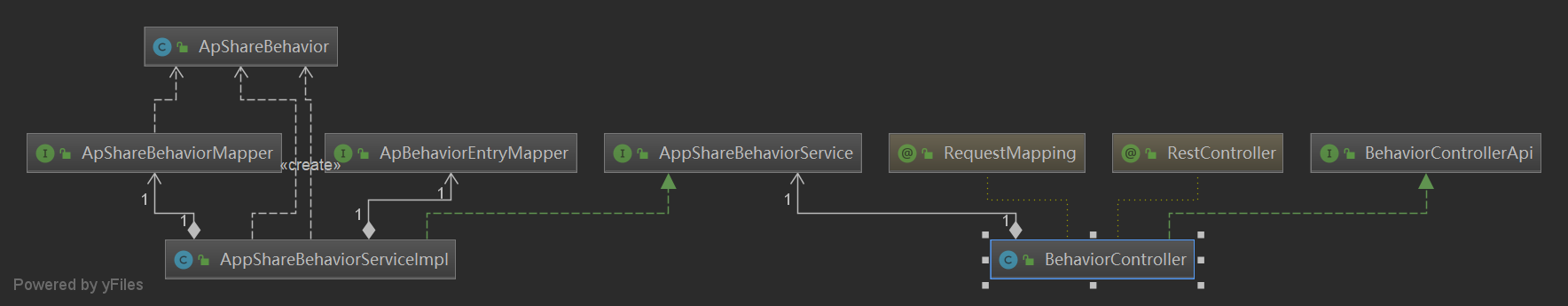
由于框架封装只对JSON反序列化自增ID，需要请求文章ID需要封装为DTO.

|  |  |
| --- | --- |
| 参考标准 | 请参考通用接口规范 |
| 接口名称 | /api/v1/behavior/share\_behavior |
| 请求DTO | com.heima.behavior.mysql.core.model.dtos.ShareBehaviorDto |
| 响应DTO | 输出插入数据条数的数量 |

#### CODE定义

|  |  |
| --- | --- |
| PARAM\_INVALID | PARAM\_INVALID(501,"无效参数"), |
| PARAM\_REQUIRE | PARAM\_REQUIRE(500,"缺少参数") |

### 类定义



Controller相关的类已创建此增量代码即可，AppShareBehavior\*在behavior模块中定义。

### Mapper实现

#### ApShareBehavior

创建类com.heima.article.mysql.core.model.pojos.app.ApShareBehavior

生成的ApShareBehavior注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。另外注意articleId、entryId需要增加@IdEncrypt注解，以作输出混淆。

@Data  
public class ApShareBehavior {  
 private Long id;  
 @IdEncrypt  
 private Integer entryId;  
 @IdEncrypt  
 private Integer articleId;  
 private Short type;  
 private Date createdTime;  
}

#### ApShareBehaviorMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApShareBehaviorMapper

定义分享行为插入方法：

public interface ApShareBehaviorMapper {

int insert(ApShareBehavior record);

}

#### ApShareBehaviorMapper.xml

创建文件resources/mappers/app/ApShareBehaviorMapper.xml

ApShareBehavior是按照entry\_id字段进行分库分表，由Mycat管理自增主键，SQL如下：

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApShareBehaviorMapper" >

<resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApShareBehavior" >

<id column="id" property="id" />

<result column="entry\_id" property="entryId" />

<result column="article\_id" property="articleId" />

<result column="type" property="type"/>

<result column="created\_time" property="createdTime" />

</resultMap>

<sql id="Base\_Column\_List" >

id, entry\_id, article\_id, type, created\_time

</sql>

<insert id="insert" parameterType="com.heima.article.mysql.core.model.pojos.app.ApShareBehavior" >

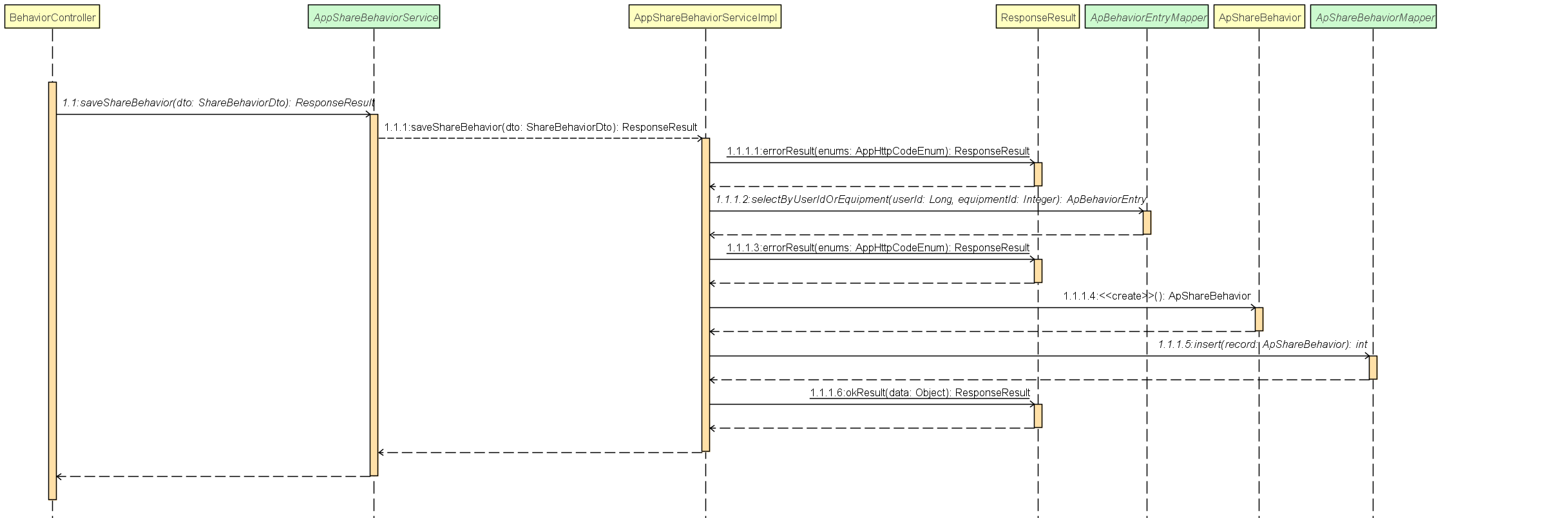
insert into ap\_share\_behavior (entry\_id, article\_id,type, created\_time)

values (#{entryId}, #{articleId},#{type,jdbcType=TINYINT}, #{createdTime})

</insert>

</mapper>

### 时序说明



* 1.1.1.1如果行为实体的ID存在，则1.1.1.2查询其对象，否则返回PARAM\_REQUIRE错误
* 1.1.1.3判断行为实体是否存在，如果不存在则返回PARAM\_INVALID错误，否则1.1.1.4创建对象，并在1.1.1.5插入数据库。

### 代码实现

#### AppShareBehaviorService

创建类：com.heima.behavior.service.AppShareBehaviorService

定义分享行为数据存储接口：

public interface AppShareBehaviorService {

/\*\*

\* 存储分享数据

\* @param dto

\* @return

\*/

public ResponseResult saveShareBehavior(ShareBehaviorDto dto);

}

#### AppShareBehaviorServiceImpl

创建类com.heima.behavior.service.impl.AppShareBehaviorServiceImpl，并实现saveShareBehavior方法，这里的分析行为数据是增量新增。

@Service

public class AppShareBehaviorServiceImpl implements AppShareBehaviorService {

@Autowired

private ApShareBehaviorMapper apShareBehaviorMapper;

@Autowired

private ApBehaviorEntryMapper apBehaviorEntryMapper;

@Override

public ResponseResult saveShareBehavior(ShareBehaviorDto dto){

ApUser user = AppThreadLocalUtils.getUser();

// 用户和设备不能同时为空

if(user==null&& dto.getEquipmentId()==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_REQUIRE);

}

Long userId = null;

if(user!=null){

userId = user.getId();

}

ApBehaviorEntry apBehaviorEntry = apBehaviorEntryMapper.selectByUserIdOrEquipment(userId, dto.getEquipmentId());

// 行为实体找以及注册了，逻辑上这里是必定有值得，除非参数错误

if(apBehaviorEntry==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_INVALID);

}

ApShareBehavior alb = new ApShareBehavior();

alb.setEntryId(apBehaviorEntry.getId());

alb.setCreatedTime(new Date());

alb.setArticleId(dto.getArticleId());

alb.setType(dto.getType());

return ResponseResult.okResult(apShareBehaviorMapper.insert(alb));

}

}

#### ShareBehaviorDto

创建类：com.heima.behavior.mysql.core.model.dtos.ShareBehaviorDto

此类在model模块中创建，定义请求入参，实现如下：

@Data

public class ShareBehaviorDto {

// 设备ID

@IdEncrypt

Integer equipmentId;

// 文章ID

@IdEncrypt

Integer articleId;

/\*\*

\* 分享渠道

\* 0 微信

\* 1 微信朋友圈

\* 2 QQ

\* 3 QQ 空间

\* 4 微博

\*/

Short type;

}

#### BehaviorControllerApi

在类com.heima.behavior.apis.BehaviorControllerApi中增加saveShareBehavior方法

ResponseResult saveShareBehavior( ShareBehaviorDto dto);

#### BehaviorController

在com.heima.behavior.controller.v1.BehaviorController类中实saveShareBehavior接口方法，调用对应的service接口即可。

@Autowired

private AppShareBehaviorService appShareBehaviorService;

@Override

@PostMapping("/share\_behavior")

public ResponseResult saveShareBehavior(@RequestBody ShareBehaviorDto dto) {

return appShareBehaviorService.saveShareBehavior(dto);

}

### 单元测试

此接口就不演示单元测试，后续在集成前端后，通过界面演示。

## 收藏接口

收藏一般是登录用户的特有功能，但是头条也支持未登录用户的文章推荐，此时是依靠设备ID进行实现的，所以此功能放置了behavior模块中。另外收藏的内容用户在APP端是可访问查询的，相关查询接口应定义在user模块中。

### 接口定义

#### 基本定义

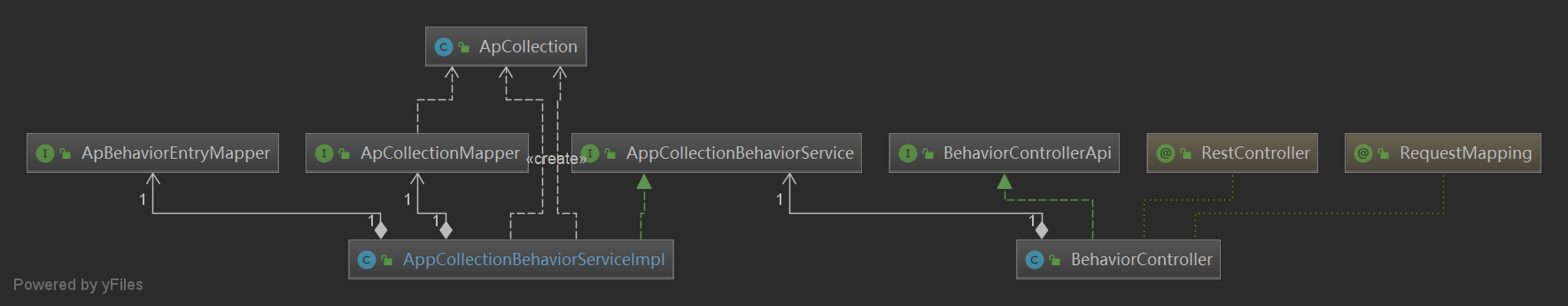
由于框架封装只对JSON反序列化自增ID，需要请求文章ID需要封装为DTO.

|  |  |
| --- | --- |
| 参考标准 | 请参考通用接口规范 |
| 接口名称 | /api/v1/behavior/ collection\_behavior |
| 请求DTO | com.heima.behavior.mysql.core.model.dtos.CollectionBehaviorDto |
| 响应DTO | 输出插入数据条数的数量 |

#### CODE定义

|  |  |
| --- | --- |
| PARAM\_INVALID | PARAM\_INVALID(501,"无效参数"), |
| PARAM\_REQUIRE | PARAM\_REQUIRE(500,"缺少参数") |
| DATA\_NOT\_EXIST | DATA\_NOT\_EXIST(1000,"数据不存在") |
| DATA\_EXIST | DATA\_EXIST(1000,"数据已经存在") |

### 类定义



Controller相关的类已创建此增量代码即可，AppCollectionBehavior\*在behavior模块中定义。

### Mapper实现

#### ApCollectionMapper

在com.heima.article.mysql.core.model.mappers.app. ApCollectionMapper中定义关注信息插入（收藏时调用）和删除（取消收藏时）的方法：

int insert(ApCollection record);

int deleteById(String burst,Long id);

#### ApCollectionMapper.xml

在resources/mappers/app/ApCollectionMapper.xml中，实现对应Mapper方法，注意以下两点：

* 删除时，使用Mycat注解确定数据路由DN
* 插入时，需插入ID注解，并在程序中调用Sequences生产ZK自增ID

<delete id="deleteById">

/\*!mycat:sql=select id from ap\_collection where burst='${burst}'\*/

delete from ap\_collection where id=#{id}

</delete>

<insert id="insert" parameterType="com.heima.article.mysql.core.model.pojos.app.ApCollection" >

insert into ap\_collection (id, behavior\_entry\_id, entry\_id,

type, collection\_time, published\_time,burst

)

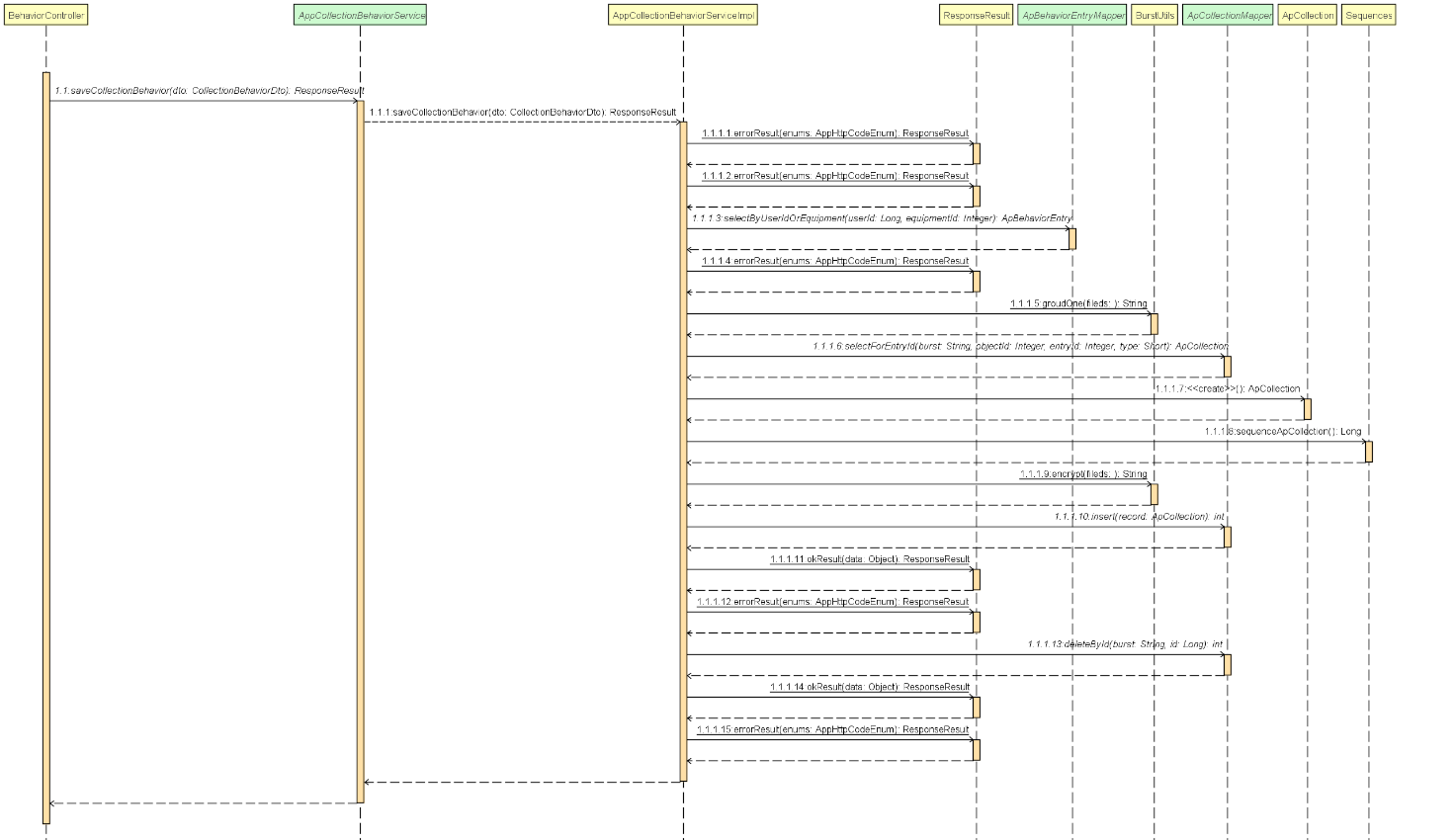
values (#{id}, #{behaviorEntryId}, #{entryId},

#{type}, #{collectionTime}, #{publishedTime},#{burst}

)

</insert>

### 时序说明



* 1.1.1.1如果行为实体的ID存在，且1.1.12operation参数合法，则1.1.1.3查询其对象，否则返回PARAM\_REQUIRE错误
* 1.1.1.4判断行为实体是否存在，如果不存在则返回PARAM\_INVALID错误，否则查询当前收藏数据1.1.16
* 1.1.1.7如果是收藏操作，且不存在收藏数据则创建并1.1.1.10插入收藏数据，最后返回成功
* 1.1.1.12如果是收藏操作，但是存在了收藏数据则返回DATA\_EXIST错误
* 1.1.1.13如果是取消收藏操作，且存在收藏数据，则删除收藏数据，并返回成功
* 1.1.1.15如果是取消操作，但不存在收藏数据，则返回DATA\_NOT\_EXIST错误

### 代码实现

#### AppCollectionBehaviorService

创建类：com.heima.behavior.service.AppCollectionBehaviorService

定义收藏行为接口：

public interface AppCollectionBehaviorService {

public ResponseResult saveCollectionBehavior(CollectionBehaviorDto dto);

}

#### AppCollectionBehaviorServiceImpl

创建类：com.heima.behavior.service.impl.AppCollectionBehaviorServiceImpl，并实现saveCollectionBehavior方法，注意不同操作类型下的处理逻辑：

@Service

public class AppCollectionBehaviorServiceImpl implements AppCollectionBehaviorService {

@Autowired

private ApCollectionMapper apCollectionMapper;

@Autowired

private ApBehaviorEntryMapper apBehaviorEntryMapper;

@Autowired

private Sequences sequences;

@Override

public ResponseResult saveCollectionBehavior(CollectionBehaviorDto dto){

ApUser user = AppThreadLocalUtils.getUser();

// 用户和设备不能同时为空

if(user==null&& dto.getEquipmentId()==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_REQUIRE);

}

Long userId = null;

if(user!=null){

userId = user.getId();

}

// 操作类型错误

if(dto.getOperation()==null||dto.getOperation()<0||dto.getOperation()>1){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_INVALID,"无效的operation");

}

ApBehaviorEntry apBehaviorEntry = apBehaviorEntryMapper.selectByUserIdOrEquipment(userId, dto.getEquipmentId());

// 行为实体找以及注册了，逻辑上这里是必定有值得，除非参数错误

if(apBehaviorEntry==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_INVALID);

}

ApCollection alb = apCollectionMapper.selectForEntryId(BurstUtils.groudOne(apBehaviorEntry.getId()),apBehaviorEntry.getId(),dto.getEntryId(),dto.getType());

// 收藏

if(dto.getOperation()==0) {

if (alb == null) {

alb = new ApCollection();

alb.setId(sequences.sequenceApCollection());

alb.setBehaviorEntryId(apBehaviorEntry.getId());

alb.setCollectionTime(new Date());

alb.setPublishedTime(dto.getPublishedTime());

alb.setEntryId(dto.getEntryId());

alb.setType(dto.getType());

alb.setBurst(BurstUtils.encrypt(alb.getId(), alb.getBehaviorEntryId()));

return ResponseResult.okResult(apCollectionMapper.insert(alb));

} else {

return ResponseResult.errorResult(AppHttpCodeEnum.DATA\_EXIST);

}

}else{

// 取消收藏

if(alb!=null){

return ResponseResult.okResult(apCollectionMapper.deleteById(alb.getBurst(),alb.getId()));

}else{

return ResponseResult.errorResult(AppHttpCodeEnum.DATA\_NOT\_EXIST);

}

}

}

}

#### CollectionBehaviorDto

创建类：com.heima.behavior.mysql.core.model.dtos.CollectionBehaviorDto

此类在model模块中创建，定义请求入参，实现如下：

@Data

public class CollectionBehaviorDto {

// 设备ID

@IdEncrypt

Integer equipmentId;

// 文章、动态ID

@IdEncrypt

Integer entryId;

/\*\*

\* 收藏内容类型

\* 0文章

\* 1动态

\*/

Short type;

/\*\*

\* 操作类型

\* 0收藏

\* 1取消收藏

\*/

Short operation;

Date publishedTime;

}

#### BehaviorControllerApi

在类com.heima.behavior.apis.BehaviorControllerApi中增加saveLikesBehavior方法

ResponseResult saveCollectionBehavior( CollectionBehaviorDto dto) ;

#### BehaviorController

在com.heima.behavior.controller.v1.BehaviorController类中实现saveCollectionBehavior接口方法，调用对应的service接口即可。

@Autowired

private AppCollectionBehaviorService appCollectionBehaviorService;

@Override

@PostMapping("/collection\_behavior")

public ResponseResult saveCollectionBehavior(@RequestBody CollectionBehaviorDto dto) {

return appCollectionBehaviorService.saveCollectionBehavior(dto);

}

### 单元测试

此接口就不演示单元测试，后续在集成前端后，通过界面演示。

## 转发接口

一般转发流程是：

* 点击转发按钮，进入到转发内容编辑页面（可输入转发内容）
* 点击转发保存按钮，后端创建转发动态内容，并存储转发转发行为
* 后端通知粉丝相关转发动态

在这里不实现以上流程功能，简化实现转发行为数据的记录功能，为之后文章推荐算法提供数据支持。另外此处限定一个文章只能被一个用户转发一次，避免重复转发。

### 接口定义

#### 基本定义

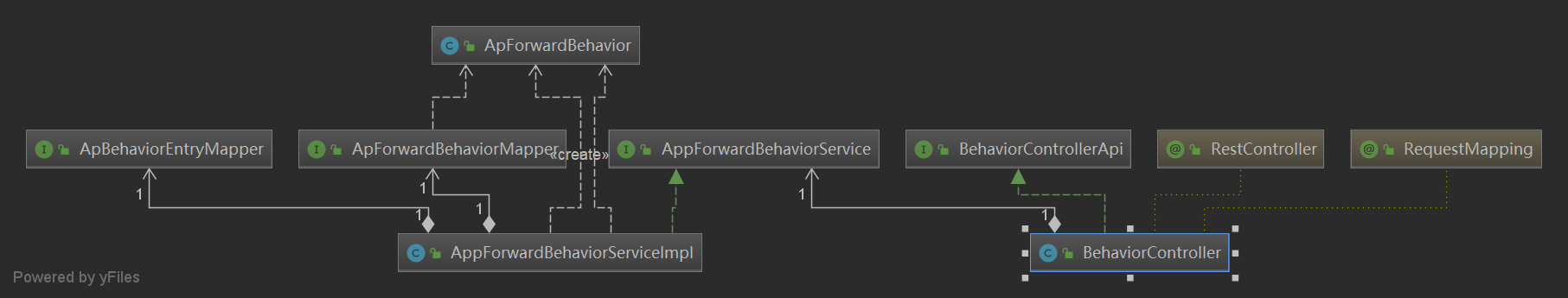
由于框架封装只对JSON反序列化自增ID，需要请求文章ID需要封装为DTO.

|  |  |
| --- | --- |
| 参考标准 | 请参考通用接口规范 |
| 接口名称 | /api/v1/behavior/forward\_behavior |
| 请求DTO | com.heima.behavior.mysql.core.model.dtos.ForwardBehaviorDto |
| 响应DTO | 输出插入数据条数的数量 |

#### CODE定义

|  |  |
| --- | --- |
| PARAM\_INVALID | PARAM\_INVALID(501,"无效参数"), |
| PARAM\_REQUIRE | PARAM\_REQUIRE(500,"缺少参数") |
| DATA\_EXIST | DATA\_EXIST(1000,"数据已经存在") |

### 类定义



Controller相关的类已创建此增量代码即可，AppForwardBehavior\*在behavior模块中定义。

### Mapper实现

#### ApForwardBehavior

创建类com.heima.article.mysql.core.model.pojos.app.ApForwardBehavior

生成的ApForwardBehavior注释和get方法可以删除，然后使用lombok @Data注解，优雅的实现pojo方法。另外注意entryId需要增加@IdEncrypt注解，以作输出混淆。

@Data  
public class ApForwardBehavior {  
 private Long id;  
 @IdEncrypt  
 private Integer entryId;  
 private Integer articleId;  
 private Integer dynamicId;  
 private Date createdTime;  
}

#### ApForwardBehaviorMapper

创建类com.heima.article.mysql.core.model.mappers.app.ApForwardBehaviorMapper

定义插入和查询转发记录的方法：

public interface ApForwardBehaviorMapper {  
 int insert(ApForwardBehavior record);  
 ApForwardBehavior selectByApForwardBehavior(Integer entryId,Integer articleId,Integer dynamicId);  
}

#### ApForwardBehaviorMapper.xml

创建文件resources/mappers/app/ApForwardBehaviorMapper.xml

ApForwardBehavior是按照entry\_id字段进行分库分表，由Mycat管理自增主键，SQL如下：

<mapper namespace="com.heima.article.mysql.core.model.mappers.app.ApForwardBehaviorMapper" >

<resultMap id="BaseResultMap" type="com.heima.article.mysql.core.model.pojos.app.ApForwardBehavior" >

<id column="id" property="id" />

<result column="entry\_id" property="entryId" />

<result column="article\_id" property="articleId" />

<result column="dynamic\_id" property="dynamicId" />

<result column="created\_time" property="createdTime" />

</resultMap>

<sql id="Base\_Column\_List" >

id, entry\_id, article\_id, dynamic\_id, created\_time

</sql>

<select id="selectByApForwardBehavior" resultMap="BaseResultMap" parameterType="com.heima.article.mysql.core.model.pojos.app.ApForwardBehavior" >

select <include refid="Base\_Column\_List" /> from ap\_forward\_behavior where entry\_id = #{entryId}

<if test="articleId!=null">

and article\_id=#{articleId}

</if>

<if test="dynamicId!=null">

and dynamic\_id=#{dynamicId}

</if>

</select>

<insert id="insert" parameterType="com.heima.article.mysql.core.model.pojos.app.ApForwardBehavior" >

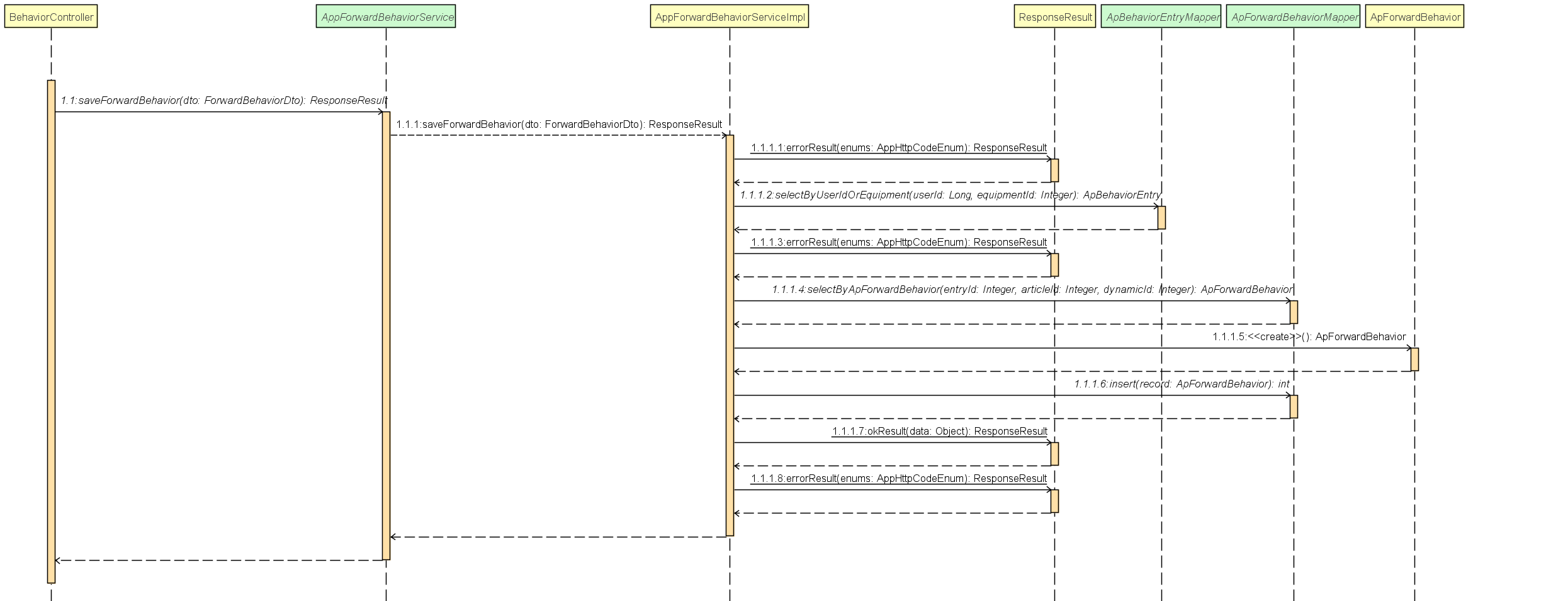
insert into ap\_forward\_behavior (entry\_id, article\_id, dynamic\_id, created\_time)

values (#{entryId}, #{articleId}, #{dynamicId}, #{createdTime})

</insert>

</mapper>

### 时序说明



* 1.1.1.1如果行为实体的ID存在，则1.1.1.2查询其对象，否则返回PARAM\_REQUIRE错误
* 1.1.1.3判断行为实体是否存在，如果不存在则返回PARAM\_INVALID错误，否则1.1.1.4查询转发数据，如果未存在则1.1.1.6插入数据，并返回成功，否则返回1.1.1.8 DATA\_EXIST错误。

### 代码实现

#### AppForwardBehaviorService

创建类com.heima.behavior.service.AppForwardBehaviorService定义转发行为存储接口：

public interface AppForwardBehaviorService {

/\*\*

\* 存储转发数据

\* @param dto

\* @return

\*/

public ResponseResult saveForwardBehavior(ForwardBehaviorDto dto);

}

#### AppForwardBehaviorServiceImpl

创建类：com.heima.behavior.service.impl.AppForwardBehaviorServiceImpl，并实现saveForwardBehavior方法：

@Service

public class AppForwardBehaviorServiceImpl implements AppForwardBehaviorService {

@Autowired

private ApForwardBehaviorMapper apForwardBehaviorMapper;

@Autowired

private ApBehaviorEntryMapper apBehaviorEntryMapper;

@Autowired

private Sequences sequences;

@Override

public ResponseResult saveForwardBehavior(ForwardBehaviorDto dto){

ApUser user = AppThreadLocalUtils.getUser();

// 用户和设备不能同时为空

if(user==null&& dto.getEquipmentId()==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_REQUIRE);

}

Long userId = null;

if(user!=null){

userId = user.getId();

}

ApBehaviorEntry apBehaviorEntry = apBehaviorEntryMapper.selectByUserIdOrEquipment(userId, dto.getEquipmentId());

// 行为实体找以及注册了，逻辑上这里是必定有值得，除非参数错误

if(apBehaviorEntry==null){

return ResponseResult.errorResult(AppHttpCodeEnum.PARAM\_INVALID);

}

ApForwardBehavior temp = apForwardBehaviorMapper.selectByApForwardBehavior(apBehaviorEntry.getId(),dto.getArticleId(),dto.getDynamicId());

if(temp==null) {

ApForwardBehavior alb = new ApForwardBehavior();

alb.setEntryId(apBehaviorEntry.getId());

alb.setCreatedTime(new Date());

alb.setArticleId(dto.getArticleId());

alb.setDynamicId(dto.getDynamicId());

return ResponseResult.okResult(apForwardBehaviorMapper.insert(alb));

}else{

return ResponseResult.errorResult(AppHttpCodeEnum.DATA\_EXIST);

}

}

}

#### ForwardBehaviorDto

创建类：com.heima.behavior.mysql.core.model.dtos.ForwardBehaviorDto

此类在model模块中创建，定义请求入参，实现如下：

@Data

public class ForwardBehaviorDto {

// 设备ID

@IdEncrypt

Integer equipmentId;

// 文章ID

@IdEncrypt

Integer articleId;

@IdEncrypt

Integer dynamicId;

}

#### BehaviorControllerApi

在类com.heima.behavior.apis.BehaviorControllerApi中增加saveForwardBehavior方法

ResponseResult saveForwardBehavior( ForwardBehaviorDto dto);

#### BehaviorController

在com.heima.behavior.controller.v1.BehaviorController类中实现saveForwardBehavior接口方法，调用对应的service接口即可。

@Autowired

private AppForwardBehaviorService appForwardBehaviorService;

@Override

@PostMapping("/forward\_behavior")

public ResponseResult saveForwardBehavior(@RequestBody ForwardBehaviorDto dto) {

return appForwardBehaviorService.saveForwardBehavior(dto);

}

### 单元测试

此接口就不演示单元测试，后续在集成前端后，通过界面演示。

## 后端开发总结

### 开发原则

* 自顶向下的设计原则：功能应该从表现层分析再到控制层、服务层、持久层逐层设计
* 自底向上的开发原则：上层需调用下层，因此开发应从底层向上层逐层开发

项目中开发的层次次序参考DB->中间件->持久层->服务层->控制层

* 单一职责的开发原则：类或者方法提供的功能应该单一明确，特别越底层越应单一职责，以便维护

项目中Mapper方法必须功能单一，参数明确，拒绝两种以上的持久逻辑使用同一个Mapper方法

* 依赖倒置的开发原则：上层依赖下层，是依赖下层接口，并不是依赖下层的实现

项目中每层都是通过接口调用Controller->Service->Mapper

### 开发步骤

* 明确类定义：明确哪些是重用类，哪些是需要新增的类
* 明确主键规则：确认操作表的ID生成规则，是Mycat主键，还是Zk主键
* Mapper实现：查、改、删时注意是否使用mycat注解确认DN，插入时是否要插入主键id
* Service实现：可用通过时序图帮助我们梳理实现逻辑
* ControllerApi定义
* Controller实现：简单的Service层调用
* 单元测试

### 开发优化

* 通过观察可发现获取行为实体的代码较为通用，可以抽象出功能方法，放置在一个BaseServiceImpl类中，相关Service都继承这个类。在这里就不做相关代码演示。
* 通过代码观察，了解到对于代码的异常没做过多的异常处理，这部分内容将在后续进行统一拦截演示。
* 通过代码观察，了解到目前写代码的规范没做过多的注重，这部分内将在后续进行统一演示。