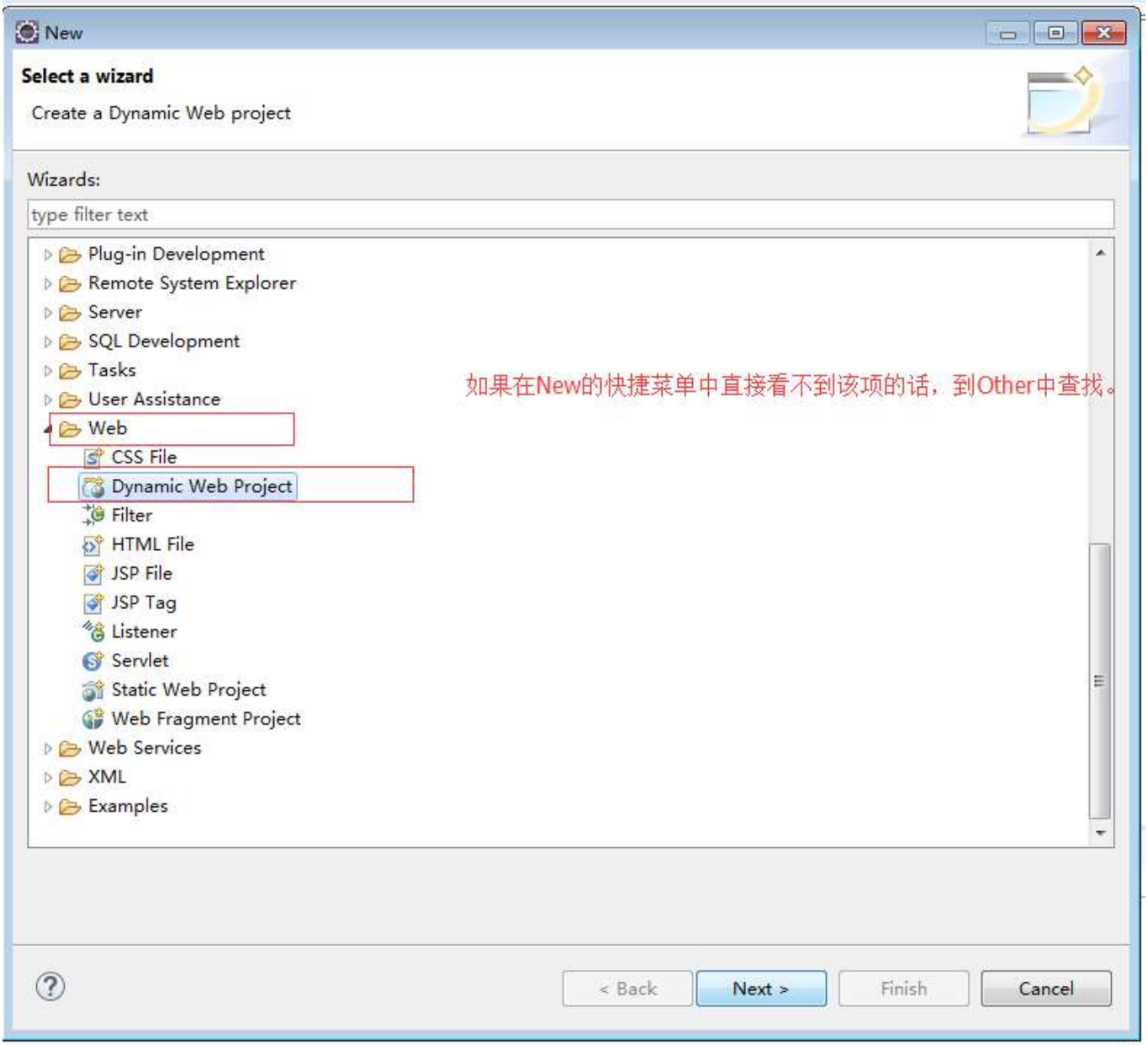
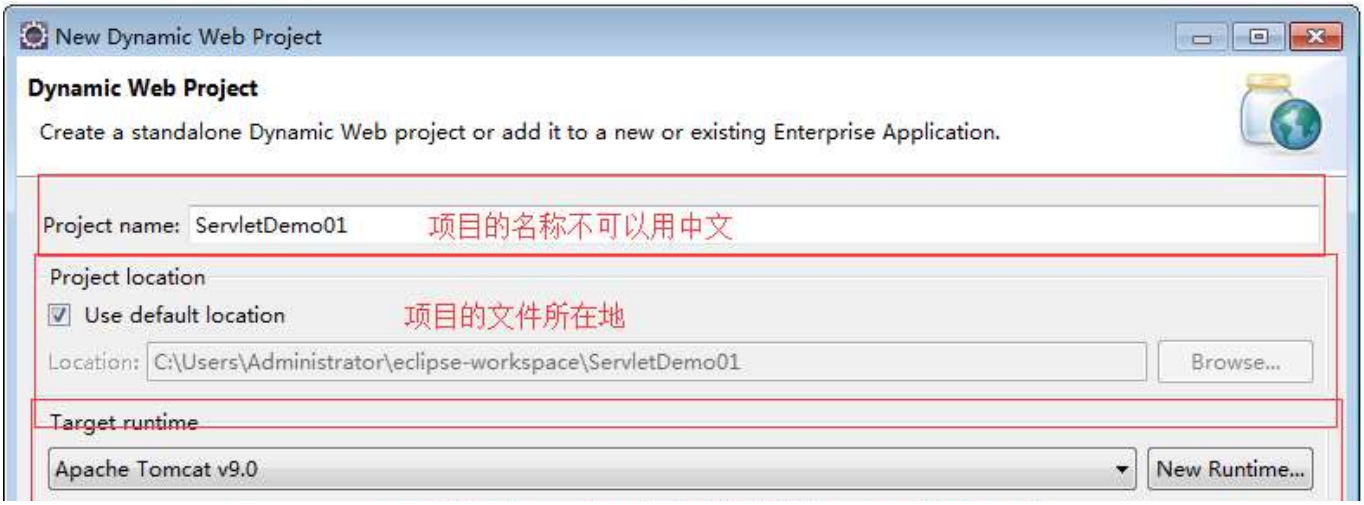
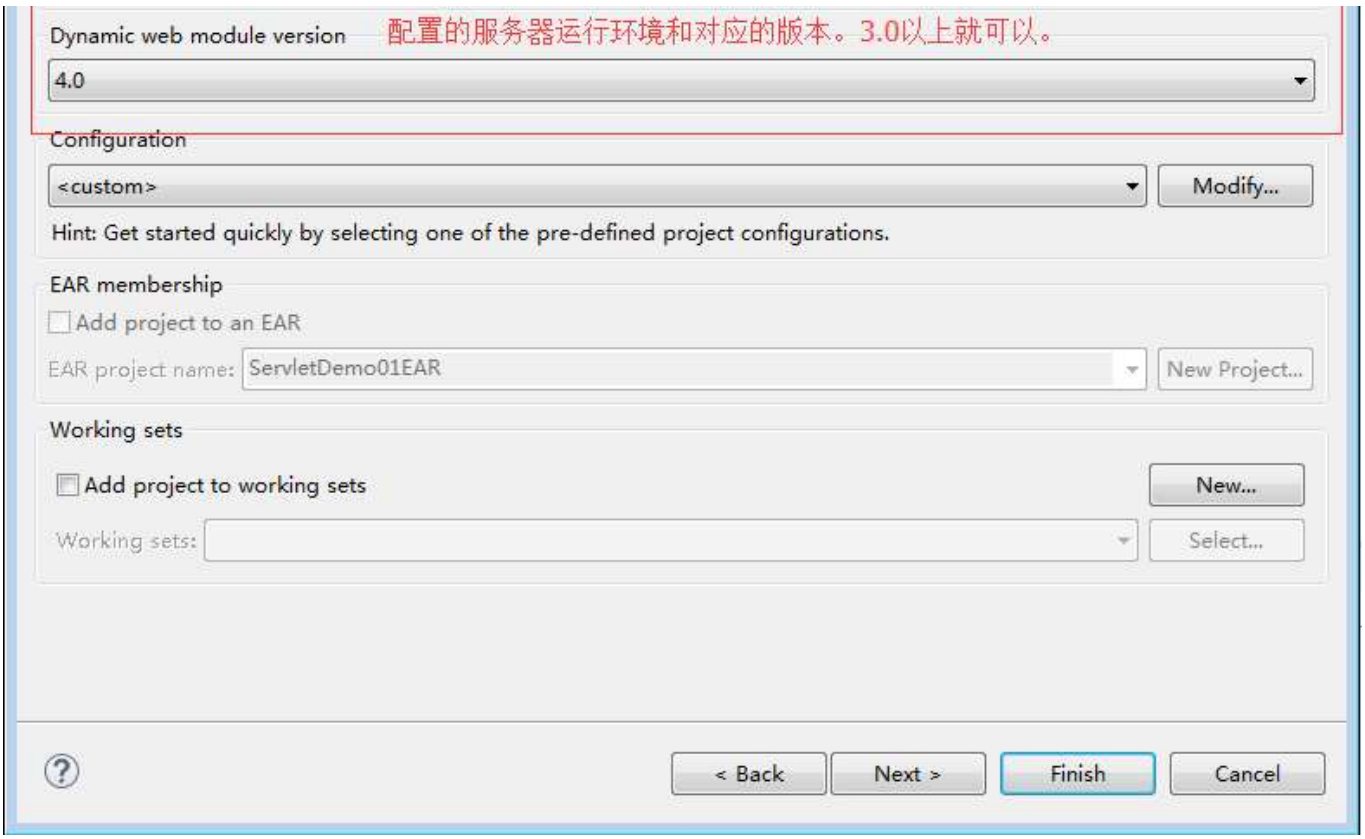
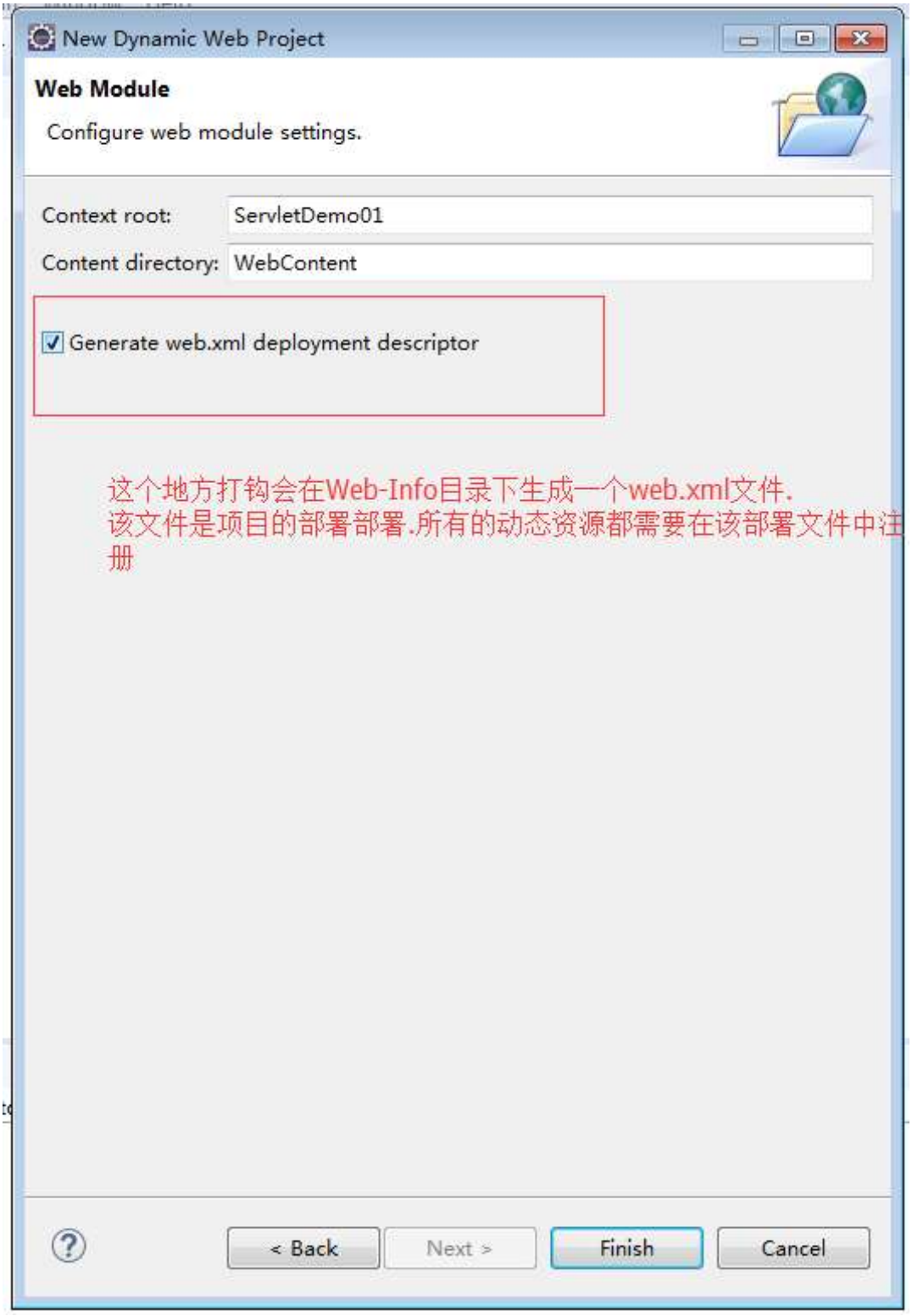
Servlet 笔记

在Eclipse找那个建立Web项目

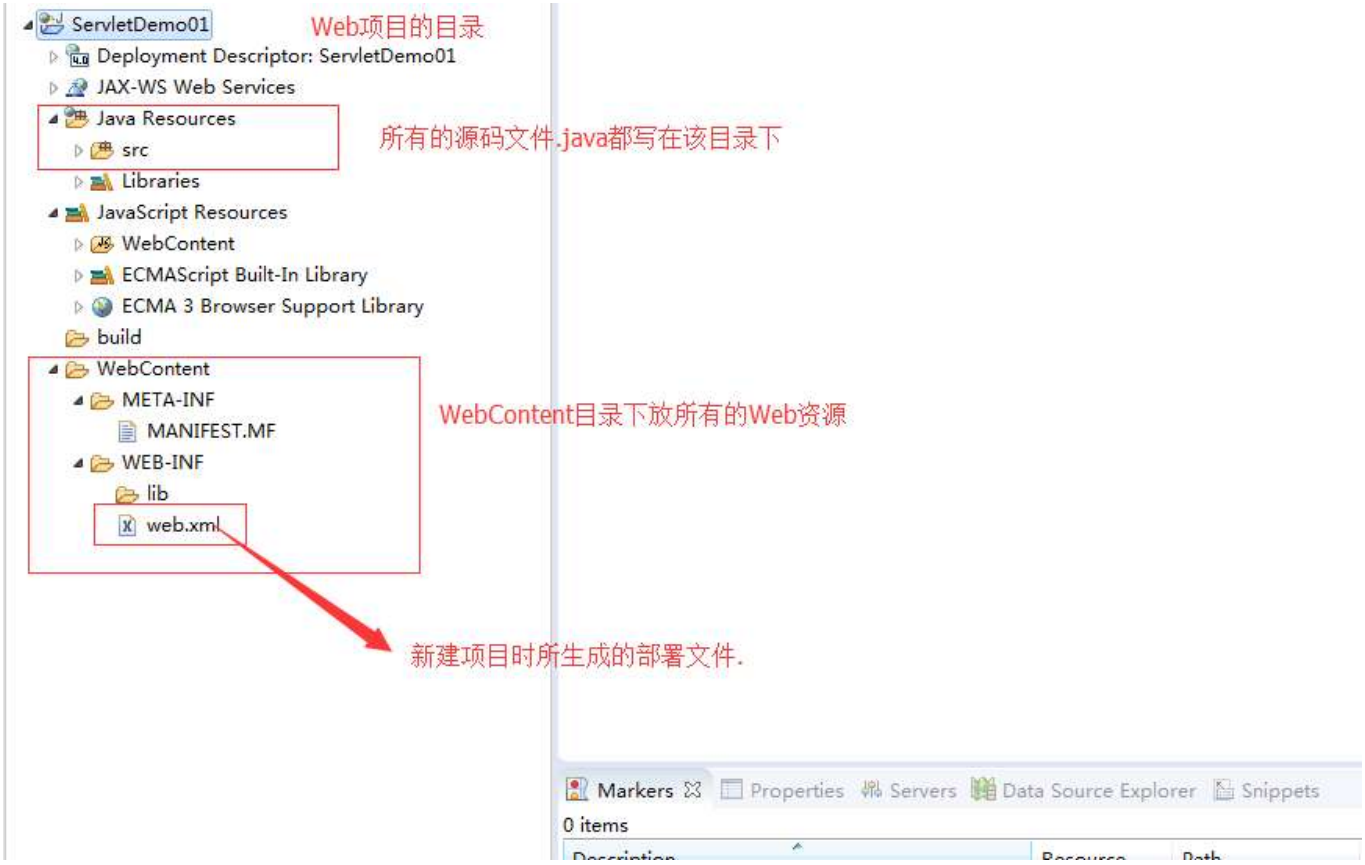

New
Select a wizard
Create a Dynamic Web project
Wizards:
type filter text
Plug-in Development
Remote System Explorer
Server
SQL Development
Tasks
User Assistance
Web
CSS File
Dynamic Web Project
Filter
HTML File
JSP File
JSP Tag
Listener
Servlet
Static Web Project
Web Fragment Project
Web Services
XML
Examples
如果在New的快捷菜单中直接看不到该项的话，到Other中查找。
< Back
Next >
Finish
Cancel


New Dynamic Web Project
Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.
Project name: ServletDemo01
项目的名称不可以用中文
Project location
Use default location
项目的文件所在地
Location: C:\Users\Administrator\eclipse-workspace\ServletDemo01
Browse...
Target runtime
Apache Tomcat v9.0
New Runtime...

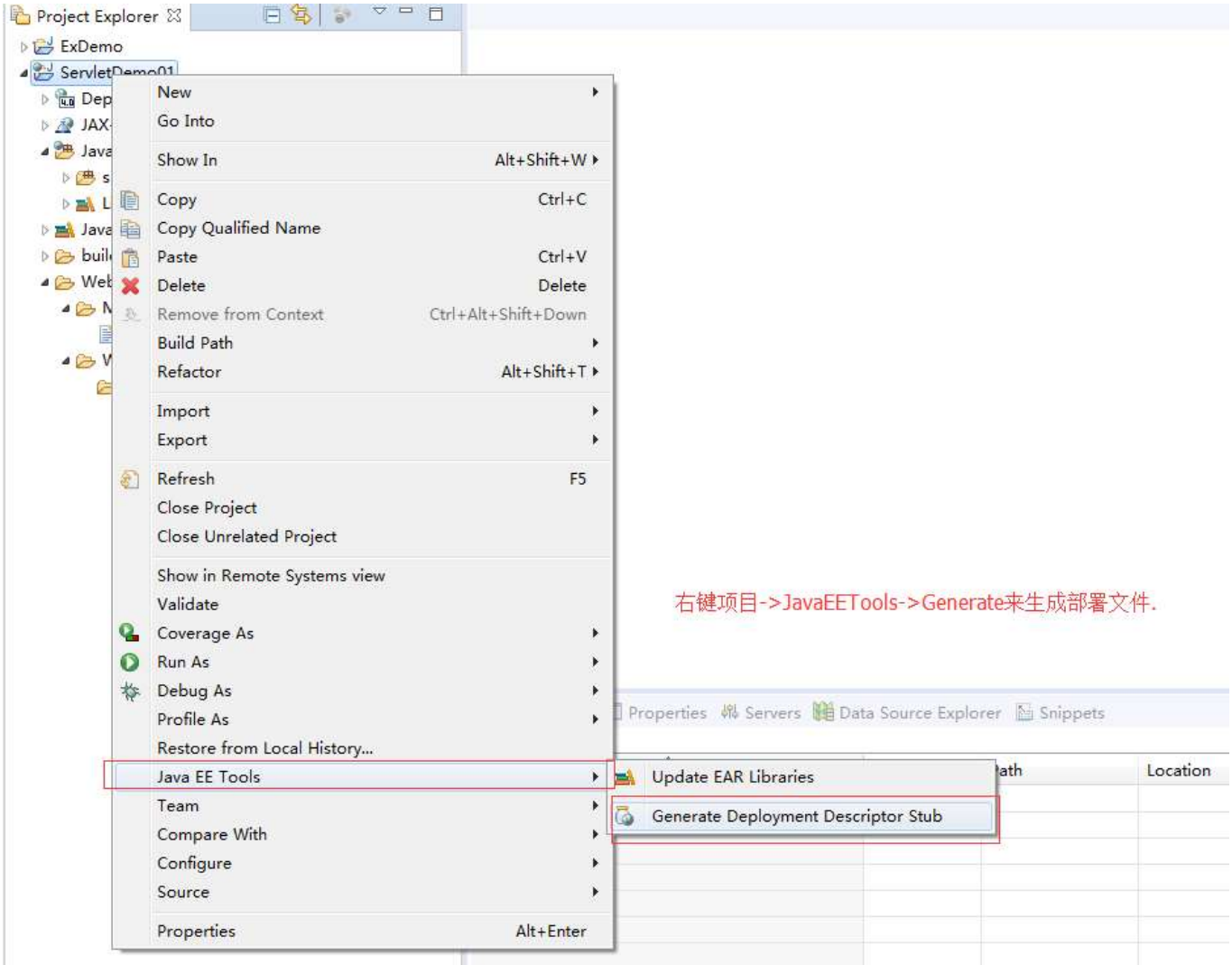

Dynamic web module version
配置的服务器运行环境和对应的版本。3.0以上就可以。
4.0
Configuration
<custom>
Modify...
Hint: Get started quickly by selecting one of the pre-defined project configurations.
EAR membership
Add project to an EAR
EAR project name:
ServletDemo01EAR
New Project...
Working sets
Add project to working sets
New...
Working sets:
Select...
< Back
Next >
Finish
Cancel


New Dynamic Web Project
Web Module
Configure web module settings.
Context root:
ServletDemo01
Content directory:
WebContent
Generate web.xml deployment descriptor
这个地方打钩会在Web-Info目录下生成一个web.xml文件.
该文件是项目的部署部署.所有的动态资源都需要在该部署文件中注
册
< Back
Next >
Finish
Cancel

项目目录结构介绍



在新建项目时没有选择创建部署文件怎么办



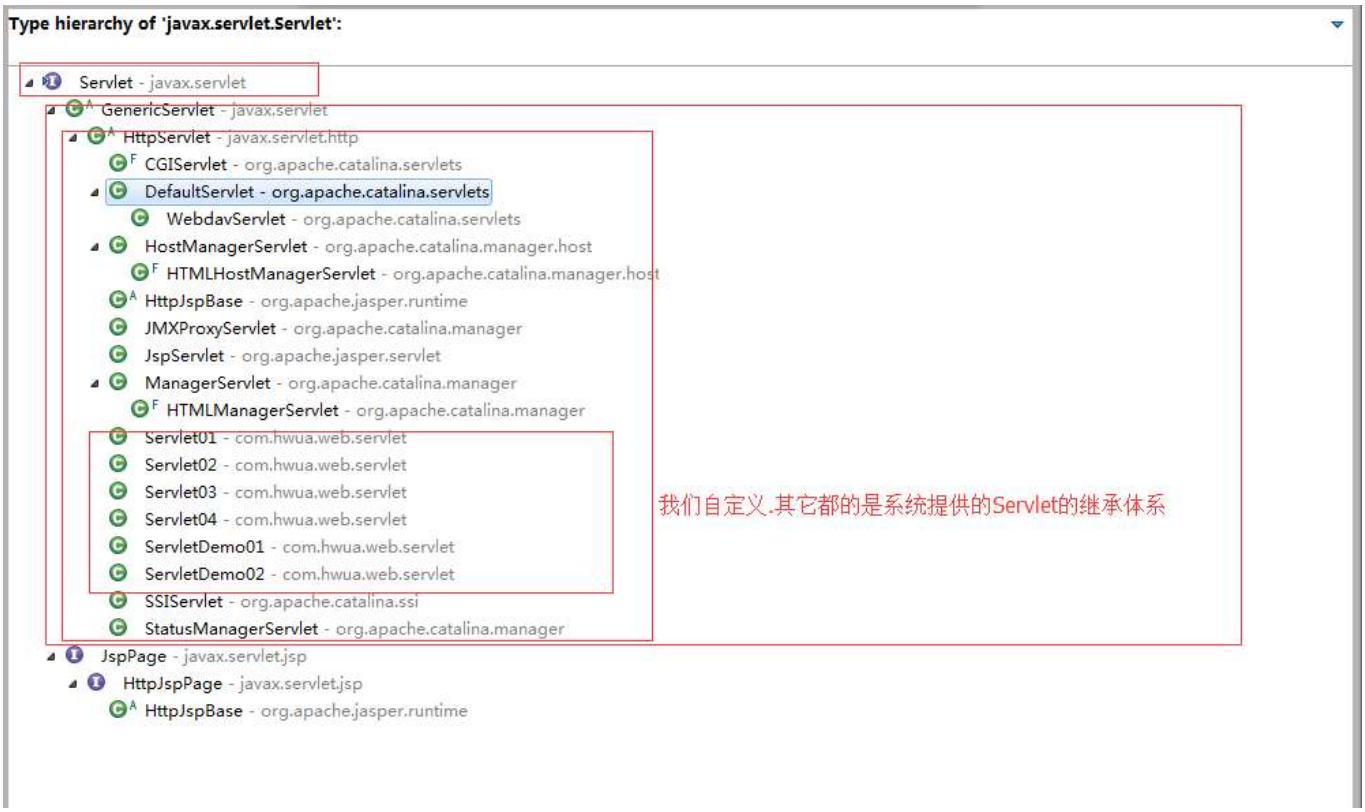
什么是 Servlet

在 ***Servlet.java***的类头有这么一段注释

```
A servlet is a small Java program that runs within a Web server.
Servlets receive and respond to requests from Web clients, usually across HTTP, the
HyperText Transfer Protocol
```

大致含义可以理解为：Servlet 就是一个运行在 Web 服务器上的Java 应用程序.Servlet会接收和响应那些来自于 Web 客户端的请求，一般这些请求都是通过 HTTP 协议传递的。

Servlet的关系树



创建第一个Servlet

> 实现 Servlet 接口

```java
package com.hwua.web.servlet;

import java.io.IOException;

import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebServlet;

/**
 *
 * @author Administrator
 *
 */
@WebServlet(name = "Servlet05", urlPatterns = { "/Servlet05" }, loadOnStartup = 1)
public class Servlet05 implements Servlet {

    @Override
    public void init(ServletConfig config) throws ServletException {
        System.out.println("Servlet被初始化");

    }

    @Override
    public ServletConfig getServletConfig() {
        System.out.println("config");
        return null;
    }

    @Override
    public void service(ServletRequest req, ServletResponse res) throws ServletException,
IOException {
        System.out.println("收到一个请求");
    }

    @Override
    public String getServletInfo() {
        System.out.println("info");
        return null;
    }

    @Override
    public void destroy() {
        System.out.println("销毁Servlet");
    }
}
```

如何配置 Servlet

配置Servlet就是向Tomcat中注册一个Servlet.注册方式有两种

- 在部署文件中(web.xml)中注册
- 使用注解进行注册

> 是Servlet3.0以后的内容.

XML 配置方式

在部署文件中注册Servlet需要提供两个内容

- servlet
- servlet的映射内容

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
    version="4.0" metadata-complete="true">
    <display-name>ServletDemo02</display-name>
    <!-- 把Servlet05添加到部署文件中 -->
    <servlet>
        <!-- name是任意有效的标识符都可以 -->
        <servlet-name>ABC</servlet-name>
        <!-- 告诉Tomcat这个Servlet的源文件在哪里.所以这里配置的是完整类名 -->
        <servlet-class>com.hwua.web.servlet.Servlet05</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <!-- name是任意有效的标识符都可以.且一定要和上面的name一致. -->
        <servlet-name>ABC</servlet-name>
        <!-- 请求路径 随便写的. -->
        <url-pattern>/ABC</url-pattern>
    </servlet-mapping>
</web-app>
```

还要独立说明的就是 `metadata-complete=true` .这个是告诉Tomcat你的部署文件中的内容是否完整.如果为true就代表web.xml这个部署文件中的内容是完整的,那就不会扫描注解,即使用注解做的servlet配置无效.默认是false.就代表部署文件中的配置不完整,既然不完整Tomcat会扫描部署文件和所有的注解.是Tomcat3.0以上版本的内容.

注解配置方式

注解的配置方式很简单就是在任意Servlet的类头添加一下内容就可以了

```
@WebServlet(name = "Servlet05", urlPatterns = { "/Servlet05" }, loadOnStartup = 1)
```

Servlet 的执行过程

参考Word文档.

Servlet 的通用写法

> HttpServlet 介绍

我们一般在写Servlet时很少去使用实现接口的方式.并且我们的Web开发一般都是针对HTTP协议的.所以使用比较多的是通过继承HttpServlet来写.

生命周期中的方法.在子类没有重写的情况下会默认执行父类的.所以当我们选择继承HttpServlet的时候我们需要重写的方法只有两个

- doGet
- doPost

如果有特殊需要可以自行重写生命周期中的方法.具体代码如下

```
package com.hwua.web.servlet;

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

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

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Servlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        response.getWriter().append("Served at: ").append(request.getContextPath());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }

}
```

Servlet的生命周期

```
This interface defines methods to initialize a servlet, to service requests,
 * and to remove a servlet from the server. These are known as life-cycle
 * methods and are called in the following sequence:
 * <ol>
 * <li>The servlet is constructed, then initialized with the <code>init</code>
 * method.
 * <li>Any calls from clients to the <code>service</code> method are handled.
 * <li>The servlet is taken out of service, then destroyed with the
 * <code>destroy</code> method, then garbage collected and finalized.
 * </ol>
 *
 * <p>
```

在Servlet接口的源码中有上面这段注释.它告诉我们Servlet需要遵守以下的声明周期

- init

> 用于初始化Servlet

- service

> 用于响应来自于客户端的所有请求

- destroy

> 当Servlet从WebServer中移除的(从Tomcat中移除的)时候.将会调用该方法来销毁Servlet

init方法.

> 用来实例化Servlet

简单说Servlet是单例的.init方法只会执行一次,在Servlet对象第一次被创建的时候执行.默认情况下只有当请求来的时候才会去创建对应的Servlet.如果希望在服务器启动(Tomcat启动的)时就完成对Servlet的实例化操作需要在配置中添加配置***loadOnStartup***.给与的值只要大于1就可以.如果有多个Servlet都配置了该属性根据值的大小来选择优先.如果有多个Servlet配置了相同的值.根据在XML中的注册顺序来选择优先.

service方法.

```
This method is only called after the servlet's <code>init()</code> method has completed
successfully.
```

只有当Servlet的init方法执行成功后才会执行service方法.service方法在HtppServlet中提供了实现,具体代码如下

```
@Override
    public void service(ServletRequest req, ServletResponse res)
        throws ServletException, IOException {

        HttpServletRequest  request;
        HttpServletResponse response;

        try {
            request = (HttpServletRequest) req;
            response = (HttpServletResponse) res;
        } catch (ClassCastException e) {
            throw new ServletException("non-HTTP request or response");
        }
        service(request, response);
    }
```

我们从这段代码可以发现.把参数ServletRequest和ServletResponse转换成与之对应的HttpServletRequest和HttpServletResponse.并将两个局部变量(线程安全)传递给自定义的service.也就是说.Servlet生命周期中的service方法的处理核心在自定义的service中.该方法的代码如下

```
protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

        String method = req.getMethod();

        if (method.equals(METHOD_GET)) {
            long lastModified = getLastModified(req);
            if (lastModified == -1) {
                // servlet doesn't support if-modified-since, no reason
                // to go through further expensive logic
                doGet(req, resp);
            } else {
                long ifModifiedSince;
                try {
                    ifModifiedSince = req.getDateHeader(HEADER_IFMODSINCE);
                } catch (IllegalArgumentException iae) {
                    // Invalid date header - proceed as if none was set
                    ifModifiedSince = -1;
                }
                if (ifModifiedSince < (lastModified / 1000 * 1000)) {
                    // If the servlet mod time is later, call doGet()
                    // Round down to the nearest second for a proper compare
                    // A ifModifiedSince of -1 will always be less
                    maybeSetLastModified(resp, lastModified);
                    doGet(req, resp);
                } else {
                    resp.setStatus(HttpServletResponse.SC_NOT_MODIFIED);
                }
            }

        } else if (method.equals(METHOD_HEAD)) {
            long lastModified = getLastModified(req);
            maybeSetLastModified(resp, lastModified);
            doHead(req, resp);

        } else if (method.equals(METHOD_POST)) {
            doPost(req, resp);

        } else if (method.equals(METHOD_PUT)) {
            doPut(req, resp);

        } else if (method.equals(METHOD_DELETE)) {
            doDelete(req, resp);

        } else if (method.equals(METHOD_OPTIONS)) {
            doOptions(req,resp);

        } else if (method.equals(METHOD_TRACE)) {
            doTrace(req,resp);

        } else {
            //
            // Note that this means NO servlet supports whatever
            // method was requested, anywhere on this server.
            //

            String errMsg = lStrings.getString("http.method_not_implemented");
            Object[] errArgs = new Object[1];
            errArgs[0] = method;
            errMsg = MessageFormat.format(errMsg, errArgs);
```

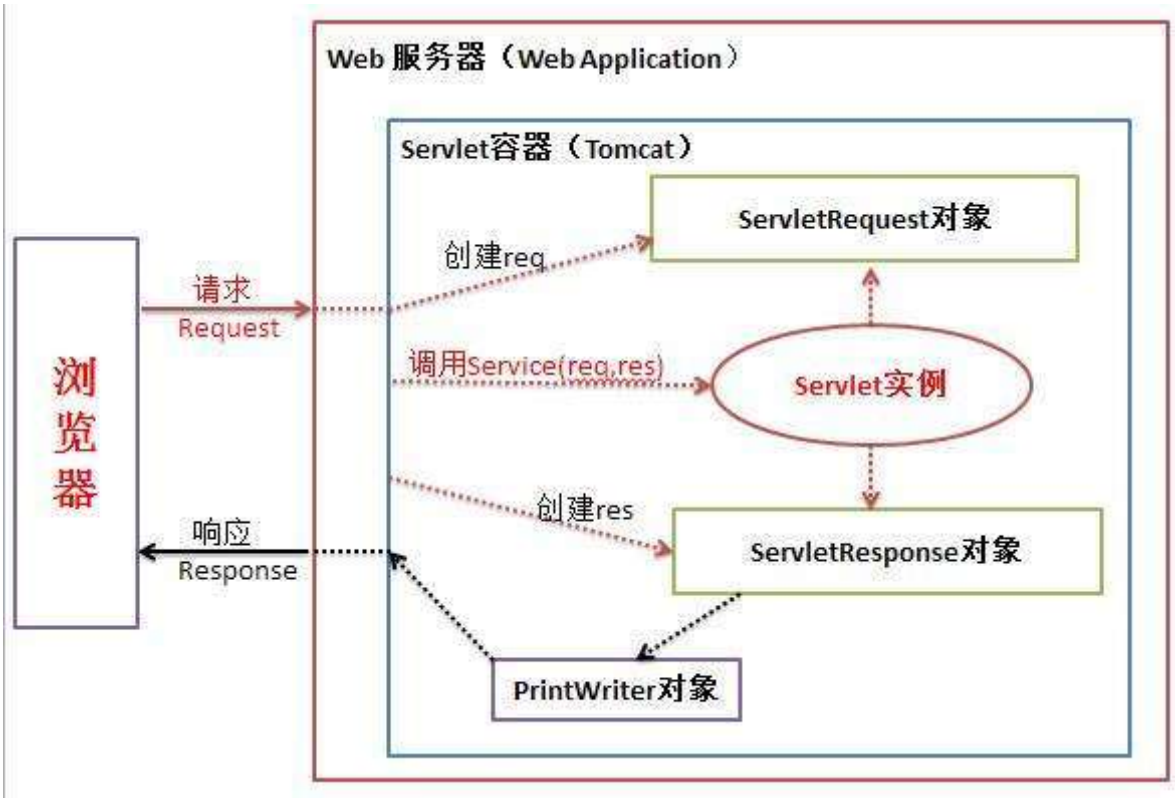
```
            resp.sendError(HttpServletResponse.SC_NOT_IMPLEMENTED, errMsg);
        }
    }
```

其中 `String method = req.getMethod();` 所获取的method就是请求行中的请求方法.也就是说在HttpServlet中自定义的service方法做了以下几件事情

- 获取请求行中的请求方法
- 使用switch来匹配所获取的请求方法
- 调用与之对应的doXXX方法
  - 如果是get就调用doGet()
  - 如果是post就调用doPost()
  - 其它依次类推.

destroy()方法

当Servlet从Tomcat中移除的时候被调用用来执行销毁程序.



Servlet 的URL映射配置问题

> 明天说.

缺省 Servlet 配置

当某个Servlet的URL路径为/的时候.那这个Servlet就是当前应用程序的缺省Servlet.当在部署文件web.xml中不到servlet-mapping元素的url时,就会将请求转交给缺省Servlet.也就是说缺省Servlet是用于处理所有Servlet都不处理的请求.

测试:请求AServlet,但是该Servlet没有配置.提供缺省Servlet看该Servlet是否执行就可以了.

Servlet 的线程安全问题

> 了解就好

家庭作业

1. 在Servlet中设置一个全局变量i初始值为1.在doGet方法中对该变量进行自增1操作.求多个页面同时访问该Servlet是i的取值问题.(把i的值在页面上打印出来!)

参考

1. Servlet的执行流程