第一章

【思考题】

请按照以下要求设计实现 PreparedStatement 对象的相关批处理操作。

指定 SQL 语句如下：

String sql = "INSERT INTO users(name,password) VALUES(?,?)";

扫描右方二维码，查看思考题答案！

------答案------

请按照以下要求设计实现 PreparedStatement 对象的相关批处理操作。

```
public class Example01{
 public static void main(String[] args) {
    Connection conn = null;
    PreparedStatement  preStmt = null;
    try {
     // 加载并注册数据库驱动
     conn=JDBCUtils.getConnection();
     String sql = "INSERT INTO users(name,password)VALUES(?,?)";
     preStmt = conn.prepareStatement(sql);
     for (int i = 0; i < 5; i++) {
     preStmt.setString(1, "name" + i);
     preStmt.setString(2, "password" + i);
     preStmt.addBatch();
    }
      preStmt.executeBatch();
    } catch (Exception e) {
      e.printStackTrace();
    } finally { // 释放资源
    JDBCUtils.release(null, preStmt, conn);
    }
   }
}
```

第二章

【思考题】

1、请简述数据库连接池的工作机制。

2、已知存在 src/c3p0-config.xml 配置文件，并且默认配置结点为 itcast。请写出 C3P0 获取数据源的代码。

扫描右方二维码，查看思考题答案！

------第 1 题答案------

1、请思考数据库连接池的工作机制是什么？

数据库连接池在初始化时将创建一定数量的数据库连接放到连接池中，当应用程序访问数据库时并不是直接创建 Connection，而是向连接池“申请”一个 Connection。如果连接池中有空闲的 Connection，则将其返回，否则创建新的 Connection。使用完毕后，连接池会将该 Connection 回收，并交付其他的线程使用，以减少创建和断开数据库连接的次数，提高数据库的访问效率

------第 2 题答案------

2、按要求写出 C3P0 获取数据源的代码。

```
public class Example02 {
public static DataSource ds = null;
// 初始化 C3P0 数据源
static {
// 使用 c3p0-config.xml 配置文件中的 named-config 节点中 name 属性的值
ComboPooledDataSource cpds = new ComboPooledDataSource("itcast");
ds = cpds;
}
public static void main(String[] args) throws SQLException {
System.out.println(ds.getConnection());
}
}
```

第三章

【思考题】

BeanHandler、BeanListHandler 和 BeanMapHandler 实现类是将结果集中的数据封装到对应的 JavaBean 实例中，这也是实际开发中最常用的结果集处理方法。请用代码来展示这三个类的具体使用。

扫描右方二维码，查看思考题答案！

------答案------

如何使用 BeanHandler、BeanListHandler 和 BeanMapHandler 实现类将结果集中的数据封装到对应的 JavaBean 实例中

BeanHandler 的使用：

（1）创建一个 JavaBean 实例类 User

```
public class User {
 private int id;
 private String name;
 private String password;
 public int getId() {
     return id;
 }
 public void setId(int id) {
     this.id = id;
 }
 public String getName() {
     return name;
 }
 public void setName(String name) {
     this.name = name;
 }
 public String getPassword() {
     return password;
 }
 public void setPassword(String password) {
     this.password = password;
 }
}
```

（2）创建类 ResultSetTest3 类

```
import java.sql.SQLException;
import org.apache.commons.dbutils.handlers.BeanHandler;
import cn.itcast.jdbc.example.domain.User;
public class ResultSetTest3 {
```

```
public static void testBeanHandler() throws SQLException {
    BaseDao basedao=new BaseDao();
    String sql="select * from user where id=?";
    User user = (User) basedao.query(sql,new BeanHandler(User.class),1);
    System.out.print("id为1的User对象的name值为："+user.getName());
    }
    public static void main(String[] args) throws SQLException {
    testBeanHandler ();
}
}
```

（3）运行类 ResultSetTest3，由输出结果可以看出，BeanHandler 成功将 id 为 1 的数据存入到了实体对象 user 中。

BeanListHandler 的使用：

（1）创建类 ResultSetTest4，写一个 testBeanListHandler()方法演示 BeanListHandler 类对结果集的处理。

```
import java.sql.SQLException;
import java.util.ArrayList;
import org.apache.commons.dbutils.handlers.BeanListHandler;
import cn.itcast.jdbc.example.domain.User;
public class ResultSetTest4 {
   public static void testBeanListHandler() throws SQLException {
    BaseDao basedao = new BaseDao();
    String sql = "select * from user ";
    ArrayList<User> list = (ArrayList<User>) basedao.query(sql,
            new BeanListHandler(User.class));
    for (int i = 0; i < list.size(); i++) {
        System.out.println("第" + (i + 1) + "条数据的username值为:"
                + list.get(i).getName());
    }
}
   public static void main(String[] args) throws SQLException {
    testBeanListHandler ();
}
}
```

（2）运行类 ResultSetTest4，由输出结果可以看出，testBeanListHandler ()方法可以将每一行的数据都封装到 user 实例中，并存放到 list 中。

BeanMapHandler 的使用：

（1）创建类 ResultSetTest5，写一个 testBeanMapHandler()方法来演示 BeanMapHandler 类对结果集的处理。

```
import java.sql.SQLException;
import java.util.Map;
import org.apache.commons.dbutils.handlers.BeanMapHandler;
import cn.itcast.jdbc.example.domain.User;
public class ResultSetTest5 {
```

```
    public static void testBeanMapHandler() throws SQLException {
        String sql = "select id, name, password from user";
        Map<Integer, User> map = (Map<Integer, User>) BaseDao.query(sql,
                new BeanMapHandler<Integer, User>(User.class, "id"));
        // zhangsan 的 id 值为 1
        User u = map.get(1);
        String uName = u.getName();
        String uPassword = u.getPassword();
    System.out.print("id 为 1 的 User 对象的 name 值为：" + uName + ",password 值为："
                + uPassword);
    }
    public static void main(String[] args) throws SQLException {
        testBeanMapHandler();
    }
    }
```

（2）运行类 ResultSetTest5，由输出结果可以看出，testBeanMapHandler()方法成功的将每一行的数据都封装到 user 实例中，并存放到 Map 中。

第四章

【思考题】

请思考什么是装饰者设计模式？并使用具体例子来展示普通类在使用装饰者设计模式后的变化。

扫描右方二维码，查看思考题答案！

------答案------

2、什么是装饰者设计模式？并举例展示。

所谓装饰设计模式，指的是通过包装类的方式，动态增强某个类的功能。下面使用装饰者设计模式，对 Bicycle 类进行简单包装。

（1）编码

```
class Bicycle {
private String bicycleType;
public Bicycle(String bicycleType){
 this.bicycleType = bicycleType;
}
public void ride(){
 System.out.println(bicycleType + "可以骑");
}
```

```
}
class MountainBike{
 public Bicycle bicycle;
 public MountainBike(Bicycle bicycle){
     this.bicycle = bicycle;
 }
 public void ride(){
     bicycle.ride();
     System.out.println("可以变速行驶");
 }
}
public class BicycleDemo{
 public static void main(String[] args) {
     Bicycle bicycle = new Bicycle("普通自行车");
     System.out.println("-------普通自行车-------");
     bicycle.ride();
     MountainBike mountainBike = new MountainBike(bicycle);
     System.out.println("--------包装后的普通自行车-----");
     mountainBike.ride();
 }
}
```

（2）结论

MountainBike 对 Bicycle 类进行包装后，MountainBike 类型的对象不仅具有了“可以骑”的功能，还具有了“可以变速行驶”的功能。

第五章

【思考题】

请用代码展示如何对 ServletContext、HttpSession 和 ServletRequest 这三个域对象属性的变更进行监听？。

扫描右方二维码，查看思考题答案！

-----答案------

监听域对象属性的变更。

（1）编写一个 testattribute.jsp 页面，以观察各个域对象属性事件监听器的作用。

```
<%@ page language="java" contentType="text/html; charset=utf-8"
 pageEncoding="utf-8"%>
<html>
<head>
<title>Insert title here</title>
```

```
    </head>
    <body>
        <h3>这是一个测试对象属性信息监听器的页面</h3>
        <%
            getServletContext().setAttribute("username", "itcast");
            getServletContext().setAttribute("username", "itheima");
            getServletContext().removeAttribute("username");
            session.setAttribute("username", "itcast");
            session.setAttribute("username", "itheima");
            session.removeAttribute("username");
            request.setAttribute("username", "itcast");
            request.setAttribute("username", "itheima");
            request.removeAttribute("username");
        %>
    </body>
    </html>
```

（2）编程一个 MyAttributeListener 类，该类实现了 ServletContextAttributeListener、HttpSessionAttributeListener 和 ServletRequestAttributeListener 接口，并实现该接口中的所有方法。

```
    import javax.servlet.*;
    import javax.servlet.http.*;
public class MyAttributeListener implements ServletContextAttributeListener,
            HttpSessionAttributeListener, ServletRequestAttributeListener {
        public void attributeAdded(ServletContextAttributeEvent sae) {
            String name = sae.getName();
            System.out.println("ServletContext 添加属性: " + name + "="
                    + sae.getServletContext().getAttribute(name));
        }
        public void attributeRemoved(ServletContextAttributeEvent sae) {
            String name = sae.getName();
            System.out.println("ServletContext 移除属性: " + name);
        }
        public void attributeReplaced(ServletContextAttributeEvent sae) {
            String name = sae.getName();
            System.out.println("ServletContext 替换属性: " + name + "="
                    + sae.getServletContext().getAttribute(name));
        }
        public void attributeAdded(HttpSessionBindingEvent hbe) {
            String name = hbe.getName();
            System.out.println("HttpSession 添加属性: " + name + "="
                    + hbe.getSession().getAttribute(name));
        }
        public void attributeRemoved(HttpSessionBindingEvent hbe) {
            String name = hbe.getName();
```

```
            System.out.println("HttpSession 移除属性: " + name);
        }
        public void attributeReplaced(HttpSessionBindingEvent hbe) {
            String name = hbe.getName();
            System.out.println("HttpSession 替换属性: " + name + "="
                    + hbe.getSession().getAttribute(name));
        }
        public void attributeAdded(ServletRequestAttributeEvent sra) {
            String name = sra.getName();
            System.out.println("ServletRequest 添加属性: " + name + "="
                    + sra.getServletRequest().getAttribute(name));
        }
        public void attributeRemoved(ServletRequestAttributeEvent sra) {
            String name = sra.getName();
            System.out.println("ServletRequest 移除属性: " + name);
        }
        public void attributeReplaced(ServletRequestAttributeEvent sra) {
            String name = sra.getName();
            System.out.println("ServletRequest 替换属性: " + name + "="
                    + sra.getServletRequest().getAttribute(name));
        }
    }
```

（3）在 web.xml 文件中，部署 MyAttributeListener 事件监听器。

```
    <listener>
        <listener-class>
            cn.itcast.chapter05.listener.MyAttributeListener
        </listener-class>
    </listener>
```

（4）访问 testattribute.jsp 页面，查看控制台窗口的显示结果。

第六章

【思考题】

编写一个实现文件上传功能的程序（限制上传文件的大小和类型。）

扫描右方二维码，查看思考题答案！

------1 题答案------

（1）编写 upload.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="gbk"%>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>通过 commons-fileupload 限制上传文件类型</title>
        <style type="text/css">
        table{
            font-size: 13px;
        }
        input{
            font-size: 12px;
        }
    </style>
  </head>
   <body>
    <form method="post" action="UploadServlet" enctype="multipart/form-data">
    <table background="bg.jpg" width="400" height="200">
    <tr>
     <td height="20" colspan="2"></td>
    </tr>
    <tr>
     <td align="center" colspan="2">【应用 commons-fileUpload 限制上传文件类型】</td>
    </tr>
      <tr>
     <td height="20" colspan="2"></td>
    </tr>
    <tr>
     <td align="right">选择文件：</td>
     <td > <input type="file" name="file1"></td>
    </tr>
    <tr>
     <td align="center"></td>
     <td > <input type="submit" value="开始上传"></td>
    </tr>
      <tr>
     <td height="40" colspan="2">
      <%
            String result = (String) request.getAttribute("result");
            if (result != null) {
                out.println("<script >alert('" + result + "');</script>");
            }
        %>
    </td>
   </tr>
   </table>
```

```
    </form>
  </body>
</html>
```

（2）创建 UploadServlet 类

```
package cn.itcast.chapter06;
import java.io.*;
import java.util.*;
import org.apache.commons.fileupload.*;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.apache.commons.io.filefilter.SuffixFileFilter;
import javax.servlet.*;
import javax.servlet.http.*;
public class UploadServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doPost(request,response);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse
            response) throws ServletException, IOException {
        //定义上传文件的服务器路径
         String uploadPath = this.getServletContext().getRealPath("/")+"upload";
        //根据该路径创建 File 对象
        File uploadFolder = new File(uploadPath);
        //如果路径不存在，则创建
        if(!uploadFolder.exists())
            uploadFolder.mkdirs();
        String message = "文件上传成功! ";
        try{
            if(ServletFileUpload.isMultipartContent(request)){
                //创建磁盘工厂，用来配置上传组件 ServletFileUpload
                DiskFileItemFactory factory = new DiskFileItemFactory();
                //设置内存中允许存储的字节数
                factory.setSizeThreshold(20*1024);
                //设置存放临时文件的目录
                factory.setRepository(factory.getRepository());
                //创建新的上传文件句柄
                ServletFileUpload upload = new ServletFileUpload(factory);
                //定义上传文件的大小
                int maxSize = 5*1024*1024; //定义上传文件的大小
                //从请求中得到所有上传域列表
                List<FileItem> files = upload.parseRequest(request);
                //限制上传的文件类型
                String[] suffixs =new String[]{".exe",".bat",".jsp"};
```

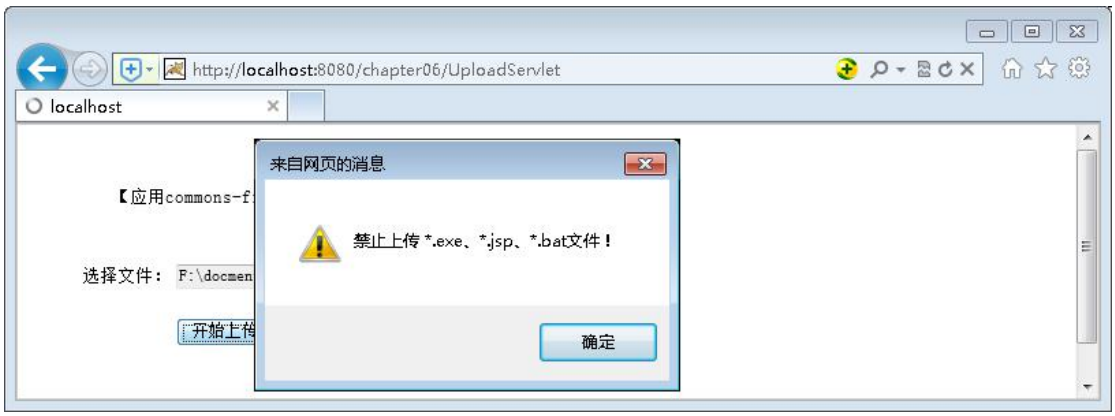
```
            SuffixFileFilter filter = new SuffixFileFilter(suffixs);
            //遍历上传文件集合
            for(FileItem fileItem:files){
            //忽略其他不是文件域的所有表单信息
                if(!fileItem.isFormField()){
                    //获取文件全路径名
                    String filePath = fileItem.getName();
                    String fileName="";
                    int startIndex = filePath.lastIndexOf("\\");
                    //对文件名进行截取
                    if(startIndex!=-1){
                        fileName = filePath.substring(startIndex+1);
                    }else{
                        fileName=filePath;
                    }
                    //限制文件大小
                    if(fileItem.getSize()>maxSize){
                        message = "上传文件不得超过 5MB！";
                        break;
                    }
                    if((fileName == null)
                        ||(fileName.equals(""))&&(fileItem.getSize()==0))
                        continue;
                    //在上传路径创建文件对象
                    File file = new File(uploadPath,fileName);
                    boolean res = filter.accept(file);
                    if(res){
                        message = "禁止上传 *.exe、*.jsp、*.bat 文件！";
                        break;
                    }else{
                        //向文件写数据
                        fileItem.write(file);
                    }
                }
            }
        }
    }
    catch(Exception ex){
        ex.printStackTrace();
    }
    //将提示信息保存在 request 对象中
    request.setAttribute("result", message);
    request.getRequestDispatcher("upload.jsp").forward(request,
    response);
```

```
    }
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }
}
```

（3）在 web.xml 中配置 UploadServlet 类的信息

```
<servlet>
  <servlet-name>UploadServlet</servlet-name>
  <servlet-class>cn.itcast.chapter06.UploadServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>UploadServlet</servlet-name>
  <url-pattern>/UploadServlet</url-pattern>
</servlet-mapping>
```

（4）启动服务器，在浏览器地址栏中输入 http://localhost:8080/chapter06/upload.jsp，运行结果如下图所示。



第七章

【思考题】

自定义并使用 EL 函数实现将文本框中的内容反向输出的功能。

扫描右方二维码，查看思考题答案！

------1 题答案------

（1）创建 ELReverse.java 文件

```
package cn.itcast.chapter07.custom;
public class ELReverse {
    // 反向输出
    public static String reverse(String text) {
        return new StringBuffer(text).reverse().toString();
```

```
    }
}
```

（2）编写 function.tld 文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
  version="2.0">
  <description>library</description>
  <display-name>functions</display-name>
  <tlib-version>1.1</tlib-version>
  <short-name>fn</short-name>
  <function>
    <description>reverse</description>
    <name>reverse</name>
    <function-class>cn.itcast.chapter07.custom.ELReverse</function-class>
    <function-signature>java.lang.String
reverse( java.lang.String )</function-signature>
  </function>
</taglib>
```

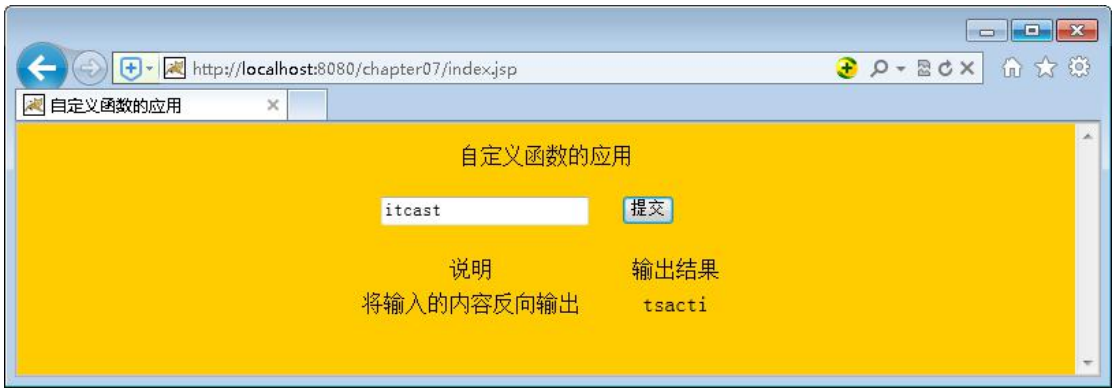
（3）编写 index.jsp 文件

```
<%@ page contentType="text/html; charset=gb2312" language="java"
import="java.sql.*" errorPage="" %>
<%@ taglib prefix="fn" uri ="/WEB-INF/funcation.tld"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312"/>
<link href="css/style.css" type="text/css" rel="stylesheet" />
<title>自定义函数的应用</title>
</head>
<style type="text/css">
<!--
body {
    background-color: #FFCC00;
}
-->
</style>
<body>
<p align="center">自定义函数的应用</p>
<form name="form1" method="post" action="index.jsp">
  <table width="300" border="0" align="center" cellpadding="0" cellspacing="0">
    <tr>
      <td width="272" align="center">
```

```
        <input type="text" name="foo" value="${param.foo}">  
    <input type="submit" name="Submit" value="提交">
      </td>
    </tr>
  </table>
</form>

<table width="300" border="0" align="center" cellpadding="0" cellspacing="0">
  <tr align="center">
    <td width="189" height="25">说明</td>
    <td width="111">输出结果</td>
  </tr>
  <tr align="center">
    <td height="25">将输入的内容反向输出</td>
    <td>${fn:reverse(param.foo)}</td>
  </tr>
</table>
</body>
</html>
```

（4）启动服务器，访问 http://localhost:8080/chapter07/index.jsp，运行结果如下图所示。



第八章

【思考题】

编写一个 index1.jsp 文件，应用 JSTL 核心库中的<c:forEach>标签输出 10 以内的全部奇数。

扫描右方二维码，查看思考题答案！

------1 题答案------

编写 index1.jsp 文件

```
<%@ page language="java" contentType="text/html; charset=GB18030"
    pageEncoding="GB18030"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=GB18030">
<title>列举 10 以内的全部奇数</title>
</head>
<body>
    <b>10 以内的全部奇数为：</b>
    <c:forEach var="i" begin="1" end="10" step="2">
        ${i }  
    </c:forEach>
</body>
</html>
```

第九章

【思考题】

实现一个自定义简单标签，模拟 JSTL 的<c:foreach>标签的功能。

扫描右方二维码，查看思考题答案！

------1 题答案------

步骤如下：

1）编写标签<itcast:foreach>的标签处理器类 MoniforEachTag 继承 BodyTagSupport 类

2）在 simpletag.tld 文件中增加一个 Tag 元素，对标签处理器类 MoniforEachTag 进行注册。

3）编写 foreach.jsp 文件对自定义标签进行测试。

具体代码实现如下：

（1）MoniforEachTag.java

```
package cn.itcast.chapter09.simpletag;
import java.util.Collection;
import java.util.Iterator;
import java.util.Map;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.BodyTagSupport;
public class MoniforEachTag extends BodyTagSupport {
    private static final long serialVersionUID = 1L;
    private String var;
    private Iterator<?> iterator;
```

```
    public String getVar() {
        return var;
    }
    public void setVar(String var) {
        this.var = var;
    }
    public void setItem(Object item) {
        if (item instanceof Map) {
            Map items = (Map) item;
            this.iterator = items.entrySet().iterator();
        } else {
            Collection<?> c = (Collection) item;
            this.iterator = c.iterator();
        }
    }
    public int doAfterBody() throws JspException {
        if (this.process()) {
            return EVAL_BODY_AGAIN;
        } else {
            return EVAL_PAGE;
        }
    }
    public int doStartTag() throws JspException {
        if (this.process())
            return EVAL_BODY_INCLUDE;
        else
            return EVAL_PAGE;
    }
    private boolean process() {
        if (null != iterator && iterator.hasNext()) {
            Object item = iterator.next();
            pageContext.setAttribute(var, item);
            return true;
        } else
            return false;
    }
}
```

（2）simpletag.tld

```
<?xml version="1.0" encoding="GBK" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">
```

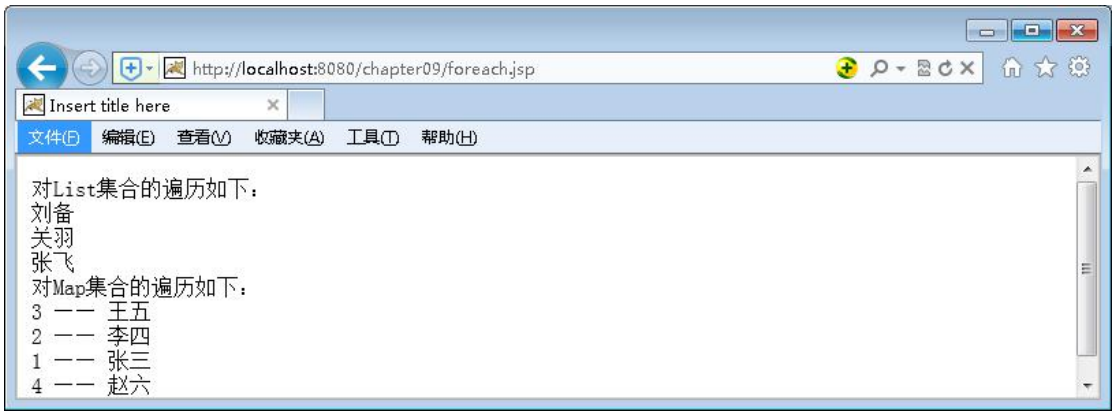
```
    <tlib-version>1.0</tlib-version>
    <short-name>itcast</short-name>
    <uri>http://www.itcast.cn</uri>
    <tag>
        <name>foreach</name>
        <tag-class>cn.itcast.chapter09.simpletag.MoniforEachTag</tag-class>
        <body-content>JSP</body-content>
        <attribute>
            <name>var</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>item</name>
            <rtexprvalue>true</rtexprvalue>
            <type>java.lang.Object</type>
        </attribute>
    </tag>
</taglib>
```

（3）foreach.jsp

```
<%@ page language="java" pageEncoding="GBK"%>
<%@taglib uri="http://www.itcast.cn" prefix="itcast"%>
<%@page import="java.util.*"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<%
   List<String> list = new ArrayList<String>();
   list.add("刘备");
   list.add("关羽");
   list.add("张飞");
   Map map = new HashMap();
   map.put("1","张三");
   map.put("2","李四");
   map.put("3","王五");
   map.put("4","赵六");
%>
<body>
对 List 集合的遍历如下：<br/>
<itcast:foreach var="l" item="<%=list %>" >
   ${l}<br/>
</itcast:foreach>
```

```
对 Map 集合的遍历如下：<br/>
<itcast:foreach var="m" item="<%=map %>" >
    ${m.key} —— ${m.value}<br/>
</itcast:foreach>
</body>
</html>
```

（4）启动程序，在浏览器地址栏中请求 http://localhost:8080/chapter09/foreach.jsp，运行结果如下图所示。



第十章

【思考题】

1、请简要概述什么是国际化？

2、请说出用于设置统一的请求消息的字符集编码的标签，并对该标签的功能特点进行解释。

扫描右方二维码，查看思考题答案！

------1 题答案------

1、请简要概述什么是国际化？

所谓的国际化就是指软件在开发时就应该具备支持多种语言和地区的功能，也就是说开发的软件能同时应对不同国家和地区的用户访问，并针对不同国家和地区的用户，提供相应的、符合来访者阅读习惯的页面和数据。

------2 题答案------

2、请列举出国际化标签库中用于设置全局信息的标签，并对该标签的功能进行解释。

<fmt:requestEncoding>标签：用于设置统一的请求消息的字符集编码，该标签内部调用request.setCharacterEncoding()方法，以便 Web 容器将请求消息中的参数值按该字符集编码转换成 Unicode 字符串返回。

第十一章

【思考题】

请简单描述购物车模块的设计思路？

扫描右方二维码，查看思考题答案！

------1 题答案------

1、请简单描述购物车模块的设计思路？

设计思路：

购物车模块的功能是基于 Session 实现的，Session 充当了一个临时信息存储平台，对购物车中商品的操作实质是对 Session 中数据的操作，当 Session 失效后，保存的购物车信息也将全部丢失。在购物车中应该具备的功能有添加选购的新商品、自动更新商品数量、清空购物车、删除购物车中的单件商品、自动调整商品的总价格和生成订单信息等，在操作购物车各项功能之前必须保证用户是已登录状态。

第十二章

【思考题】

请简单描述商品管理模块的设计思路和实现流程？

扫描右方二维码，查看思考题答案！

------1 题答案------

1、请简单描述商品管理模块的设计思路和实现流程？

设计思路：

商品管理是对图书信息的管理，管理系统中的商品管理模块主要实现的是查询商品信息、添加商品信息、编辑商品信息和删除商品信息这四个功能。

用户进入模块首先需要进行查询商品信息操作，将查询的结果展示在首页页面的商品列表中。商品列表中的每一行数据表示的就是一个商品信息的展示，针对每一个商品我们可以进行编辑和删除操作，以便对商品的信息进行更改和对系统中不需要的商品信息进行删除。新商品入库我们时，还需要进行商品信息添加操作，在商品管理首页面上添加一个按钮，点击按钮弹出商品添加的页面，填写具体信息然后保存，将商品信息保存在数据库中。

实现流程：

商品管理各功能的实现流程大致相同，这里以添加商品信息为例来说明。它的功能流程为，点击商品管理首页面上的添加按钮->添加页面->填写商品信息->保存商品信息->回到首页面。该功能的实现流程是，从添加页面提交表单到 Servlet，再从 Servlet 到调用 DAO 层的方法将提交的数据保存到数据表中。