博客系统设计文档

一、用例图

1. 参与者

· 博主：博客内容的创作者和管理者。
· 读者：博客内容的浏览者。
· 管理员：负责博客系统的整体管理。

2. 用例

· 博主用例：发布博客、编辑博客、删除博客、管理评论等。
· 读者用例：浏览博客、发表评论、点赞博客、搜索博客等。
· 管理员用例：用户管理、博客审核、系统设置等。

3. 关系

· 包含关系：发布博客包含保存草稿用例。
· 扩展关系：浏览博客在特定条件下可以扩展为查看热门博客用例。
· 泛化关系：博主和读者都泛化自用户。

二、系统顺序图

1. 博主发布博客

步骤	消息发送者	消息接收者	消息内容	消息类型
1	博主	博客系统	发起发布博客请求	同步消息
2	博客系统	博客数据库	保存博客数据	同步消息
3	博客数据库	博客系统	返回保存成功信息	返回消息
4	博客系统	博主	返回发布成功信息	返回消息

2. 读者阅读博客

步骤	消息发送者	消息接收者	消息内容	消息类型
1	读者	博客系统	发起阅读博客请求	同步消息
2	博客系统	博客数据库	查询博客数据	同步消息
3	博客数据库	博客系统	返回博客数据	返回消息
4	博客系统	读者	返回博客内容	返回消息

3. 管理员审核博客

步骤	消息发送者	消息接收者	消息内容	消息类型
1	管理员	博客系统	发起审核博客请求	同步消息
2	博客系统	博客数据库	查询待审核博客数据	同步消息
3	博客数据库	博客系统	返回待审核博客数据	返回消息
4	管理员	博客系统	提交审核结果（通过或不通过）	同步消息
5	博客系统	博客数据库	更新博客审核状态	同步消息
6	博客数据库	博客系统	返回更新成功信息	返回消息
7	博客系统	管理员	返回审核结果处理成功信息	返回消息

三、概念类图

1. 概念类

- User（用户）
- Article（文章）
- Category（分类）
- Tag（标签）
- Comment（评论）
- Like（点赞）

2. 关联关系

- User – Article：一对多
- Article – Category：一对多
- Article – Tag：多对多
- User – Comment：一对多
- Article – Comment：一对多
- User – Like：多对多

3. 属性

- User类：用户名、密码、邮箱、注册时间等。
- Article类：标题、内容、发布时间、作者等。
- Comment类：发布时间、评论内容等。

四、OCL合约

1. 不变量

- 文章发布时间约束：文章的发布时间必须小于等于当前时间。

```
context Article
inv postTimeConstraint: self.postTime <= CurrentDate()
```

- 评论发布时间约束：评论的发布时间必须小于等于当前时间，且大于等于文章的发布时间。

```
context Comment
inv commentTimeConstraint: self.postTime <= CurrentDate() and
self.postTime >= self.article.postTime
```

· 点赞数量非负约束：文章的点赞数量必须是非负整数。

```
context Article
inv likeCountConstraint: self.likeCount >= 0
```

2. 前提

· 用户删除文章前提：只有文章的作者或者管理员才能删除文章。

```
context Article::deleteArticle()
pre deletePermission: self.author = currentUser or currentUser.role =
'管理员'
```

· 用户发表评论前提：用户必须处于登录状态才能发表评论。

```
context Comment::postComment()
pre userLoggedIn: currentUser.isLoggedIn = true
```

3. 后置

· 用户点赞文章后置：文章的点赞数量加1。

```
context Article::likeArticle()
post likeCountIncrease: self.likeCount = self.likeCount@pre + 1
```

· 用户删除评论后置：文章的评论数量减1。

```
context Article::deleteComment(comment: Comment)
post commentCountDecrease: self.commentCount = self.commentCount@pre -
1
```

4. 派生

· 文章的热门程度：文章的热门程度可以根据文章的点赞数、评论数和阅读数来计算。

```
context Article
def: popularity: Integer = self.likeCount + self.commentCount +
self.readCount
```

· 用户的文章数量：用户发布的文章数量可以通过关联关系派生出来。

```
context User
def: articleCount: Integer = self.articles->size()
```