# Servlet新特性

# 本章内容：共2小节，6个知识点

- 第1节： Servlet3.0新特性
- 第2节： Servlet4.0新特性

# 本章目标

- 了解Servlet3.0版本的主要新特性；
- 熟悉Servlet3.0中的注解；
- 理解Servlet3.0的非阻塞特征；
- 能够使用Servlet3.0版本实现文件上传；
- 了解Servlet4.0版本的主要新特性；
- 熟悉掌握如何获取全新 PushBuilder；
- 掌握全新 servlet 映射使服务器能够对URL执行运行时检查；
- 熟悉Servlet 4.0 的细微变化；

# 第1节【Servlet3.0新特性】

- 知识点1：非阻塞
- 知识点2：注解
- 知识点3：文件上传

# 知识点1【非阻塞】-1

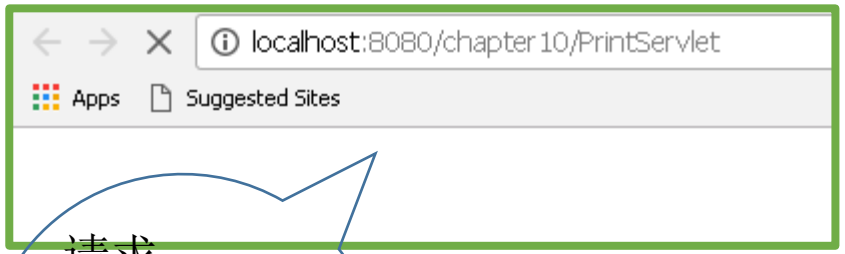
课堂案例：
PrintServlet.java

- 我们编写一个PrintServlet，在一个死循环中打印输出：

```
out.println(Thread.currentThread().getName()+": Hello，我是PrintServlet,马上要进入死循环打印了！ ");
out.close();
while(true){
    System.out.println("正在打印，请稍后.....");
}
```

- 在之前版本的Servlet中，访问该Servlet后，再次访问就不再有响应，因为当前线程阻塞，在处理死循环；


请求PrintServlet，在后台死循环打印。



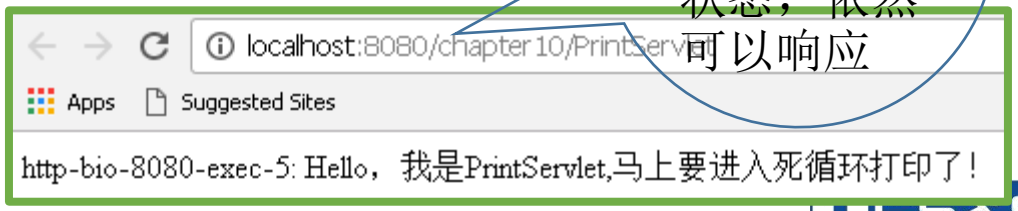

请求PrintServlet，处于阻塞状态，一直在加载状态。

# 知识点1【非阻塞】-2

课堂案例：
PrintServlet.java

- Servlet3.0版本，增加了非阻塞特性：接收请求后，启动一个异步线程处理请求，客户端可以继续请求这个Servlet，而不必等待上一次请求返回；
- 默认情况下，Servlet3.0版本Servlet并没有这个特性，需要在web.xml中进行配置：

```
<servlet>
        <servlet-name>PrintServlet</servlet-name>
        <servlet-class>com.chiansofti.chapter10.section01.PrintServlet</servlet-class>
        <async-supported>true</async-supported>
</servlet>
```

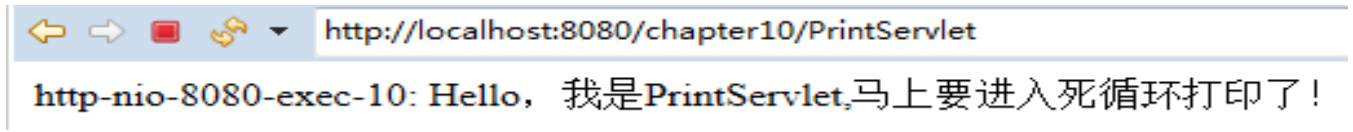




请求PrintServlet，处于非阻塞状态，依然可以响应

# 知识点1【非阻塞】-3

课堂案例：
PrintServlet.java

```
@WebServlet("/PrintServlet")
public class PrintServlet extends HttpServlet {
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
//		设置响应的内容类型
		response.setContentType("text/html;charset=utf-8");
//		获得输出流
		PrintWriter out=response.getWriter();
	out.println(Thread.currentThread().getName()+": Hello，我是PrintServlet,马上要进入死循环打印了！ ");
		out.close();
		while(true){
			System.out.println("正在打印，请稍后.....");
		}
	}
```



# 知识点2【注解】-1

- Servlet3.0中可以使用注解（annotation）替代web.xml进行配置；
- 通常用的注解有五种类型：
  - @WebServlet：对Servlet进行配置
  - @WebInitParam：配置Servlet初始化参数
  - @WebFilter：配置过滤器
  - @WebListener：配置监听器
  - @MultipartConfig：对文件上传的支持

# 知识点2【注解】-2

- @WebServlet：对Servlet进行配置示例，包括的属性如下，这些属性除了vlaue或urlPatterns是必选的，其他的都是可选的：
  - 1）name：等价于web.xml配置文件中的 <servlet-name>。如果没有指定， Servlet 的<servlet-name>取值为类的全限定名，比如XXX.XXX.XXX。
  - 2）urlPatterns：等价于web.xml配置文件中的 <url-pattern> 标签
  - 3）value：等价于 urlPatterns 属性。
  - 4）loadOnStartup：等价于web.xml配置文件中的<load-on-startup> 标签
  - 5）initParams ：等价于web.xml配置文件中的<init-param> 标签，他的参数是@WebInitParam注解的集合（此注解之后介绍）
  - 6）asyncSupported：等价于web.xml配置文件中的<async-supported> 标签
  - 7）description：等价于web.xml配置文件中的<description> 标签
  - 8）displayName：等价于web.xml配置文件中的 <display-name> 标签

# 知识点2【注解】-3

- @WebServlet：对Servlet进行配置示例

注解

```
@WebServlet(urlPatterns = {"/demo"},
asyncSupported = true,
loadOnStartup = -1,
name = "DemoServlet",
displayName = "chinasofti",
initParams = {
@WebInitParam(name = "username", value = "etc")
}
)
```

当值小于0或者没有指定时，则表示容器在该servlet被选择时才会去加载。

正数的值越小，该servlet的优先级越高，应用启动时就越先加载。

web.xml

```
<servlet>
   <display-name> chinasofti </display-name>
   <servlet-name>DemoServlet</servlet-name>
   <servlet-class>
com.chinasofti.servlet.DemoServlet</servlet-class>
   <load-on-startup>-1</load-on-startup>
   <async-supported>true</async-supported>
   <init-param>
      <param-name>username</param-name>
      <param-value>etc</param-value>
   </init-param>
</servlet>
<servlet-mapping>
   <servlet-name> DemoServlet </servlet-name>
   <url-pattern>/demo</url-pattern>
</servlet-mapping>
```

# 知识点2【注解】-4

- @WebInitParam：配置Servlet初始化参数，常用属性有三个，这三个属性当中只有description为可选属性：
  - name：等价于web.xml配置文件中的 <param-name>
  - value ： 等价于web.xml配置文件中的<param-value>
  - description：等价于web.xml配置文件中的<description>

# 知识点2【注解】-5

- @WebInitParam：配置Servlet初始化参数示例

注解

```
initParams = {
@WebInitParam(name = "username", value = "etc")
}
```

=

web.xml

```
<init-param>
    <param-name>username</param-name>
    <param-value>etc</param-value>
</init-param>
```

# 知识点2【注解】-6

- @WebFilter：配置过滤器此注解为声明一个过滤器，主要属性有以下几个。
- 在这些属性当中value、urlPatterns、servletNames 三个属性至少要包含其中的一个，并且 value 和 urlPatterns 属性只能有一个，如果两个同时配置，一般情况下value取值将会被忽略。其他的都是可选属性。
  - filterName：等价于web.xml配置文件中的 <filter-name>标签
  - value：该属性等价于 urlPatterns 属性
  - urlPatterns：等价于web.xml配置文件中的 <url-pattern> 标签
  - servletNames：指定该过滤器将应用的范围。如果是注解的话取值是 @WebServlet 中的 name 属性的取值，如果servlet这 web.xml 中配置的话，取值是 <servlet-name> 的取值
  - dispatcherTypes：过滤器的转发模式。取值包括：
  - ASYNC（异步）、ERROR（错误）、FORWARD（请求转发）、INCLUDE（包含）、REQUEST（请求）。
  - initParams：等价于web.xml配置文件中的<init-param> 标签
  - asyncSupported：等价于web.xml配置文件中的<async-supported> 标签
  - description：等价于web.xml配置文件中的<description> 标签
  - displayName：等价于web.xml配置文件中的<display-name> 标签

# 知识点2【注解】-7

- @WebFilter：配置过滤器示例

注解

```
@WebFilter(servletNames =
{"LoginServlet"},filterName="LoginFilter")
public class LoginFilter implements Filter{
.....
}
```

=

web.xml

```
<filter>
   <filter-name> LoginFilter </filter-name>
   <filter-class> com.chinasofti.filter.LoginFilter </filter-
class>
</filter>
<filter-mapping>
   <filter-name> LoginFilter </filter-name>
   <servlet-name> LoginServlet </servlet-name>
</filter-mapping>
```

# 知识点2【注解】-8

- @WebListener：配置监听器，此注解是用来声明监听器，它主要的属性只有一个：
  - value：这个属性表示的是监听器的描述信息，整个配置可以简写成@WebListener("XXX")

# 知识点2【注解】-9

- @WebListener：配置监听器示例

注解

```
@WebListener("this is a listener")
public class CounterListener implements
ServletContextListener{
……
```

=

web.xml

```
<listener>
        <listener-class>
         com.chinasofti.chapter10.section01.ListenerTest
        </listener-class>
</listener>
```

# 知识点3【文件上传】-1

- Servlet3.0以前版本没有对文件上传进行支持，只能用第三方组件实现；
- Servlet3.0中对文件上传进行了支持，核心接口是Part接口，该接口中的核心方法如下：

| 方法声明 | 方法描述 |
| --- | --- |
| void delete() | 删除part对象对应文件项的基本存储，包括删除任何相关的临时磁盘文件 |
| String getContentType() | 请求上传文件的类型 |
| String getHeader(String name): | 获取上传文件内容的指定名字的请求头信息 |
| Collection<String> getHeaderNames() | 获取上传文件请求的全部请求头名称，返回的是一个包含请求头名称的集合 |
| Collection<String> getHeaders(String name) | 通过请求头名称，获取全部对应的请求信息，返回的是一个集合 |
| InputStream getInputStream() | 获取输入流 |
| String getName() | 获取控件的名字 |
| Long getSize() | 获取上传文件的大小 |
| void write(String fileName) | 将文件写入到物理磁盘 |

# 知识点3【文件上传】-2

课堂案例：
upload.jsp

- 在Servlet3.0版本中，请求接口提供了获取Part实例的方法；

| 方法声明 | 方法描述 |
| --- | --- |
| Part getPart(String name) | 根据上传控件名称获取上传文件对应的Part对象 |
| Collection<Part> getParts() | 获取所有上传文件对应的Part对象。 |

- 在upload.jsp中定义上传表单；

```
<form action="UploadServelt" method="post"   enctype="multipart/form-data“ >
   姓名: <input type="text" name="userName"> <br>
   文件: <input  type="file" name="file"> <br>
 <input type="submit"  value="上传">
</form>
```

enctype就是encodetype就是编码类型的意思。

multipart/form-data是指表单数据有多部分构成，既有文本数据，又有文件等二进制数据的意思。

需要注意的是：默认情况下，enctype的值是application/x-www-form-urlencoded，不能用于文件上传，只有使用了multipart/form-data，才能完整的传递文件数据。

# 知识点3【文件上传】-3

课堂案例：
UploadServlet.java

- 定义UploadServlet，实现上传功能：

```
@WebServlet("/UploadServelt")
@MultipartConfig
public class UploadServelt extends HttpServlet {
        @Override
        protected void service(HttpServletRequest req, HttpServletResponse res) throws ServletException,
IOException {
                req.setCharacterEncoding("utf-8");
                res.setContentType("text/html; charset=utf-8");
               String name = req.getParameter("userName");
                Part file = req.getPart("file");
                String path = req.getServletContext().getRealPath("/upload/");
                System.out.println(path);
                System.out.println(file.getSubmittedFileName());
                //文件大小
                System.out.println(file.getSize());
                //文件名称
                System.out.println(file.getName());
```

# 知识点3【文件上传】-4

课堂案例：
UploadServlet.java

- 定义UploadServlet，实现上传功能；核心代码如下：

```
    //表单名称
                    File f = new File(path);
                    if (!f.exists()) { f.mkdirs();     }
                    String fn = file.getSubmittedFileName();
                            if (fn.toLowerCase().endsWith(".jpg") || fn.toLowerCase().endsWith(".png")) {
                            //getSubmittedFileName() 返回的是上传的文件名称
                            file.write(path+"/"+file.getSubmittedFileName());
                            //生成文件新名字
                            String nfn = new SimpleDateFormat("yyyyMMddhhmmss").format(new Date()) +
fn.substring(fn.lastIndexOf("."));
                            file.write(path + "/" + nfn);
                            req.setAttribute("file", file.getSubmittedFileName());
                            } else {
                                    req.setAttribute("file", "文件上传失败，必须为jpg或png图像文件");
                                    }
                    req.setAttribute("user", name);
                    req.getRequestDispatcher("/uploadsuccess.jsp").forward(req, res);      }}
```

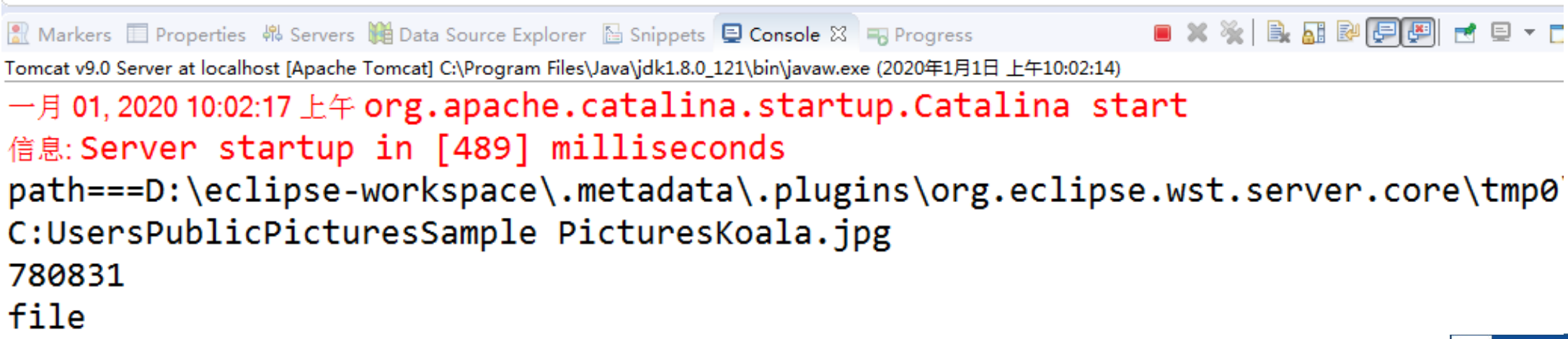
# 知识点3【文件上传】-5

课堂案例：
uploadsuccess.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<center>
            <h3>上传完成</h3>
            上传人：<h3>${user}</h3>
            上传文件为：<h3>${file}</h3>
</center>
</body>
</html>
```

# 知识点3【文件上传】-6


http://localhost:8080/chapter10/UploadServlet
上传完成
上传人：
BianXinWen
上传文件为：
C:UsersPublicPicturesSample PicturesKoala.jpg


Markers
Properties
Servers
Data Source Explorer
Snippets
Console
Progress
Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk1.8.0_121\bin\javaw.exe (2020年1月1日 上午10:02:14)
一月 01, 2020 10:02:17 上午 org.apache.catalina.startup.Catalina start
信息: Server startup in [489] milliseconds
path===D:\eclipse-workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0
C:UsersPublicPicturesSample PicturesKoala.jpg
780831
file

# 知识点3【文件上传】-7

让Servlet支持上传，需要做两件事情

1. 需要添加MultipartConfig注解
2. 从request对象中获取Part文件对象

```
@WebServlet("/uploadservelt")
@MultipartConfig(
        fileSizeThreshold = 1024 * 1024 * 2,
        maxFileSize = 1024 * 1024 * 1000,
        maxRequestSize = 1024 * 1024 * 50
)
```

MultipartConfig注解

| 属性名 | 类型 | 是否可选 | 描述 |
| --- | --- | --- | --- |
| fileSizeThreshold | int | 是 | 当数据量大于该值时，内容将被写入文件。单位：byte |
| location | String | 是 | 存放生成的文件地址。 |
| maxFileSize | long | 是 | 允许上传的文件最大值。默认值为 -1，表示没有限制。单位：byte |
| maxRequestSize | long | 是 | 针对该 multipart/form-data 请求的最大数量，默认值为 -1，表示没有限制。单位：byte |

# 本节总结提问【Servlet3.0新特性】

- Servlet3.0的非阻塞特性是什么意思？如何实现？
- Servlet3.0中有哪几个常用的注解类型？
- Servlet3.0对文件上传进行了哪些支持？

# 本节总结【 Servlet3.0新特性】

- Servlet3.0中增加了非阻塞特性支持，可以在web.xml中或者使用注解进行配置，使得Servlet可以异步处理请求；
- Servlet3.0可以使用注解替代web.xml的部分内容，常用的注解类型有@WebServlet、@WebFilter、@WebListener、@WebInitParam。
- Servlet3.0中对文件上传进行了支持，提供了Part接口以及@MultipartConfig注解，可以方便地实现文件上传；

# 第2节【Servlet4.0新特性】

- 知识点1：如何获取全新PushBuilder
- 知识点2：HttpServletMapping接口
- 知识点3：Servlet 4.0 的细微变化

# 知识点1：如何获取全新PushBuilder -1

- HTTP/2 （原名HTTP/2.0）即超文本传输协议 2.0，是下一代HTTP协议。是由互联网工程任务组（IETF）的Hypertext Transfer Protocol Bis (httpbis)工作小组进行开发。是自1999年http1.1发布后的首个更新。HTTP 2.0在2013年8月进行首次合作共事性测试。在开放互联网上HTTP 2.0将只用于https://网址，而 http://网址将继续使用HTTP/1，目的是在开放互联网上增加使用加密技术，以提供强有力的保护去遏制主动攻击

# 知识点1：如何获取全新PushBuilder -2

- HTTP/2 的首要目标是改善 Web 应用程序用户的体验。作为一个二进制协议，它拥有包括轻量型、安全和快速在内的所有优势。HTTP/2 保留了原始 HTTP 协议的语义，但更改了在系统之间传输数据的方式。这些复杂细节主要由客户端和服务器管理，所以网站和应用程序无需重大更改即可享受 HTTP/2 的优势。

# 知识点1：如何获取全新 PushBuilder-3

- Servlet 4.0 的主要新功能为服务器推送和全新 API，该 API 可在运行时发现 servlet 的 URL 映射。
- 服务器推送是最直观的 HTTP/2 强化功能，通过 PushBuilder 接口在 servlet 中公开。
- 全新 servlet 映射发现接口 HttpServletMapping 使框架能够获取有关激活给定 servlet 请求的 URL 信息。这可能对框架尤为有用，这些框架需要这一信息来运行内部工作。

# 知识点1：如何获取全新 PushBuilder-4

- 服务器推送使服务器能预测客户端请求的资源需求。然后，在完成请求处理之前，它可以将这些资源发送到客户端。
- 要了解服务器推送的好处，可以考虑一个包含图像和其他依赖项（比如 CSS 和 JavaScript 文件）的网页。客户端发出一个针对该网页的请求。服务器然后分析所请求的页面，确定呈现它所需的资源，并主动将这些资源发送到客户端的缓存。
- 在执行所有这些操作的同时，服务器仍在处理原始网页请求。客户端收到响应时，它需要的资源已经位于缓存中。

# 知识点1：如何获取全新 PushBuilder-5

- 什么是 Server Push
- Server Push 是 HTTP/2 规范中引入的一种新技术，即服务端在没有被客户端明确的询问下，抢先地“推送”一些网站资源给客户端（浏览器），该特性可以极大的改善页面访问效果。

# 知识点1：如何获取全新 PushBuilder-6


**未开启 HTTP/2 Server Push**

WEB 浏览器访问 WEB 服务端遵循着请求--响应模式。也即 WEB 浏览器请求一个资源，WEB 服务器响应一个资源。以常规的网页为例，当请求一个 /index.html 后，WEB 服务端响应一个 /index.html 页面给 WEB 浏览器，此时 WEB 浏览器会去解析该 /index.html 页面，发现还需要去加载 JS、CSS、图片等资源，此时客户端会依次去请求这些资源。这无形当中影响了首屏渲染的时间，不利于页面快速加载和渲染。

# 知识点1：如何获取全新 PushBuilder-7


已开启 HTTP/2 Server Push

- 使用 Server Push 技术之后，当 WEB 浏览器请求 /index.html 之后，WEB 服务端会直接将需要推送的资源一并发给WEB浏览器，而不需要WEB浏览器依次进行请求，这减少了WEB浏览器进行请求所消耗的时间。

# 知识点1：如何获取全新 PushBuilder-8

- Servlet 4.0 通过 PushBuilder 接口公开服务器推送。为了能够进行访问，需要通过调用 newPushBuilder() 方法，从 HttpServletRequest 获取 PushBuilder 实例。下面示例展示了如何获取 PushBuilder 实例。

# 知识点1：如何获取全新 PushBuilder-9

课堂案例：
PushBuilderTest.java

```
@WebServlet("/PushBuilderTest")
public class PushBuilderTest extends HttpServlet {
        private static final long serialVersionUID = 1L;

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

  PushBuilder pushBuilder = request.newPushBuilder();

  }
}
```

# 知识点1：如何获取全新 PushBuilder-10

- 每次调用 newPushBuilder() 方法时，都将返回 PushBuilder 的新实例。如果服务器推送不可用，newPushBuilder() 将返回 null。在某些情况下，客户端可能会为请求事务拒绝服务器推送。如果客户端没有使用安全连接，服务器推送也不会起作用。因此，务必要在对 PushBuilder 实例调用方法之前，针对 null 返回值进行测试。

```
信息:Server startup in [593] milliseconds
pb=======null
```

- 顾名思义，PushBuilder 实现 Builder 模式。在这一实现过程中，通过链接赋值方法构建推送请求。这些赋值方法通过设置 HTTP 标头、方法类型（GET 是唯一的可接受值）、查询字符串、会话 ID 和资源路径（即，将要推出资源的路径），来配置 PushBuilder 实例。

# 知识点1：如何获取全新 PushBuilder-11


## 无法显示此页

在高级设置中启用 TLS 1.0、TLS 1.1 和 TLS 1.2，然后尝试再次连接到 **https://localhost:8080**。如果此错误依然存在，则可能是因为此站点使用了不受支持的协议或不安全的密码套件，例如 RC4 (详细信息链接)。请与站点管理员联系。

更改设置

# 知识点1：如何获取全新 PushBuilder-12

**server.xml 中释放注解**

```
<!--
<Connector port="8443" protocol="org.apache.coyote.http11.Http11AprProtocol"
           maxThreads="150" SSLEnabled="true" >
    <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />
    <SSLHostConfig>
        <Certificate certificateKeyFile="conf/localhost-rsa-key.pem"
                     certificateFile="conf/localhost-rsa-cert.pem"
                     certificateChainFile="conf/localhost-rsa-chain.pem"
                     type="RSA" />
    </SSLHostConfig>
</Connector>
-->
```

# 知识点2：HttpServletMapping接口-1

- Servlet 4.0 的全新 servlet 映射发现 API 使服务器能够对 URL（可调用 servlet）执行运行时检查。例如，对 1.txt, /brain 和 /brain/1.txt 的请求将通过 URL 模式 /brain/* 和 *.txt 激活 servlet。
- HttpServletMapping 接口支持运行时发现 servlet 的映射 URL。可以在 HttpServletRequest 实例上调用 getHttpServletMapping()，获取接口的实例。

# 知识点2：HttpServletMapping接口-2

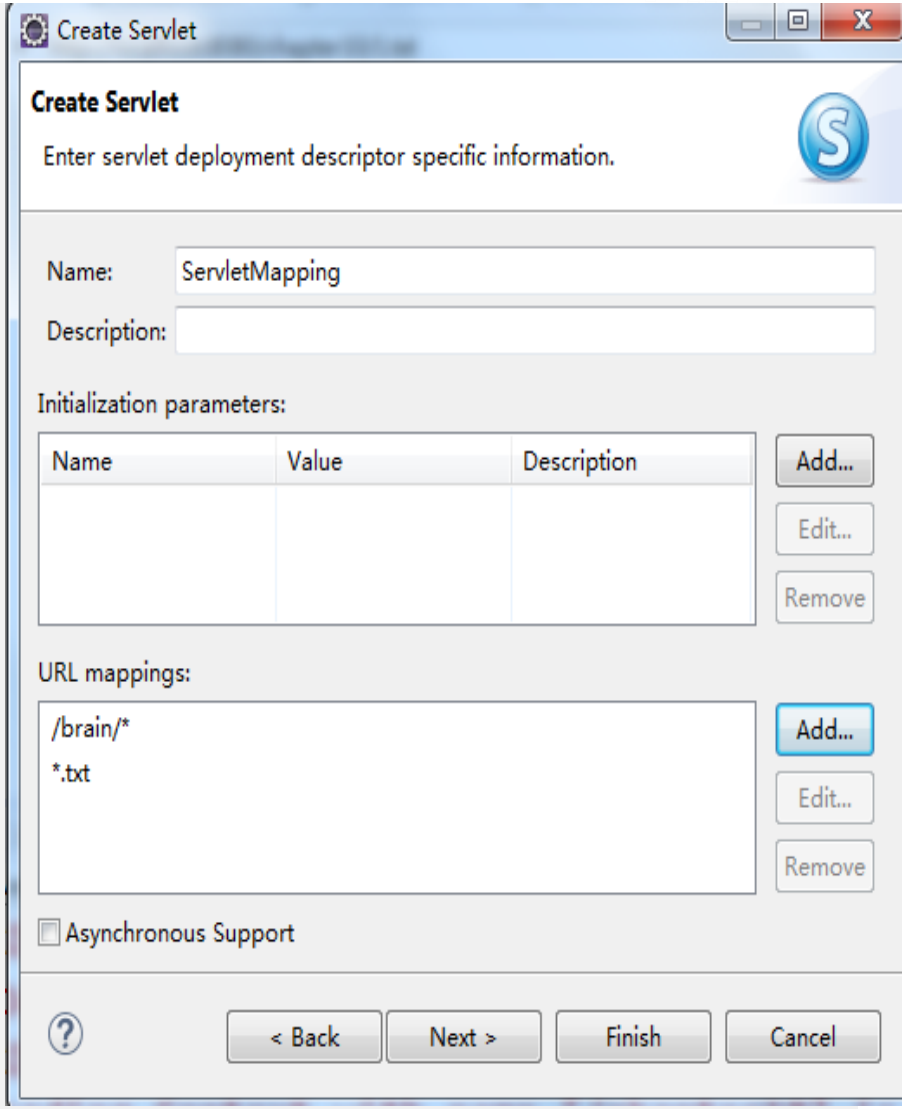
- 可以使用以下方法获取有关 servlet 映射 URL 的信息：
- getMatchValue() 返回部分 URI 路径，该路径会导致请求匹配。
- getPattern() 返回 URL 模式的 String 表示形式。
- getServletName() 返回 servlet 名称的 String 表示形式。
- getMappingMatch() 返回匹配的类型，表示为 MappingMatch枚举值，

该枚举值将为以下值之一：CONTEXT_ROOT、DEFAULT、EXACT、EXTENSION 或 PATH。

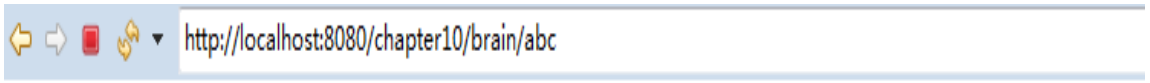
# 知识点2：HttpServletMapping接口-4

课堂案例：
ServletMapping.java

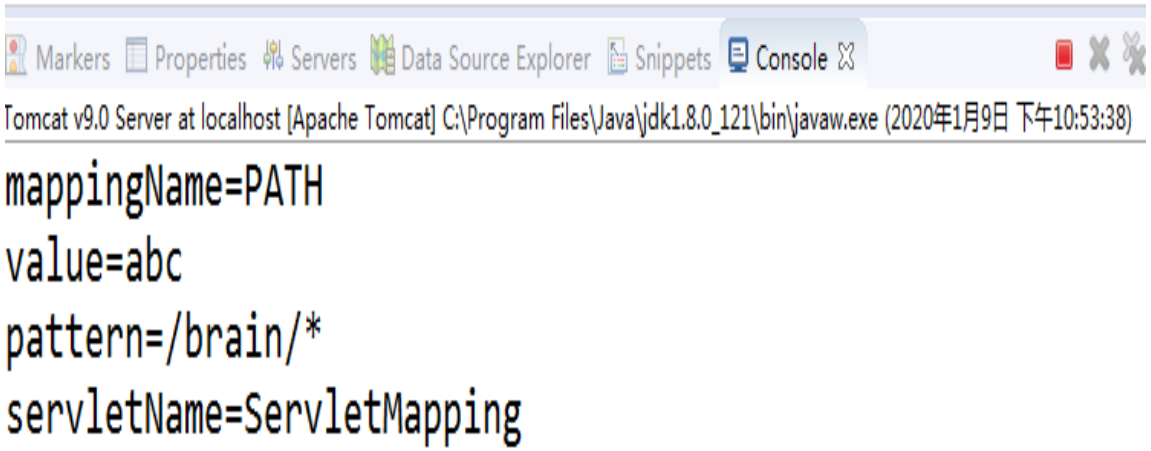
```
@WebServlet({"/brain/*", "*.txt"})
public class ServletMapping extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response)
 throws IOException {
                HttpServletMapping mapping = request.getHttpServletMapping();
                 String mappingName = mapping.getMappingMatch().name();
                  System.out.println("mappingName="+mappingName);

                  String value = mapping.getMatchValue();
                  System.out.println("value="+value);

                  String pattern = mapping.getPattern();
                  System.out.println("pattern="+pattern);

                  String servletName = mapping.getServletName();
                  System.out.println("servletName="+servletName);
 }}
```

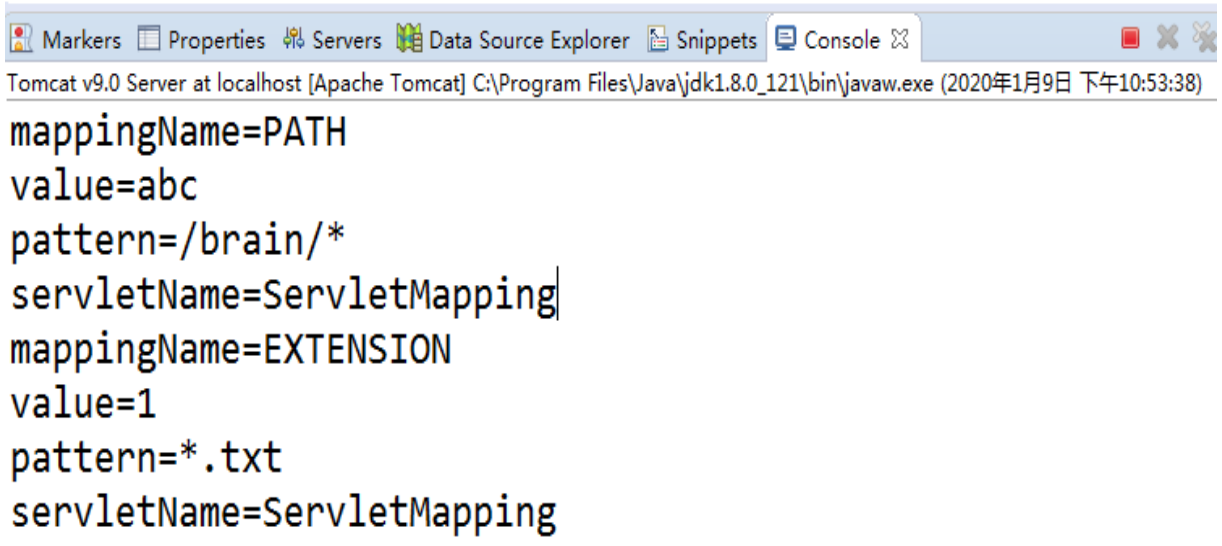


# 知识点2：HttpServletMapping接口-4 访问结果







# 知识点3：Servlet 4.0 的细微变化

- 除了服务器推送和全新 HttpServletMapping 接口，Servlet 4.0 还包括少量值得注意的新增功能和变更。
- 1. Trailer 响应标头支持发送方在分块消息的末尾包含额外字段。这用于提供在发送消息主体时可能会动态生成的元数据，例如，消息完整性检查、数字签名或后期处理状态。
- 2. Servlet 4.0 添加了 GenericFilter 和 HttpFilter 抽象类，这些抽象类通过最低限度地实现生命周期方法 init() 和 destroy()，简化了编写过滤器。
- 3. Servlet 4.0 还集成了全新的 HTTP Trailer，支持发送方在分块消息的末尾包含额外的字段。

# 知识点3：Servlet 4.0 的细微变化

- 4. ServletContext 接口采用了一些新方法：

•addJspFile() 可将带有给定 JSP 文件的 servlet 添加到 servlet 上下文中。

•getSessionTimeout() 和 setSessionTimeout() 可提供对会话超时的访问权限。

•getRequestCharacterEncoding() 和 setRequestCharacterEncoding() 可为当前 的 servlet 上下文提供访问权限，并改变默认的请求字符编码。

- 5. HttpServletRequest 接口上的 isRequestedSessionIdFromUrl() 方法已被弃用。
- 6. 由于升级到 Java SE 8，默认方法已被添加到侦听器接口中。

Servlet 4.0 API Documentation - Apache Tomcat 9.0.0.M26

隐藏 上一步 前进 主页 打印

目录(C) 索引(N)

键入关键字进行查找(W):

HttpServletMapping

All Classes
All Classes
API Help
AsyncContext
AsyncEvent
AsyncListener
Class Hierarchy
Constant Field Values
Cookie
Deprecated List
DispatcherType
Filter
FilterChain
FilterConfig
FilterRegistration
FilterRegistration.Dynamic
GenericFilter
GenericServlet
HandlesTypes
HttpConstraint
HttpConstraintElement
HttpFilter
HttpMethodConstraint
HttpMethodConstraintElement
HttpServlet
HttpServletMapping
HttpServletRequest
HttpServletRequestWrapper

OVERVIEW PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

javax.servlet.http

# Interface HttpServletMapping

---

```
public interface HttpServletMapping
```

Represents how the request from which this object was obtained was mapped to the associated servlet.

Since:

4.0

## *Method Summary*

All Methods | Instance Methods | Abstract Methods

| Modifier and Type | Method and Description |
| --- | --- |
| MappingMatch | getMappingMatch() |

# 本节总结提问【Servlet4.0新特性】

- Servlet4.0通过哪个接口公开服务器推送?

- 除了服务器推送和全新 HttpServletMapping 接口，Servlet 4.0 还包括哪些值得注意的新增功能和变更?

# 本节总结【 Servlet4.0新特性】

- Servlet 4.0 的主要新功能为服务器推送和全新 API，服务器推送是最直观的 HTTP/2 强化功能，通过 PushBuilder 接口在 servlet 中公开；
- PushBuilder API 实现了服务器推送功能。即使仍在服务器端处理请求，它也能够将后续请求推送到客户端；
- Servlet 4.0 还包括少量值得注意的新增功能和变更

Trailer、添加了 GenericFilter 和 HttpFilter 抽象类、集成了全新的 HTTP Trailer、支持发送方在分块消息的末尾包含额外的字段、ServletContext 接口采用了一些新方法、HttpServletRequest 接口上的 isRequestedSessionIdFromUrl() 方法已被弃用、 由于升级到 Java SE 8，默认方法已被添加到侦听器接口中。

# 本章总结

- 非阻塞
- 注解
- 如何获取全新PushBuilder
- HttpServletMapping接口
- Servlet 4.0 的细微变化

# 本章作业

- 作业1:

题目：在注册功能中，增加上传头像功能。使用Servlet3.0的非阻塞特性、注解、文件上传相关API。

难度：中

- 作业2:

题目：使用Servlet 4.0 的全新 servlet 映射发现 API 使服务器能够对 URL（可调用 servlet）执行运行时检查。

难度：中

# Web课程回顾与总结

# Web课程回顾与总结

- 第1章-Web快速入门

  第1节：基本概念及Web应用入门

  B/S、C/S、RIA

  第2节：介绍第一个Web应用

  HelloWorldServlet、生命周期

- 第2章-Servlet入门

  第1节：Servlet入门

  javax.servlet. ServletRequest、javax.servlet. ServletResponse

# Web课程回顾与总结

- 第3章-JSP入门

  第1节：概述　----什么是JSP

  第2节：页面元素及内置对象概念

  脚本元素、表达式元素、声明元素

  第3节：Servlet与JSP作用总结 ----Dispatcher、Forward

- 第4章-会话跟踪

  第1节：会话跟踪概述

  第2节：　Cookie　----客户端

  第3节：　Session　----服务器

# Web课程回顾与总结

- 第5章-上下文

  第1节：ServletContext接口 ----上下文对象的概念、作用

  第2节：数据作用域 ---- 4个作用域范围

- 第6章-监听器与过滤器

  第1节：监听器 ----8种监听器接口

  第2节：过滤器 ----Filter接口

# Web课程回顾与总结

- 第7章-JSP其他主题

  第1节：内置对象 ----9大内置对象

  第2节：指令与动作 ----三种指令、若干动作

- 第8章-MVC模式

  第1节：实现MVC模式 ----M、V、C

  第2节：MVC思考 ----解决重复提交

# Web课程回顾与总结

- 第9章-EL与JSTL

  第1节：表达式语言EL　----表达式语言

  第2节：标准标签库JSTL ----JSP标准标签库

- 第10章-Servlet新特性

  第1节：　Servlet3.0新特性 ----非阻塞、注解、文件上传

  第2节：　Servlet4.0新特性 ----PushBuilder、HttpServletMapping接口及细微变化