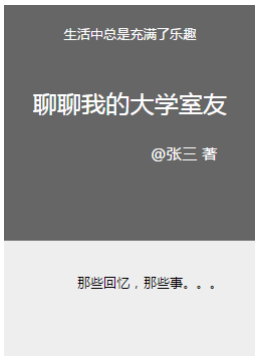
【手把手】JavaWeb 入门级项目实战 -- 文章发布系统 （第九节）

1. 根据静态页面完成JavaBean设计

在上一节中，我们完成了文章封面的制作，这些都属于静态页面的部分。



从图片中可以看到，一篇文章的主要信息有：文章标题，文章名称，作者，还有摘要描述。

在《用大白话聊聊JavaSE -- 如何理解Java Bean（一）》中，我们已经讨论关于JavaBean的一些问题。

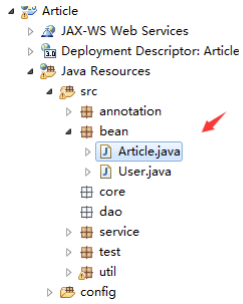
一般来说，JavaBean分为必要字段和辅助字段，文章标题，文章名称，作者，还有摘要描述，还有文章内容这些，应该属于必要字段的范畴。

至于辅助字段，我就不搞那么复杂了，简单设置几个吧，比如发布时间，最后更新时间，是否发布，是否删除。

当然，我们还需要知道这篇文章是谁写的，所以还要再加一个userid字段，这样的话才能和user表关联起来。

最后，还需要有一个分类字段，一篇文章，肯定是属于某一个类别的，所以这个也需要加上。

嗯，就添加这几个辅助字段吧，我们弄简单一点。



我们在bean包里面新建一个Article类。

设置属性如下：

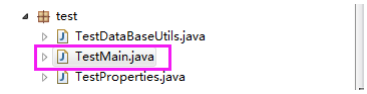
```
package bean;

import java.util.Date;

import annotation.Column;
import annotation.Table;

@Table(tableName = "t_article")
public class Article {

    @Column(field = "id" , type = "varchar(100)" , primaryKey = true)
    private String id; //主键

    @Column(field = "header" , type = "varchar(100)")
    private String header; //标题

    @Column(field = "name" , type = "varchar(60)")
    private String name; //文章名称

    @Column(field = "content" , type = "text")
    private String content; //文章内容

    @Column(field = "author" , type = "varchar(30)")
    private String author; //作者

    @Column(field = "description" , type = "varchar(100)")
    private String description; //概要

    @Column(field = "is_published" , type = "int(1)")
    private Integer isPublished; //是否发布 0 未发布 1 发布

    @Column(field = "is_delete" , type = "int(1)")
    private Integer isDelete;      //是否删除    0 未删除 1 已删除

    @Column(field = "create_time" , type = "datetime")
    private Date createTime;//创建时间

    @Column(field = "update_time" , type = "timestamp" , defaultNull = false)
    private Date updateTime;//最后更新时间

    @Column(field = "user_id" , type = "varchar(100)" , defaultNull = false)
    private String userId;//作者id

    @Column(field = "category_id" , type = "int(2)" , defaultNull = false)
    private Integer categoryId;//分类ID


}
```

然后，别忘了生成get，set以及toString方法。

2. Mysql建表

2.1 文章表

在TestMain方法中再生成一下sql语句。




```
package test;

import bean.Article;
import util.TableUtils;

public class TestMain {
    public static void main(String[] args) {
        String sql = TableUtils.getCreateTableSQl(Article.class);
        System.out.println(sql);
    }
}
```

运行

这是生成出来的sql语句

```
DROP TABLE IF EXISTS t_article;
DROP TABLE IF EXISTS t_article;
create table t_article(
    id varchar(100) DEFAULT NULL,
    header varchar(100) DEFAULT NULL,
    name varchar(60) DEFAULT NULL,
    content text DEFAULT NULL,
    author varchar(30) DEFAULT NULL,
    description varchar(100) DEFAULT NULL,
    is_published int(1) DEFAULT NULL,
    is_delete int(1) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    update_time timestamp NOT NULL,
    user_id varchar(100) NOT NULL,
    category_id int(2) NOT NULL,

) DEFAULT CHARSET=utf8
```

因为 update_time 是timestamp类型，也就是时间戳，那么我们给他一个默认值，默认就是当前时间。

改成：

update_time timestamp NOT NULL
DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

在mysql数据库里面运行一下，发现报错了

[Err] 1064 - You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ') DEFAULT CHARSET=utf8' at line 13

哦，原来在属性的最后一行不能有逗号。

```
DROP TABLE IF EXISTS t_article;
DROP TABLE IF EXISTS t_article;
create table t_article(
    id varchar(100) DEFAULT NULL,
    header varchar(100) DEFAULT NULL,
    name varchar(60) DEFAULT NULL,
    content text DEFAULT NULL,
    author varchar(30) DEFAULT NULL,
    description varchar(100) DEFAULT NULL,
    is_published int(1) DEFAULT NULL,
    is_delete int(1) DEFAULT NULL,
    create_time datetime DEFAULT NULL,
    update_time timestamp NOT NULL,
    user_id varchar(100) NOT NULL,
    category_id int(2) NOT NULL,

) DEFAULT CHARSET=utf8
```

看来之前写的方法还有点问题，这边我们先把逗号去掉吧。

再次运行sql语句，OK，成功建表了。

3. 制造测试数据，JUint初探

接下来，我们来虚拟一些数据。

我们在test包下新建一个类，叫做TestInsertOperation，就是测试INSERT操作的意思。

我们用JUint来测试。

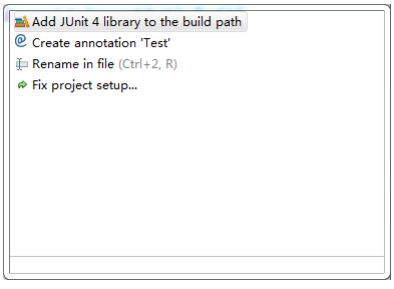
JUnit是一个基于Java语言的单元测试框架，用起来比较方便。它的源代码很轻巧，而且优雅地运用了多种设计模式，应该来说，这是一个非常优秀的框架。

首先在这个TestInsertOperation类中添加一个方法

```
/**
 *  测试：给文章插入数据
 */
@Test
public void insertArticle(){

}
```

@Test是一个注解，加上它以后，才会被JUint测试框架识别。

把光标放在@Test上面，**ctrl + 1**



这个东西就跳出来了，点击第一项，JUint的依赖包就被加载进来了。

接下来，在测试方法 insertArticle 中写上测试代码:

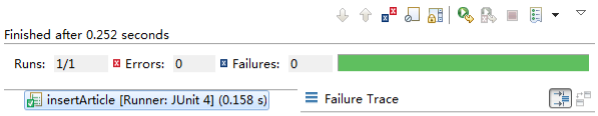
```
String sql = "INSERT INTO t_article(id,header,name,content,author,"
           + "description,is_published,is_delete,create_time,update_time"
           + ",user_id,category_id) VALUES (?,?,?,?,?,?,?,?,?,?,?,?) ";
String id = UUID.randomUUID().toString(); //主键
String header = "Java Web实用技术";
String name  = "如何将MyEclipse项目导入eclipse";
String content = "我们经常会在网上下载一些开源项目，或者从别的地方迁移一些项目进来，但经常会发现导入后各种报错。这是初学java肯定会遇到的问题，本文对一些常见的处理方案做一个总结。（本文将MyEclipse项目导入eclipse的
String author = "Jack";
String description = "解决项目导入的冲突问题...";
int isPublished = 1 ;
int isDelete = 0;
String create_time = "2016-10-19 10:43:10";
String update_time = "2016-10-19 10:43:10";
String userId = "319600c3-550a-4f9f-80cf-deebe2376528";
int categoryId = 2;
DataBaseUtils.update(sql, id,header,name,content,author,description,isPublished,isDelete,create_time,update_time,userId,categoryId);
System.out.println("新增成功！");
```

鼠标双击方法名

```
@Test
public void insertArticle(){
    String sql = "INSERT INTO t_article(id,header,name,content,author,"
               + "description,is_published,is_delete,create_time,update_time"
               + ",user_id,category_id) VALUES (?,?,?,?,?,?,?,?,?,?,?,?) ";
    String id = UUID.randomUUID().toString(); //主键
    String header = "Java Web实用技术";
    String name  = "如何将MyEclipse项目导入eclipse";
    String content = "我们经常会在网上下载一些开源项目，或者从别的地方迁移一些项目进来，但经常
    String author = "Jack";
    String description = "解决项目导入的冲突问题...";
    int isPublished = 1 ;
    int isDelete = 0;
    String create_time = "2016-10-19 10:43:10";
    String update_time = "2016-10-19 10:43:10";
    String userId = "319600c3-550a-4f9f-80cf-deebe2376528";
    int categoryId = 2;
    DataBaseUtils.update(sql, id,header,name,content,author,description,isPubli
    System.out.println("新增成功！");
}
```

按一下F11，开始测试（如果F11不起作用，那么就右键，Run As， JUnit Test）

测试结果：



OK，没有错误。

控制台也没有报错，而且成功打印了 **"新增成功！"** 这几个字。

我已经在库里查到这条数据了，现在，用jdbc的方式将刚刚插入的数据查询出来。

在库里看到它的 ID 为 2145ed72-164a-401c-af29-248625a775b8。

好的，现在新写一个方法来获取这条数据：

```
public void getArticle(){
    String sql = "select * from t_article where id = ?";
    Article article = DataBaseUtils.queryForBean(sql, Article.class, "2145ed72-164a-401c-af29-248625a775b8");
    System.out.println(article);
}
```

测试结果：

Article [id = 2145ed72-164a-401c-af29-248625a775b8,
header = Java Web实用技术,
name = 如何将MyEclipse项目导入eclipse,
content = 我们经常会在网上下载一些开源项目，或者从别的地方迁移一些项目进来，但经常会发现导入后各种报错。这是初学java肯定会遇到的问题，本文对一些常见的处理方案做一个总结。（本文将MyEclipse项目导入eclipse的过程为例，其他情况也可参考这个流程），
author = Jack,
description = 解决项目导入的冲突问题...,
isPublished = 1,
isDelete = 0,
createTime = Wed Oct 19 10:43:10 CST 2016,
updateTime = Wed Oct 19 10:43:10 CST 2016,
userId = 319600c3-550a-4f9f-80cf-deebe2376528,
categoryId = 2]

这样就成功了。

2.2 分类表

分类表的话比较简单，为了简单起见，我们就不写JavaBean了，直接在mysql中建表吧。

建表语句：

```
DROP TABLE IF EXISTS `t_category`;
```

```
CREATE TABLE `t_category` (
  `category_id` int(11) NOT NULL AUTO_INCREMENT,
  `category_name` varchar(20) CHARACTER SET utf8 DEFAULT NULL,
  PRIMARY KEY (`category_id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;
```

ID是自增长的。

制造数据：

```
INSERT INTO `t_category` VALUES ('1', '连载小说');
INSERT INTO `t_category` VALUES ('2', '编程代码类');
INSERT INTO `t_category` VALUES ('3', '生活感悟');
```

insert 操作可以直接在mysql中进行操作，也可以用jdbc来操作。

jdbc操作的代码如下

```
/**
 * 插入分类数据
 */
@Test
public void insertCategory(){
    DataBaseUtils.update("insert into t_category set category_name = ?", "连载小说");
    DataBaseUtils.update("insert into t_category set category_name = ?", "编程代码类");
    DataBaseUtils.update("insert into t_category set category_name = ?", "生活感悟");
}
```

测试一下就行了。

好的，插入完毕后，我们新建一个测试方法来查询一下。

```
/**
 * 获取分类列表
 */
@Test
public void getCategoryList(){
    String sql = "select * from t_category where 1 = 1";
    List list = DataBaseUtils.queryForList(sql);
    System.out.println(list);
}
```

结果：

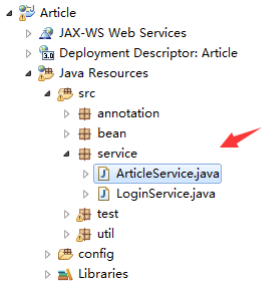
[{category_name=连载小说, category_id=1},
{category_name=编程代码类, category_id=2},
{category_name=生活感悟, category_id=3}]

嗯，OK了。

4. service层开发

上面的测试代码主要是dao部分的，但因为本项目省去了dao层，所以，有什么操作的话，我们直接在service层解决掉算了。

新建一个 ArticleService 类



首页的文章列表：

从静态页面中，我们可以看到，文章被分为几个不同的类别，比如连载小说，就是一个单一的类别，我们应该是通过类别去加载相应的文章。

在数据库表中，连载小说的分类ID为1，那么我们如果想要查询出该分类下的文章，就会写出这样的sql语句：

select * from t_article where category_id = 1;

我们先不管到底怎么和首页对接的，先把后台逻辑写好再说。

在 ArticleService 类中定义一个查询方法

```
/**
 * 通过类别获取文章列表
```

```
 * @param categoryId
 * @param start
 * @param end
 */
public List<Map<String,Object>> getArticlesByCategoryId(Integer categoryId,Integer start,Integer end){
    String sql = "select id,header,name,author,"
        + "description from t_article where 1 = 1 "
        + " and is_delete = 0"
        + " and is_published = 1"
        + " and category_id = ?"
        + "  order by update_time desc limit ?,?";
    return DataBaseUtils.queryForList(sql, categoryId,start,end);
}
```

测试代码：

```
ArticleService ArticleService = new ArticleService();
List list = ArticleService.getArticlesByCategoryId(2,0,10);
System.out.println(list);
```

我测试了一下，应该没问题。sql查询的话，我做了一个简单的排序，就是根据最后更新时间倒序排序。

相信你也已经看出来了，因为我们已经有了 DataBaseUtils 这个工具类，所以大大减少了我们的java代码。

不然的话，我们还需要手动去获取连接，然后生成 PreparedStatement 的实例，一大堆 try catch，最后还要关闭连接。

有了 DataBaseUtils，这些重复的代码就可以省略了。

在这个 getArticlesByCategoryId 方法中，我故意没有把文章内容查询出来。

原因很简单，因为文章内容不需要展示在首页，我就是查询出来也没用。

我把id查出来了，这样的话，当用户通过点击文章封面，进入详情页的时候，就可以获取文章id，有了这个id，我们是不是就可以去数据库把文章内容给查出来了呢？

所以，我们肯定还需要一个方法，就是通过文章id查询出文章内容的方法。

代码：

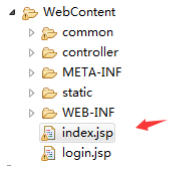
```
/**
 * 通过文章id获取文章内容
 * @param id
 * @return
 */
public List<Map<String,Object>> getContentByArticleId(String id){
    String sql = "select content from t_article where id = ?";
    return DataBaseUtils.queryForList(sql,id);
}
```

测试了一下，也是没问题的哈。

5. 与前台页面对接

好的，后台已经写好了，我们现在和前台对接一下。

打开index.jsp



找到编程代码类：

```
<div class='category'>
    <div class='title'>编程代码类</div>
    <ul class='items'>
        <li class='item'></li>
        <li class='item'></li>
        <div style='clear:both'></div>
    </ul>
</div>
```

现在，我们要把它变成动态的。

首先，在index.jsp页面顶部的地方，导入必要的包。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ page language="java" import="service.ArticleService" pageEncoding="UTF-8"%>
```

然后，新建一个 ArticleService 的实例。

```
<% ArticleService articleService = new ArticleService(); %>
```

接下来，在 编程代码类 的div上方获取 list 数据。

```
<%
    //查询出编程代码类的相关文章
    List<Map<String,Object>>  articles2 = articleService.getArticlesByCategoryId(2, 0, 6);
    pageContext.setAttribute("articles2", articles2);
%>
${articles2}
<div class='category'>
    <div class='title'>编程代码类</div>
    <ul class='items'>
        <li class='item'></li>
        <li class='item'></li>
        <div style='clear:both'></div>
    </ul>
</div>
```

pageContext是JSP九大隐式对象的一员，顾名思义，它的作用域就是当前页面。

${articles2}表示在html代码中显示articles2的具体信息，注意，这个信息是Java代码获取的。

我们只要刷新一下页面，这些代码逻辑就会被执行。

好的，我们刷新一下。

HTTP Status 500 - /index.jsp (line: 2, column: 1) Page directive must not have multiple occurrences of pageencoding

type Exception report

message /index.jsp (line: 2, column: 1) Page directive must not have multiple occurrences of pageencoding

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
org.apache.jasper.JasperException: /index.jsp (line: 2, column: 1) Page directive must not have multiple occurrences of pageencoding
        org.apache.jasper.compiler.DefaultErrorHandler.jspError(DefaultErrorHandler.java:41)
        org.apache.jasper.compiler.ErrorDispatcher.dispatch(ErrorDispatcher.java:275)
        org.apache.jasper.compiler.ErrorDispatcher.jspError(ErrorDispatcher.java:107)
        org.apache.jasper.compiler.Validator$DirectiveVisitor.visit(Validator.java:196)
        org.apache.jasper.compiler.Node$PageDirective.accept(Node.java:571)
        org.apache.jasper.compiler.Node$Nodes.visit(Node.java:2376)
        org.apache.jasper.compiler.Node$Visitor.visitBody(Node.java:2428)
        org.apache.jasper.compiler.Node$Visitor.visit(Node.java:2434)
        org.apache.jasper.compiler.Node$Root.accept(Node.java:464)
        org.apache.jasper.compiler.Node$Nodes.visit(Node.java:2376)
        org.apache.jasper.compiler.Validator.validateDirectives(Validator.java:1813)
        org.apache.jasper.compiler.Compiler.generateJava(Compiler.java:196)
        org.apache.jasper.compiler.Compiler.compile(Compiler.java:356)
        org.apache.jasper.compiler.Compiler.compile(Compiler.java:336)
        org.apache.jasper.compiler.Compiler.compile(Compiler.java:323)
        org.apache.jasper.JspCompilationContext.compile(JspCompilationContext.java:585)
        org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:363)
        org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:396)
        org.apache.jasper.servlet.JspServlet.service(JspServlet.java:340)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

note The full stack trace of the root cause is available in the Apache Tomcat/8.0.23 logs.

报错了。

没关系，看看它说什么。

错误信息：

message /index.jsp (line: 2, column: 1) Page directive must not have multiple occurrences of pageencoding

哦，它说我们must not have，一定不能有。

一定不能有什么呢？继续往下看。

multiple occurrences of pageencoding（多个pageencoding出现）

哦，一定不能出现多个 pageencoding 。

我们来看看页面。

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ page language="java" import="service.ArticleService" pageEncoding="UTF-8"%>
```

嗯，我们真的定义了多个pageEncoding。

好的，我们删掉多余的pageEncoding。

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page language="java" import="java.util.*"%>
<%@ page language="java" import="service.ArticleService"%>
```

再来一次，刷新页面。

[{id=2145ed72-164a-401c-af29-248625a775b8, author=Jack, description=解决项目导入的冲突问题..., name=如何将MyEclipse项目导入eclipse, header=Java Web实用技术}]

效果出来了。

${articles2}变成了：

[{id=2145ed72-164a-401c-af29-248625a775b8, author=Jack, description=解决项目导入的冲突问题..., name=如何将MyEclipse项目导入eclipse, header=Java Web实用技术}]

然后，我们需要用JSTL标签库中的一个叫做 **c:forEach** 标签。

它的作用是循环遍历，我们直接上代码吧。

```
<%
    //查询出编程代码类的相关文章
    List<Map<String,Object>>  articles2 = articleService.getArticlesByCategoryId(2, 0, 6);
    pageContext.setAttribute("articles2", articles2);
%>

<div class='category'>
    <div class='title'>编程代码类</div>
    <ul class='items'>
        <c:forEach items="${articles2}" var="item">
            <li class='item'>
                <div class='item-banner'>
                    <div class='item-header'>${item.header}</div>
                    <div class='item-name'>${item.name}</div>
                    <div class='item-author'>@${item.author} 著</div>
                </div>
```

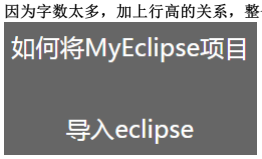
```
                <div class='item-description'>${item.description}</div>
            </li>
        </c:forEach>
        <div style='clear:both'></div>
    </ul>
</div>
```

我们用了一个forEach标签，将${articles2}进行了遍历。因为${articles2}是一个list，所以是可以遍历的。

var="item" 是遍历出来的每一个对象。

效果：

因为字数太多，加上行高的关系，整个封面被挤下来了。



嗯，我手动来调一下css样式吧。

让文章名称强制不换行，溢出部分用 ... 显示

```
.item-name {
    font-size: 22px;
    line-height: 74px;
       white-space:nowrap;
       overflow:hidden;
       text-overflow: ellipsis;
}
```

鼠标划上去的时候，显示一个 tip 提示

```
<div class='item-name' title = "${item.name}">${item.name}</div>
```

这样就OK了。

好的，与前台对接完毕了。

我又弄了几条测试数据。

假模假式的，稍微有那么点样子了。

嗯，今天就到这里了，下一节咱们继续。

源码：点击下载