Spring开发web项目

```
1、Spring的普通java项目有main方法，执行代码是执行main方法的
    在main方法中创建容器对象
    ApplicationContext ac =
ClassPathXmlApplicationContext("applicationContext.xml");

2、web项目是在Tomcat容器中运行的。和服务器一起运行，项目一直运行
实例步骤：
    1）、创建maven项目，选择webapp模板
    2）、加入依赖，拷贝e-spring-mybatis-05中的依赖
        加入jsp和Servlet的依赖
    3）、拷贝e-spring-mybatis-05中的代码和配置文件
    4）、创建一个jsp页面作为注册页面，参数有loginName、loginPwd、realName
    5）、创建Servlet实现类，接收参数请求，调用service类中的方法，底层调用dao方法完成数据插
入，完成注册
    6）、创建一个jsp页面作为转发页面显示结果，也可以直接输出在浏览器上
```

新建一个maven的web项目模块，i-spring-web-09，骨架模板选择maven-archetype-webapp，使用e-spring-mybatis-05中的代码和配置，主要实现的功能是注册功能。

1）、选择的骨架模板需要自己创建java主程序目录和资源目录。还有就是web.xml文件中使用的约束文件等部分使用的是最低版本的，需要自己修改，步骤：Project Structure->Modules->选中web项目，打开，选择Web->右边的Deployment Descriptors中将原来的删掉，选中点击-号->在点击+号->点击web.xml->然后会为你自动选择4.0版本的，然后名字改一下（不改不生成）ok就行了，最后再把名字改回原来的

2）、加入jsp和Servlet依赖

```
<!--servlet依赖-->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
      <scope>provided</scope>
    </dependency>

    <!--jsp依赖-->
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>jsp-api</artifactId>
      <version>2.2.1-b03</version>
      <scope>provided</scope>
    </dependency>
```

3)、创建自定义异常exceptions.InformationNotLegalException

```
package com.studymyself.exceptions;

public class InformationNotLegalException extends RuntimeException{

    public InformationNotLegalException() {
        super();
    }

    public InformationNotLegalException(String message) {
        super(message);
    }
}
```

4）、创建一个controller包，其中新建一个Servlet实现类，实现注册功能，如下com.studymyself.controller.RegisterServlet

```
package com.studymyself.controller;

import com.studymyself.entity.User;
import com.studymyself.exceptions.InformationNotLegalException;
import com.studymyself.service.UserService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

public class RegisterServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

    }

    /**
     * 调用service中的方法完成用户注册
     * @param request
     * @param response
     * @throws ServletException
     * @throws IOException
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        //告诉Tomcat服务器POST请求体中的信息是UTF-8编码
```

```
        request.setCharacterEncoding("UTF-8");
        //获取请求中的注册信息
        String loginName = request.getParameter("loginName");
        String loginPwd = request.getParameter("loginPwd");
        String realName = request.getParameter("realName");

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        /**
         * 一般方式获取容器对象和容器中的对象
         */
        ApplicationContext ac = new
ClassPathXmlApplicationContext("applicationContext.xml");
        //输出容器对象信息
        out.print("容器对象信息：" +ac);
        UserService userService = (UserService) ac.getBean("userService");

        User user = new User(loginName,loginPwd,realName);
        int count = 0;
        if (loginName!=null || loginPwd!=null ||realName!=null
            || loginName!=""|| loginPwd!="" ||realName!=""
        ){
            count = userService.addUser(user);
        }
        if (count!=0){
            out.print("<br>用户：<br>"+user+"<br>注册成功!");
           //
request.getRequestDispatcher("/result.jsp").forward(request,response);
        }else {
            out.print("用户名或密码或真实姓名不合法!");
            throw new InformationNotLegalException("用户名或密码或真实姓名不合法");
        }
    }

}
```

5）、web.xml文件如下

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <servlet>
        <servlet-name>RegisterServlet</servlet-name>
        <servlet-class>com.studymyself.controller.RegisterServlet</servlet-
class>
    </servlet>
    <servlet-mapping>
        <servlet-name>RegisterServlet</servlet-name>
        <url-pattern>/regs</url-pattern>
    </servlet-mapping>
```

```
</web-app>
```

6）、创建一个index.jsp页面作为欢迎和注册页面，其中内容是输入注册信息

```
<%--
  Created by IntelliJ IDEA.
  User: 钟荣杰
  Date: 2021/2/13
  Time: 17:02
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>注册页面</title>
</head>
<body>
    <p>注册用户</p>
    <form action="" method="post">
        <table>
            <tr>
                <td>登录名</td>
                <td><input type="text" name="loginName"></td>
            </tr>
            <tr>
                <td>登录密码</td>
                <td><input type="password" name="loginPwd"></td>
            </tr>
            <tr>
                <td>真实姓名</td>
                <td><input type="text" name="realName"></td>
            </tr>
            <tr>
                <td><input type="submit" name="注册"></td>
            </tr>
        </table>
    </form>
</body>
</html>
```

7）、启动Tomcat服务器进行测试。当我们在服务器运行项目期间，进行两次注册用户时，输出的容器对象信息如下：

```
容器对象信息：org.springframework.context.support.ClassPathXmlApplicationContext@5f0b2fc1, started on Sat Feb 13 18:29:54 CST 2021
用户：
User{id=null, loginName='110120119@email.com', loginPwd='88888888', realName='å□½å®¶æ³□å□¶æ□»å±□'}
注册成功!

容器对象信息：org.springframework.context.support.ClassPathXmlApplicationContext@5796b1e3, started on Sat Feb 13 18:34:15 CST 2021
用户：
User{id=null, loginName='666666@email.com', loginPwd='666666', realName='AlanStation'}
注册成功!
```

上面出现乱码的原因是接收前端请求体中的信息，我们没有告诉Tomcat该信息是UTF-8编码的，所以接收的中文信息是乱码。具体解决可以回顾前面servlet学习部分

对比两次的容器对象信息，我们知道，每一次发送注册的请求都会新创建容器对象，导致容器总是创建Spring配置文件中配置的javabean，如果请求很频繁的话就会占用服务器大量的内存，资源耗费高。所以需要改进。

8）、改进：web项目中容器对象只需要被创建一次，容器中的对象也只需要创建一次。所以需要把创建的容器对象放到项目的全局作用域ServletContext中，使用的时候从里面拿出来就可以了。

实现：使用监听器，在ServletContext对象被创建的同时创建容器对象，然后将容器对象存到ServletContext中

监听器作用：

```
1、创建容器对象，执行
ApplicationContext ac = new
ClassPathXmlApplicationContext("applicationContext.xml");
2、把容器对象ac存到ServletContext中，执行
ServletContext.setAttribute(key,ac);
这个监听器可以自己创建，也可以使用框架中提供好的ContextLoaderListener类，就像是一个工具类。
```

需要我们在pom文件中添加有监听器的依赖：

```
 <!--可以使用监听器的依赖-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>5.2.3.RELEASE</version>
    </dependency>
```

然后在web.xml文件中注册监听器

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <servlet>
        <servlet-name>RegisterServlet</servlet-name>
        <servlet-class>com.studymyself.controller.RegisterServlet</servlet-
class>
    </servlet>
    <servlet-mapping>
        <servlet-name>RegisterServlet</servlet-name>
        <url-pattern>/regs</url-pattern>
    </servlet-mapping>

    <!--注册监听器:ContextCleanupListener
        监听器被创建对象，会读取/WEB-INF/applicationContext.xml
```

```
        监听器读取Spring配置文件是为了创建ApplicationContext容器对象
        而路径/WEB-INF/applicationContext.xml是监听器默认读取Spring配置文件的路径
        但是我们的存放在的是resources目录下，编译后存储的位置是target/classes目录下
        所以我们需要修改监听器的读取路径，使用context-param标签
    -->
    <!--自定义监听器读取的Spring配置文件路径-->
    <context-param>
        <!--contextConfigLocation:表示配置文件的路径，路径前要加classpath-->
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>
    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

</web-app>
```

首先我们可以到ContextLoaderListener类中查看一下源代码，可以看到里面有这样一个创建容器对象的方法contextInitialized(...)：

```
public class ContextLoaderListener extends ContextLoader implements
ServletContextListener {

    无参数有参数的构造方法...

    public void contextInitialized(ServletContextEvent event) {
        this.initWebApplicationContext(event.getServletContext());
    }

   ...
    }
}
```

可以看到方法里面调用了一个initWebApplicationContext(event.getServletContext())方法，该方法传进去的是ServletContext全局范围对象，我们再点击该方法查看源代码，该方法是ContextLoader类中的，其中主要语句如下：

```
 ...
this.context = this.createWebApplicationContext(servletContext);
...
 servletContext.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_
ATTRIBUTE, this.context);
```

首先第一句是创建容器对象的语句，返回的是WebApplicationContext类型的对象

第二句是将容器对象放进ServletContext对象中，key的值是WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE

因为ApplicationContext对象是javaSE项目中创建的容器对象，而WebApplicationContext是web项目中创建的容器对象，这个类是继承ApplicationContext类的，我们可以到WebApplicationContext类中查看源代码：

```
public interface WebApplicationContext extends ApplicationContext {
    String ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE =
WebApplicationContext.class.getName() + ".ROOT";
    String SCOPE_REQUEST = "request";
    String SCOPE_SESSION = "session";
    String SCOPE_APPLICATION = "application";
    String SERVLET_CONTEXT_BEAN_NAME = "servletContext";
    String CONTEXT_PARAMETERS_BEAN_NAME = "contextParameters";
    String CONTEXT_ATTRIBUTES_BEAN_NAME = "contextAttributes";

    @Nullable
    ServletContext getServletContext();
}
```

从上面的可以看出，key值是该接口类的一个String类型属性，继承ApplicationContext，到时获取容器对象时key值填的是ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE

所以Servlet实现类可改为如下内容：

```
package com.studymyself.controller;

import com.studymyself.entity.User;
import com.studymyself.exceptions.InformationNotLegalException;
import com.studymyself.service.UserService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.context.support.WebApplicationContextUtils;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;


public class RegisterServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

    }

    /**
     * 调用service中的方法完成用户注册
     * @param request
     * @param response
     * @throws ServletException
     * @throws IOException
     */
    @Override
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        //告诉Tomcat服务器POST请求体中的信息是UTF-8编码
        request.setCharacterEncoding("UTF-8");
        //获取请求中的注册信息
        String loginName = request.getParameter("loginName");
        String loginPwd = request.getParameter("loginPwd");
        String realName = request.getParameter("realName");

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        /**
         * 一般方式获取容器对象和容器中的对象
         */
//         ApplicationContext ac = new
ClassPathXmlApplicationContext("applicationContext.xml");
//         //输出容器对象信息
//         out.print("容器对象信息：" +ac);
//         UserService userService = (UserService) ac.getBean("userService");

        /**
         * 获取全局范围内的容器对象
         */
        ApplicationContext ac = null;

//         //容器对象在ServletContext中的key值
//         String key =
WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE;
//         //获取容器对象,获取ServletContext对象的this可以省略
//         Object attr = getServletContext().getAttribute(key);
//         if(attr instanceof ApplicationContext){
//             ac = (ApplicationContext) attr;
//         }

        //使用框架中的工具类方法获取容器对象
        ServletContext sc = getServletContext();
        //工具类中已经帮你强制转换好了
        ac = WebApplicationContextUtils.getWebApplicationContext(sc);

        //输出容器对象信息
        out.print("容器对象信息：" +ac);

        UserService userService = (UserService) ac.getBean("userService");
        User user = new User(loginName,loginPwd,realName);
        int count = 0;
        if (loginName!=null && loginPwd!=null && realName!=null
            && loginName!=""&& loginPwd!="" &&realName!=""
        ){
            count = userService.addUser(user);
        }
        if (count!=0){
            out.print("<br>用户：<br>"+user+"<br>注册成功!");
           //
request.getRequestDispatcher("/result.jsp").forward(request,response);
        }else {
            out.print("用户名或密码或真实姓名不合法!");
```

```
            throw new InformationNotLegalException("用户名或密码或真实姓名不合法");
        }
    }

}
```

可以看到两次请求结果截图如下：

容器对象信息：Root WebApplicationContext, started on Sat Feb 13 22:17:27 CST 2021
用户：
User{id=null, loginName='admin@mail.com', loginPwd='123456', realName='管理员'}
注册成功!

容器对象信息：Root WebApplicationContext, started on Sat Feb 13 22:17:27 CST 2021
用户：
User{id=null, loginName='696969@mail.com', loginPwd='9999999', realName='制裁者'}
注册成功!

上面的容器对象，都是同一个容器对象