优极限

“极限教育，挑战极限”

www.yjxxt.com

极限教育，挑战极限。优极限是一个让 95%的学生年薪过 18 万的岗前培训公司，让我们的学员具备优秀的互联网技术和职业素养，勇攀高薪，挑战极限。公司位于上海浦东，拥有两大校区，共万余平。累计培训学员超 3 万名。我们的训练营就业平均月薪 19000，最高年薪 50 万。

核心理念：让学员学会学习，拥有解决问题的能力，拿到高薪职场的钥匙。

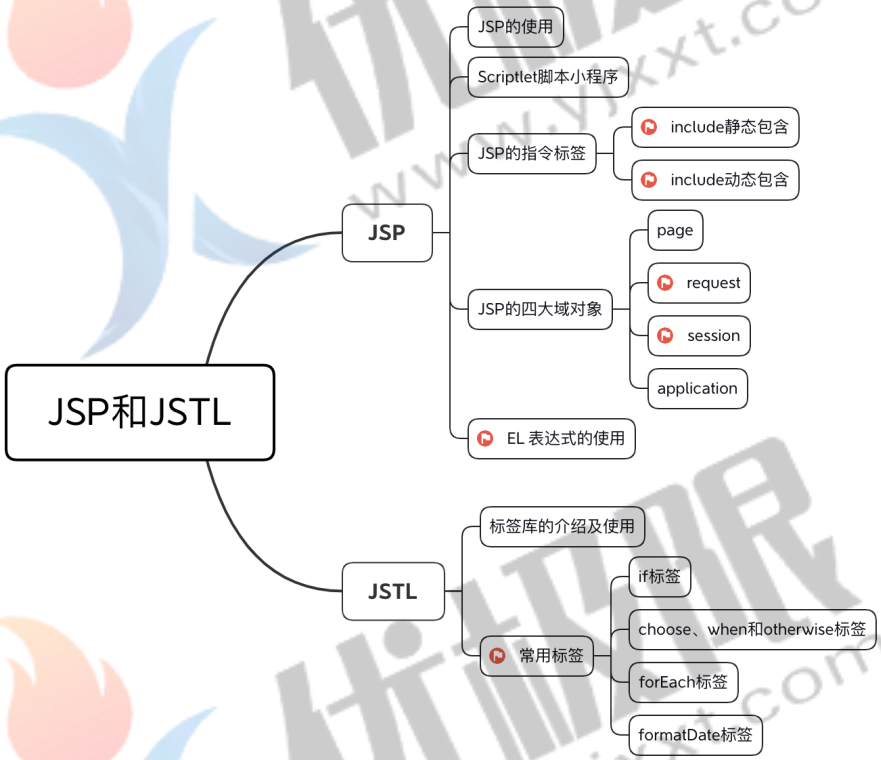
项目驱动式团队协作、一对一服务、前瞻性思维、教练式培养模型-培养你成为就业明星。首创的老学员项目联盟给学员充分的项目、技术支撑，利用优极限平台这根杠杆，不断挑战极限，勇攀高薪，开挂人生。

扫码关注优极限微信公众号：

（获取最新技术相关资讯及更多源码笔记）

JSP 和 JSTL

主要内容



JSP

JSP的基础语法

简介

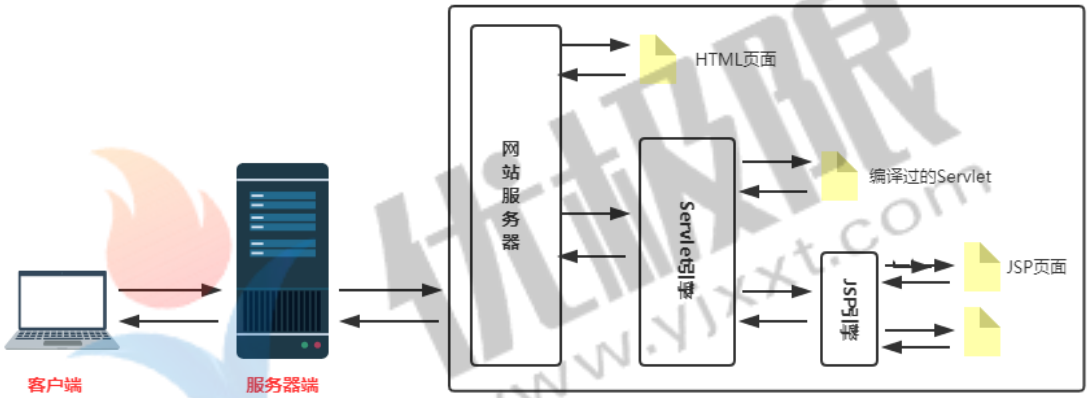
JSP：Java Server Page SUN 公司提供的动态网页编程技术，是 Java Web 服务器端的动态资源。

它相比 html 而言，html 只能为用户提供静态数据，而 Jsp 技术允许在页面中嵌套 java 代码，为用户提供动态数据。

相比 servlet 而言，servlet 很难对数据进行排版，而 jsp 除了可以用 java 代码产生动态数据的同时，也很容易对数据进行排版。

不管是 JSP 还是 Servlet，虽然都可以用于开发动态 web 资源。但由于这 2 门技术各自的特点，在长期的软件实践中，人们逐渐把 servlet 作为 web 应用中的控制器组件来使用， 而把 JSP 技术作为数据显示模板来使用。

其实 Jsp 就是一个 Servlet，当我们第一次访问 Jsp 的时候，Jsp 引擎都会将这个 Jsp 翻译 成一个 Servlet，这个文件存放在tomcat（源码目录） 中的 work 目录中。

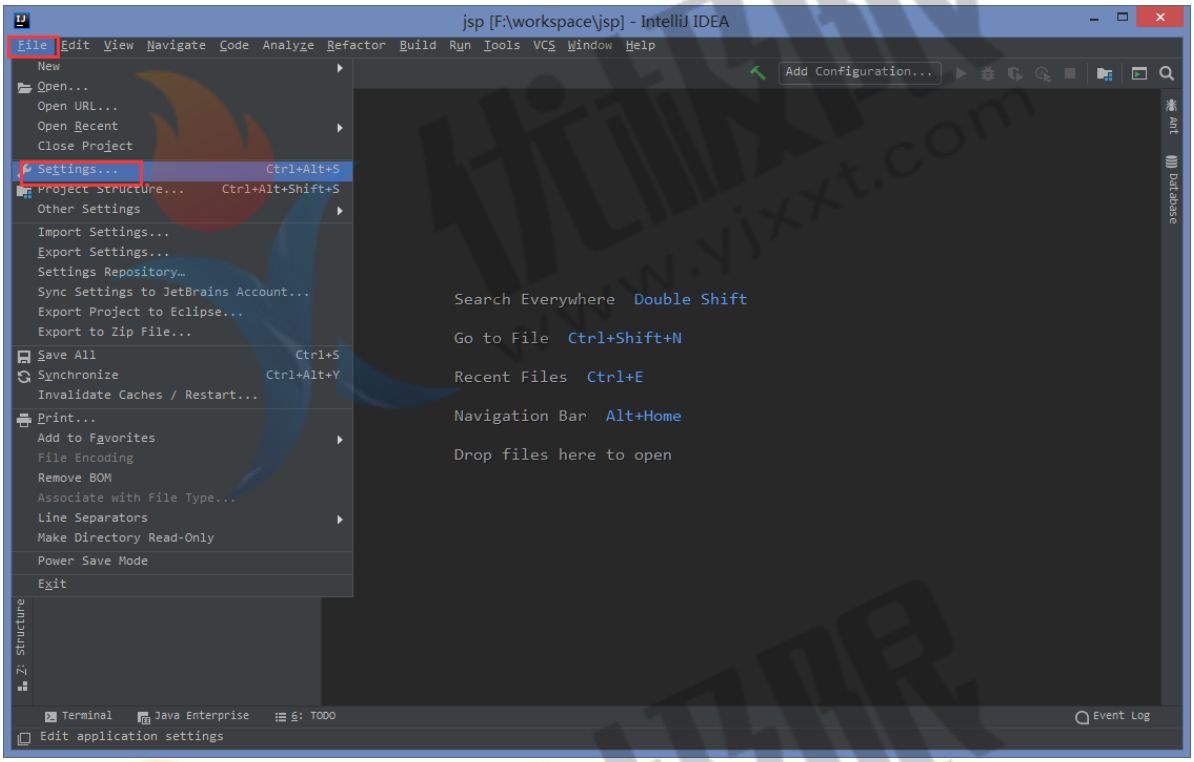


准备工作

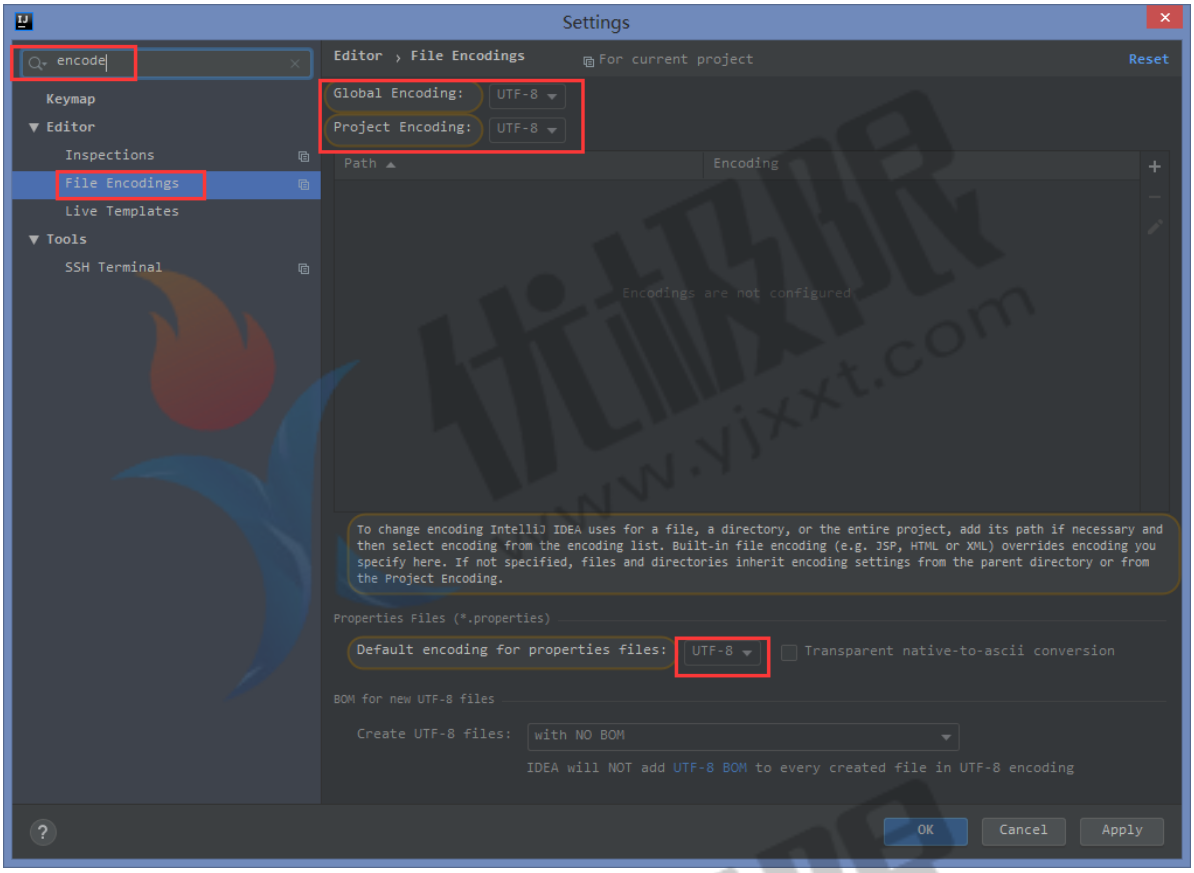
配置IDEA

这一步不是必须的，当然由于 编辑器中有些默认的配置项我们觉得不是很完美，比如"编码格式"、页面模板等。我们可以在新建 JSP 页面之前就先修改为我们需要的。

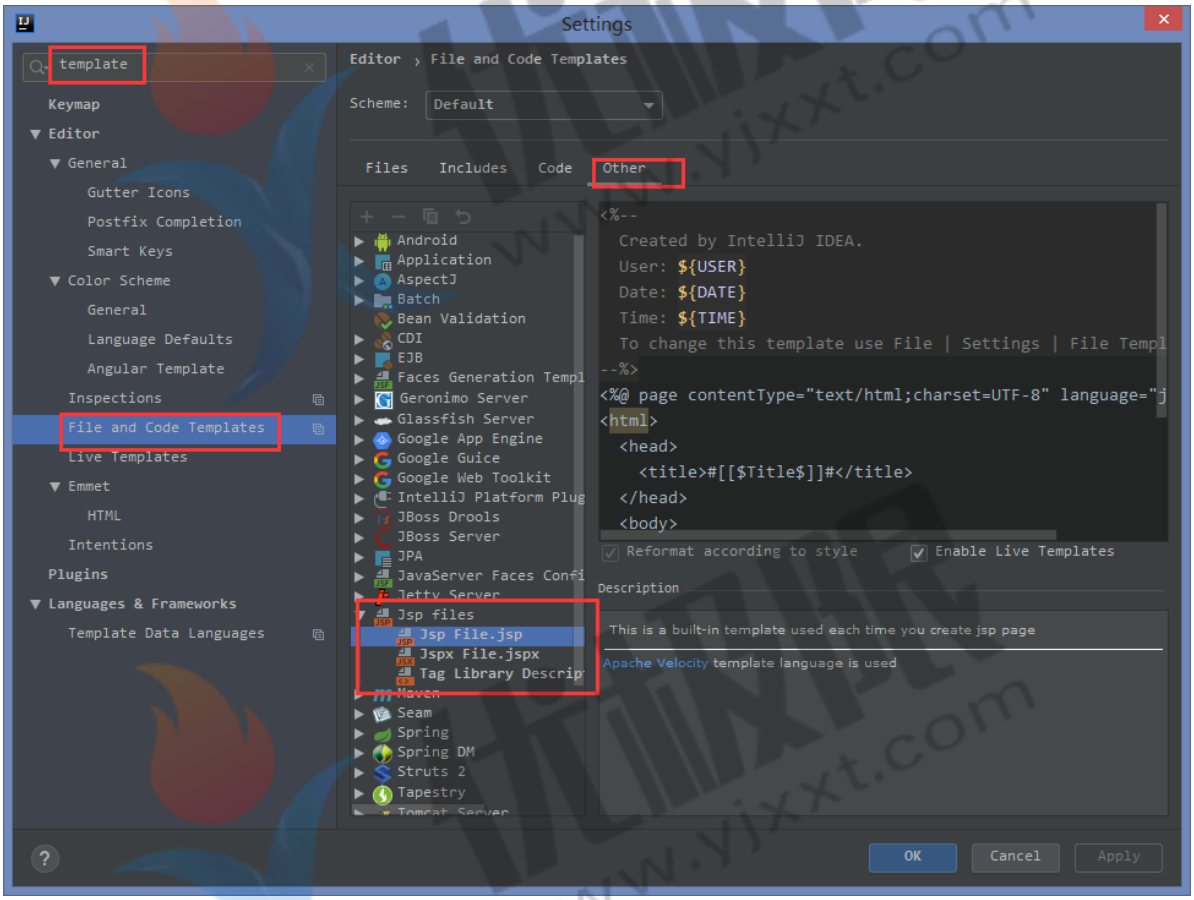
1.选择"File" —> "Settings..."



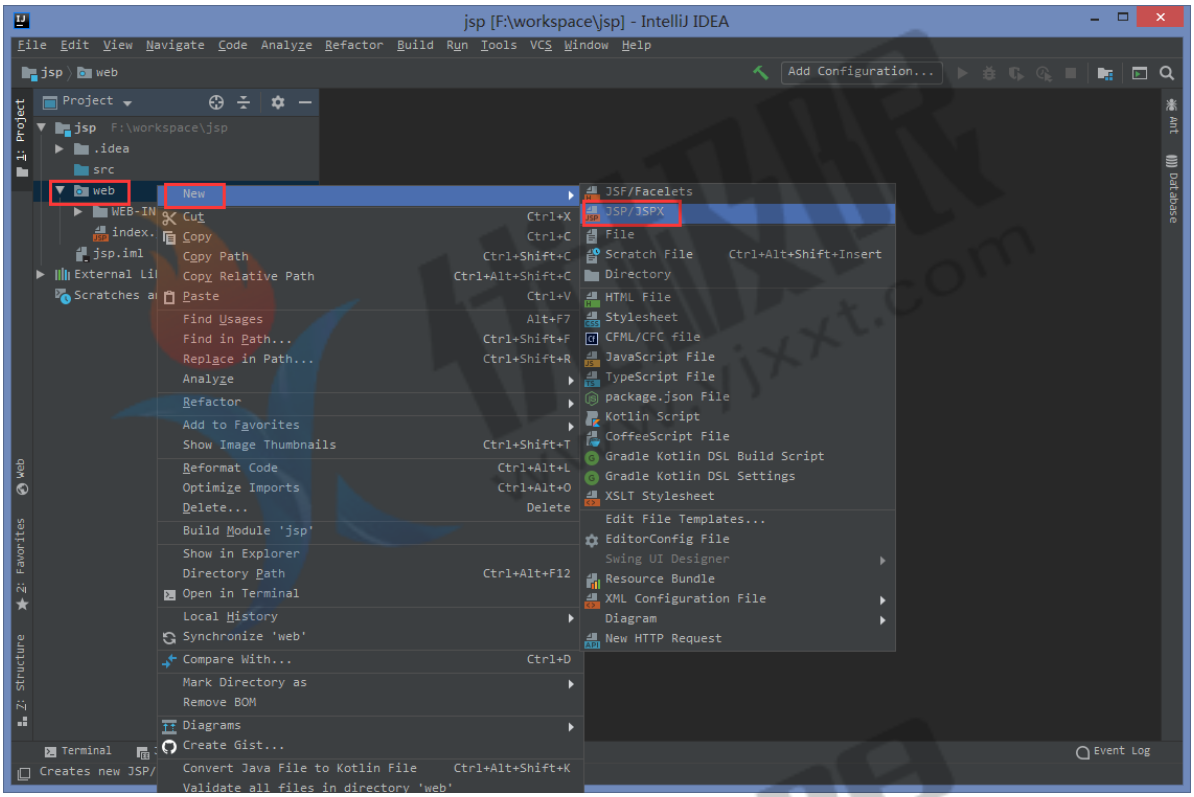
2.设置编码格式。搜索"encode"，选择"File Encoding"



3.设置页面模板。搜索"template"，选择"File and Code Templates"，选择右侧的"Other"，选择下方的"Jsp File"



新建JSP页面



注释

在 JSP 中支持两种注释的语法操作：

一种是显示注释，这种注释是允许客户端看见的； 另一种是隐式注释，此种注释是客户端无法看见的

① 显示注释语法：从 HTML 风格继承而来

② 隐式注释语法：从 JAVA 风格继承；JSP 自己的注释

JSP 的三种注释方式：

```
1） // 注释，单行注释  /* 多行注释*/

2）<!--  HTML风格的注释 -->

3）<%--   JSP注释   --%>
```

Scriptlet

在 JSP 中最重要的部分就是 Scriptlet（**脚本小程序**），**所有嵌入在 HTML 代码中的 Java 程序。**

在 JSP 中一共有三种 Scriptlet 代码：**都必须使用 Scriptlet 标记出来**

```
第一种：<%    %>：  java 脚本段，可以定义局部变量、编写语句

第二种：<%!   %>：声明，可以定义全局（成员）变量、方法、类

第三种：<%=  %>：表达式，数据一个变量或具体内容
```

通过观察解析为 java 文件的 jsp 代码理解三种小脚本

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE >
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Scriptlet</title>
    </head>
    <body>
        <%
            String str = "Hello JSP";
            System.out.println(str);
            response.getWriter().write(str);
        %>
        <%!
            String memberStr = "a member String";
        %>
        <%=memberStr%>
        <h1>This is a JSP page!!</h1>
    </body>
</html>
```

JSP的指令标签

使用包含操作，可以将一些重复的代码包含进来继续使用，从正常的页面组成来看，有时可能分为几个区域。而其中的一些区域可能是一直不需要改变的，改变的就其中的一个具体内容区域。现在有两种方法可以实现上述功能。

方法一：在每个 JSP 页面（HTML）都包含工具栏、头部信息、尾部信息、具体内容

方法二：将工具栏、头部信息、尾部信息都分成各个独立的文件，使用的时候直接导入

很明显，第二种方法比第一种更好，第一种会存在很多重复的代码，并且修改很不方便，在 JSP 中如果要想实现包含的操作，有两种做法：**静态包含**、**动态包含**，静态包含使用 include 指令即可，动态包含则需要使用 include 动作标签。

include 静态包含

```
<%@ include file="要包含的文件路径" %>  <!-- 相对路径 -->
```

例如：

```
<%@include file="include.jsp" %>
或
<%@include file="include.html" %>
```

静态包含就是将内容进行了直接的替换，就好比程序中定义的变量一样，是在 servlet 引擎转译时，就把此文件内容包含了进去（两个文件的源代码整合到一起， 全部放到_jspService 方法中），所以只生成了一个 servlet，所以两个页面不能有同名的变量。 运行效率高一点点。耦合性较高，不够灵活。

include 动态包含

动态包含在代码的编译阶段，包含和被包含部分是两个独立的部分，只有当运行时，才会动态包含进来，好比方法的调用。

```
<jsp:include page="include.jsp"></jsp:include>
```

注意：动态包含，中**间不要加任何内容**，包括空格，除非确认要使用参数，否则报错！

```
<jsp:include page="include.html"></jsp:include>
<%
    String a = "hello.jsp";
%>
<jsp:include page="<%=a %>"></jsp:include>
```

使用动态包含还可以通过在页面之间传参。

接收参数通过 **request.getParameter(name);**

```
<jsp:include page="hello.jsp" flush="true">
    <jsp:param name="uname" value="zhangsan"/>
</jsp:include>
```

hello.jsp

```
<!-- 接收参数 -->
<%=request.getParameter("uname")%>
```

JSP的四大域对象

四种属性范围

在JSP中提供了四种属性的保存范围，所谓的属性保存范围，指的就是一个设置的对象，可以再多少个页面中保存并可以继续使用

1. page范围

   pageContext : 只在一个页面中保存属性，跳转之后无效

2. request范围

   request : 只在一次请求中保存，服务器跳转后依然有效

3. session范围

session : 在一次会话范围中，无论何种跳转都可以使用

4. application范围

application : 在整个服务器上保存

方法	类型	描述
public void setAttribute(String name, Object o)	普通	设置属性的名称及内容
public Object getAttribute(String name)	普通	根据属性名称取属性
public void removeAttribute(String name)	普通	删除指定的属性

验证属性范围的特点

1. page

本页面取得，服务器端跳转（）后无效

2. request

服务器跳转有效，客户端跳转无效

如果是客户端跳转，则相当于发出了两次请求，那么第一次的请求将不存在了；如果希望不管是客户端还是服务器跳转，都能保存的话，就需要继续扩大范围。

3. session

无论客户端还是服务器端都可以取得，但是现在重新开启一个新的浏览器，则无法取得之前设置的session了，因为每一个session只保存在当前的浏览器当中，并在相关的页面取得。

对于服务器而言，每一个连接到它的客户端都是一个session

如果想要让属性设置一次之后，不管是否是新的浏览器打开都能取得则可以使用application

4. application

所有的application属性直接保存在服务器上，所有的用户(每一个session)都可以直接访问取得

只要是通过application设置的属性，则所有的session都可以取得，表示公共的内容，但是如果此时服务器重启了，则无法取得了，因为关闭服务器后，所有的属性都消失了，所以需要重新设置。

问：使用哪个范围呢?

答：在合理范围尽可能小

EL表达式的使用

EL表达式的语法

EL（Expression Language） 是为了使 JSP 写起来更加简单。表达式语言的灵感来自于 ECMAScript 和 XPath 表达式语言，它提供了在 JSP 中简化表达式的方法，让 Jsp 的代码更加简化。

```
语法结构非常简单： ${expression}
```

EL 表达式一般操作的都是域对象中的数据，操作不了局部变量。

域对象的概念在 JSP 中一共有四个：**pageContext**, **request**, **session**, **application**；范围依次是，**本页面**，**一次请求**，**一次会话**，**整个应用程序。**

当需要指定从某个特定的域对象中查找数据时可以使用四个域对象对应的空间对象，分别是：pageScope, requestScope, sessionScope, applicationScope。

而 EL 默认的查找方式为从小到大查找，找到即可。当域对象全找完了还未找到则返回空字符串""。

EL表达式的使用

获取数据

设置域对象中的数据

```
<%
    pageContext.setAttribute("uname","zhangsan"); // page作用域
    request.setAttribute("uname","lisi"); // request作用域
    session.setAttribute("uname","wangwu"); // session作用域
    application.setAttribute("uname","zaholiu"); // application
%>
```

获取域对象的值

```
<%-- 获取域对象中的数据：默认查找方式为从小到大，找到即止。若四个范围都未找到，则返回空字符串。--%>
${uname} <!-- 输出结果为：zhangsan -->
```

获取指定域对象的值

```
${pageScope.uname}          <!-- page作用域 -->
${requestScope.uname}       <!-- request作用域 -->
${sessionScope.uname}       <!-- session作用域 -->
${applicationScope.uname}   <!-- application作用域 -->
```

获取List

```
<%
    List<String> list = new ArrayList<String>();
    list.add("aaa");
    list.add("bbb");
    list.add("ccc");
    request.setAttribute("list", list);
%>
```

```
<%--
    获取List中指定下标的数据
        ${list[下标] }
    获取集合的长度
        ${list.size()}
    注：
       list代表的是存在域对象中的变量名（限域变量名）
--%>
${list[1] }
```

获取Map

```
<%
    Map map = new HashMap();
    map.put("aaa", "111");
    map.put("bbb", 2222);
    map.put("ccc-a", 333);
    request.setAttribute("map", map);
%>
```

```
<%--
    获取Map中指定值
       ${map["key"] } 或 ${map.key }
    注：
        map代表的是存在域对象中的变量名（限域变量名）
--%>
${map.aaa }
${map["bbb"]}
```

获取JavaBean对象

User.java

```
public class User {

    private Integer userId;
    private String uname;
    private String upwd;

    public Integer getUserId() {
        return userId;
    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }

    public String getUname() {
        return uname;
    }

    public void setUname(String uname) {
        this.uname = uname;
    }
```

```
    public String getUpwd() {
        return upwd;
    }

    public void setUpwd(String upwd) {
        this.upwd = upwd;
    }
}
```

```
<%
    User user = new User();
    user.setUserId(1);
    user.setUname("zhangsan");
    user.setUpwd("123456");
    request.setAttribute("user",user);
%>
```

```
<%-- JavBean中的属性字段需要提供get方法 --%>
${user} <%-- 获取对象 --%>
${user.uname} <%--获取对象中的属性--%>
```

empty

```
<%--
    empty
        判断域对象是否为空。为空，返回true；不为空返回false；
            ${empty 限域变量名 }
        判断对象是否不为空。
            ${!empty 限域变量名 }
--%>
${empty uname}
${empty list}
${empty map}
${empty user}
```

EL运算

```
<%
    request.setAttribute("a", 10);
    request.setAttribute("b", 2);
    request.setAttribute("c", "aa");
    request.setAttribute("d", "bb");
%>
```

等值判断

```
<%--
    比较两个值是否相等，返回true或false
        ==  或  eq
--%>
${a == b }
${c == d }
${c eq d }
${a == 5 }
${c == 'aa' }
```

算术运算

```
<%--
    加法：  +
    减法：  -
    乘法：  *
    除法：  / 或 div
--%>
${a + b }
${a / b } 或 ${a div b }
```

大小比较

```
<%--
    大于：>
    小于：<
    大于等于：>=
    小于等于：<=
--%>
${a > b}
${a + 1 > 10 }
${a + b >= 10 }
${a > b && b > 5 }
${a + b > 10 || a - b > 5 }
```

JSTL

标签的使用

Java Server Pages Standard Tag Libray(JSTL)：JSP 标准标签库，是一个定制标签类库的集合，用于解决一些常见的问题，例如迭代一个映射或者集合、条件测试、XML 处理，甚至数据库和访问数据库操作等。

我们现在只讨论 JSTL 中最重要的标签，迭代集合以及格式化数字和日期几个标签。

核心标签库:

http://java.sun.com/jsp/jstl/core

包含 Web 应用的常见工作，比如：循环、表达式赋值、基本输入输出等。

格式化标签库:

http://java.sun.com/jsp/jstl/fmt

用来格式化显示数据的工作，比如：对不同区域的日期格式化等。

为了在 JSP 页面使用 JSTL 类库，必须以下列格式使用 taglib 指令：

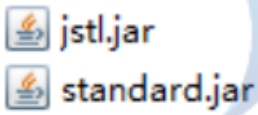
```
<%@taglib uri="" prefix="" %>
```

例如：

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

前缀可以是任意内容，遵循规范可以使团队中由不同人员编写的代码更加相似；所以，建议使用事先设计好的前缀。

此时需要导入两个jar 包



从Apache的标准标签库中下载的二进包(jakarta-taglibs-standard-current.zip)。

官方下载地址：http://archive.apache.org/dist/jakarta/taglibs/standard/binaries/

下载 **jakarta-taglibs-standard-1.1.2.zip** 包并解压，将 **jakarta-taglibs-standard-1.1.2/lib/** 下的两个 jar 文件：**standard.jar** 和 **jstl.jar** 文件拷贝到项目的指定目录下。

条件动作标签

条件动作指令用于处理页面的输出结果依赖于某些输入值的情况，在 Java 中是利用 if、 if...else 和 switch 语句来进行处理的。在 JSTL 中也有 4 个标签可以执行条件式动作指令：if、 choose、when 和 otherwise。

if 标签

if 标签先对某个条件进行测试，如果该条件运算结果为 true, 则处理它的主体内容，测试结果保存在一个 Boolean 对象中，并创建一个限域变量来引用 Boolean 对象。可以利用 var 属性设置限域变量名，利用 scope 属性来指定其作用范围。

语法格式

```
<c:if test="<boolean>" var="<string>" scope="<string>">
   ...
</c:if>
```

属性

if 标签有如下属性：

属性	描述	是否必要	默认值
test	条件	是	无
var	用于存储条件结果的变量（限域变量名）	否	无
scope	var属性的作用域 可取值：page|request|session|application	否	page

示例

```
<%
    request.setAttribute("flag",true);
    request.setAttribute("num",1);
%>
<c:if test="${flag}">
   <p>结果为true<p>
</c:if>
<c:if test="${num > 0}">
   <p>num的值比0大<p>
</c:if>
```

注： JSTL中没有else标签，为了模拟 else 的情景，需要使用两个 if 标签，并且这两个标签为**相反**的条件。

choose、when 和 otherwise 标签

choose 和 when 标签的作用与 Java 中的 switch 和 case 关键字相似，用于在众多选项中做出选择。也就是说：他们为相互排斥的条件式执行提供相关内容。

switch语句中有case，而choose标签中对应有when，switch语句中有default，而choose标签中有otherwise。

语法格式

```
<c:choose>
    <c:when test="<boolean>">
        ...
    </c:when>
    <c:when test="<boolean>">
        ...
    </c:when>
    ...
    ...
    <c:otherwise>
        ...
    </c:otherwise>
</c:choose>
```

属性

- choose标签没有属性。
- when标签只有一个test属性。
- otherwise标签没有属性。

示例

```
<%
    request.setAttribute("score", 90);
%>
<c:choose>
    <c:when test="${score < 60 }">
        <h3>你个小渣渣！</h3>
    </c:when>
    <c:when test="${score >= 60 && score < 80 }">
        <h3>革命尚未成功，同志仍需努力！</h3>

    </c:when>
    <c:otherwise>
        <h3>你很棒棒哦！</h3>
    </c:otherwise>
</c:choose>
```

注意点

- choose标签和otherwise标签没有属性，而when标签必须设置test属性
- choose标签中必须有至少一个when标签，可以没有otherwise标签
- otherwise标签必须放在最后一个when标签之后
- choose标签中只能有when标签和otherwise标签，when标签和otherwise标签可以嵌套其他标签
- otherwise标签在所有的when标签不执行的情况下才会执行

迭代标签

forEach 是将一个主体内容迭代多次，或者迭代一个对象集合。可以迭代的对象包括所有的 java.util.Collection 和 java.util.Map 接口的实现，以及对象或者基本类型的数组。他还可 以迭代 java.util.Iterator 和 java.util.Enumeration,但不能在多个动作指令中使用 Iterator 或者 Enumeration, 因为 Iterator 或者 Enumeration 都不能重置（reset）。 各属性含义如下：

forEach标签

语法格式

```
<c:forEach
    items="<object>"
    begin="<int>"
    end="<int>"
    step="<int>"
    var="<string>"
    varStatus="<string>">
</c:forEach>
```

属性

属性	描述	是否必要	默认值
items	要被循环的数据	否	无
begin	开始的元素（0=第一个元素，1=第二个元素）	否	0
end	最后一个元素（0=第一个元素，1=第二个元素）	否	Last element
step	每一次迭代的步长	否	1
var	代表当前条目的变量名称	否	无
varStatus	代表循环状态的变量名称	否	无

forEach varStatus 属性

- **index**: 当前这次迭代从 0 开始的迭代索引
- **count**: 当前这次迭代从 1 开始的迭代计数
- **first**: 用来表明当前这轮迭代是否为第一次迭代的标志
- **last**: 用来表明当前这轮迭代是否为最后一次迭代的标志

示例

1.遍历主体内容多次

```
<c:forEach begin="开始数" end="结束数" step="迭代数" var="限域变量名">

</c:forEach>

相当于java的for循环：

for(int i = 0; i < 10; i++) {

}
```

如下：

```
<!-- 遍历主体内容多次  -->
<c:forEach begin="0" end="10" var="i" >
    标题${i }<br>
</c:forEach>
```

2.循环

```
<c:forEach items="被循环的集合" var="限域变量名" varStatus="当前成员对象的相关信息">

</c:forEach>

相当于java的foreach循环：

for(String str : list) {

}
```

如下:

```
 <%
    List<String> list = new ArrayList<String>();
    for (int i = 1; i <= 10; i++) {
        list.add("A:" + i);
    }
    pageContext.setAttribute("li", list);
%>

<!-- 循环集合 -->
<c:forEach items="${li }" var="item">
    ${item }
</c:forEach>
<hr>
<table align="center" width="800" border="1" style="border-collapse: collapse;">
    <tr>
        <th>名称</th>
        <th>当前成员下标</th>
        <th>当前成员循环数</th>
        <th>是否第一次被循环</th>
        <th>是否最后一次被循环</th>
    </tr>
    <c:forEach items="${li }" var="item" varStatus="itemp">
        <tr>
            <td>${item }</td>
            <td>${itemp.index }</td>
            <td>${itemp.count }</td>
            <td>${itemp.first }</td>
            <td>${itemp.last }</td>
        </tr>
    </c:forEach>
</table>

<!-- 循环对象集合 -->
<%
    List<User> userList = new ArrayList<User>();
    User user = new User(1,"zhangsan","123456");
    User user2 = new User(2,"lisi","123321");
    User user3 = new User(3,"wangwu","654321");
    userList.add(user);
    userList.add(user2);
    userList.add(user3);
    // 将数据设置到作用域中
    request.setAttribute("userList", userList);
%>
<c:if test="${!empty userList }">
    <table align="center" width="800" border="1" style="border-collapse:
collapse;">
        <tr>
            <th>用户编号</th>
            <th>用户名称</th>
            <th>用户密码</th>
            <th>用户操作</th>
        </tr>
        <c:forEach items="${userList }" var="user">
            <tr align="center">
```

```
                <td>${user.userId }</td>
                <td>${user.uname }</td>
                <td>${user.upwd }</td>
                <td>
                    <button>修改</button>
                    <button>删除</button>
                </td>
            </tr>
        </c:forEach>
    </table>
</c:if>

<!-- 遍历Map -->
<%
    Map<String,Object> map = new HashMap<String,Object>();
    map.put("map1", "aaa");
    map.put("map2", "bbb");
    map.put("map3", "ccc");
    pageContext.setAttribute("map", map);
%>
<c:forEach items="${map }" var="mymap">
    键：${mymap.key }-值：${mymap.value } <br>
</c:forEach>
```

格式化动作标签

JSTL 提供了格式化和解析数字和日期的标签,我们讨论里面有：formatNumber、formatDate、parseNumber及parseDate。

formatNumber标签

formatNumber标签用于格式化数字，百分比，货币。该标签用指定的格式或精度来格式化数字。（将数值型数据转换成指定格式的字符串类型。）

语法格式

```
<fmt:formatNumber
  value="<string>"
  type="<string>"
  var="<string>"
  scope="<string>"/>
```

属性

属性	描述	是否必要	默认值
value	要显示的数字	是	无
type	NUMBER，CURRENCY，或 PERCENT类型	否	Number
var	存储格式化数字的变量	否	Print to page
scope	var属性的作用域	否	page

注意：

1. 如果设置了var属性，则格式化后的结果不会输出，需要通过el表达式获取var对应的限域变量名
2. 默认的类型（type）的取值为number。可取值：number数值型、percent百分比类型、currency货币型

示例

```
<fmt:formatNumber value="10" type="number" var="num" /> ${num } <br>
<fmt:formatNumber value="10" type="percent"  />  <br>
<fmt:formatNumber value="10" type="currency"  />  <br>
<!-- 设置时区  -->
<fmt:setLocale value="en_US"/>
<fmt:formatNumber value="10" type="currency"  />  <br>
```

formatDate标签

formatDate标签用于使用不同的方式格式化日期。（将Date型数据转换成指定格式的字符串类型。）

语法格式

```
<fmt:formatDate
  value="<string>"
  type="<string>"
  dateStyle="<string>"
  timeStyle="<string>"
  pattern="<string>"
  timeZone="<string>"
  var="<string>"
  scope="<string>"/>
```

属性

属性	描述	是否必要	默认值
value	要显示的日期	是	无
type	DATE, TIME, 或 BOTH	否	date
dateStyle	FULL, LONG, MEDIUM, SHORT, 或 DEFAULT	否	default
timeStyle	FULL, LONG, MEDIUM, SHORT, 或 DEFAULT	否	default
pattern	自定义格式模式	否	无
timeZone	显示日期的时区	否	默认时区
var	存储格式化日期的变量名	否	显示在页面
scope	存储格式化日志变量的范围	否	页面

标签格式模式

代码	描述	实例
y	不包含纪元的年份。如果不包含纪元的年份小于 10，则显示不具有前导零的年份。	2002
M	月份数字。一位数的月份没有前导零。	April & 04
d	月中的某一天。一位数的日期没有前导零。	20
h	12 小时制的小时。一位数的小时数没有前导零。	12
H	24 小时制的小时。一位数的小时数没有前导零。	0
m	分钟。一位数的分钟数没有前导零。	45
s	秒。一位数的秒数没有前导零。	52

示例

```
<%
    request.setAttribute("myDate", new Date());
%>

${myDate } <br/>
<fmt:formatDate value="${myDate }" /><br/>
<fmt:formatDate value="${myDate }" type="date"/><br/>
<fmt:formatDate value="${myDate }" type="time"/><br/>
<fmt:formatDate value="${myDate }" type="both"/><br/>
<fmt:formatDate value="${myDate }" type="both" dateStyle="full"/><br/>
<fmt:formatDate value="${myDate }" type="both" dateStyle="long"/><br/>
<fmt:formatDate value="${myDate }" type="both" dateStyle="short"/><br/>
<fmt:formatDate value="${myDate }" type="both" timeStyle="full"/><br/>
<fmt:formatDate value="${myDate }" type="both" timeStyle="long"/><br/>
<fmt:formatDate value="${myDate }" pattern="HH:mm yyyy/MM/dd"/><br/>
```

parseNumber标签

parseNumber标签用来解析数字，百分数，货币。（parseNumber 标签可以将数字、货币或百分比类型的字符串转换成数值型。）

语法格式

```
<fmt:parseNumber
  value="<string>"
  type="<string>"
  var="<string>"
  scope="<string>"/>
```

属性

属性	描述	是否必要	默认值
value	要解析的数字	否	Body
type	NUMBER,，CURRENCY，或 PERCENT	否	number
var	存储待解析数字的变量	否	Print to page
scope	var属性的作用域	否	page

示例

```
<fmt:parseNumber value="100"  /> <br>
<fmt:parseNumber value="100"  type="number" /> <br>
<fmt:parseNumber value="100%"  type="percent" /> <br>
<fmt:parseNumber value="￥10.00"  type="currency" /> <br>
```

parseDate标签

parseDate标签用于解析日期。（将指定格式的字符串转换成Date类型。）

语法格式

```
<fmt:parseDate
   value="<string>"
   type="<string>"
   dateStyle="<string>"
   timeStyle="<string>"
   pattern="<string>"
   var="<string>"
   scope="<string>"/>
```

属性

属性	描述	是否必要	默认值
value	要显示的日期	是	无
type	DATE, TIME, 或 BOTH	否	date
dateStyle	FULL, LONG, MEDIUM, SHORT, 或 DEFAULT	否	default
timeStyle	FULL, LONG, MEDIUM, SHORT, 或 DEFAULT	否	default
pattern	自定义格式模式	否	无
var	存储格式化日期的变量名	否	显示在页面
scope	存储格式化日志变量的范围	否	页面

示例

```
<fmt:parseDate value="2020-01-06" type="date" /> <br>
<fmt:parseDate value="2020/01/06" pattern="yyyy/MM/dd" /> <br>
```