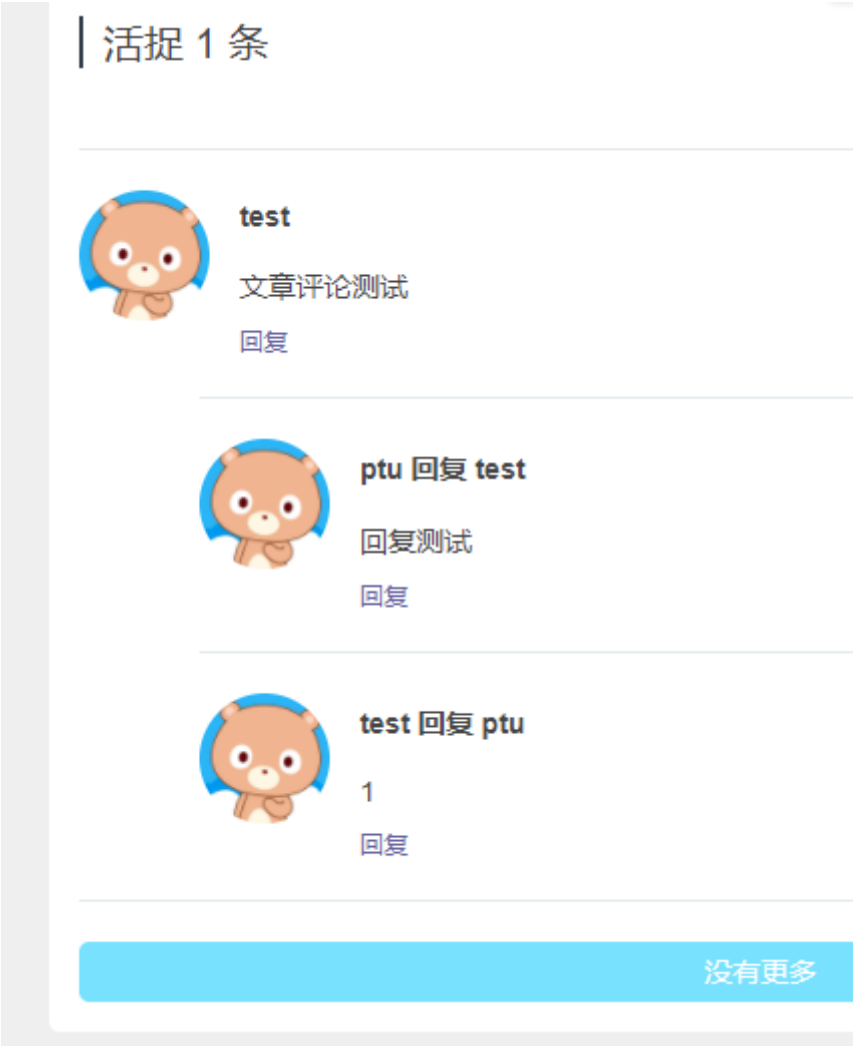
3.10 查询评论列表接口

3.10.1 需求

文章详情页面要展示这篇文章下的评论列表。

效果如下:



3.10.2 评论表分析

通过需求去分析需要有哪些字段。

表结构前面已经给到大家，并且生成了代码

3.10.3 接口设计

请求方式	请求地址	请求头
GET	/comment/commentList	不需要token请求头

Query格式请求参数:

articleId:文章id

pageNum: 页码

pageSize: 每页条数

响应格式:


```
{
    "code": 200,
    "data": {
        "rows": [
            {
                "articleId": "1",
                "children": [
                    {
                        "articleId": "1",
                        "content": "回复测试",
                        "createBy": "3",
                        "id": "3",
                        "rootId": "2",
                        "toCommentId": "2",
                        "toCommentUserId": "1",
                        "toCommentUserName": "test",
                        "username": "ptu"
                    }
                ],
                "content": "文章评论测试",
                "createBy": "1",
                "id": "2",
                "rootId": "-1",
                "toCommentId": "-1",
                "toCommentUserId": "-1",
                "username": "test"
            }
        ],
        "total": "1"
    },
    "msg": "操作成功"
}
```


3.10.4 代码实现

3.10.4.1 不考虑子评论

CommentController

```
@RestController
@RequestMapping("/comment")
public class CommentController {

    @Autowired
    private ICommentService commentService;

    @GetMapping("/commentList")
    public ResponseResult commentList(Long articleId,Integer pageNum,Integer
pageSize){
        return commentService.commentList(articleId,pageNum,pageSize);
    }
}
```

CommentService

```
public interface ICommentService extends IService<Comment> {

    ResponseResult commentList(Long articleId, Integer pageNum, Integer
pageSize);
}
```

CommentServiceImpl

```
@Service
public class CommentServiceImpl extends ServiceImpl<CommentMapper, Comment>
implements ICommentService {

    @Autowired
    private IUserService userService;

    @Override
    public ResponseResult commentList(Long articleId, Integer pageNum, Integer
pageSize) {

        //查询对应文章的根评论
        LambdaQueryWrapper<Comment> queryWrapper = new LambdaQueryWrapper<>();
        queryWrapper.eq(Comment::getArticleId,articleId)
                    .eq(Comment::getRootId,-1)
                    .orderByAsc(Comment::getCreateTime);
        //分页查询
        Page page = new Page(pageNum, pageSize);
        page(page,queryWrapper);
        List<CommentVo> commentVoList = toCommentVoList(page.getRecords());

        return ResponseResult.okResult(new
PageVo(commentVoList,page.getTotal()));
    }

    private List<CommentVo> toCommentVoList(List<Comment> list) {
```

```
        List<CommentVo> commentVoList = BeanCopyUtils.copyBeanList(list,
CommentVo.class);
        for (CommentVo commentVo : commentVoList) {
            //通过creatyBy查询用户的昵称并赋值
            String nickName =
userService.getById(commentVo.getCreateBy()).getNickName();
            commentVo.setUsername(nickName);
            //通过toCommentUserId查询用户的昵称并赋值
            //如果toCommentUserId不为-1才进行查询
            if (commentVo.getToCommentUserId()!=-1){

 commentVo.setToCommentUserName(userService.getById(commentVo.getToCommentUserId
()).getNickName());
            }
        }
        return commentVoList;
    }
}
```

Bean拷贝工具封装

```
public class BeanCopyUtils {

    private BeanCopyUtils() {
    }

    public static <V> V copyBean(Object source,Class<V> clazz) {
        //创建目标对象
        V result = null;
        try {
            result = clazz.newInstance();
            //实现属性copy
            BeanUtils.copyProperties(source, result);
        } catch (Exception e) {
            e.printStackTrace();
        }
        //返回结果
        return result;
    }
    public static <O,V> List<V> copyBeanList(List<O> list,Class<V> clazz){
        return list.stream()
                .map(o -> copyBean(o, clazz))
                .collect(Collectors.toList());
    }
}
```

CommentVo

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class CommentVo {
    private Long id;
```

```
    //文章id
    private Long articleId;
    //根评论id
    private Long rootId;
    //评论内容
    private String content;
    //所回复的目标评论的userid
    private Long toCommentUserId;
    private String toCommentUserName;
    //回复目标评论id
    private Long toCommentId;

    private Long createBy;

    private Date createTime;

    private String username;
}
```

3.10.4.2 查询子评论

CommentVo在之前的基础上增加了 private List children;

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class CommentVo {
    private Long id;
    //文章id
    private Long articleId;
    //根评论id
    private Long rootId;
    //评论内容
    private String content;
    //所回复的目标评论的userid
    private Long toCommentUserId;
    private String toCommentUserName;
    //回复目标评论id
    private Long toCommentId;

    private Long createBy;

    private Date createTime;

    private String username;

    private List<CommentVo> children;
}
```

CommentServiceImpl

```java
@Service
public class CommentServiceImpl extends ServiceImpl<CommentMapper, Comment>
implements ICommentService {

    @Autowired
    private IUserService userService;

    @Override
    public ResponseResult commentList(Long articleId, Integer pageNum, Integer
pageSize) {

        //查询对应文章的根评论
        LambdaQueryWrapper<Comment> queryWrapper = new LambdaQueryWrapper<>();
        queryWrapper.eq(Comment::getArticleId,articleId)
                    .eq(Comment::getRootId,-1)
                    .eq(Comment::getType,0)
                    .orderByAsc(Comment::getCreateTime);
        //分页查询
        Page page = new Page(pageNum, pageSize);
        page(page,queryWrapper);
        List<CommentVo> commentVoList = toCommentVoList(page.getRecords());

        //查询所有根评论对应的子评论集合，并且赋值给对应的属性
        for (CommentVo commentVo : commentVoList) {
            //查询对应的子评论
            List<CommentVo> children = getChildren(commentVo.getId());
            commentVo.setChildren(children);
        }
        return ResponseResult.okResult(new
PageVo(commentVoList,page.getTotal()));
    }

    /**
     * 根据根评论的id查询所对应的子评论的集合
     * @param id
     * @return
     */
    private List<CommentVo> getChildren(Long id) {
        LambdaQueryWrapper<Comment> queryWrapper = new LambdaQueryWrapper<>();
        queryWrapper.eq(Comment::getRootId,id)
                    .orderByAsc(Comment::getCreateTime);
        List<Comment> comments = list(queryWrapper);
        return toCommentVoList(comments);
    }

    private List<CommentVo> toCommentVoList(List<Comment> list) {
        List<CommentVo> commentVoList = BeanCopyUtils.copyBeanList(list,
CommentVo.class);
        for (CommentVo commentVo : commentVoList) {
            //通过creatyBy查询用户的昵称并赋值
            String nickName =
userService.getById(commentVo.getCreateBy()).getNickName();
            commentVo.setUsername(nickName);
            //通过toCommentUserId查询用户的昵称并赋值
            //如果toCommentUserId不为-1才进行查询
            if (commentVo.getToCommentUserId()!=-1){
```

```
 commentVo.setToCommentUserName(userService.getById(commentVo.getToCommentUserId
()).getNickName());
            }
        }
        return commentVoList;
    }
```

3.11 发表评论接口

3.11.1 需求

用户登录后可以对文章发表评论，也可以对评论进行回复。

用户登录后也可以在友链页面进行评论。

3.11.2 接口设计

请求方式	请求地址	请求头
POST	/comment	需要token头

请求体：

回复了文章：

```
{"articleId":1,"type":0,"rootId":-1,"toCommentId":-1,"toCommentUserId":-1,"conte
nt":"评论了文章"}
```

回复了某条评论：

```
{"articleId":1,"type":0,"rootId":"3","toCommentId":"3","toCommentUserId":"1","co
ntent":"回复了某条评论"}
```

如果是友链评论，type应该为1

响应格式：

```
{
    "code":200,
    "msg":"操作成功"
}
```

3.11.3 代码实现

CommentController

```
    @PostMapping("/comment")
    public ResponseResult comment(@RequestBody Comment comment){
        return commentService.addComment(comment);
    }
```

CommentService

```
ResponseResult addComment(Comment comment);
```

CommentServiceImpl

```
    @Override
    public ResponseResult addComment(Comment comment) {
        //评论内容不能为空
        if(!StringUtils.hasText(comment.getContent())){
            throw new SystemException(AppHttpCodeEnum.CONTENT_NOT_NULL);
        }
        save(comment);
        return ResponseResult.okResult();
    }
```

SecurityUtils

```
public class SecurityUtils
{

    /**
     * 获取用户
     **/
    public static LoginUser getLoginUser()
    {
        return (LoginUser) getAuthentication().getPrincipal();
    }

    /**
     * 获取Authentication
     */
    public static Authentication getAuthentication() {
        return SecurityContextHolder.getContext().getAuthentication();
    }

    public static Boolean isAdmin(){
        Long id = getLoginUser().getUser().getId();
        return id != null && 1L == id;
    }

    public static Long getUserId() {
        return getLoginUser().getUser().getId();
    }
}
```

配置MP字段自动填充

```
@Component
public class MyMetaObjectHandler implements MetaObjectHandler {

    @Override
    public void insertFill(MetaObject metaObject) {
        Long userId = null;
        try {
            userId = SecurityUtils.getUserId();
        }catch (Exception e){
            e.printStackTrace();
            userId = -1L;//表示是自己创建
        }
        this.setFieldValByName("createTime", LocalDateTime.now(), metaObject);
        this.setFieldValByName("createBy", userId, metaObject);
        this.setFieldValByName("updateTime", LocalDateTime.now(), metaObject);
        this.setFieldValByName("updateBy", userId, metaObject);
    }

    @Override
    public void updateFill(MetaObject metaObject) {
        this.setFieldValByName("updateTime", new Date(), metaObject);
        this.setFieldValByName("updateBy", SecurityUtils.getUserId(),
metaObject);
    }
}
```

用注解标识哪些字段在什么情况下需要自动填充

```
    /**
     * 创建人的用户id
     */
    @TableField(fill = FieldFill.INSERT)
    private Long createBy;
    /**
     * 创建时间
     */
    @TableField(fill = FieldFill.INSERT)
    private Date createTime;
    /**
     * 更新人
     */
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private Long updateBy;
    /**
     * 更新时间
     */
    @TableField(fill = FieldFill.INSERT_UPDATE)
    private Date updateTime;
```

3.12 友链评论列表

3.12.1 需求

友链页面也需要查询对应的评论列表。

3.12.2 接口设计

请求方式	请求地址	请求头
GET	/comment/linkCommentList	不需要token请求头

Query格式请求参数：

pageNum: 页码

pageSize: 每页条数

响应格式:


```
{
    "code": 200,
    "data": {
        "rows": [
            {
                "articleId": "1",
                "children": [
                    {
                        "articleId": "1",
                        "content": "回复友链评论3",
                        "createBy": "1",
                        "createTime": "2022-01-30 10:08:50",
                        "id": "23",
                        "rootId": "22",
                        "toCommentId": "22",
                        "toCommentUserId": "1",
                        "toCommentUserName": "sg333",
                        "username": "sg333"
                    }
                ],
                "content": "友链评论2",
                "createBy": "1",
                "createTime": "2022-01-30 10:08:28",
                "id": "22",
                "rootId": "-1",
                "toCommentId": "-1",
                "toCommentUserId": "-1",
                "username": "sg333"
            }
        ],
        "total": "1"
    },
    "msg": "操作成功"
}
```

3.12.3 代码实现

CommentController 修改了之前的文章评论列表接口，并且增加了新的友联评论接口

```
    @GetMapping("/commentList")
    public ResponseResult commentList(Long articleId,Integer pageNum,Integer
pageSize){
        return
commentService.commentList(SystemConstants.ARTICLE_COMMENT,articleId,pageNum,pag
eSize);
    }
    @GetMapping("/linkCommentList")
    public ResponseResult linkCommentList(Integer pageNum,Integer pageSize){
        return
commentService.commentList(SystemConstants.LINK_COMMENT,null,pageNum,pageSize);
    }
```

SystemConstants增加了两个常量

```
    /**
     * 评论类型为：文章评论
     */
    public static final String ARTICLE_COMMENT = "0";
    /**
     * 评论类型为：友联评论
     */
    public static final String LINK_COMMENT = "1";
```

CommentService修改了commentList方法，增加了一个参数commentType

```
ResponseResult commentList(String commentType, Long articleId, Integer pageNum,
Integer pageSize);
```

CommentServiceImpl修改commentList方法的代码，必须commentType为0的时候才增加articleId的判断，并且增加了一个评论类型的添加。

```
@Override
    public ResponseResult commentList(String commentType, Long articleId,
Integer pageNum, Integer pageSize) {

        //查询对应文章的根评论
        LambdaQueryWrapper<Comment> queryWrapper = new LambdaQueryWrapper<>();
        queryWrapper.eq(articleId != null,Comment::getArticleId,articleId)
                    .eq(Comment::getRootId,-1)
                    .eq(Comment::getType,commentType)
```

```
                    .orderByAsc(Comment::getCreateTime);
        //分页查询
        Page page = new Page(pageNum, pageSize);
        page(page,queryWrapper);
        List<CommentVo> commentVoList = toCommentVoList(page.getRecords());

        //查询所有根评论对应的子评论集合，并且赋值给对应的属性
        for (CommentVo commentVo : commentVoList) {
            //查询对应的子评论
            List<CommentVo> children = getChildren(commentVo.getId());
            commentVo.setChildren(children);
        }
        return ResponseResult.okResult(new
PageVo(commentVoList,page.getTotal()));
    }
```