[个人博客系统设计文档]

2016 年 4 月 28 日

1.1 开发背景

近年来，博客及博客文化成为互联网的热点，被视为继 E-mail、BBS 和 ICQ 之后的第四种网络交流方式。

博客正在改变组织沟通和社会的交流方式。目前用户在网络上发表文章、张贴内容有很大的差异，但是，由于沟通方式比电子邮件、讨论群组和论坛方式更简单和容易，博客系统已经成为广大各界用户进行沟通的主流工具，本系统是针对博客用户的需求设计的。

1.2 系统分析

1.2.1 需求分析

对于个人博客系统来说，最大的需求就是让更多的用户能够浏览博客，并且发表自己的博客文章。对于用户而言，首先要能浏览其他用户发表的文章，并且还能评论其他用户的博客；并且用户自己通过注册帐号，登录后能自己发表博客，还能查看修改自己的博客等等。

1.2.2 功能分析

该博客网站的功能主要是为博客用户设计开发的，用户进入博客网站后，有两种选择，登录和浏览博客。通过登录用户可以获取自己的博客空间，如果不进行这些操作，用户也可以浏览其他用户的博客内容。

- 发表博客
- 评论博客
- 修改个人信息
- 搜索博客的功能
- 分页功能
- 修改博客
- 修改密码
- 通过博客列表动态的显示最近更新的博主的信息
- ....

1.3 系统设计

1.3.1 绘制用例图设计系统功能

博客系统中一共包含两种权限的角色，分别是用户和游客。下面就来分析这两个角色所对应的用例图。

游客可以查看博客文章内容、访问其他用户博客、查看博客相关内容等功能，其用例图如图 1-1 所示。用户除了能够完成游客功能外，还能够完成对自己博客进行相关操作的功能，包括发表博客等功能，其用例图如图 1-2 所示。

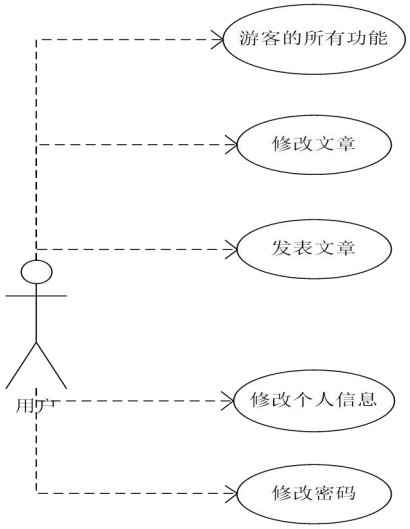

图 1-1 用户用例图

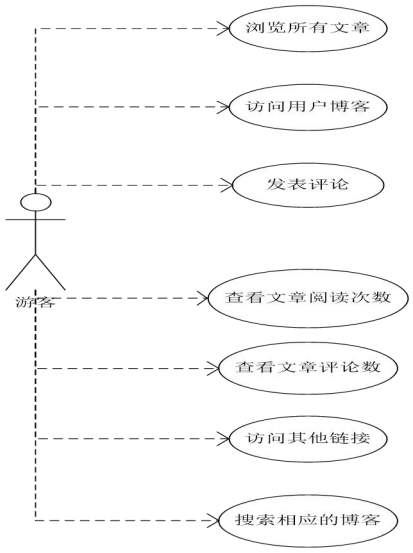

图 1-2 游客用例图

1.3.2 绘制系统流程图

本系统首先需要对用户进行身份验证，验证时判断用户是否是已注册的用户。如果是已注册用户，跳转到个人博客页面。系统流程图如图 1-3 所示。

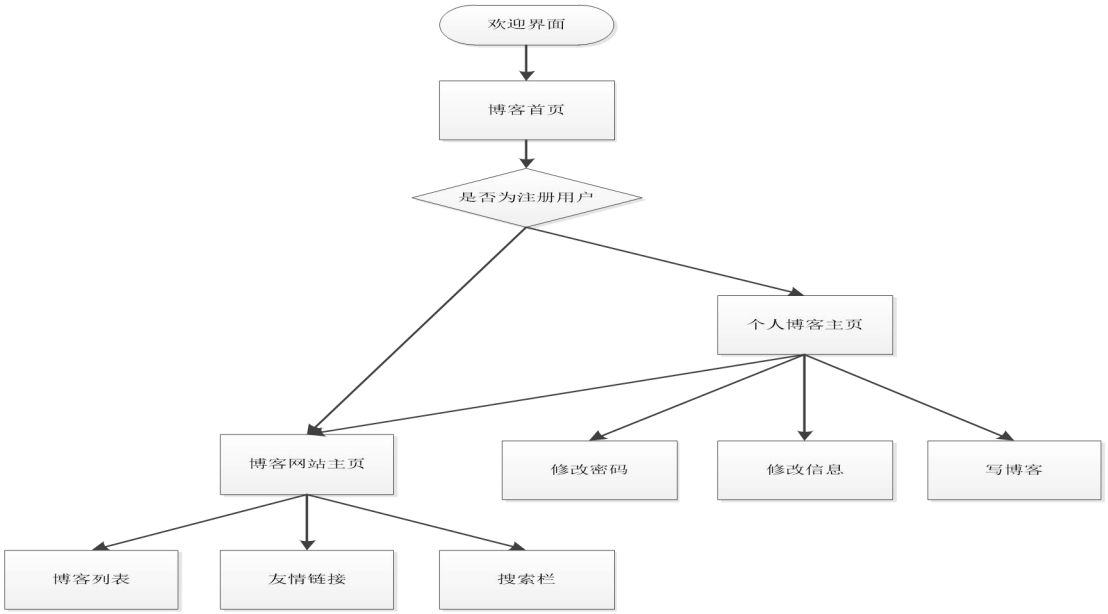

图 1-3 系统流程图

1.3.3 系统演示

打开 IE 浏览器，在地址栏输入 http://localhost:8080/MyBlog/，打开欢迎动画页面如图 1-4 所示

图 1-3 欢迎页面

点击欢迎页面，进入博客首页，如图 1-4 所示界面显示出来，有博客列表、友情链接、搜索、登录、注册等功能。

图 1-4 博客主页

点击博客列表选项，将出现其他用户等博客可以浏览，如图 1-5 所示。

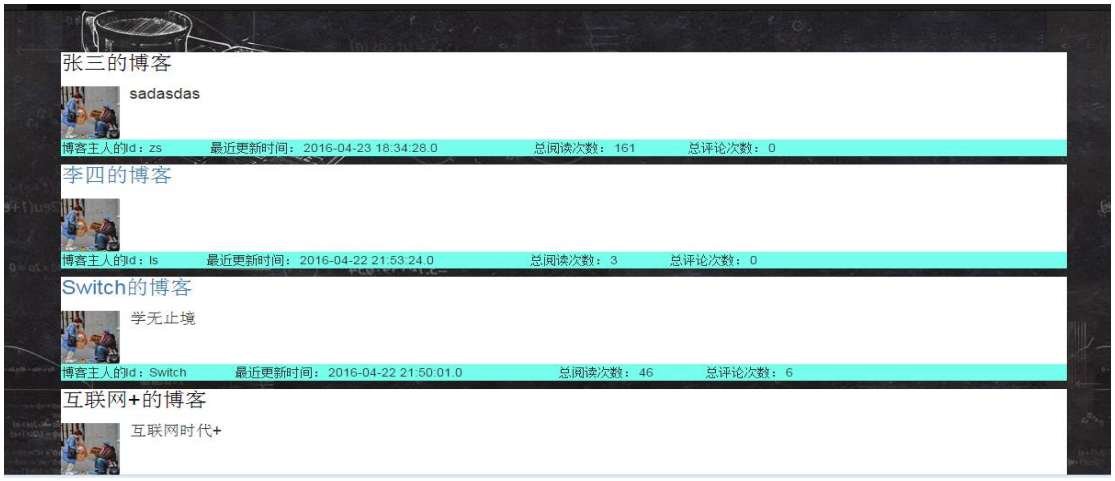

图 1-5 博客列表页面

点击友情链接出现下拉列表，会出现一些链接供查看，如图 1-6 所示。

图 1-6 友情链接页面

在搜索框中搜索相应的语句可以查询博客中相关内容。如搜索“java”一词，结果如图 1-7 所示。

图 1-7 搜索页面

点击登录选项，弹出登录页面可以登录已有的账号进入个人博客。如图 1-8 所示，登录后的页面如图 1-8 所示。

图 1-8 登录页面

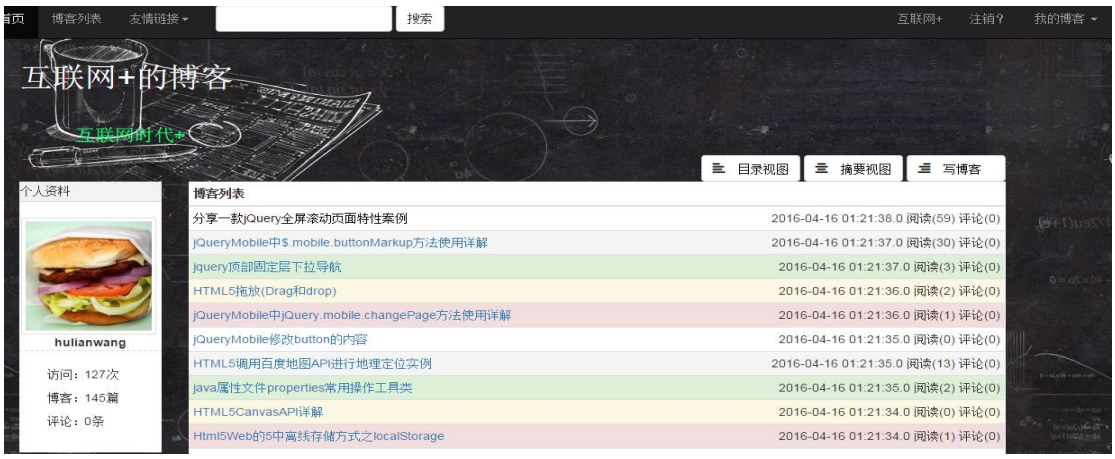

图 1-9 个人博客页面

右上角有注销账号功能，点击则会注销当前账号。还有我的博客功能，点击后会弹出如图 1-10 的页面。

图 1-10 我的博客页面

点击修改信息则可以修改，点击修改密码可以，点击写博客可以进入写博客页面。分别如图 1-11，1-12，1-13 所示。

图 1-11 修改信息页面

图 1-12 修改密码页面

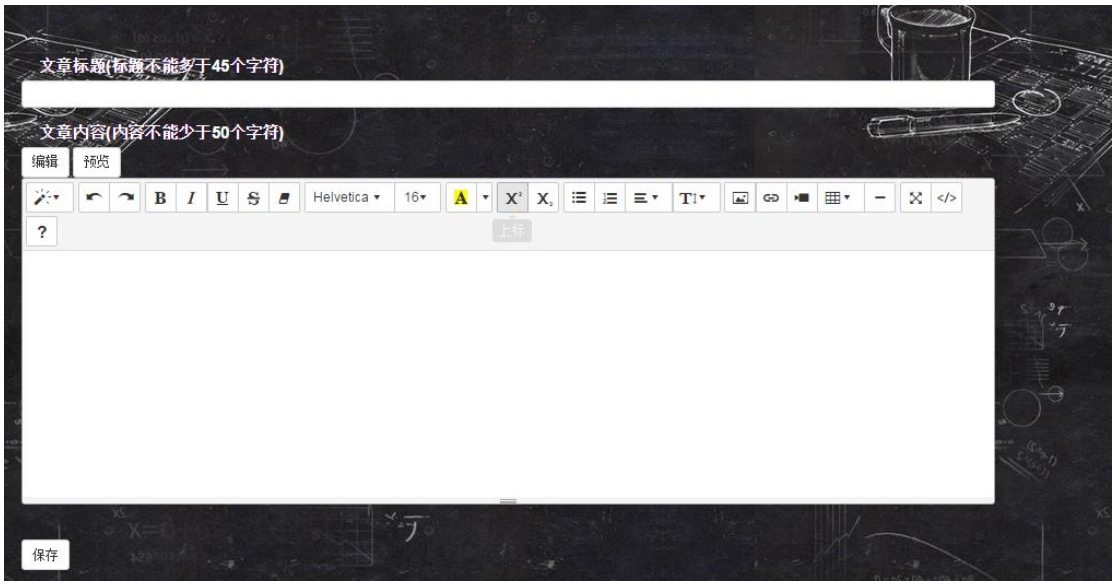

图 1-13 写博客页面

再没有登录的状态下，点击注册选项会进入注册页面，如图 1-14 所示。

图 1-14 注册页面

1.3.4 开发工具和开发技术的选择

本系统的开发工具如下。

- 系统开发平台：MyEclipse 10.
- 数据库管理系统软件：MySQL.
- 运行平台：Windows 7.
- Java 开发包：JDK6.0 以上.
- Web 服务器：Tomcat6.0.

本系统采用 SSH 框架四层架构模式开发，具体技术如下。

- 显示层：使用 Struts 1 标签技术开发
- 控制层：使用 Struts 1 技术开发
- 业务逻辑层：使用 Spring 技术进行业务处理
- 数据访问层：使用 Hibernate 进行数据库访问和操作

1.3.5 文件夹组织结构

在编写代码之前，首先需要将系统可能用到的文件夹创建好，这样可以方便网站开发工作，同时还可以规范网站的整体架构。本系统的文件夹组织结构如图 1-15 所示。

图 1-15 文件夹组织结构图

1.4 数据库分析与设计

1.4.1 数据库分析

在博客系统中，用户通过登录才能拥有自己的博客，所以在数据库中要创建用户表。

用户发表博客文章，所以要在数据库中创建文章表，从而保存所有用户发表的文章。这些数据表如图 1-16 所示。

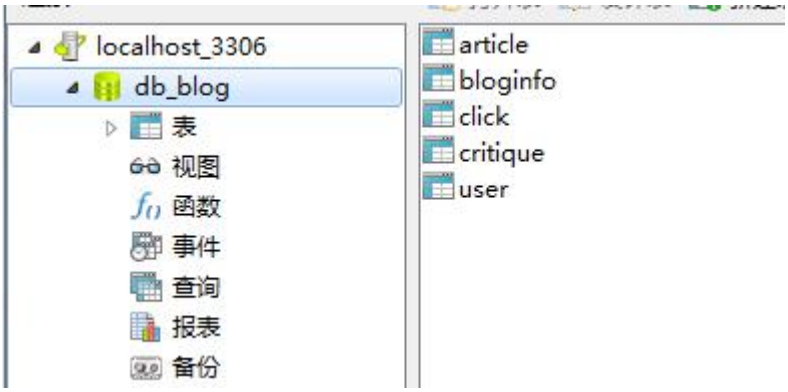

图 1-16 数据库 db_exam 中的所有数据表

1.4.2 数据库概念设计

本系统一共设计规划出 3 个实体，分别是用户实体、文章实体、评论实体。

用户实体用来保存用户的所有信息，包括用户名、登录密码、昵称、用户编号、密保问题、密保回答。用户实体 E-R 图如图 1-17 所示。

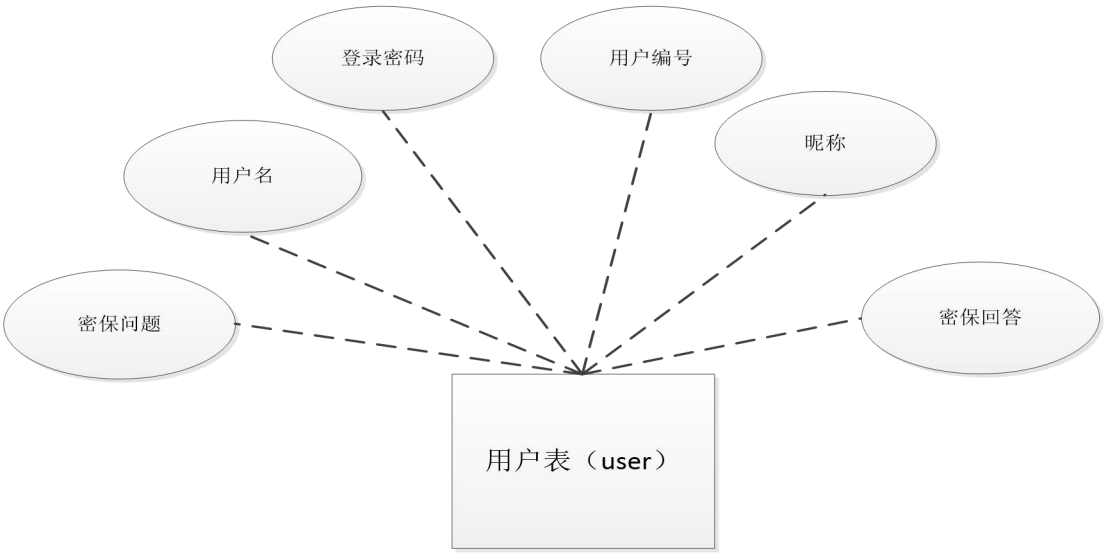

图 1-17 用户实体 E-R 图

文章实体用来保存文章的基本信息，包括文章 ID、文章内容、用户 ID、发表时间、评论数、文章标题。文章实体 E-R 图如图 1-18 所示。

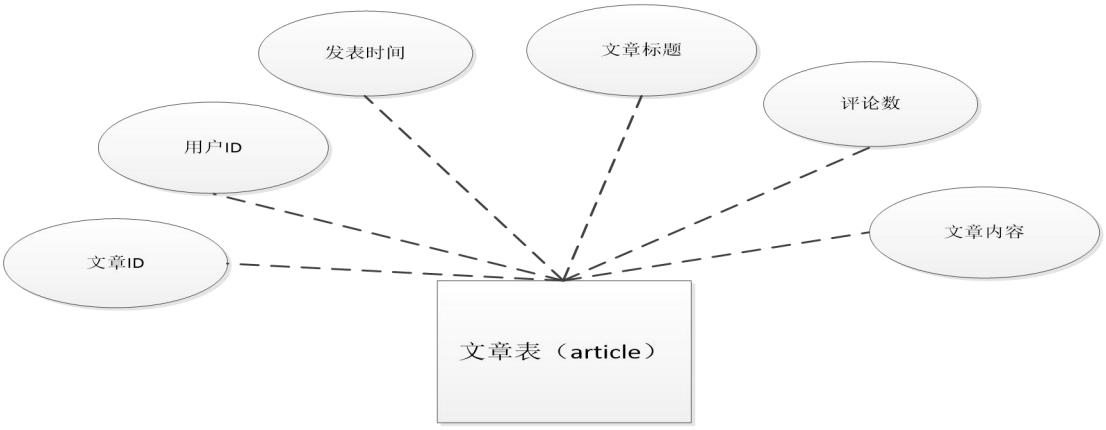

图 1-18 文章实体 E-R 图

评论实体用来保存评论的基本信息，包括评论用户 ID、评论内容、所属文章 ID、评论 ID 实体 E-R 图如图 1-19 所示。

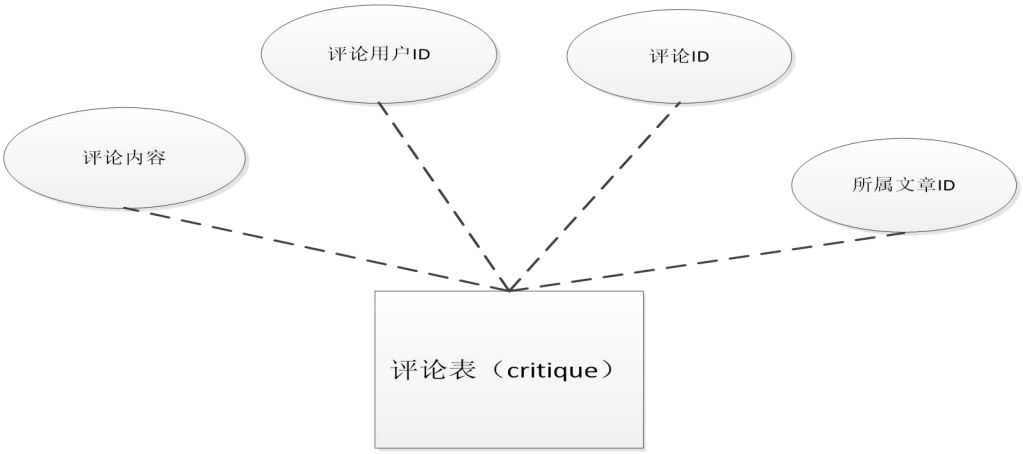

图 1-19 评论实体 E-R 图

个性设置实体用来保存博客个性化信息，包括个性设置 ID、博客标签、个性签名。如图 1-20 所示。

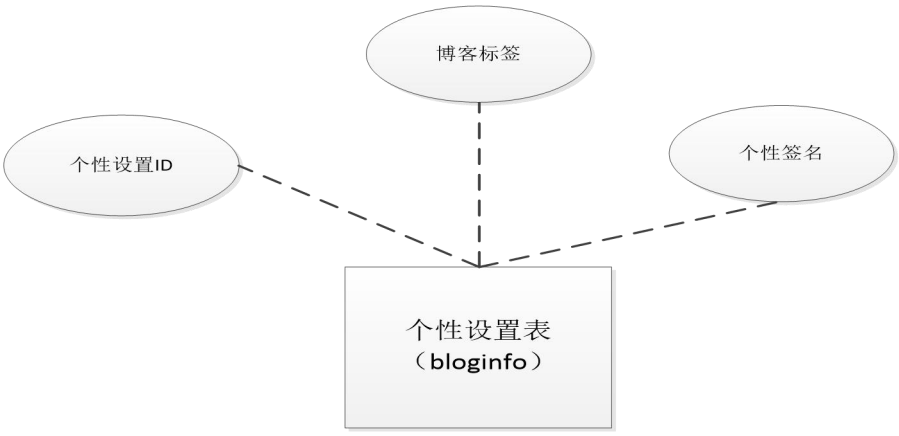

图 1-20 个性设置实体 E-R 图

点击量实体用来保存点击文章的基本信息，包括点击 ID、点击者 IP、点击时间。如图 1-21 所示。

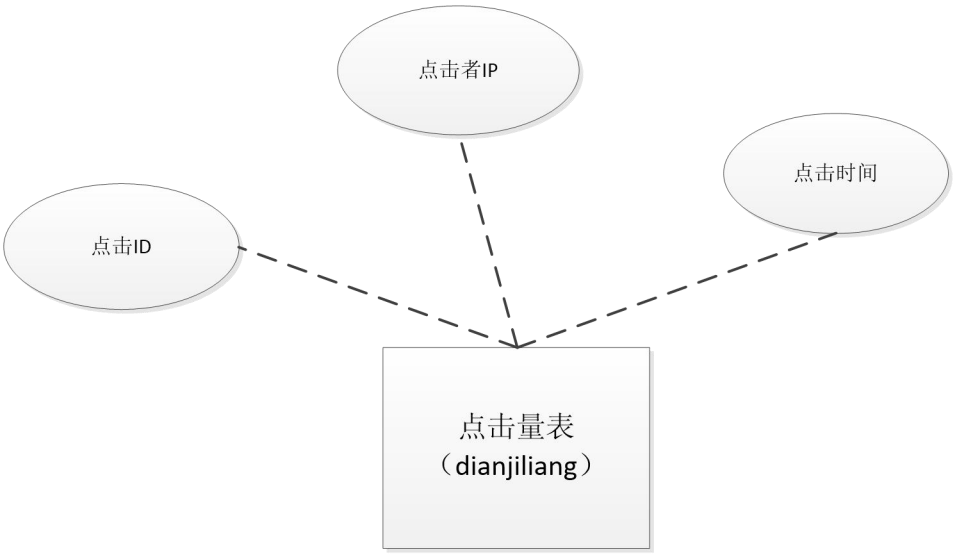

图 1-21 点击量实体 E-R 图

1.4.3 数据库逻辑结构设计

根据设计好的各实体 E-R 图创建数据库的逻辑结构，数据库各表的结构如下。

1） 用户表用来存储博客系统中建立自己博客，并进行博客操作的用户，包括用户编号、用户名、密码、昵称、密保问题、密保答案，该表的逻辑结果如表 1-1 所示。

表 1-1 用户表

字段名	数据类型	是否为主外键	描述内容
userId	整型(int)	主键	用户编号
userName	文本(varchar)	否	用户名
password	文本(varchar)	否	密码
nickName	文本(varchar)	否	昵称
question	文本(varchar)	否	密保问题
answer	文本(varchar)	否	密保答案

2) 个性设置表存储用户对个人博客进行个性设置的基本信息，包括个性设置 ID、用户 ID、博客标签、个性签名。如表 1-2 所示。

表 1-2 个性设置表

字段名	数据类型	是否为主外键	描述内容
logId	整型(int)	主键	个性设置 ID
userId	整型(int)	外键	用户 ID
logtitle	文本(varchar)	否	博客标签
diograph	文本(varchar)	否	个性签名

3) 文章表用来存储博客系统中博客文章的基本信息,包括文章 ID、用户 ID、文章标题、文章内容、发表用户名、发表时间、评论数，该表的逻辑结果如表 1-3 所示。

表 1-3 文章表

字段名	数据类型	是否为主外键	描述内容
articleId	整型(int)	主键	文章 ID
userId	整型(int)	外键	用户 ID
title	文本(varchar)	否	文章标题
content	文本（text）	否	文章内容
date	日期（datetime）	否	发表时间
hasread	整型(int)	否	评论数

4) 评论表用来存储博客系统中所有评论的基本信息，包括评论 ID、所属文章 ID、评论用户 ID、评论内容。如表 1-4 所示。

表 1-4 评论表

字段名	数据类型	是否为主外键	描述内容
critiqueId	整型(int)	主键	评论 ID
articleId	整型(int)	外键	所属文章 ID
userId	整型(int)	外键	评论用户 ID
content	文本（text）	否	评论内容

5） 点击量表用来存储博客系统发中点击文章的基本信息，包括点击 ID、所属文章 ID、点击者 IP、点击时间。如表 1-5 所。

表 1-5 点击量表

字段名	数据类型	是否为主外键	描述内容
clickId	整型(int)	主键	点击 ID
articleId	整型(int)	外键	所属文章 ID
ip	文本(varchar)	否	点击者 IP
date	日期（datetime）	否	点击时间

1.4.4 绘制表之间关系 E-R 图

根据各表关系绘制关系 E-R 图如图 1-22 所示。

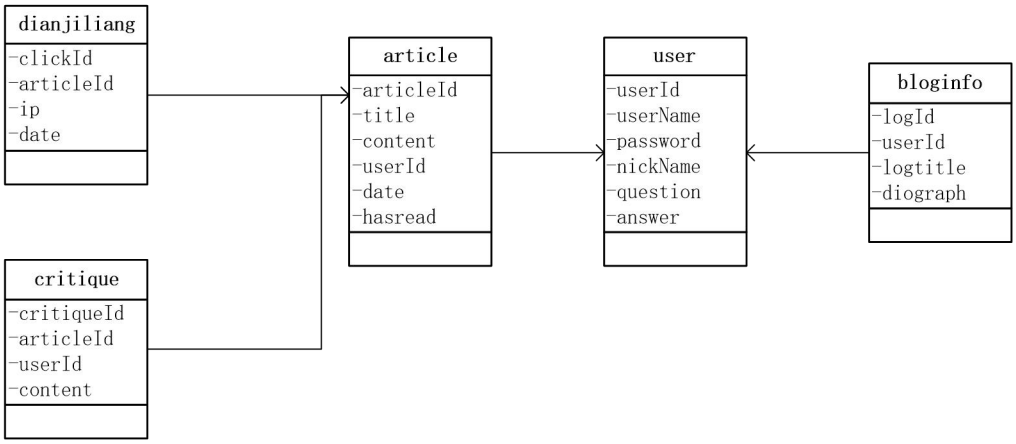

图 1-22 关系 E-R 图

1.5 公共类设计

1.5.1 用户信息类

用户信息类中封装类博客系统中用户的所有信息，包括用户编号、用户名、密码、昵称、密保问题、密保答案。

例程 1.1

```
1public class User implements java.io.Serializable {
2   private Integer userId;
3   private String userName;
4   private String password;
5   private String nickName;
6   private String question;
7   private String answer;
8   private Set articles = new HashSet(0);
9   private Set bloginfos = new HashSet(0);
10  private Set critiques = new HashSet(0);

11  // 省略各属性的 setter 和 getter 方法
    }
```

在上述代码中，第 2 行到第 10 行定义了用户的所有属性，第 11 行省略了 setter 和 getter 方法。在 hibernate 中，需要对用户实体类进行配置。

例程 1.2

```
<hibernate-mapping>
    <class name="com.myblog.domain.User" table="user"
catalog="db_blog">
        <id name="userId" type="integer">
            <column name="userId" />
            <generator class="native" />
        </id>
        <property name="userName" type="string">
            <column name="userName" length="20" not-null="true"
unique="true" />
        </property>
        <property name="password" type="string">
            <column name="password" length="50" not-null="true" />
        </property>
        <property name="nickName" type="string">
            <column name="nickName" length="20" />
        </property>
        <property name="question" type="string">
            <column name="question" length="50" not-null="true" />
        </property>
        <property name="answer" type="string">
            <column name="answer" length="50" not-null="true" />
        </property>
        <set name="articles" inverse="true"
cascade="save-update">
            <key>
                <column name="userId" not-null="true" />
```

```
            </key>
            <one-to-many class="com.myblog.domain.Article" />
        </set>
        <!-- 这里需要重构为一对一关系 -->
        <set name="bloginfos" inverse="true"
cascade="save-update">
            <key>
                <column name="userId" not-null="true"
unique="true" />
            </key>
            <one-to-many class="com.myblog.domain.Bloginfo" />
        </set>
        <set name="critiques" inverse="true"
cascade="save-update">
            <key>
                <column name="userId" not-null="true" />
            </key>
            <one-to-many class="com.myblog.domain.Critique"/>
        </set>
    </class>
</hibernate-mapping>
```

上述代码是对实体类对配置，第 2 行 name 属性对应实体类对完整类名。第 4 行是对主键 userId 进行配置，并且采用 native 策略。之后开始依次配置用户密码、昵称、密码保护问题和密码保护答案属性。

1.5.2 文章信息类

文章信息类中封装类博客系统中发表文章的所有信息，包括文章 ID、用户 ID、文章标题、文章内容、发表用户名、发表时间、评论数属性。

例程 1.3

```
1public class Article implements java.io.Serializable {
2   private Integer articleId;
3   private User user;
4   private String title;
5   private String content;
6   private Date date;
7   private Integer hasread;
8   private Set critiques = new HashSet(0);
9   private Set clicks = new HashSet(0);
10  // 省略各属性的 setter 和 getter 方法
       }
```

上述代码中定义了文章信息实体类，从第 2 行到第 9 行定义了文章类的所有属性。从第 10 行开始定义这些属性的 setter 和 getter 方法。

例程 1.4

```
<hibernate-mapping>
    <class name="com.myblog.domain.Article" table="article"
catalog="db_blog">
        <id name="articleId" type="integer">
            <column name="articleId" />
            <generator class="native" />
        </id>
        <many-to-one name="user" class="com.myblog.domain.User"
fetch="select">
            <column name="userId" not-null="true" />
        </many-to-one>
        <property name="title" type="string">
            <column name="title" length="50" not-null="true" />
        </property>
        <property name="content" type="string">
            <column name="content" length="65535" not-null="true"
/>
        </property>
        <property name="date" type="timestamp">
            <column name="date" length="19" />
        </property>
        <property name="hasread" type="integer">
            <column name="hasread" />
        </property>
        <set name="critiques" inverse="true"
cascade="save-update">
            <key>
                <column name="articleId" not-null="true" />
            </key>
            <one-to-many class="com.myblog.domain.Critique" />
        </set>
        <set name="clicks" inverse="true" cascade="save-update">
            <key>
                <column name="articleId" not-null="true" />
            </key>
            <one-to-many class="com.myblog.domain.Click" />
        </set>
    </class>
</hibernate-mapping>
```

上述代码是对文章类对配置，第 2 行文章信息类进行了配置。第 4 行是对主键 articleId 进行配置，并且采用 ***integer*** 策略。之后开始依次配置文章 ID、用户 ID、文章标题、文章内容、发表用户名、发表时间、评论数属性。

1.5.3 评论信息类

评论信息类中封装类博客系统中发表评论的所有信息，包括评论 ID、所属文章 ID、评论用户 ID、评论内容属性。

例程 1.5

```
1public class Critique implements java.io.Serializable {
2   private Integer critiqueId;
3   private User user;
4   private Article article;
5   private String content;
6   // 省略各属性的 setter 和 getter 方法
       }
```

上述代码中定义了评论信息实体类，从第 2 行到第 5 行定义了文章类的所有属性。从第 6 行开始定义这些属性的 setter 和 getter 方法。

例程 1.6

```
<hibernate-mapping>
    <class name="com.myblog.domain.Critique" table="critique"
catalog="db_blog">
        <id name="critiqueId" type="integer">
            <column name="critiqueId" />
            <generator class="native" />
        </id>
        <many-to-one name="user" class="com.myblog.domain.User"
fetch="select">
            <column name="userId" not-null="true" />
        </many-to-one>
        <many-to-one name="article"
class="com.myblog.domain.Article" fetch="select">
            <column name="articleId" not-null="true" />
        </many-to-one>
        <property name="content" type="string">
            <column name="content" length="65535" not-null="true"
/>
        </property>
    </class>
</hibernate-mapping>
```

上述代码第 2 行评论信息类进行了配置。第 4 行是对主键 critiqueId 进行配置，并且采用 *integer* 策略。之后开始依次配置评论 ID、所属文章 ID、评论用户 ID、评论内容属性。

1.6 获取创建个人博客权限模块

1.6.1 申请个人博客 Service 方法

当进行申请个人博客操作时，要将用户输入的信息保存在数据库中。在本系统项目中，进行数据库操作是通过 hibernate 技术完成的，从而使 servlet 方法应用也更加简单。

当进行申请操作时，首先要判断申请的用户名是否被他人使用，如果没有被使用，就可以申请，将用户填写的信息保存在数据库中。

例程 1.7

```
public boolean register(User user) {
    List<User> users = this.getAllUser();
    for (User u : users) {
        if(u.getUserName().equals(user.getUserName())) {
            return false;
        }
    }
    // 使用 MD5 加密密码
    user.setPassword(createMD5(user.getPassword()));
    this.add(user);
    Set<Bloginfo> bloginfos =
blogInfoService.initBlogInfoByUser(user);
    user.setBloginfos(bloginfos);
    // 测试获取 ID 是否成功(成功)
    // System.out.println(id);
    return true;
}
```

1.6.2 申请个人博客 action 方法

例程 1.8

```
public ActionForward register(ActionMapping mapping,
ActionForm form,
        HttpServletRequest request, HttpServletResponse
response)
        throws Exception {
    UserForm userForm = (UserForm) form;

    if(!checkUserForm(request, userForm)){
```

```
            return mapping.findForward("registererr");

}
}
```

1.6.3 进入个人博客 service 方法

例程 1.9

```
public User checkUser(User user) {
       String hql = "from User where userName=? and password=?";
       Object[] parameters = {user.getUserName(),
createMD5(user.getPassword())};
       List list = this.executeQuery(hql, parameters);
       if(list.size() <= 0 || list.size() >= 2) {
          return null;
       } else {
          return (User) list.get(0);
       }
   }
```

1.6.4 进入个人博客 action 方法

例程 1.10

```
   public ActionForward login(ActionMapping mapping, ActionForm
form,
          HttpServletRequest request, HttpServletResponse
response) {
       UserForm userForm = (UserForm) form;

       // 构建一个 User 对象
       User loginUserInfo = new User();
       loginUserInfo.setUserName(userForm.getUserName());
       loginUserInfo.setPassword(userForm.getPassword());

       // 测试表单数据是否填入表单对象(成功)
       // System.out.println(userForm.getUserName() + " " +
userForm.getPassword());
       if((loginUserInfo =
userService.checkUser(loginUserInfo)) != null) {
          // 测试 domain 对象是否被填入数据(成功)
          // System.out.println(loginUserInfo.getUserId() + " " +
loginUserInfo.getUserName() + " " + loginUserInfo.getPassword());
```

```
            saveUserToSessionAndCookie(request, response,
loginUserInfo);

            return mapping.findForward("gotoUserUI");
        } else {
            request.setAttribute("errinfo", "账号名或密码输入有误
");
            return mapping.findForward("loginerr");
        }
    }
```

1.7 个人博客模块

1.7.1 写文章 service 方法

例程 1.11

```
public String saveArticleByUserIdAndTitleAndContent(String
userId,
            String title, String content) throws Exception{
        String sql = "insert into article(userId,title,content)
values(?,?,?)";
        String[] parameters = {userId, title, content};
        this.executeSQLUpdate(sql, parameters);
        sql = "select max(articleId) from article";
        List list = this.executeSQLQuery(sql, null);
        return list.get(0).toString();
        }
```

1.7.2 写文章 action 方法

点击写文章 action 方法：

例程 1.12

```
    public ActionForward gotoWriteBlogUI(ActionMapping mapping,
ActionForm form,
            HttpServletRequest request, HttpServletResponse
response)
            throws Exception {
        String userIdStr = request.getParameter("userId");
        User loginUserInfo = (User)
```

```
request.getSession().getAttribute("loginUserInfo");
        if (loginUserInfo == null) {
            return mapping.findForward("opererr");
        }

        Integer userId = null;
        try {
            userId = Integer.parseInt(userIdStr);
        } catch (Exception e) {
            return mapping.findForward("opererr");
        }

        if (!userId.equals(loginUserInfo.getUserId())) {
            return mapping.findForward("opererr");
        } else {
            return mapping.findForward("gotoWriteBlogUI");
        }
        }
```

保存文章 action 方法：

例程 1.13

```
    public ActionForward saveArticle(ActionMapping mapping,
ActionForm form,
            HttpServletRequest request, HttpServletResponse
response)
            throws Exception {
        ArticleForm articleForm = (ArticleForm) form;
        String userId = articleForm.getUserId();
        String title = articleForm.getTitle();
        String content = articleForm.getContent();
        if (title.length() > 45 || content.length() < 50) {
            request.setAttribute("errInfo", "博客内容不符合要求");
            return mapping.findForward("gotoWriteBlogUI");
        }
        String articleId = null;
        try {
            articleId =
articleService.saveArticleByUserIdAndTitleAndContent(userId,
title, content);
        } catch(Exception e) {
            e.printStackTrace();
            return mapping.findForward("opererr");
        }

        prepareArticleContent(request, articleId);
```

```
        return mapping.findForward("gotoArticleContentPage");
        }
```

1.7.3 显示用户所有文章 service 方法

例程 1.14

```
public Object findById(Class clazz, Serializable id) {
        // TODO Auto-generated method stub
        return this.sessionFactory.getCurrentSession().load(clazz,
id);
    }
```

1.7.4 显示用户所有文章 action 方法

例程 1.15

```
public ActionForward gotoUserUI(ActionMapping mapping, ActionForm
form,
            HttpServletRequest request, HttpServletResponse
response) {
        // get blog username from query string
        String userName = request.getParameter("userName");
        // User loginUserInfo = (User)
request.getSession().getAttribute("loginUserInfo");
        String userId = request.getParameter("userId");
        if (userId != null && userId.length() >= 0) {
            User user = (User) userService.findById(User.class,
Integer.parseInt(userId));
            userName = user.getUserName();
        }

        if(prepareShowVisitInfo(request, userName, 1, 30)){
            return mapping.findForward("gotoUserUI");
        } else {
            return mapping.findForward("opererr");
        }
    }
```

1.8 博客首页模块

1.8.1 显示所有文章 service 方法

例程 1.16

```
public List<User> getAllUserByPageOrderByTime(int pageNow, int
pageSize) {
      String sql = "select
userId,userName,password,nickName,question,answer from (select
u.*,max(date) maxdate from user u,article a where u.userId =
a.userId GROUP BY a.userId ORDER BY userId asc) temp ORDER BY
temp.maxdate desc";
      SQLQuery sqlQuery =
sessionFactory.getCurrentSession().createSQLQuery(sql);
      sqlQuery.addScalar("userId", Hibernate.INTEGER);
      sqlQuery.addScalar("userName", Hibernate.STRING);
      sqlQuery.addScalar("password", Hibernate.STRING);
      sqlQuery.addScalar("nickName", Hibernate.STRING);
      sqlQuery.addScalar("question", Hibernate.STRING);
      sqlQuery.addScalar("answer", Hibernate.STRING);

   sqlQuery.setResultTransformer(Transformers.aliasToBean(User.
class));

      // 分页
      // 设置起始记录
      sqlQuery.setFirstResult((pageNow - 1) * pageSize);
      // 设置每页记录数
      sqlQuery.setMaxResults(pageSize);
      return sqlQuery.list();

   }
```

1.8.2 显示所有文章 action 方法

例程 1.17

```
public ActionForward gotoAllBlogUI(ActionMapping mapping,
ActionForm form,
         HttpServletRequest request, HttpServletResponse
```

```
response)
            throws Exception {
        prepareAllBlogInfo(request, 1, 20);

        return mapping.findForward("gotoAllBlogUI");
    }
```

1.8.3 查看指定文章内容

Service 方法

例程 1.18

```
public Object findById(Class clazz, Serializable id) {
        // TODO Auto-generated method stub
        return this.sessionFactory.getCurrentSession().load(clazz,
id);
    }
```

Action 方法：

例程 1.19

```
    public ActionForward gotoUserUI(ActionMapping mapping,
ActionForm form,
            HttpServletRequest request, HttpServletResponse
response) {
        // get blog username from query string
        String userName = request.getParameter("userName");
        // User loginUserInfo = (User)
request.getSession().getAttribute("loginUserInfo");
        String userId = request.getParameter("userId");
        if (userId != null && userId.length() >= 0) {
            User user = (User) userService.findById(User.class,
Integer.parseInt(userId));
            userName = user.getUserName();
        }

        if(prepareShowVisitInfo(request, userName, 1, 30)){
            return mapping.findForward("gotoUserUI");
        } else {
            return mapping.findForward("opererr");
        }
    }
```

1.8.3.1 显示文章的所有评论

例程 1.20

```
public List<Critique> getAllCritiqueByArticleId(String articleId)
throws Exception{
        String hql = "from Critique c where c.article.articleId=?";
        Object[] parameters = {Integer.parseInt(articleId)};
        return this.executeQuery(hql, parameters);
    }
```

1.8.3.2 获取文章的点击量

例程 1.21

保存点击记录

```
    public Serializable saveClickRecordByArticleIdAndIPAddress(
            String articleId, String ip) {
        String sql = "insert into click(articleId,ip) values(?,?)";
        String[] parameters = {articleId, ip};
        return this.executeSQLUpdate(sql, parameters);
    }
```

1.8.6 显示文章信息 action

例程 1.22

```
public ActionForward gotoArticleContentPage(ActionMapping
mapping, ActionForm form,
          HttpServletRequest request, HttpServletResponse
response) throws Exception{
        String articleId = request.getParameter("articleId");
        prepareArticleContent(request, articleId);
        return mapping.findForward("gotoArticleContentPage");
        }
```

附录 1 网站后台数据参数

Cookie
JSESSIONID
userName
Session
loginUserInfo
forgetUser

userMainUI.jsp Request
visitUserInfo

visitBlogInfo

visitArticleInfo

pageCount

pageNow

visitArticleCount

visitClickCount

visitCritiqueCount

allBlog.jsp Request
allUser

"userClick" + userTmp.getUserName()

"userCritique" + userTmp.getUserName()

"lastUpdateTime" + userTmp.getUserName()

pageCount

pageNow

articleMainUI.jsp Request
visitUserInfo

visitBlogInfo

visitSingleArticleInfo

visitCritiquesInfo

visitArticleCount

visitClickCount

visitCritiqueCount

articleWriteUI.jsp articleModifyUI.jsp Request
title

content

articleId

searchArticle.jsp Request
searchArticles

pageCount

pageNow

searchKey

Parameter
userName

pageNow

articleId

Question Select option value
mother

father

lover

name

birth