




# 软件开发环境

主讲教师 刘凡

[fanliu@hhu.edu.cn](mailto:fanliu@hhu.edu.cn)



# 第七章

## Java Servlet基础



# 本章主要内容

---

◆ 7.1 Servlet的部署、创建与运行

◆ 7.2 Servlet工作原理

◆ 7.3 通过JSP页面访问Servlet

◆ 7.4 共享变量

◆ 7.5 doGet、doPost方法

◆ 7.6 重定向与转发

◆ 7.7 使用session

# 概述

---

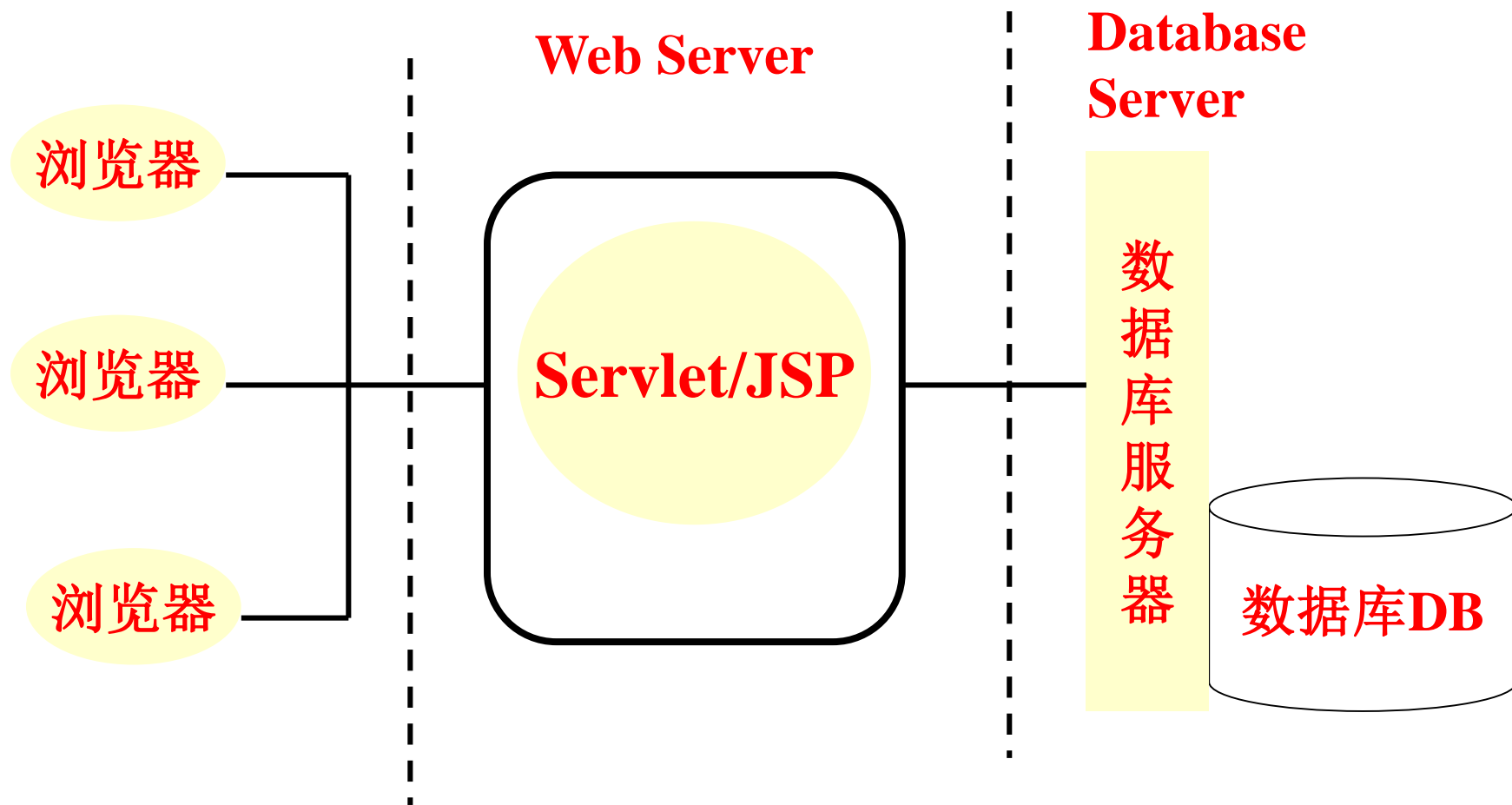
- JSP技术的根基是Servlet技术。
- Servlet技术的核心就是在服务器端创建能响应用户请求的对象，被创建的对象习惯上称为一个Servlet。
- 有些Web应用，需要JSP+Javabean+servlet来完成，即需要服务器再创建一些servlet，配合JSP页面来完成整个Web应用程序的工作。

# 概述

- Servlet程序具备以下的基本功能：
  - 获取客户端HTML的FORM表单提交的数据和URL后面的参数信息。
  - 创建和客户端的响应消息内容。
  - 访问服务器端的文件系统。
  - 连接数据库并开发基于数据库的应用。
  - 调用其他JAVA类。

# 概述

## B/S架构:



# 概述

## Servlet的特点:

- Servlet是一个供其他java程序（Servlet引擎）调用的java类，它不能独立运行。
- Servlet引擎是一种容器程序，它负责管理和维护所有Servlet对象的生命周期，因此也被称为Servlet容器或Web容器。
- Servlet的加载、执行流程，以及如何接收客户端发送的数据和如何将数据传输到客户端等具体的底层事务，都是由Servlet引擎来实现的。

## 7.1 servlet的创建、部署与运行

---

学习Java Servlet的首要任务是掌握：

- 创建servlet的类
- 保存编译这个类所得到的字节码文件
- 编写部署文件
- 运行servlet

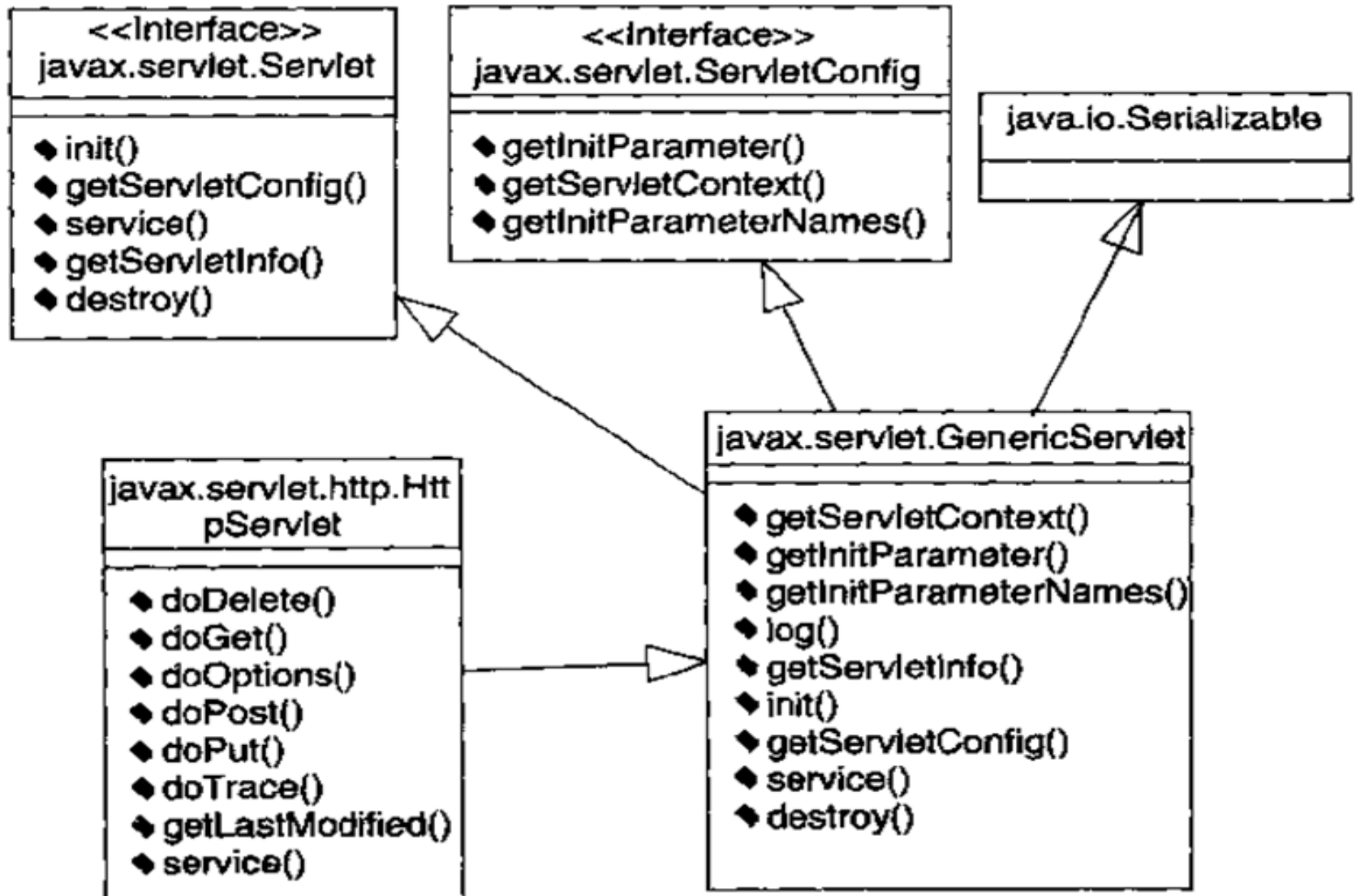


## 7.1 servlet的创建、部署与运行

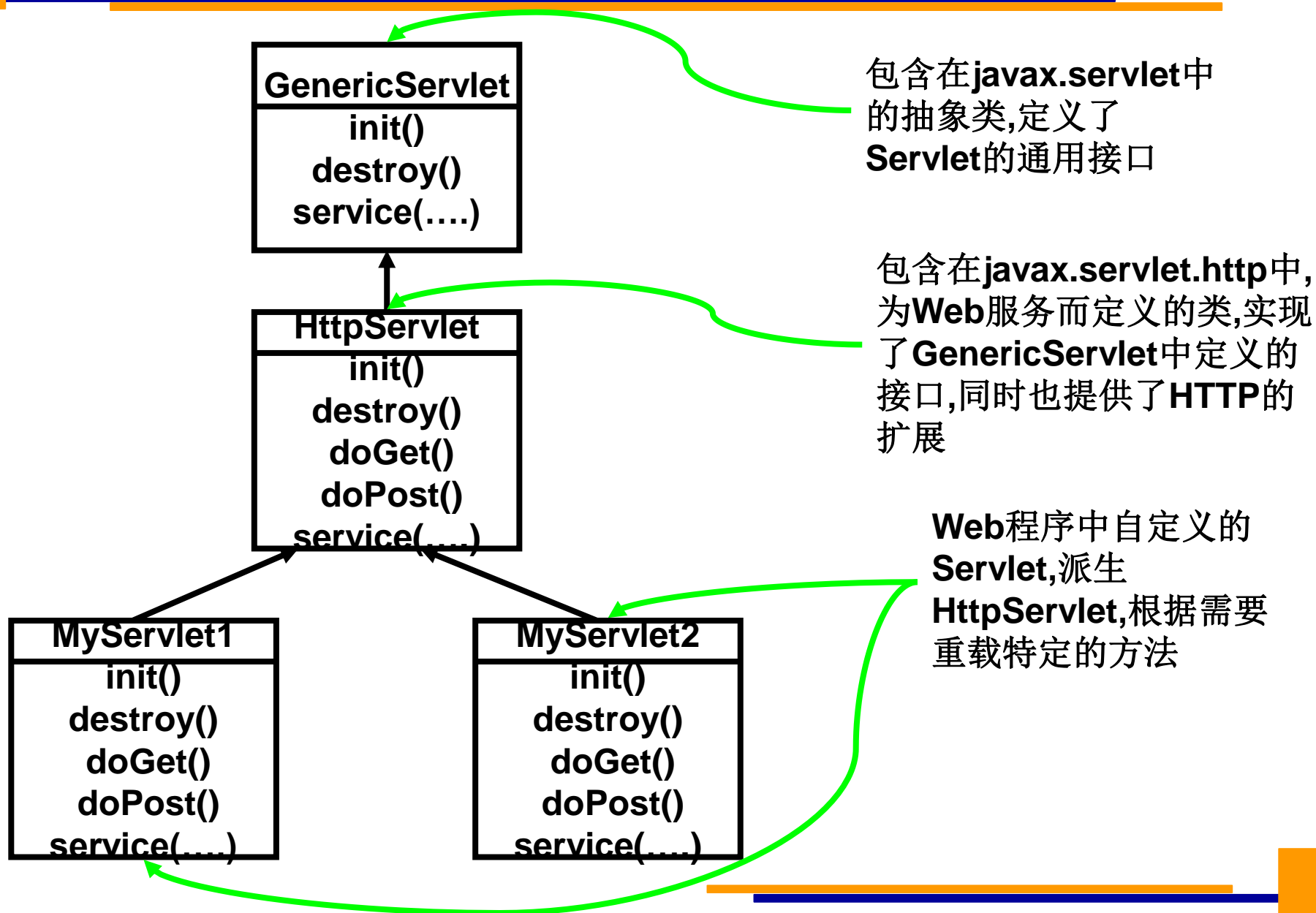
### 1、编写一个创建Servlet对象的类

- 编写一个创建Servlet对象的类就是编写一个特殊类的子类
- 这个特殊的类就是 `javax.servlet.http` 包中的 `HttpServlet` 类。
- `HttpServlet` 类实现了 **Servlet** 接口，实现了响应用户的方法。这样的子类创建的对象习惯地被称作一个 `servlet`。

## 7.1 servlet的创建、部署与运行



## 7.1 servlet的创建、部署与运行



## 7.1 servlet的创建、部署与运行

### 2、编写一个创建Servlet对象的类

- javax.servlet.http包不在JDK的核心类库中，因此需要将Tomcat安装目录lib子目录中的servlet-api.jar文件复制到Tomcat服务器所使用的JDK的扩展目录中，比如，复制到D:\jdk1.7\jre\lib\ext中。

## 7.1 servlet的创建、部署与运行

### 2、保存编译这个类所得到的字节码文件

➤ 按着其包名，保存到Web服务目录的下述子目录中：

**\WEB-INF\classes\myservlet\control**

➤ 然后按如下格式编译源文件：

**classes>javac**

**myservlet\control\Example5\_1\_Servlet.java**

## 7.1 servlet的创建、部署与运行

### 3、编写部署文件web.xml

- Servlet类的字节码保存到指定的目录后，必须为Tomcat服务器编写一个部署文件，只有这样，Tomcat服务器才会按用户的请求使用Servlet字节码文件创建对象。
- 编写的web.xml文件需要保存到Web服务目录的WEB-INF子目录中。

## 7.1 servlet的创建、部署与运行

### 3、编写部署文件web.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<web-app>
  <servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>myservlet.control.Example5_1_Servlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/lookHello</url-pattern>
  </servlet-mapping>
</web-app>
```

## 7.1 servlet的创建、部署与运行

### 4、运行servlet

- 根据 web.xml 部署文件来请求服务器执行一个 Servlet 对象。
- 根据 web.xml 文件中 <servlet-mapping> 标记指定的格式输入请求：

<http://127.0.0.1:8080/ch5/lookHello>



## 7.1 servlet的创建、部署与运行

### 4、运行servlet

<servlet-mapping>匹配规则，同一个servlet指定多个不同的url，四种不同的url形式：

- 以/开始以/字符串形式结束，精确匹配关联servlet;
- 以/开始/\*结束的形式，按纯目录形式关联servlet;
- 以\*.jsp形式，按具体文件类型关联servlet;
- 单个/形式，缺省servlet，所有没精确指定资源的url都由该servlet处理。

## 7.1 servlet的创建、部署与运行

### 4、运行servlet

考虑如下url映射规则：

Path Pattern	Servlet
/foo/bar/*	servlet1
/baz/*	servlet2
/catalog	servlet3
*.bop	servlet4

## 7.1 servlet的创建、部署与运行

### 4、运行servlet

请求行为的结果如下：

Incoming Path	Servlet
/foo/bar/index.html	<b>Servlet1</b>
/foo/bar/index.bop	<b>Servlet1</b>
/baz	<b>Servlet2</b>
/baz/inde.html	<b>Servlet2</b>
/catlog	<b>Servlet3</b>
/catlog/index.html	<b>default Servlet</b>
/catlog/racecar.bop	<b>Servlet4</b>
/inde.bop	<b>Servlet4</b>

## 7.1 servlet的创建、部署与运行

### 5、向servlet传递参数

- 在请求一个servlet时，可以在请求的url-pattern中额外的加入参数及其值，格式是：

url-pattern?参数1=值1&参数2=值…参数n=值

- 那么被请求的servlet就可以使用request对象获取参数的值：

request.getParameter(参数n)

- 比如可以在浏览器的浏览器键入：

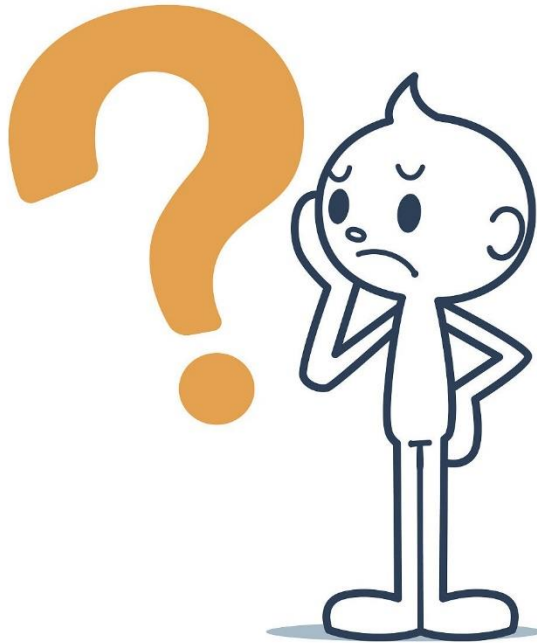
http://127.0.0.1:8080/ch5/lookHello?moon=loveliness

## 7.2 servlet的工作原理

- servlet是javax.servlet.http包中HttpServlet类的子类的一个实例、由服务器负责创建并完成初始化工作。
- 当多个用户请求一个servlet时，服务器为每个用户启动一个线程而不是启动一个进程，这些线程由服务器来管理，与传统的CGI为每个用户启动一个进程相比较，效率要高的多。

## 7.2 servlet的工作原理

### Servlet工作原理



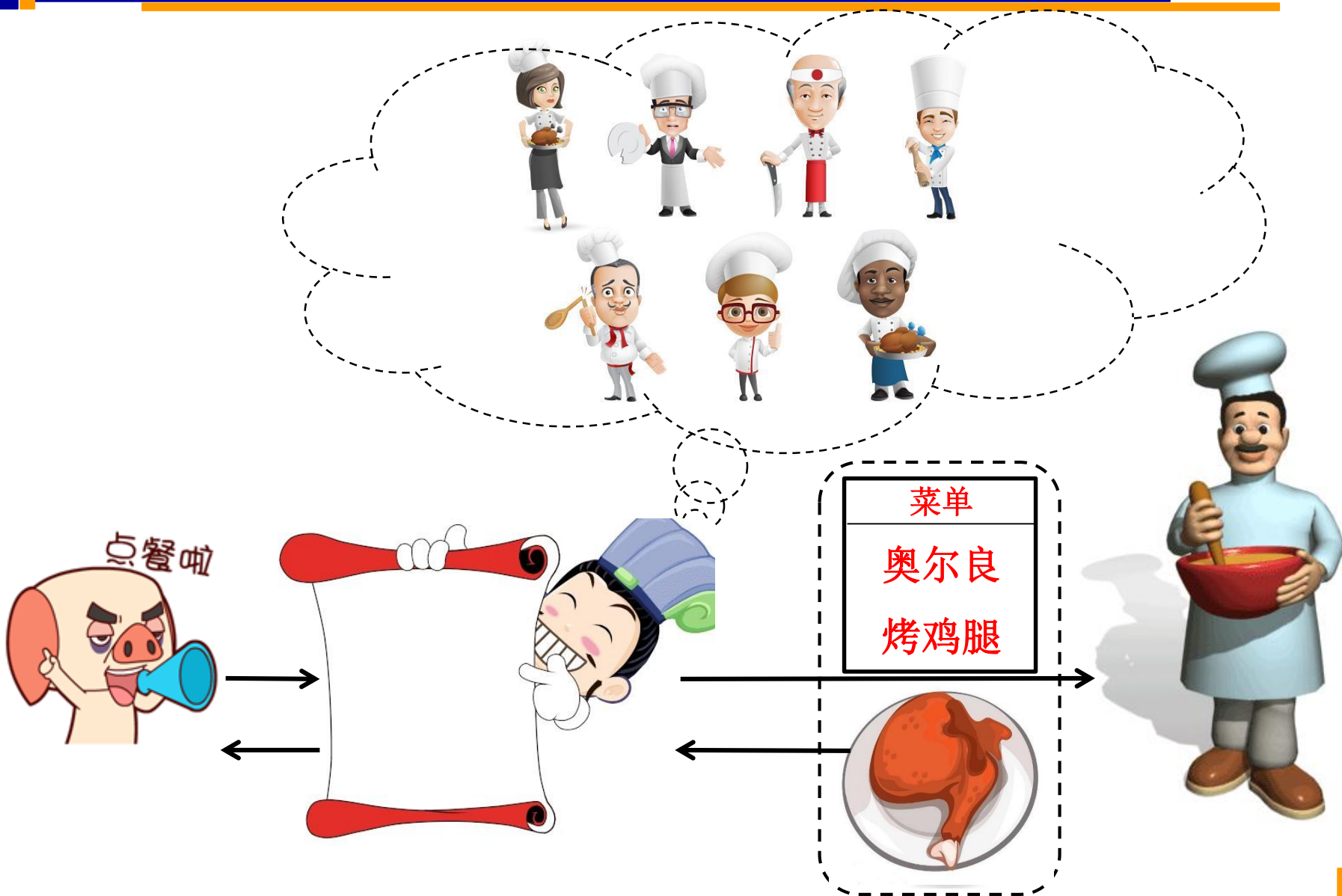
## 7.2 servlet的工作原理



# 河海大学食堂

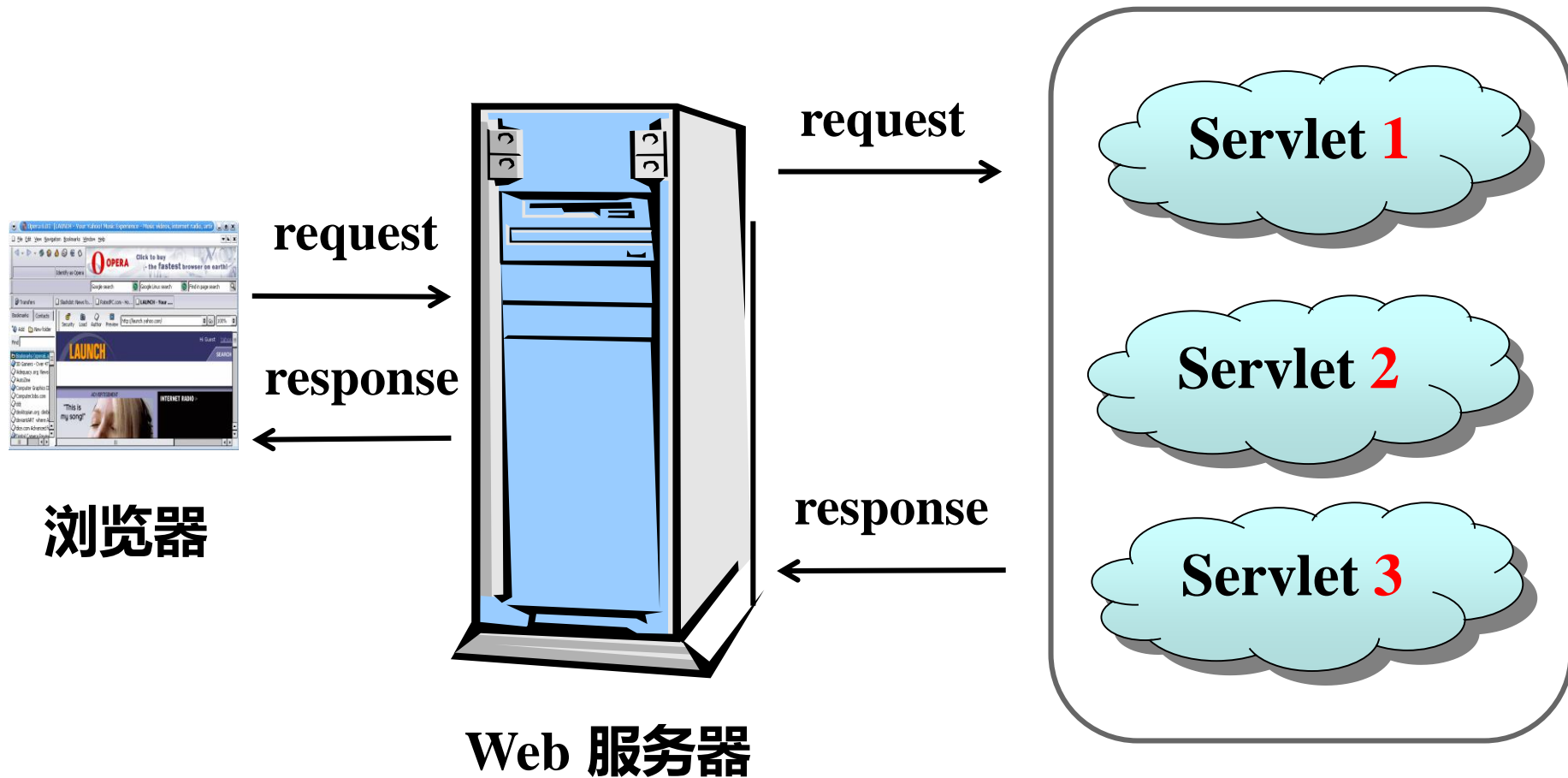


## 7.2 servlet的工作原理

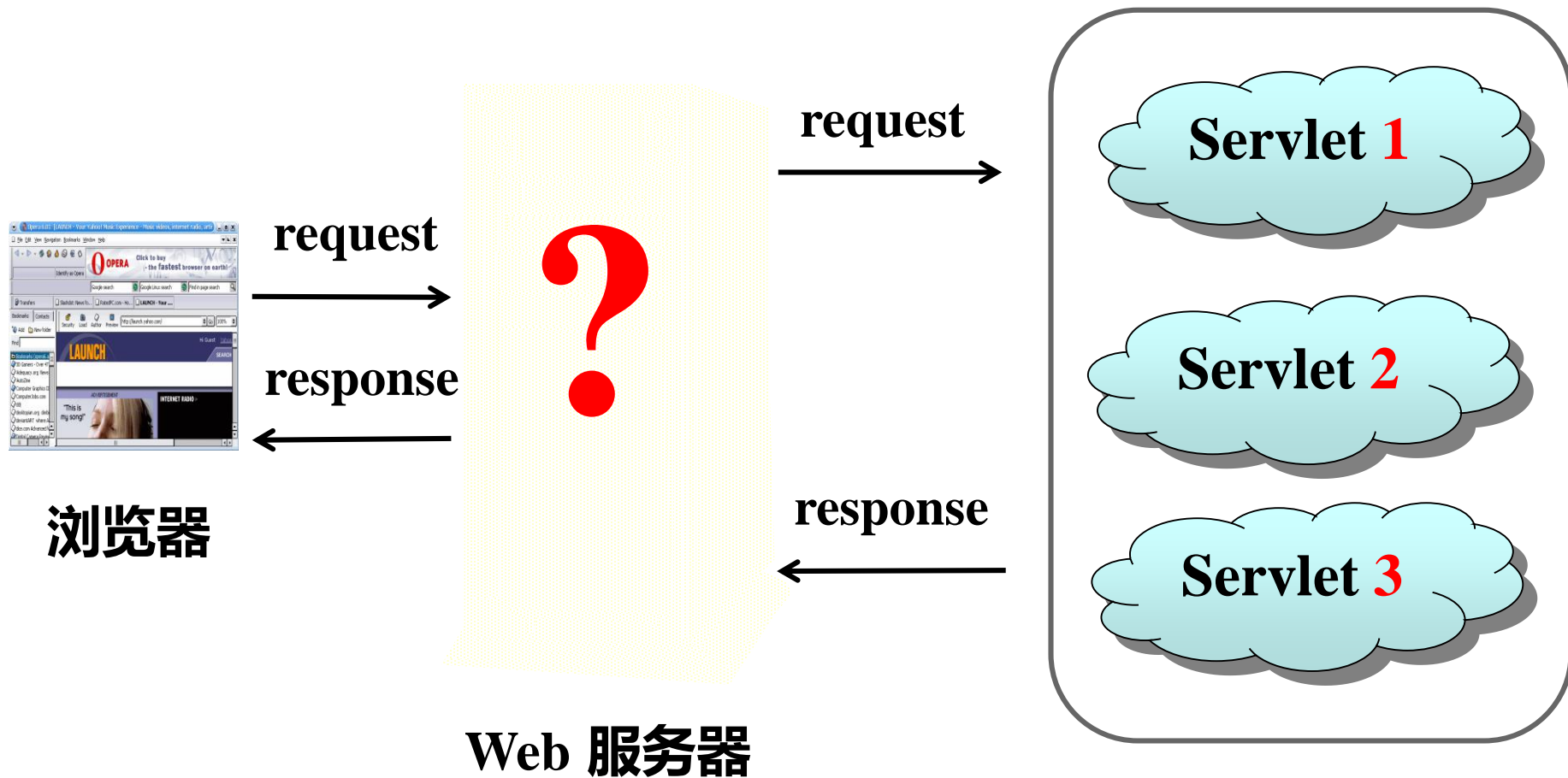




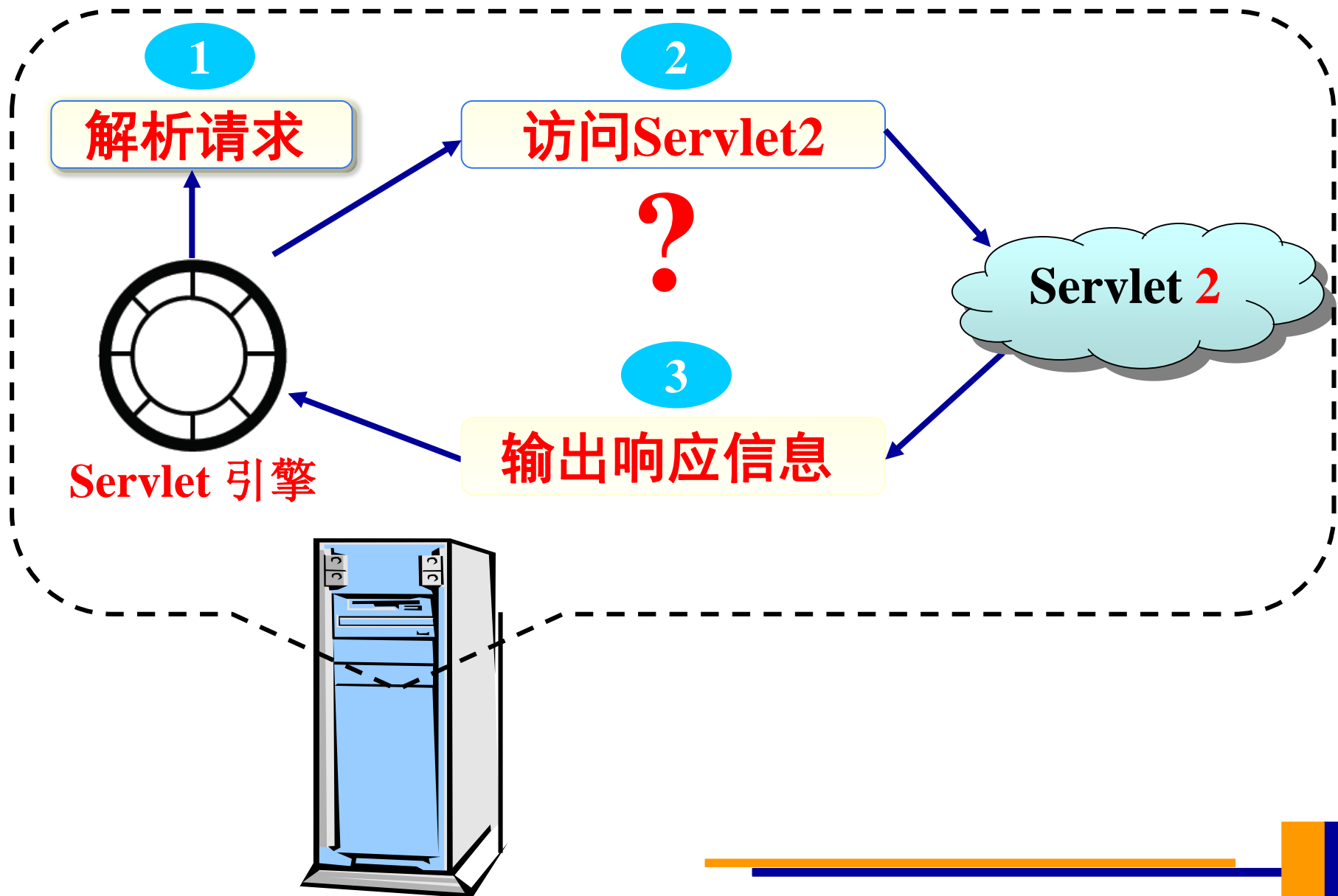
## 7.2 servlet的工作原理



## 7.2 servlet的工作原理

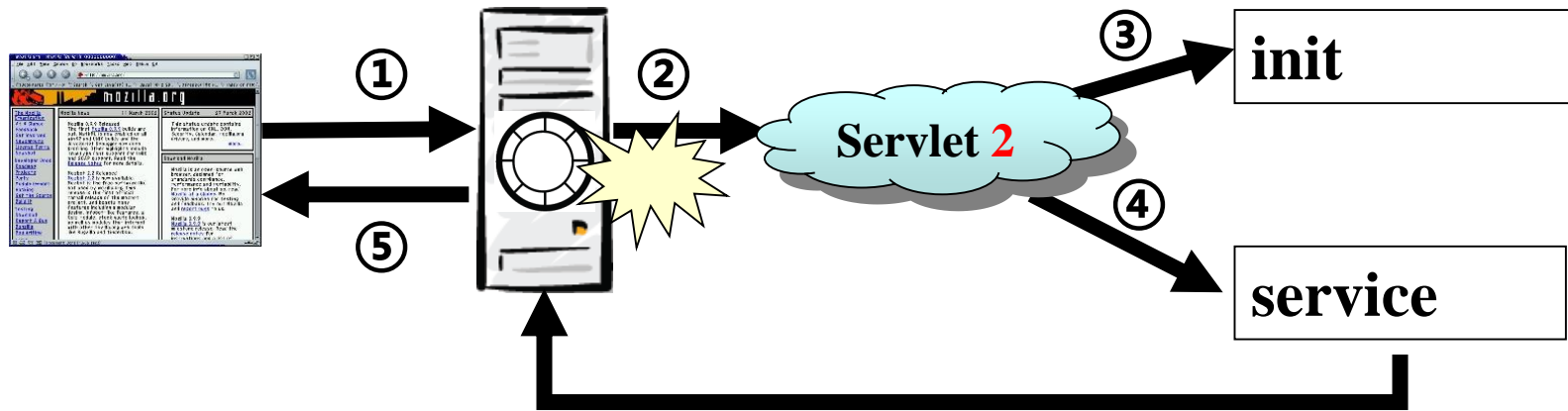


## 7.2 servlet的工作原理

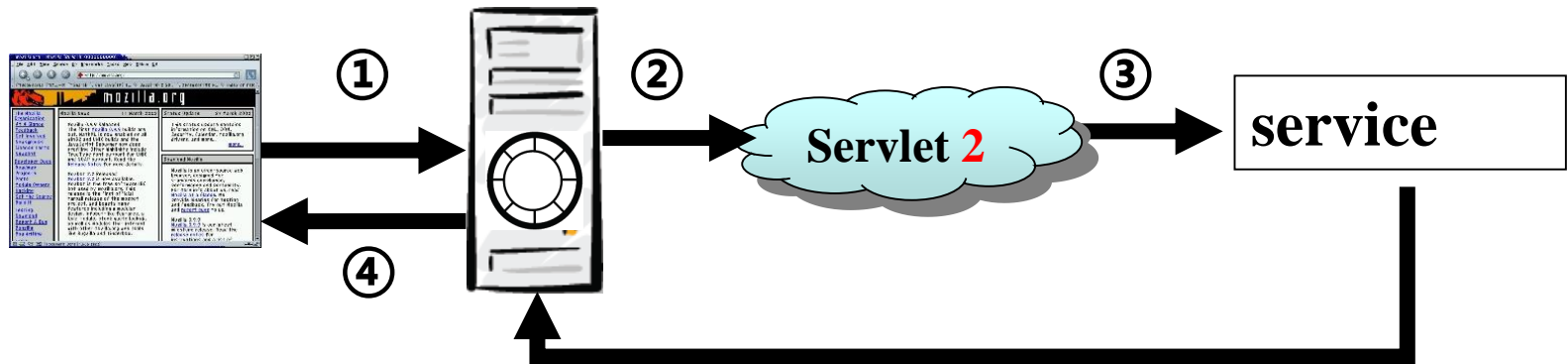


## 7.2 servlet的工作原理

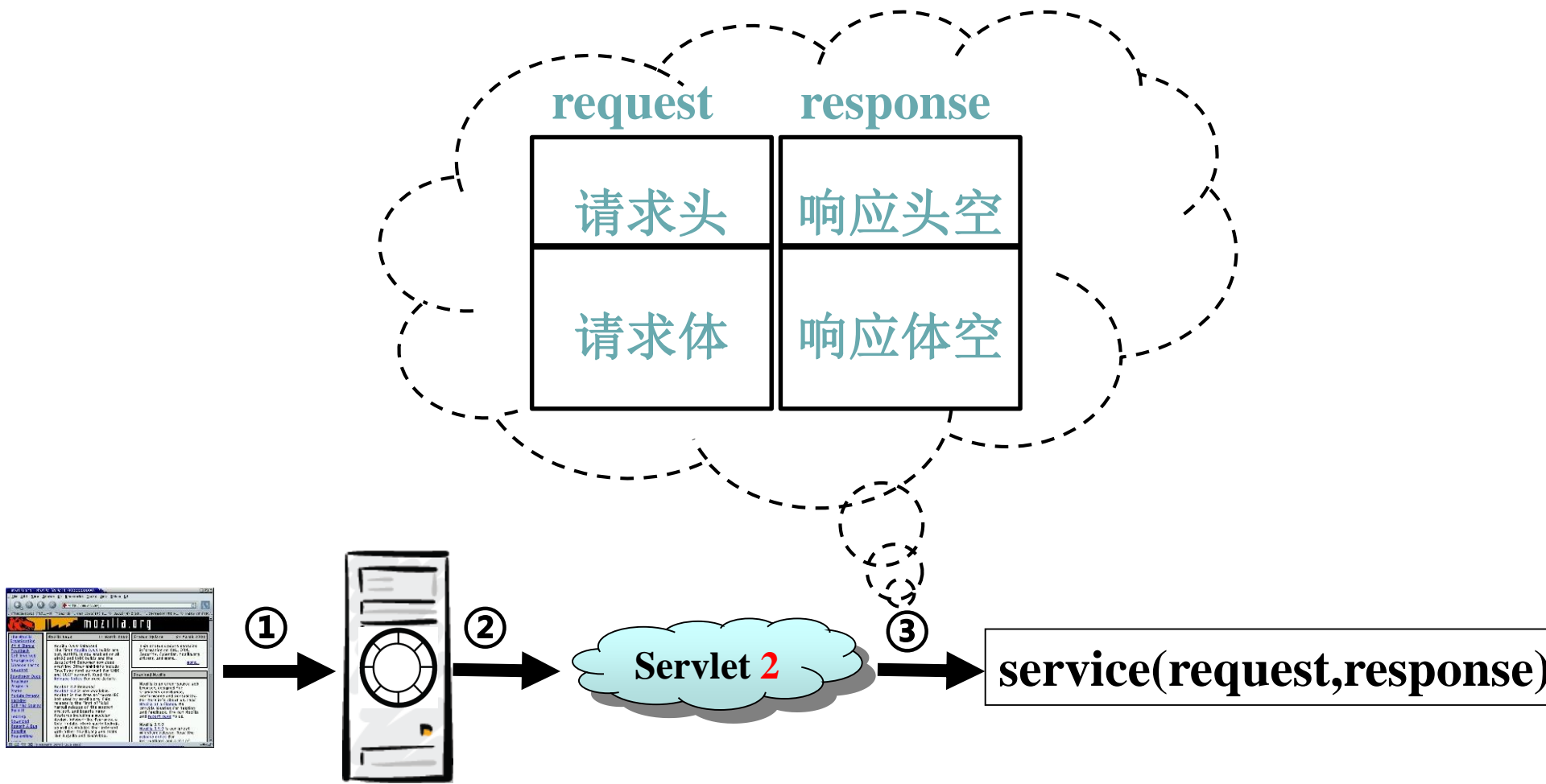
### 第一次请求Servlet



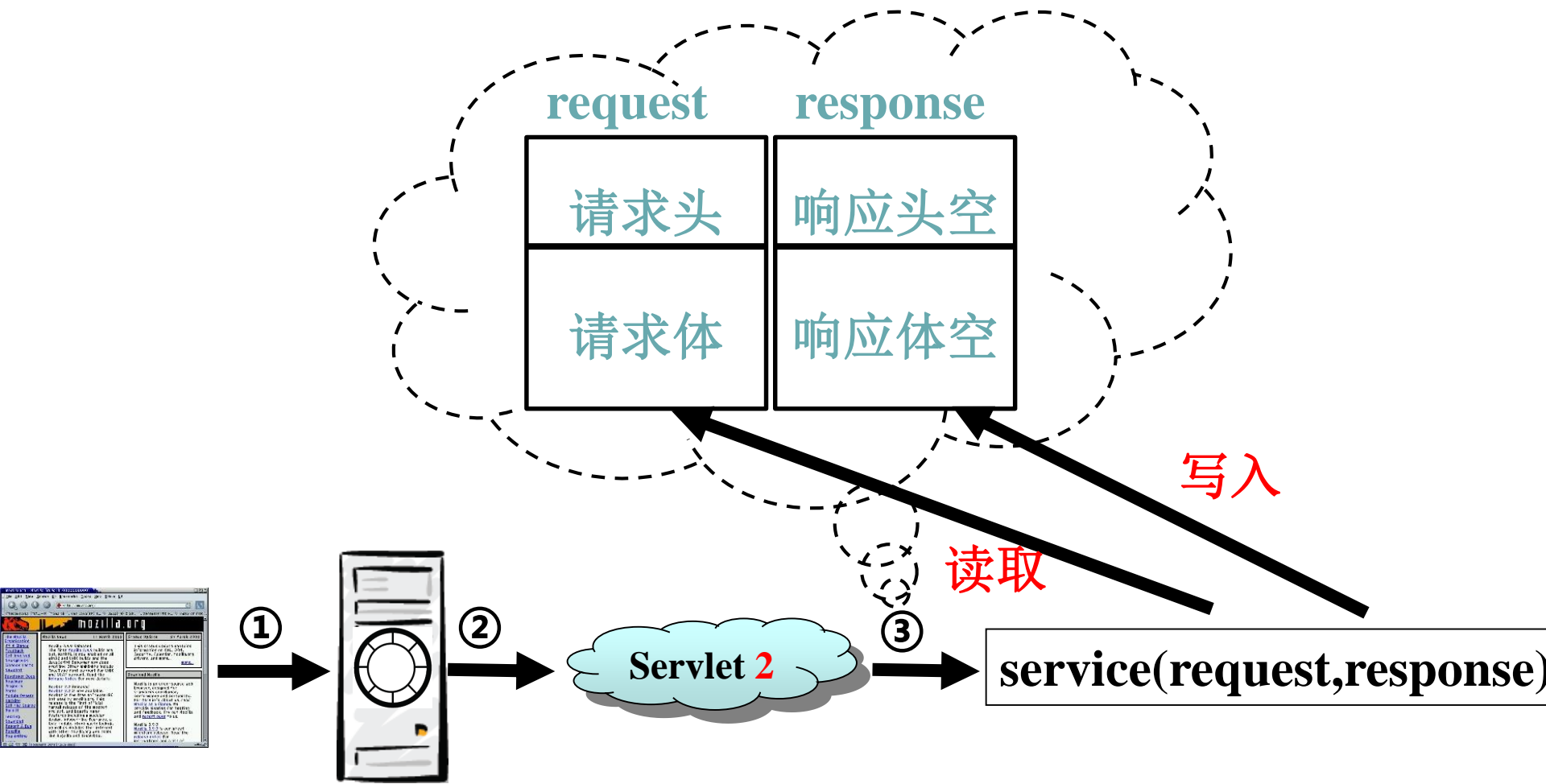
### 再次请求Servlet



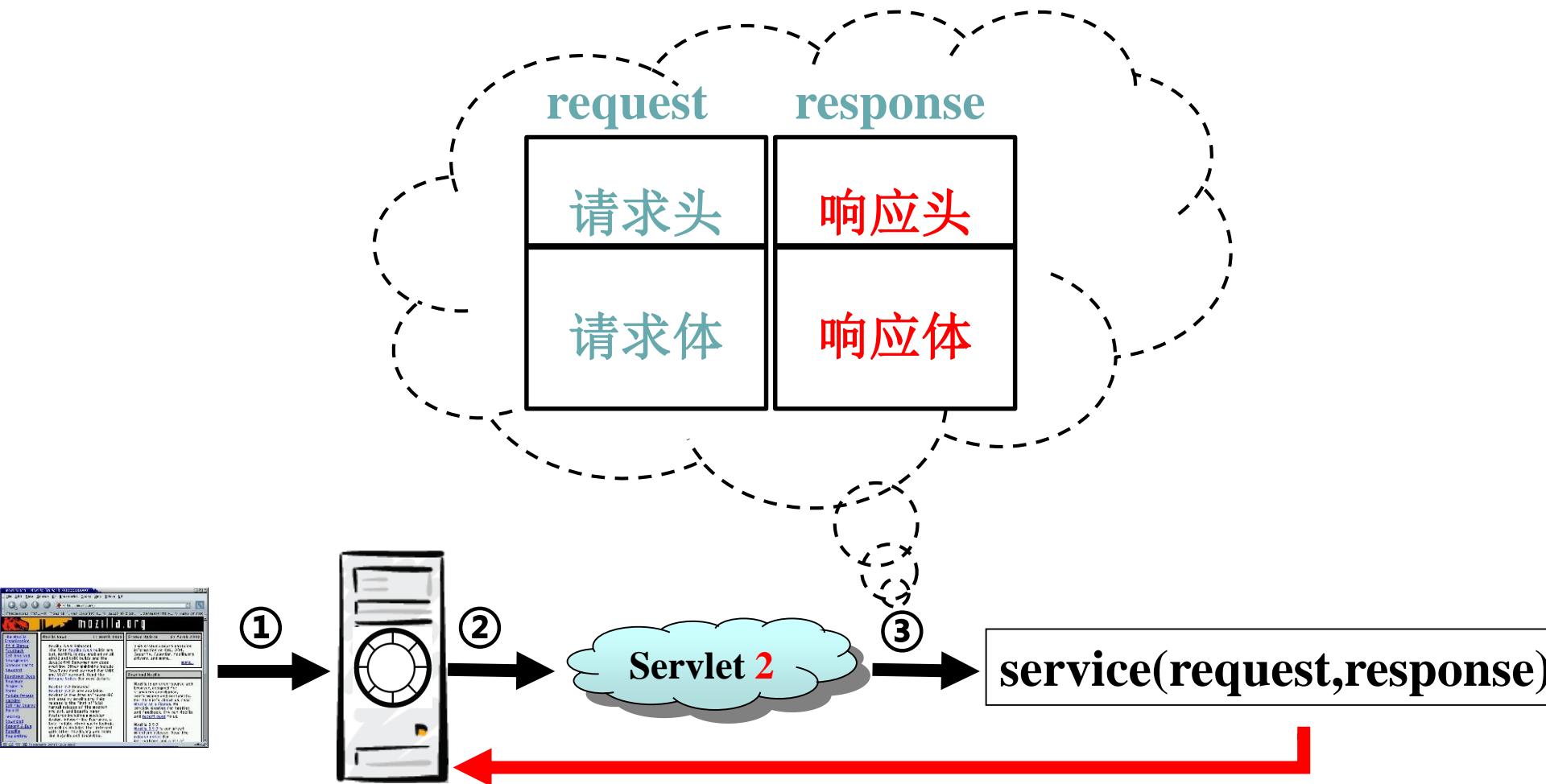
## 7.2 servlet的工作原理



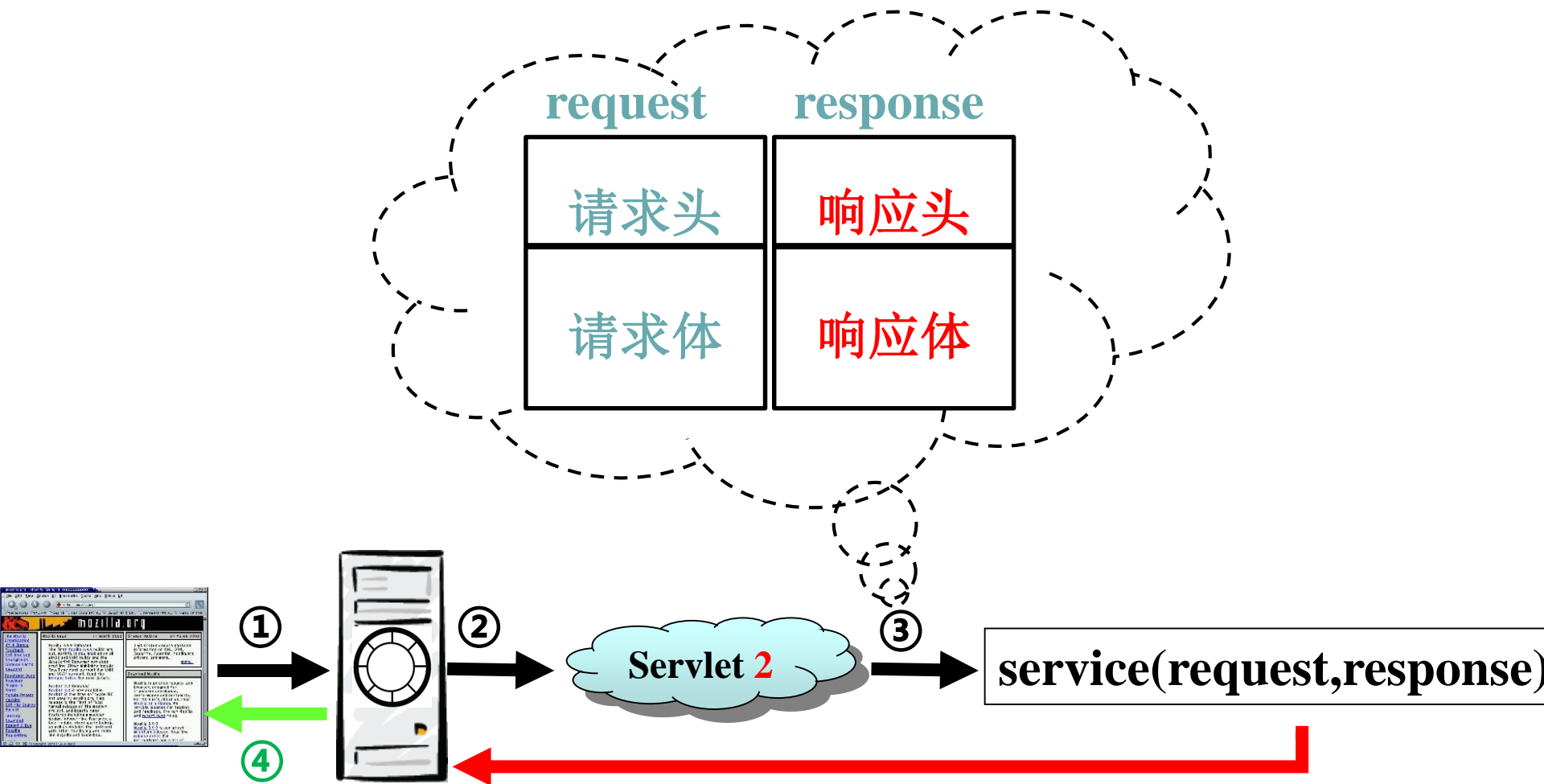
## 7.2 servlet的工作原理



## 7.2 servlet的工作原理



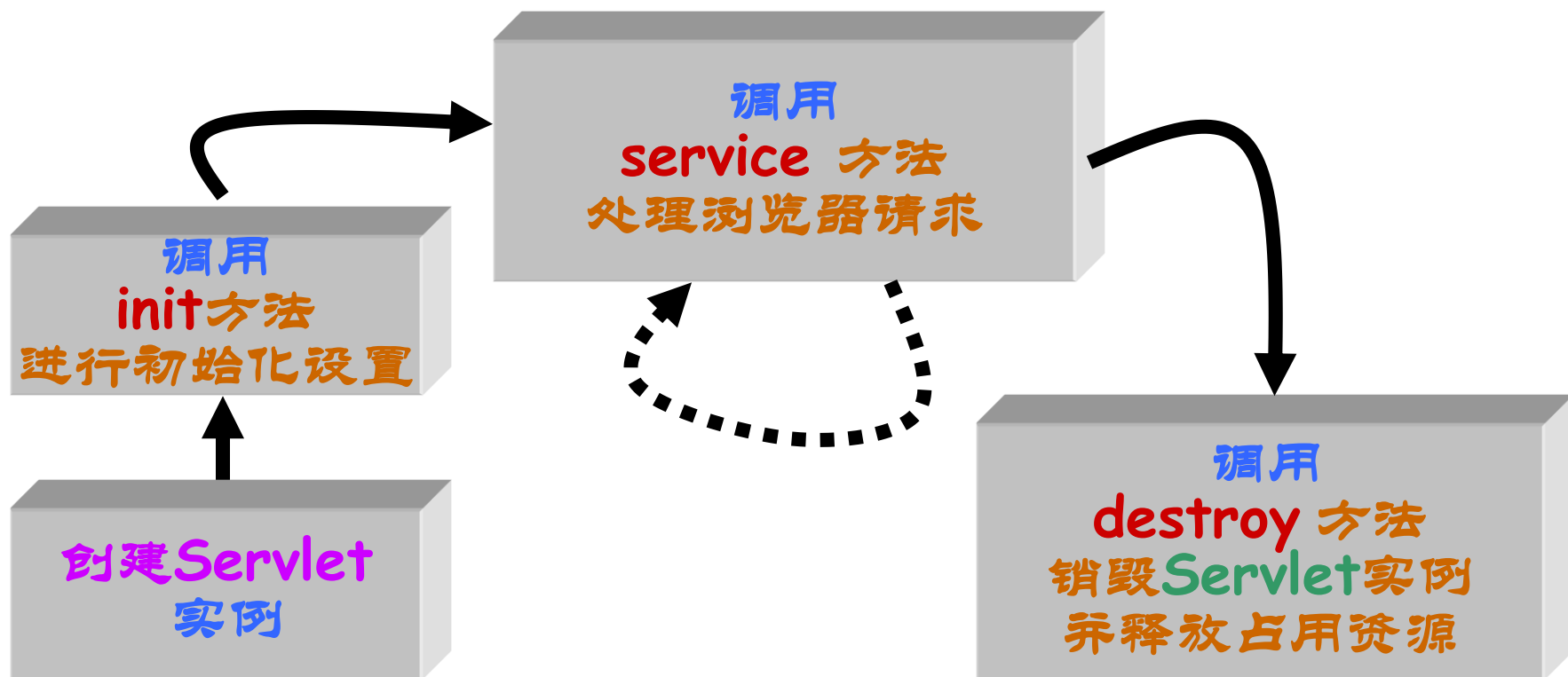
## 7.2 servlet的工作原理





## 7.2 servlet的工作原理

### Servlet的生命周期



## 7.2 servlet的工作原理

### 1、Servlet的生命周期主要由下列三个过程组成

- 调用init()方法完成必要的对象初始化工作。
- 诞生的Servlet对象再调用service()方法响应客户的请求。
- 当服务器关闭时，调用destroy()方法，消灭Servlet对象。

## 7.2 servlet的工作原理

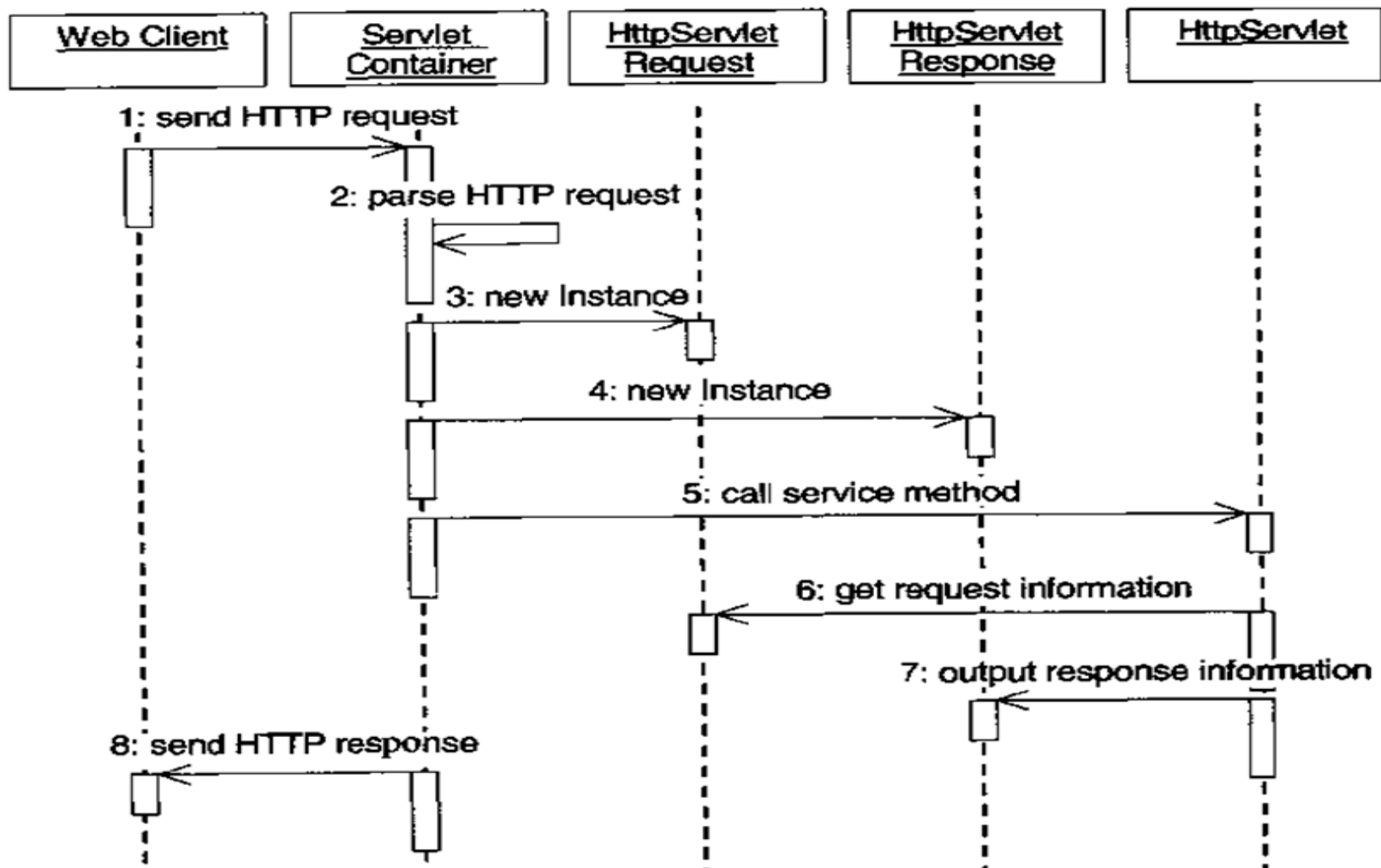
### 1、Servlet的生命周期

#### ➤ 注意：

- 在Servlet的整个生命周期内，Servlet只被初始化一次。
- 对一个Servlet的每次访问请求都导致Servlet引擎调用一次Servlet的service()方法
- 所以在Servlet的整个生命周期内，它的init()方法只被调用一次，但它的service()方法可能被调用多次。

## 7.2 servlet的工作原理

**Servlet响应Web容器客户请求流程的UML时序图:**



## 7.2 servlet的工作原理

注意：

- 对于每次访问请求，Servlet引擎都会创建一个新的 `HttpServletRequest` 请求对象和 `HttpServletResponse` 响应对象，然后将这两个对象作为参数传递给它调用的Servlet的 `service()` 方法。
- 在 `service()` 方法内部首先从请求对象中获得请求信息，接着处理请求和访问其他资源以获得需要返回的信息，然后调用响应对象的方法将响应内容写入到Servlet引擎的缓冲区中，再由Web服务器发送给客户端。
- `HttpServletRequest` 对象和 `HttpServletResponse` 对象是Servlet引擎与Servlet程序进行通信的纽带。

## 7.2 servlet的工作原理

### 3、init方法

- 该方法是HttpServlet类中的方法，可以在子类中重写这个方法，声明格式：

**public void init(ServletConfig config) throws ServletException**

- Servlet开发人员可以在这个方法中完成与构造方法类似的初始化功能。Servlet引擎在调用init方法时，会传递一个包含Servlet的配置和运行环境信息的ServletConfig对象。
- 如果init方法抛出异常，Servlet引擎将卸载Servlet。

## 7.2 servlet的工作原理

### 3、init方法

- ◆ 在Servlet类重写 `init(ServletConfig config)`时必须调用 `super.init(ServletConfig config)` 让父类GenericServlet保存config信息。
- ◆ 具体配置信息存在GenericServlet类的config变量里面，但它是个私有变量，不能被继承，而我们获取所有配置信息都是通过这个config变量获取的，所以这个config变量不能为空，否则出现NullPointerException异常。

## 7.2 servlet的工作原理

### 3、init方法

GenericServlet类中则实现了init(ServletConfig config)方法。不仅如此，还更添加了一个新的不带参数的init()方法。而且带参数的init方法中调用了不带参数的init方法。

```
public void init() throws ServletException
```

```
public void init(ServletConfig) throws ServletException
```

```
public void init(ServletConfig config) throws ServletException{
```

```
    this.config=config;//指向类的成员变量this.config
```

```
    this.init();}
```

```
public void init() throws ServletException{ }
```



## 7.2 servlet的工作原理

### 3、init方法

- 无参init方法是为了防止程序员在写Servlet类重写有参init时忘记写`super.init(ServletConfig config)`。
- 对于继承GenericServlet类的Servlet程序可以覆盖这个无参数的init()方法来编写初始化代码，这样不仅省去了覆盖有参数的init(config)方法时总要编写`super.init(config)`语句的麻烦，也可以避免忘记编写`super.init(config)`语句产生的异常。

## 7.2 servlet的工作原理

### 3、init方法

- 如果用户不需要涉及ServletConfig的一些初始化操作, 那么init (ServletConfig) 方法的实现可有可无, 可以从父类GenericServlet中继承得到。
- 如果servlet需要在初始化时读取ServletConfig中保存的信息, 那么就需要覆盖init (ServletConfig) 方法。

## 7.2 servlet的工作原理

### 4、service方法

- Service方法是Servlet的核心方法，在GenericServlet类中没有对这个方法进行实现，HttpServlet类实现了这个方法。我们可以在子类中直接继承该方法或重写这个方法。
- `public void service(HttpServletRequest request HttpServletResponse response) throw ServletException,IOException`
- 每个用户的请求都导致service方法被调用执行，调用过程运行在**不同的线程中，互不干扰**。

## 7.2 servlet的工作原理

### 5、destroy方法

- destroy方法在Web容器卸载Servlet之前被调用
- 该方法在Servlet的生命周期中也仅执行一次
- 可以通过重写destroy方法来完成与init方法相反的功能，释放被该Servlet打开的资源，例如，关闭数据库连接和I/O流。
- GenericServlet类实现的destroy方法已经满足通常的需要了，子类Servlet一般不必覆盖这个方法。

## 7.3 通过JSP页面访问servlet

- 用户除了可以在浏览器的地址栏中直接键入servlet的请求格式来请求运行一个servlet外，也可以通过JSP页面来请求一个servlet。
- **需要特别注意的是：**如果web.xml文件中<servlet-mapping>标记的子标记<url-pattern>指定的请求servlet的格式是“/lookHello”，那么JSP页面请求servlet时，必须要写成“lookHello”，不可以写成“/lookHello”，否则将变成请求root服务目录下的某个servlet。

## 7.3 通过JSP页面访问servlet

### 1、通过表单向servlet提交数据

- 如果web.xml文件中<url-pattern>指定的请求servlet的格式是“/computeBill”，那么form表单中action给出的值就是“computeBill”：

```
<form action= "computeBill " ... ..>  
</form>。
```

- 当请求一个servlet时，可以加入参数及其值  
servlet名?参数1=值1&参数2=值…参数n=值

## 7.3 通过JSP页面访问servlet

### example5\_2.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
```

```
<HTML><body bgcolor=#EEDDEE><font size=5>
```

```
<form action="computeBill" method=post>
```

```
  输入账单: <br>
```

```
    <textArea name='billMess' rows=5 cols=30 >
```

```
    洗衣粉:12.8圆, 可乐: 12圆, 泡菜:0.8圆
```

```
  </textArea>
```

```
  <br><input type=submit value="提交">
```

```
</form>
```

```
</font></body></HTML>
```

## 7.3 通过JSP页面访问servlet

**web.xml**

**<servlet>**

**<servlet-name>computeBill</servlet-name>**

**<servlet-**

**class>myservlet.control.Example5\_2\_Servlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>computeBill</servlet-name>**

**<url-pattern>/computeBill</url-pattern>**

**</servlet-mapping>**



## 7.3 通过JSP页面访问servlet

**Example5\_2\_Servlet.java**

```
package myervlet.control;
```

```
import java.io.*;
```

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
public class Example5_1_Servlet extends HttpServlet{
```

```
    public void init(ServletConfig config) throws ServletException{
```

```
        super.init(config);
```

```
    }
```

## 7.3 通过JSP页面访问servlet

```
public void service(HttpServletRequest request, HttpServletResponse
response)throws IOException{
    request.setCharacterEncoding("gb2312");
    response.setContentType("text/html;charset=gb2312");
    PrintWriter out=response.getWriter();
    out.println("<html><body bgcolor=yellow>");
    String str=request.getParameter("billMess");
    if(str==null || str.length()==0)
        return;
    String []price = str.split("[^0123456789.]+");
    double sum = 0;
    try{
        for(int i=0;i<price.length;i++) {
            if(price[i].length()>=1)
                sum+=Double.parseDouble(price[i]);}
        }
    catch(NumberFormatException e){out.print(" "+e);}
    out.print("\\"+str+"\\"<br>的消费额:"+sum+"圆");
    out.println("</body></html>");
}
```

## 7.3 通过JSP页面访问servlet

### 2、通过超链接访问servlet

- 如果web.xml文件中<url-pattern>指定的请求servlet的格式是“/triangle”，那么<a>超链接标记中href的值是“triangle”，如下所示意：

```
<a href="triangle"></a>
```

## 7.3 通过JSP页面访问servlet

### example5\_3.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
```

```
<HTML><body bgcolor=cyan><font size=3>
```

```
<br><a href="triangle?sideA=3&sideB=4&sideC=5">
```

三角形（3,4,5）的面积

```
</a>
```

```
</font></body></HTML>
```

## 7.3 通过JSP页面访问servlet

### Example5\_3\_Servlet.java

```
package myservlet.control;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Example5_3_Servlet extends HttpServlet{
    public void init(ServletConfig config) throws ServletException{
        super.init(config);
    }
    public void service(HttpServletRequest request,HttpServletResponse
        response) throws IOException{
        request.setCharacterEncoding("gb2312");
        response.setContentType("text/html;charset=gb2312");
        PrintWriter out=response.getWriter();
        out.println("<html><body bgcolor=#EEFFAA>");
        String a=request.getParameter("sideA");
        String b=request.getParameter("sideB");
        String c=request.getParameter("sideC");
```

## 7.3 通过JSP页面访问servlet

### Example5\_3\_Servlet.java

```
if(a==null || a.length()==0)
    return;
double sideA=0,sideB=0,sideC=0;
try {
    sideA = Double.parseDouble(a);
    sideB = Double.parseDouble(b);
    sideC = Double.parseDouble(c);
}
catch(NumberFormatException exp){
    return;
}
double area = 0,p=0;
p = (sideA+sideB+sideC)/2;
area = Math.sqrt(p*(p-sideA)*(p-sideB)*(p-sideC));
out.print("三角形"+a+", "+b+", "+c+"的面积是"+area);
out.println("</body></html>");
}
}
```

## 7.4 servlet共享变量

- Servlet类是HttpServlet的一个子类，那么在编写子类时就可以声明某些成员变量。
- 当用户请求加载Servlet时，服务器分别为每个用户启动一个线程，在该线程中，Servlet调用service()方法响应客户请求，那么Servlet类的成员变量是被所有线程共享的数据。

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11}$$

## 7.4 servlet共享变量

### example5\_4.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
```

```
<HTML><body bgcolor=cyan><font size=3>
```

```
<a href="computerPI" >参与计算PI的值<a>
```

```
</body></HTML>
```



## 7.4 servlet共享变量

### web.xml

**<servlet>**

**<servlet-name>**computerPI**</servlet-name>**

**<servlet-**

**class>**myservlet.control.Example5\_4\_Servlet**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>**computerPI **</servlet-name>**

**<url-pattern>**/computerPI **</url-pattern>**

**</servlet-mapping>**

## 7.4 servlet共享变量

### Example5\_4\_Servlet.java

```
package myservlet.control;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Example5_4_Servlet extends HttpServlet{
    double sum=0,i=1,j=1; //被所有用户共享
    int number=0;         //被所有用户共享
    public void init(ServletConfig config) throws ServletException{
        super.init(config);
    }
}
```

## 7.4 servlet共享变量

### Example5\_4\_Servlet.java

```
public synchronized void service(HttpServletRequest request,
HttpServletResponse response) throws IOException{
    response.setContentType("text/html;charset=gb2312");
    PrintWriter out=response.getWriter();
    out.println("<html><body>");
    number++;
    sum=sum+i/j;
    j=j+2;
    i=-i;
    out.println("servlet:"+getServletName()+"已经被请求了
"+number+"次");
    out.println("<BR>现在PI的值是:");
    out.println(4*sum);
    out.println("</body></html>");
}
}
```

## 7.5 doGet()方法和doPost()方法

- 可以在Servlet类中重写doPost()或doGet()方法来响应用户的请求。
- 如果不论用户请求类型是POST还是GET，服务器的处理过程完全相同，那么我们可以只在doPost()方法中编写处理过程，而在doGet()方法中再调用doPost()方法即可，反之也一样。
- 如果根据请求的类型进行不同的处理，就需在两个方法中编写不同的处理过程。

## 7.5 doGet()方法和doPost()方法

### example5\_5.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
```

```
<HTML><body bgcolor=yellow ><font size=2>
```

```
<form action="sumORproduct" method=post>
```

输入数字，用逗号分隔提交给servlet(post方式):

```
<br><input type=text name="number">
```

```
<input type=submit value="提交">
```

```
</form>
```

```
<form action="sumORproduct" method=get>
```

输入数字，用逗号分隔提交给servlet(get方式):

```
<br><input type=text name="number">
```

```
<input type=submit value="提交">
```

```
</form>
```

```
</body></HTML>
```

## 7.5 doGet()方法和doPost()方法

### web.xml

**<servlet>**

**<servlet-name>sumORproduct</servlet-name>**

**<servlet-class>myservlet.control.Example5\_5\_Servlet</servlet-class> </servlet>**

**<servlet-mapping>**

**<servlet-name>sumORproduct</servlet-name>**

**<url-pattern>/sumORproduct</url-pattern>**

**</servlet-mapping>**

## 7.5 doGet()方法和doPost()方法

### **Example5\_5\_Servlet.java**

```
package myservlet.control;  
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
public class Example5_5_Servlet extends HttpServlet{  
    public void init(ServletConfig config) throws ServletException{  
        super.init(config);  
    }  
}
```

## 7.5 doGet()方法和doPost()方法

### Example5\_5\_Servlet.java

```
public void doPost(HttpServletRequest request, HttpServletResponse
response)throws ServletException,IOException{
    response.setContentType("text/html;charset=gb2312");
    PrintWriter out=response.getWriter();
    out.println("<html><body bgcolor=#FFAA99>");
    String s=request.getParameter("number");
    String []a=s.split("[, ]+");
    double sum = 0;
    for(String item:a) {
        if(item.length()>=1)
            sum+=Double.parseDouble(item);
    }
    out.print("用户的请求方式是"+request.getMethod()+"<br>");
    for(String item:a) {
        if(item.length()>=1)
            out.print(item+" ");
    }
    out.print("<br>的和是"+sum);
    out.println("</body></html>");
}
```



## 7.5 doGet()方法和doPost()方法

### Example5\_5\_Servlet.java

```
public void doGet(HttpServletRequest request,HttpServletResponse
response) throws ServletException,IOException{
    response.setContentType("text/html;charset=gb2312");
    PrintWriter out=response.getWriter();
    out.println("<html><body bgcolor=#EEFFCC>");
    String s=request.getParameter("number");
    String []a=s.split("[, ]+");
    double product = 1;
    for(String item:a) {
        if(item.length()>=1)
            product*=Double.parseDouble(item);
    }
    out.print("用户的请求方式是"+request.getMethod()+"<br>");
    for(String item:a) {
        if(item.length()>=1)
            out.print(item+" ");
    }
    out.print("<br>的乘积是"+product);
    out.println("</body></html>");
}
}
```

## 7.6 重定向与转发

在Servlet程序中,有时需要调用另外一个资源来对浏览器的请求进行响应,这可以通过两种方式来实现:

- 其中一种是调用 `RequestDispatcher.forward` 方法实现的请求转发;
- 另外一种则是调用 `HttpServletResponse.sendRedirect` 方法实现的请求重定向.

## 7.6 重定向与转发

### 1、RequestDispatcher对象

- RequestDispatcher对象是由Servlet引擎创建的,它用于包装一个被其他资源调用的资源(例如:servlet,html文件,JSP文件等),并可以通过其中的方法来将客户端的请求转发给所包装的资源.
- RequestDispatcher接口中定义了两个方法:forward方法和include方法,他们分别用于将请求转发到RequestDispatcher对象封装的资源 and 将RequestDispatcher对象封装的资源作为当前响应内容的一部份包含进来.

## 7.6 重定向与转发

### 1、RequestDispatcher对象

- forward和include方法接收的两个参数必须是传递给当前Servlet的service方法的那两个ServletRequest和ServletResponse对象或者是对它们进行了包装的ServletRequestWrapper或ServletResponseWrapper对象,以便调用和被调用的两个资源共享相同的ServletRequest和ServletResponse对象.

## 7.6 重定向与转发

### 1、RequestDispatcher对象

实现转发的步骤如下：

#### ➤ 1. 得到RequestDispatcher对象

```
RequestDispatcher dispatcher=request.getRequestDispatcher("a.jsp");
```

#### ➤ 2. 转发

```
dispatcher.forward (request,response);
```

➤ 将用户对当前JSP页面或servlet的请求转变成对a.jsp页面的请求。

## 7.6 重定向与转发

### 1、RequestDispatcher对象

- 转发的目标页面或servlet可以使用request获取用户提交的数据。
- 用户在浏览器的地址栏中不能看到forward方法转发的页面的地址或servlet的地址，只能看到该页面或servlet的运行效果。

## 7.6 重定向与转发

### 1、RequestDispatcher对象

- 转发的目标页面或servlet可以使用request获取用户提交的数据。
- 用户在浏览器的地址栏中不能看到forward方法转发的页面的地址或servlet的地址，只能看到该页面或servlet的运行效果。
- RequestDispatcher对象只能包装当前Web应用程序中的资源。所以, forward方法和include方法只能在同一个Web应用程序内的资源之间转发请求和实现资源包含。

## 7.6 重定向与转发

请求转发的运行流程

步骤1:



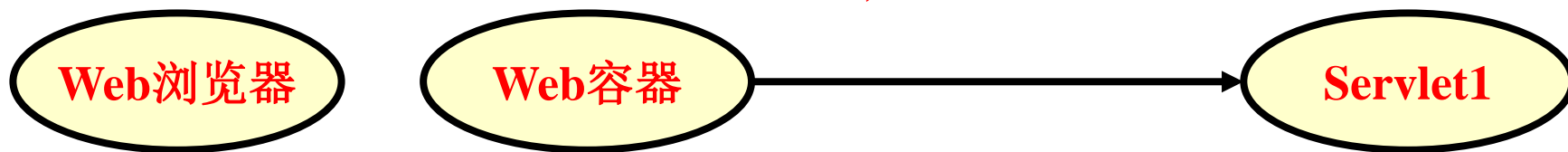


## 7.6 重定向与转发

请求转发的运行流程

步骤2:

首次访问,容器才创建目标Servlet

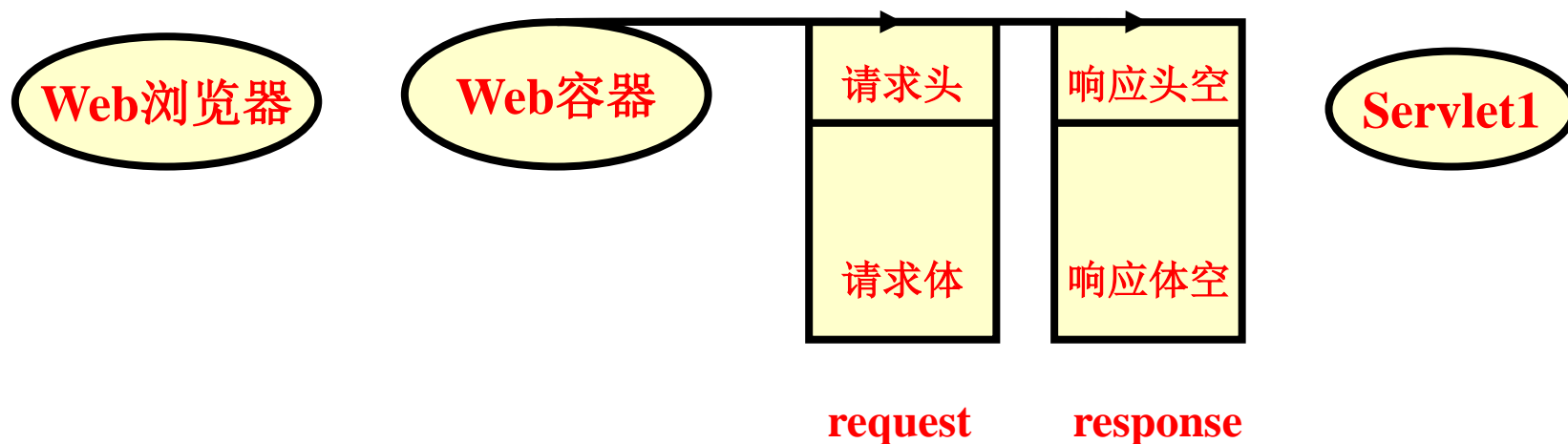


## 7.6 重定向与转发

### 请求转发的运行流程

步骤3:

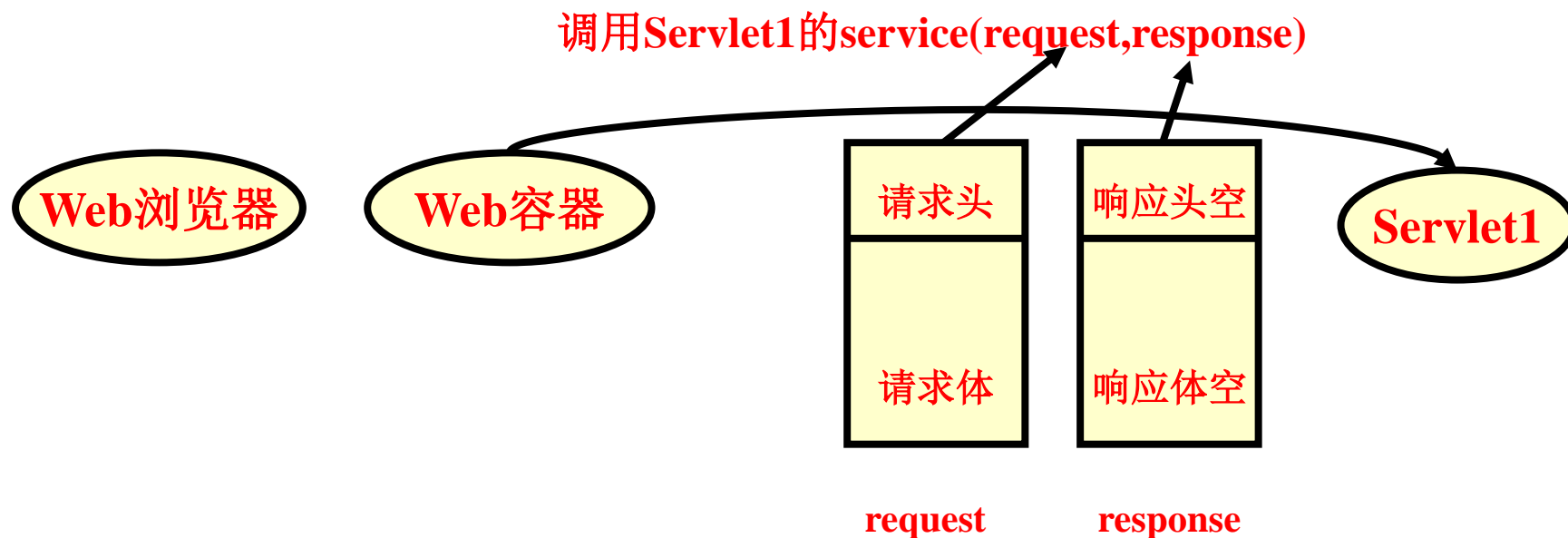
创建请求和响应对象



## 7.6 重定向与转发

### 请求转发的运行流程

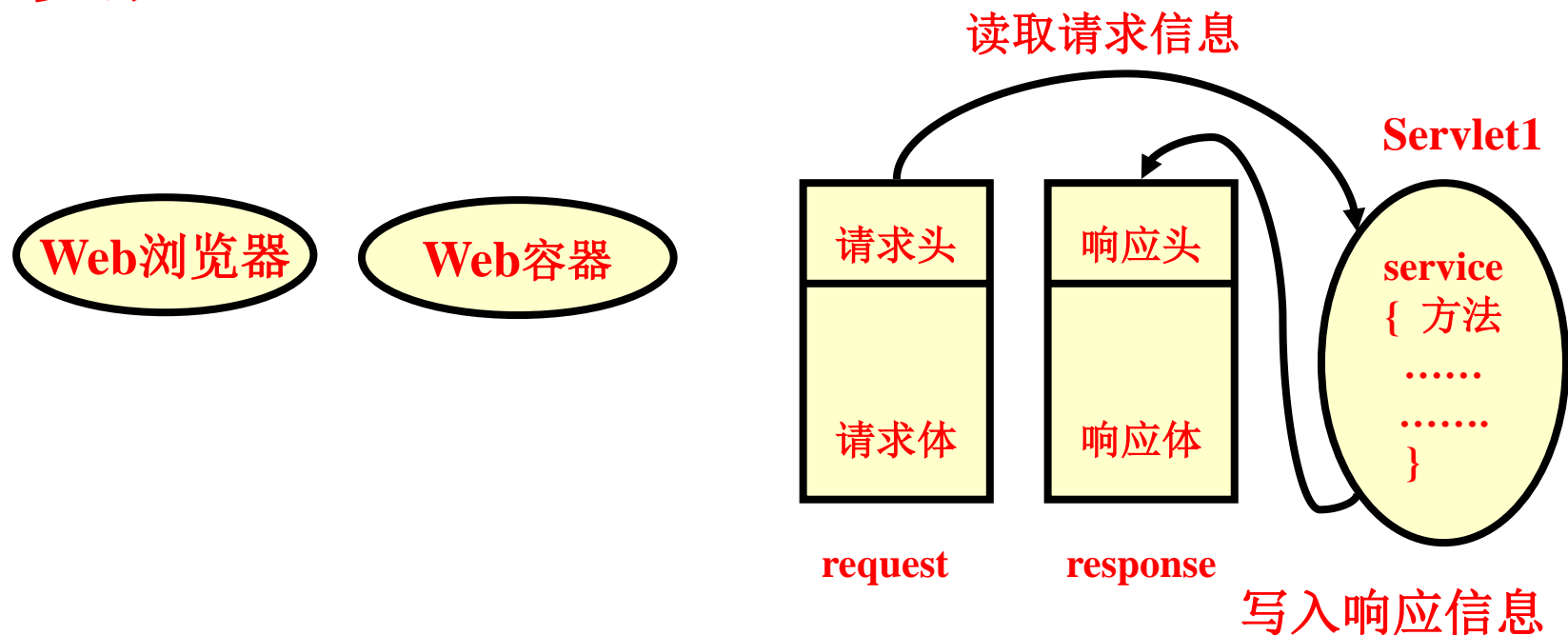
#### 步骤4:



## 7.6 重定向与转发

### 请求转发的运行流程

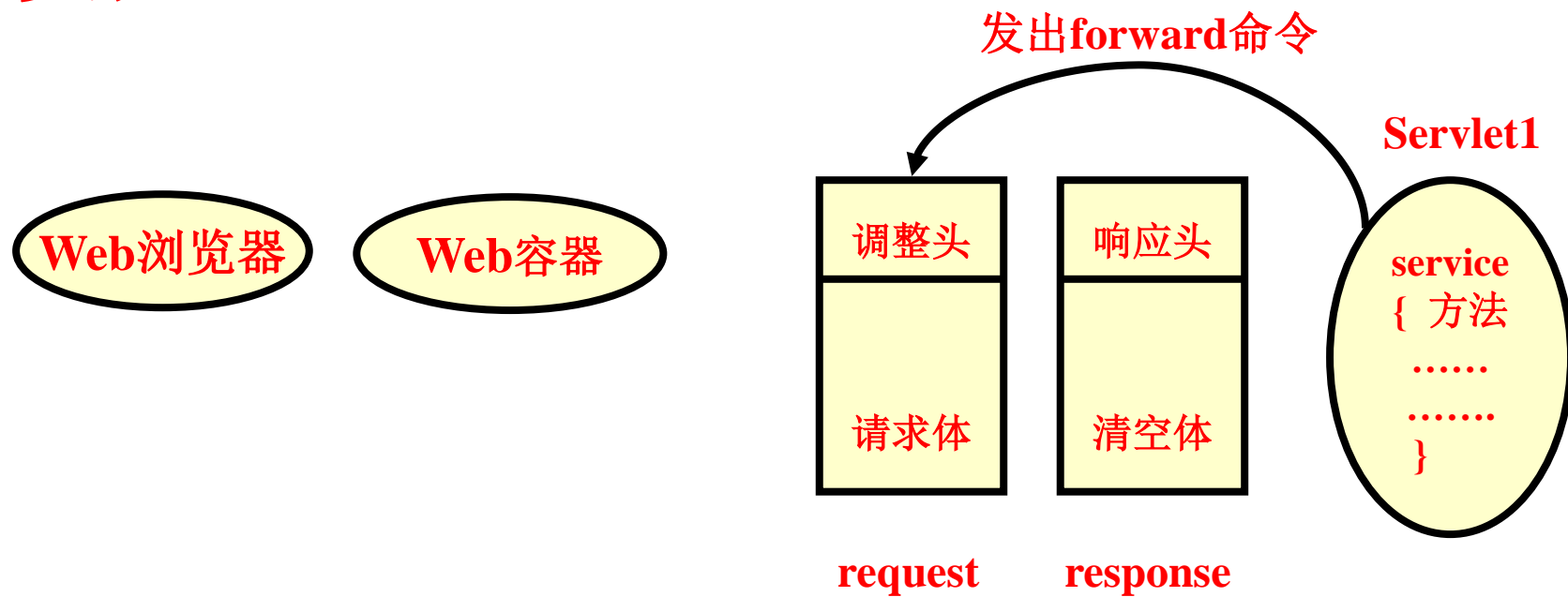
#### 步骤5:



## 7.6 重定向与转发

请求转发的运行流程

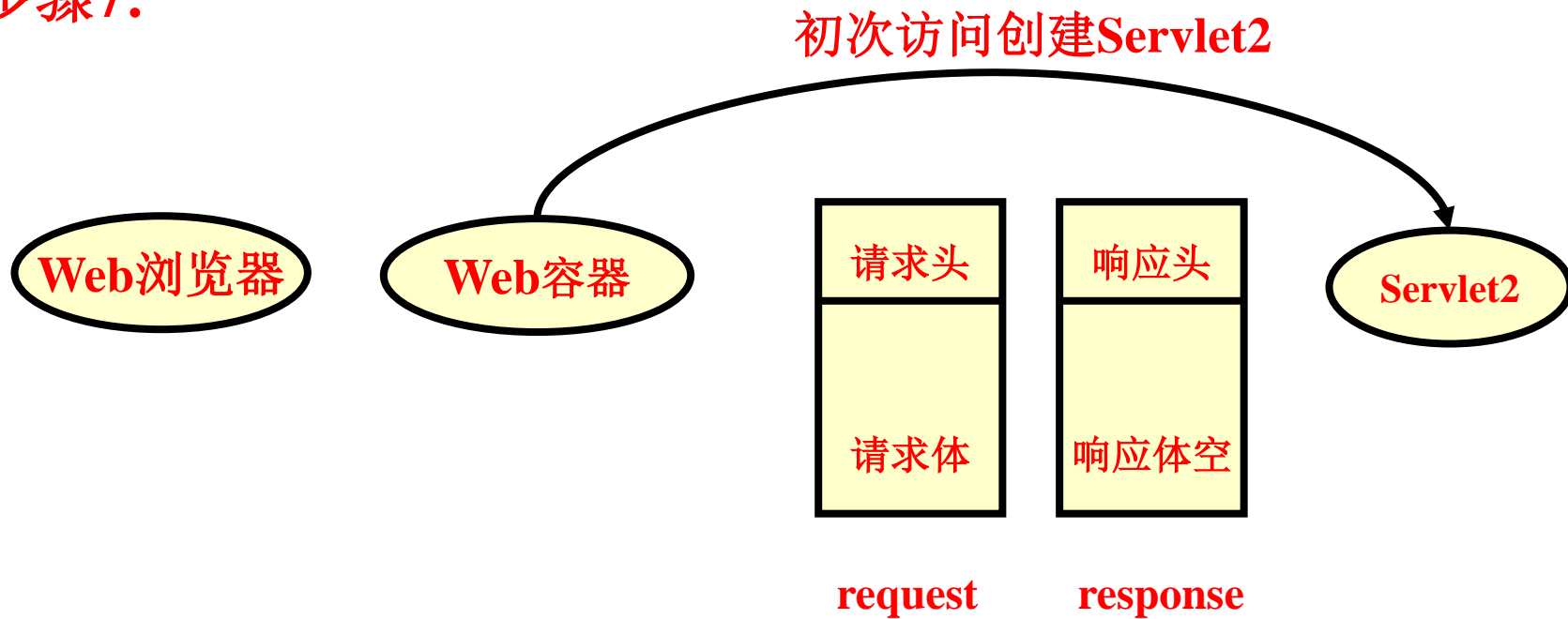
步骤6:



## 7.6 重定向与转发

请求转发的运行流程

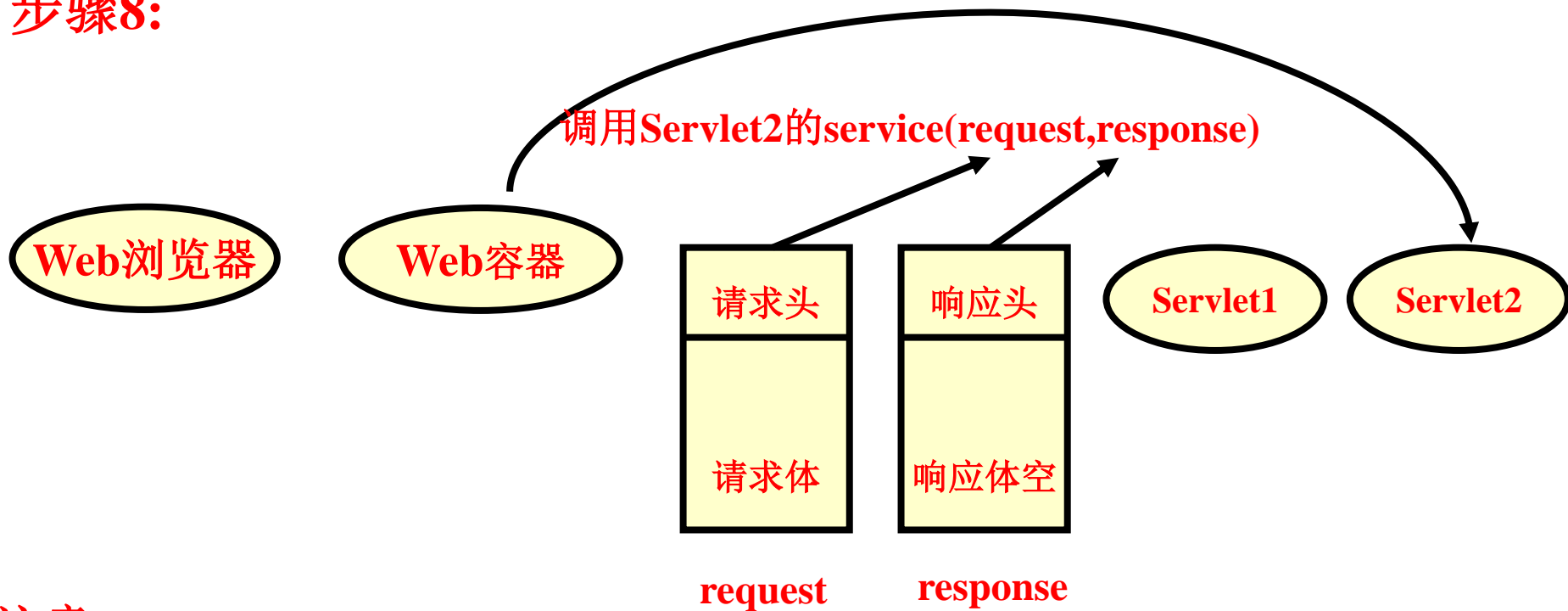
步骤7:



## 7.6 重定向与转发

请求转发的运行流程

步骤8:



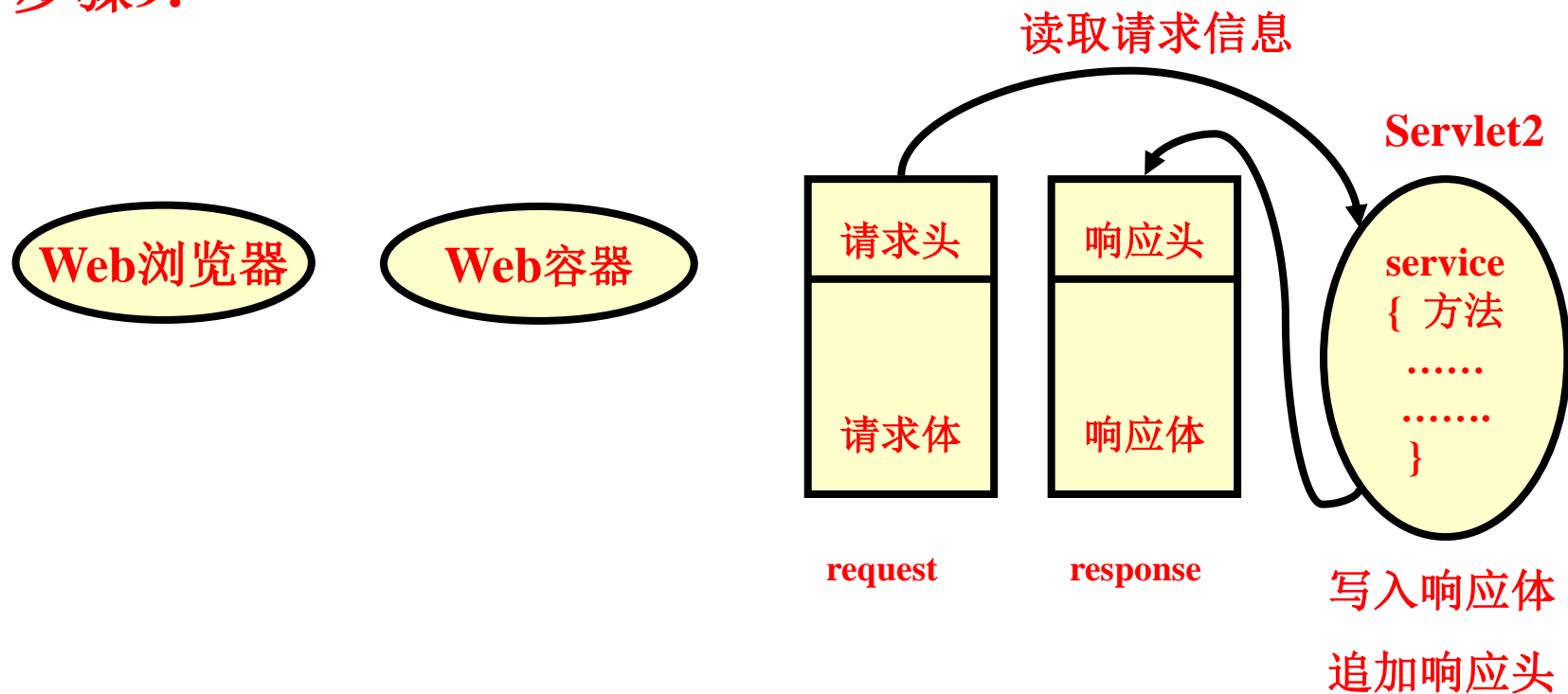
注意:

注意，这里传递的还是调用Servlet1时所创建的request和response对象.

## 7.6 重定向与转发

### 请求转发的运行流程

#### 步骤9:

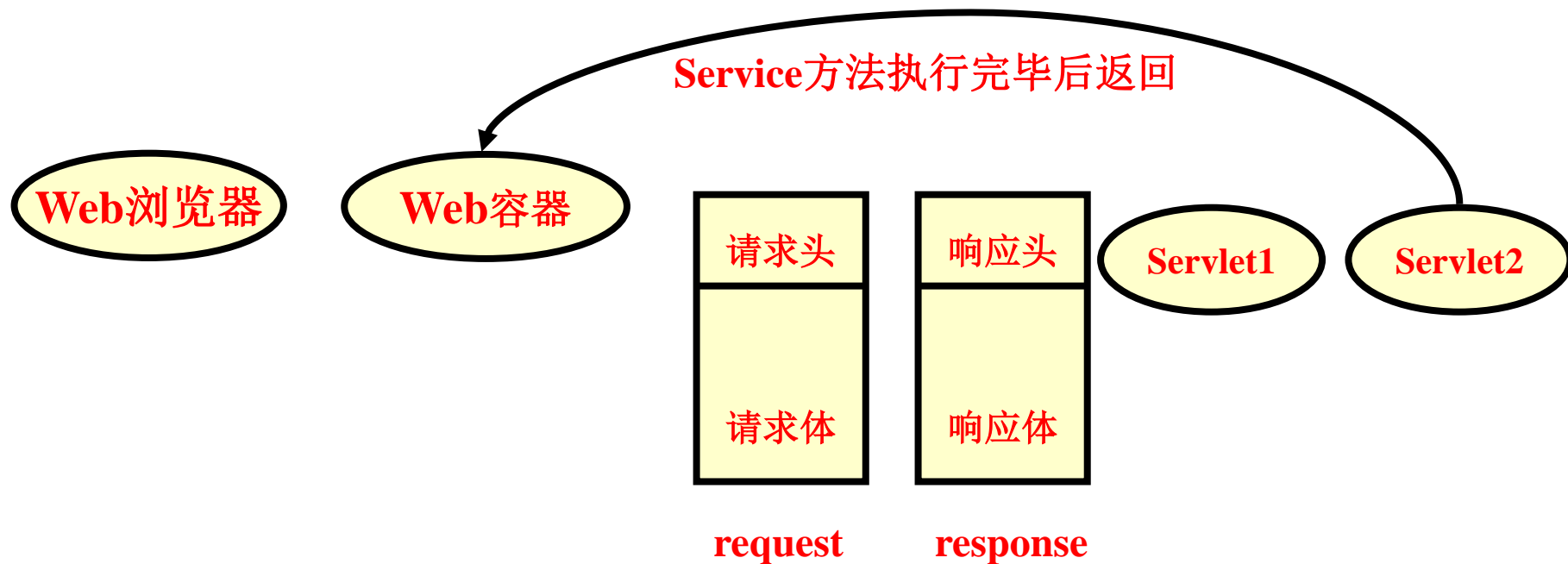




## 7.6 重定向与转发

请求转发的运行流程

步骤10:



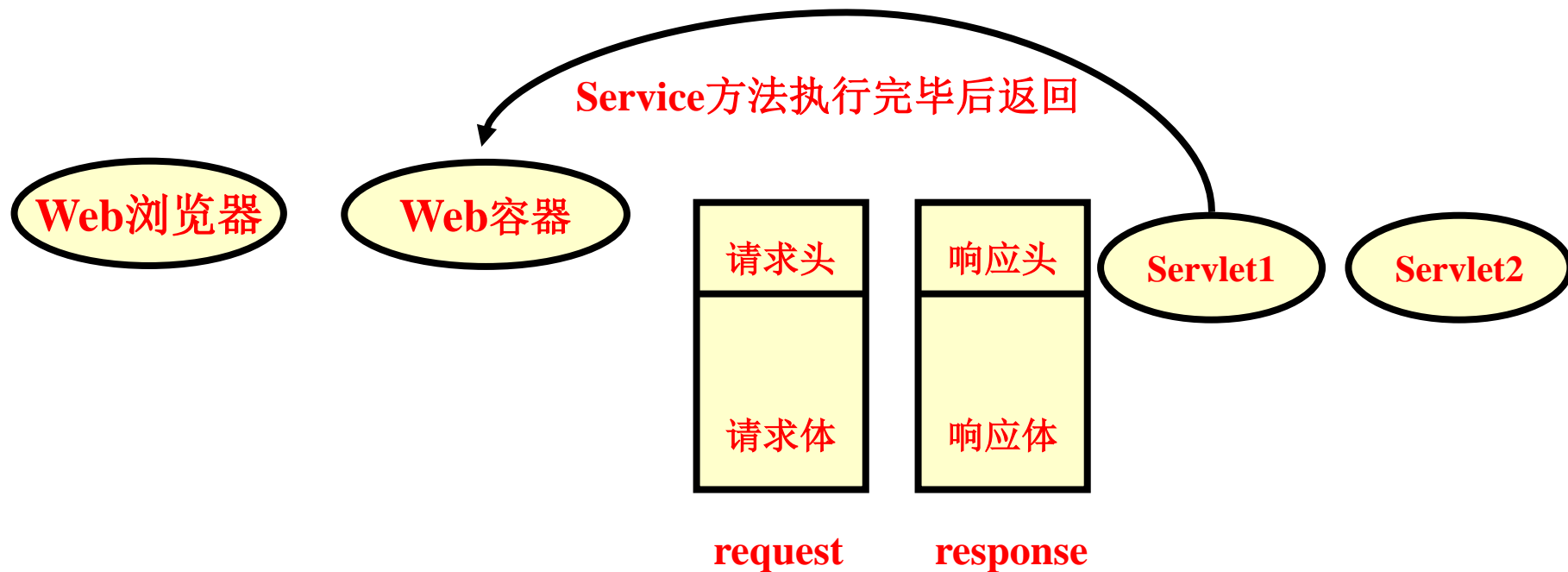
注意:

**Servlet1**的**service**方法继续执行调用**forward**方法的语句后面的代码,执行完毕后返回.

## 7.6 重定向与转发

请求转发的运行流程

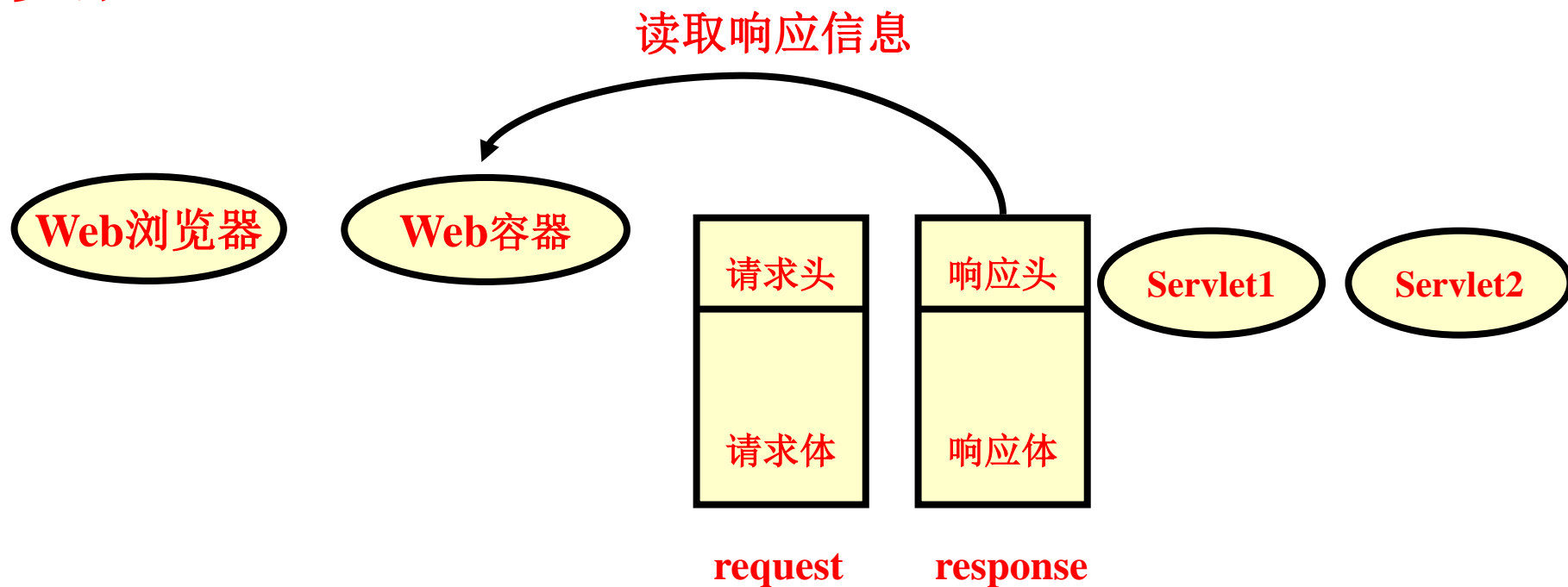
步骤11:



## 7.6 重定向与转发

请求转发的运行流程

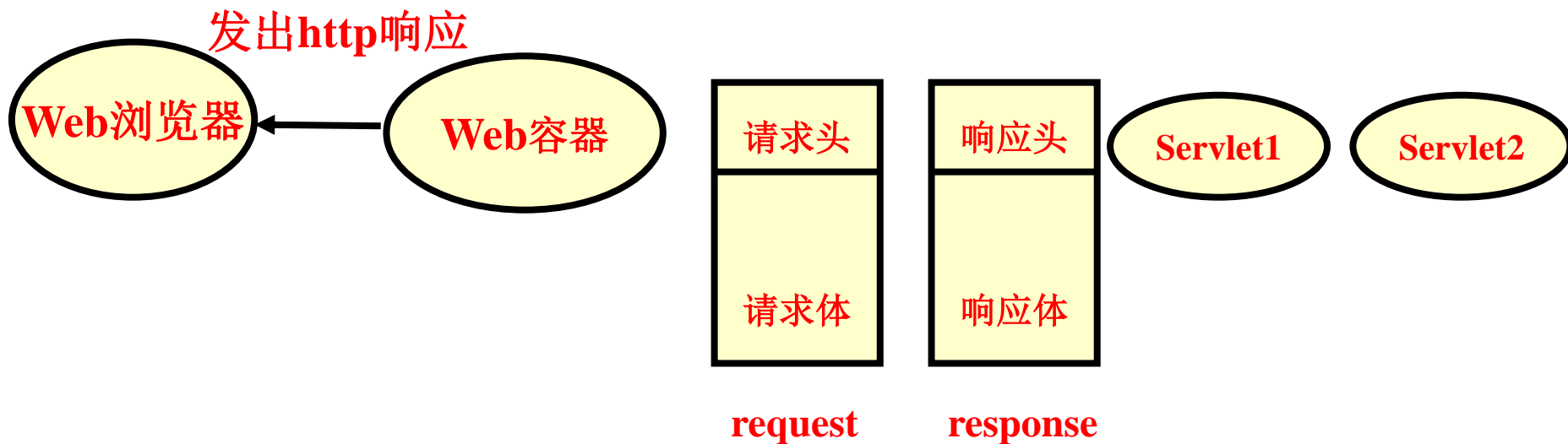
步骤12:



## 7.6 重定向与转发

请求转发的运行流程

步骤13:



注意:

Web服务器将响应信息发送给浏览器处理和显示,一次请求响应过程完全结束,request和response变成垃圾,等待垃圾收集器将其彻底从内存中清除.

## 7.6 重定向与转发

### 1、RequestDispatcher对象

使用forward方法时, 应该注意下面一些问题:

- 如果在调用forward方法之前, 向Servlet引擎的缓冲区写入的内容已经被真正地传送到客户端, forward方法将抛出IllegalStateException异常(属于系统级的异常, 运行时异常).
- 调用RequestDispatcher.Forward方法时, Servlet容器将根据目标资源路径对当前HttpServletRequest对象中的请求路径和参数信息进行调整.

## 7.6 重定向与转发

### 1、RequestDispatcher对象

使用forward方法时, 应该注意下面一些问题:

- 如果在调用forward方法之前向Servlet引擎的缓冲区中写入内容, 只要写入到输出缓冲区的内容还没有被真正输出到客户端, forward方法就可以被正常执行, 原来写入到输出缓冲区中的内容将被清空. 在调用forward方法之后, 如果调用者程序继续向Servlet引擎的缓冲区中执行写入操作, 这些写入操作的执行结果将被忽略.

## 7.6 重定向与转发

### 2、sendRedirect方法

- sendRedirect方法不仅可以重定向到当前应用程序中的其他资源, 它还可以重定向到同一个站点上的其他应用程序中的资源, 甚至是使用绝对URL重定向到其他站点的资源。
- RequestDispatcher.forward方法只能在同一个Web应用程序内的资源之间转发请求。

## 7.6 重定向与转发

### 2、sendRedirect方法

- sendRedirect方法不仅可以重定向到当前应用程序中的其他资源, 它还可以重定向到同一个站点上的其他应用程序中的资源, 甚至是使用绝对URL重定向到其他站点的资源。
- RequestDispatcher.forward方法只能在同一个Web应用程序内的资源之间转发请求。



## 7.6 重定向与转发

### 2、sendRedirect方法

- 在调用sendRedirect方法之前向Servlet引擎的缓冲区中写入了内容, 只要写入到缓冲区中的内容还没有被真正传送到客户端, sendRedirect方法就可以被正常执行, 并且将输出缓冲区中原来写入的内容清空. 位于sendRedirect方法后面的程序代码仍然会执行, 只是向Servlet引擎的缓冲区中进行写入的那些语句的执行结果将被忽略.

## 7.6 重定向与转发

请求重定向的运行流程

步骤1:

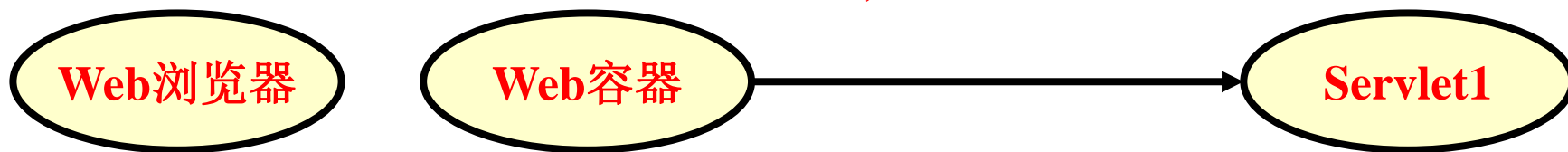


## 7.6 重定向与转发

请求重定向的运行流程

步骤2:

首次访问,容器才创建目标Servlet

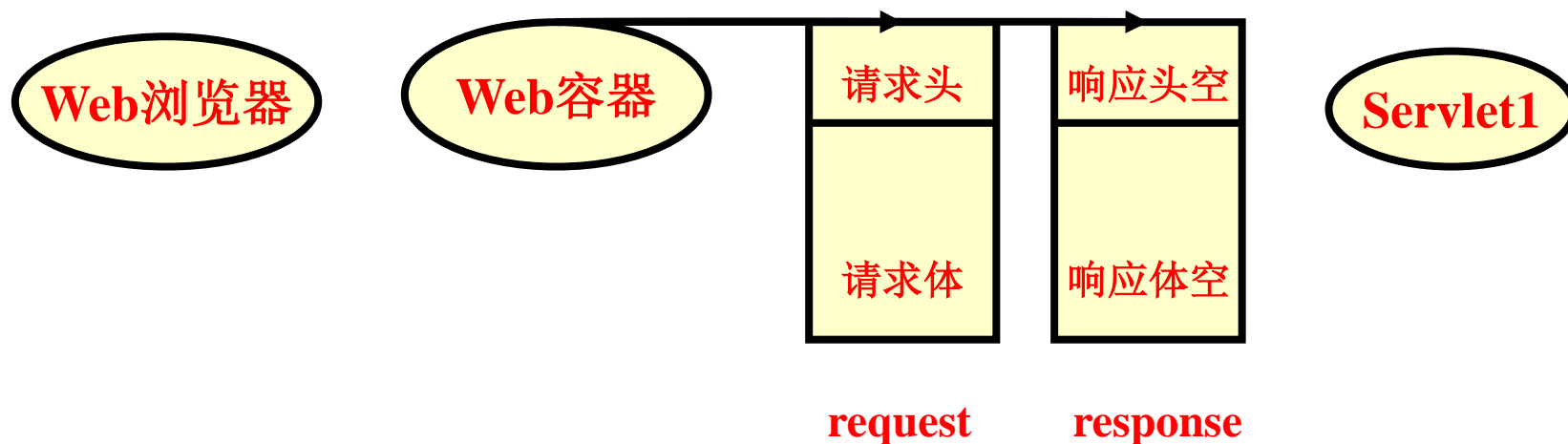


## 7.6 重定向与转发

请求重定向的运行流程

步骤3:

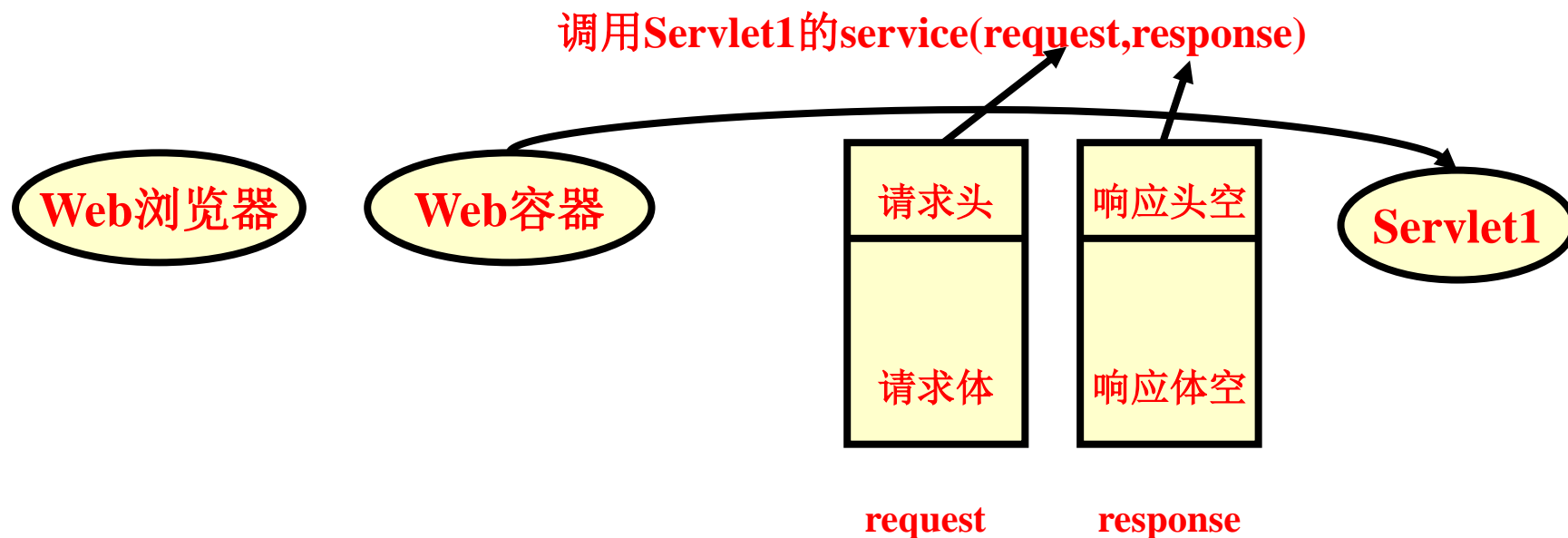
创建请求和响应对象



## 7.6 重定向与转发

## 请求重定向的运行流程

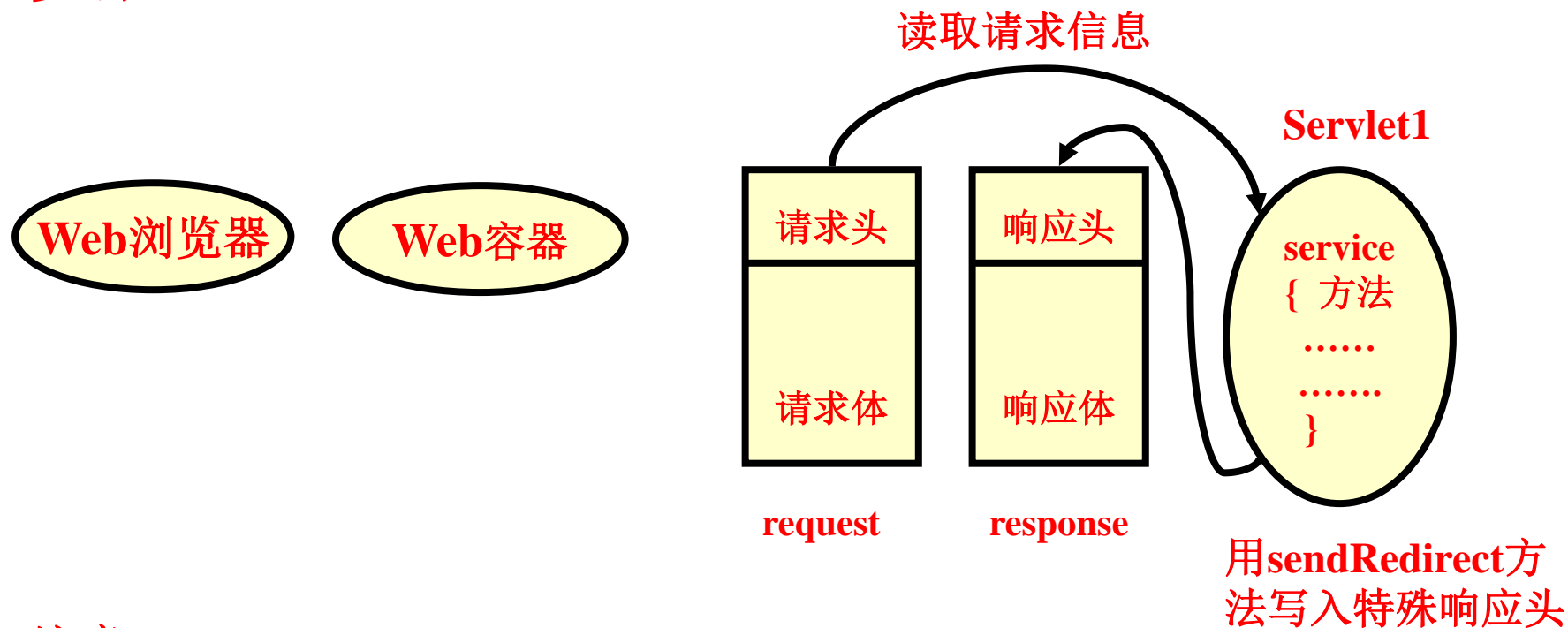
## 步骤4:



## 7.6 重定向与转发

请求重定向的运行流程

步骤5:



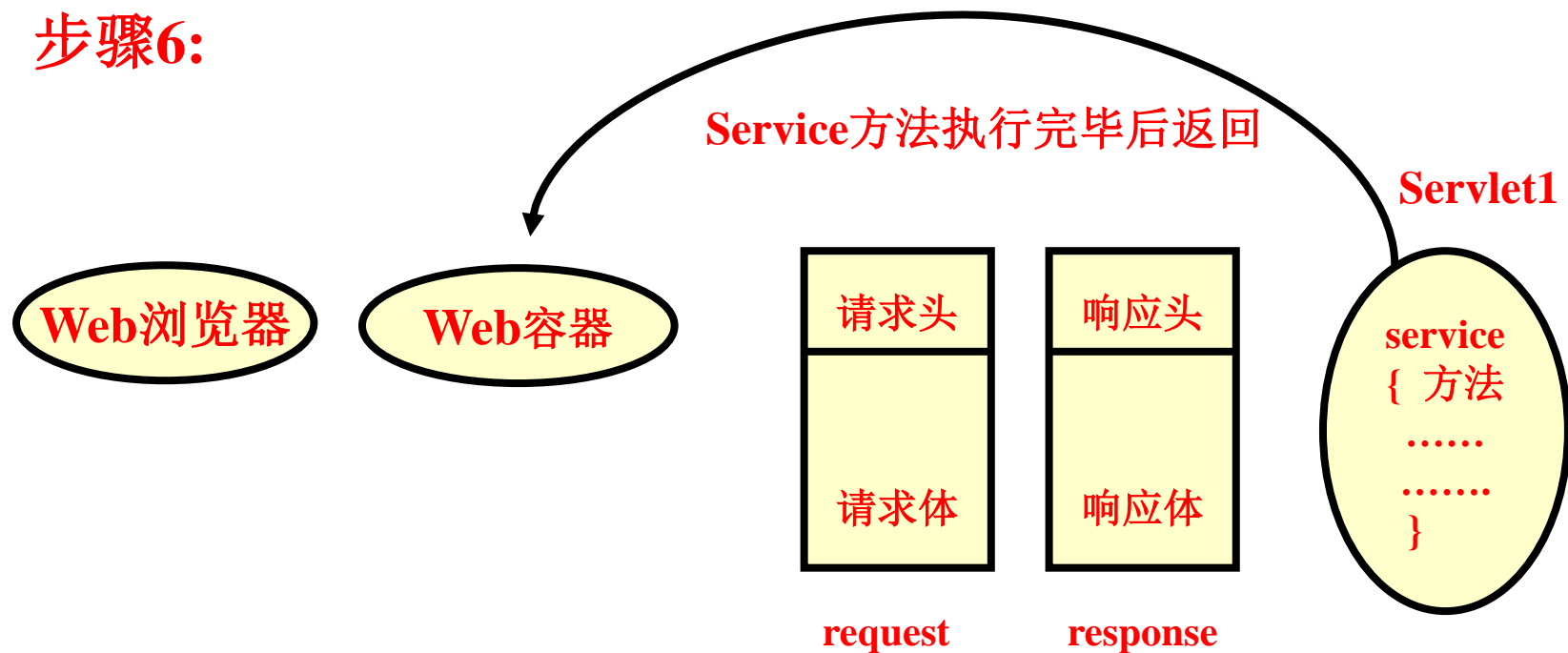
注意:

**Servlet1**对象的**service**方法从请求对象中读取请求信息，并将包含重定向的响应信息写入到响应对象中。

## 7.6 重定向与转发

请求重定向的运行流程

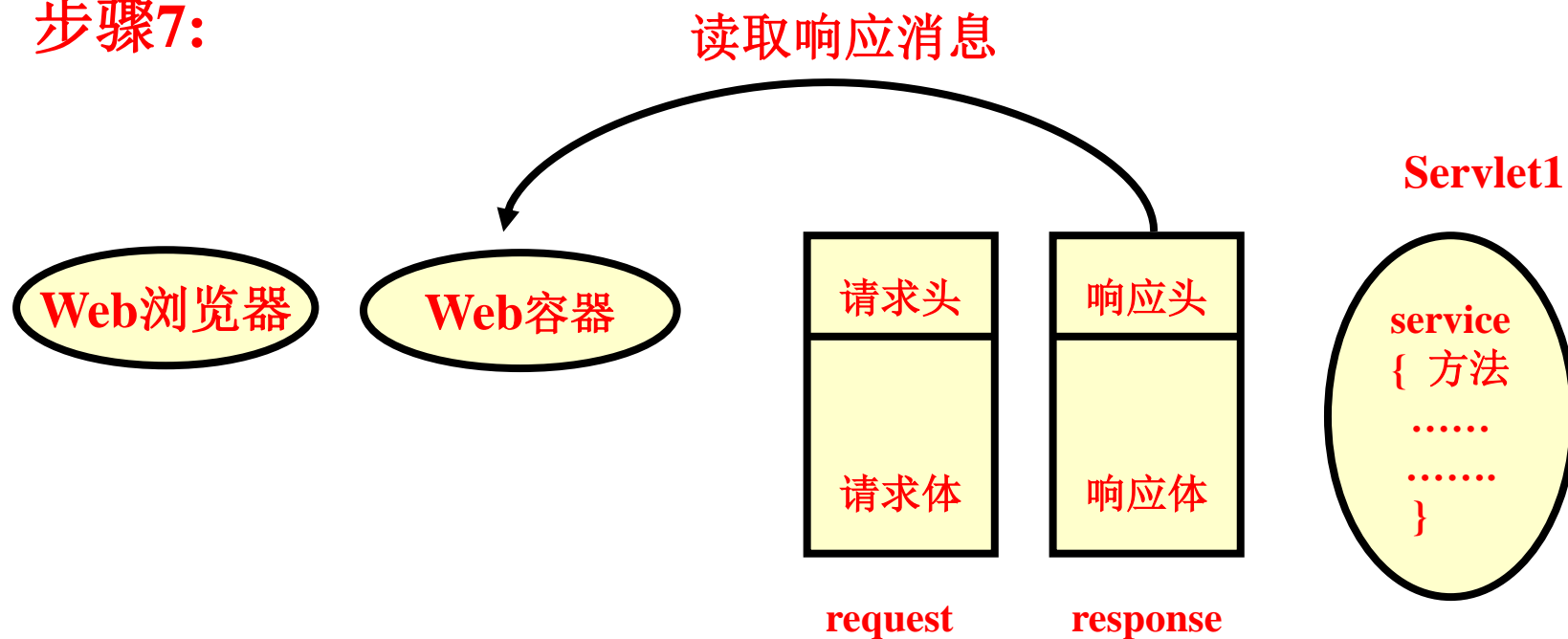
步骤6:



## 7.6 重定向与转发

请求重定向的运行流程

步骤7:

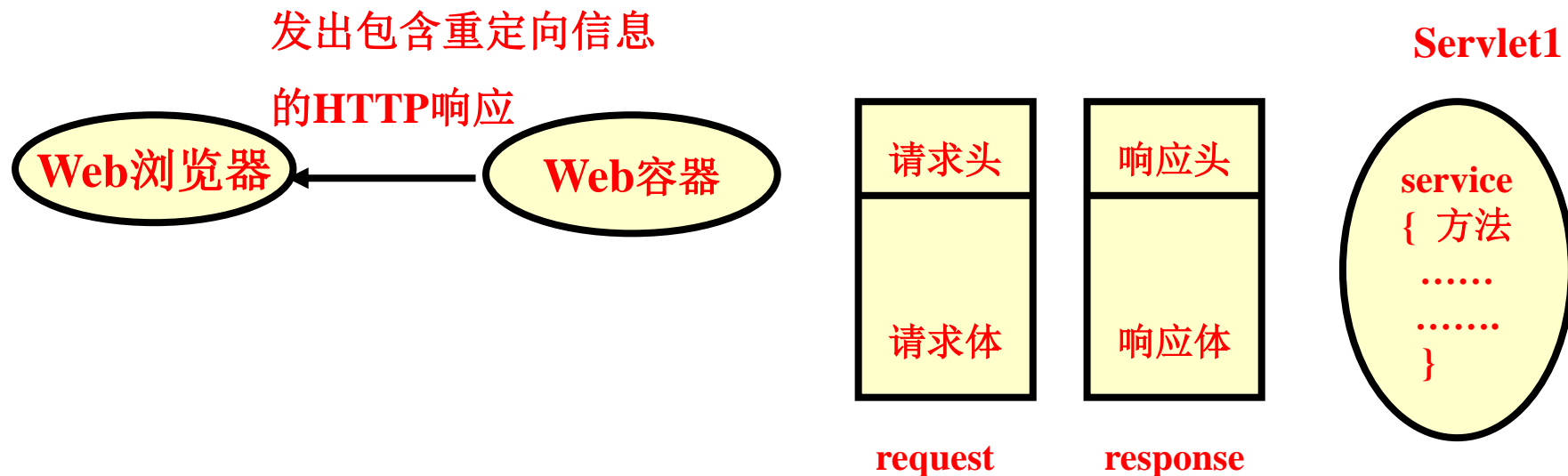




## 7.6 重定向与转发

### 请求重定向的运行流程

#### 步骤8:



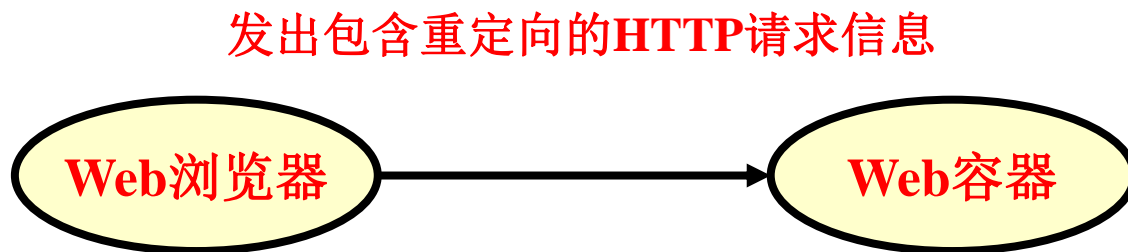
#### 注意:

Web服务器把包含重定向的响应信息发送给浏览器。一次请求响应过程完全结束，request和response变成垃圾，等待垃圾收集器将其彻底从内存中清除。

## 7.6 重定向与转发

请求重定向的运行流程

步骤9:



注意:

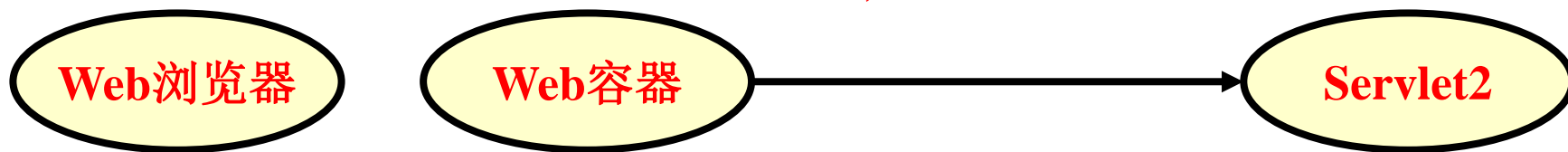
Web浏览器根据上次返回的重定向URL发出新的HTTP请求，这个请求甚至可以指向另外一个站点，Web浏览器又开始一次全新的请求响应过程。

## 7.6 重定向与转发

请求重定向的运行流程

步骤10:

首次访问,容器才创建目标Servlet

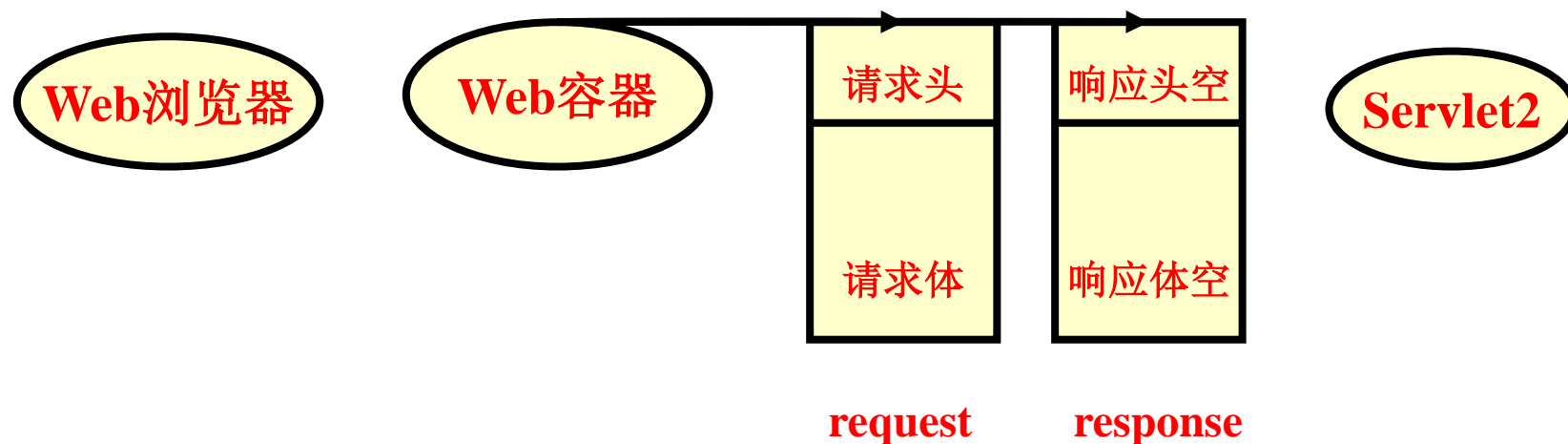


## 7.6 重定向与转发

请求重定向的运行流程

步骤11:

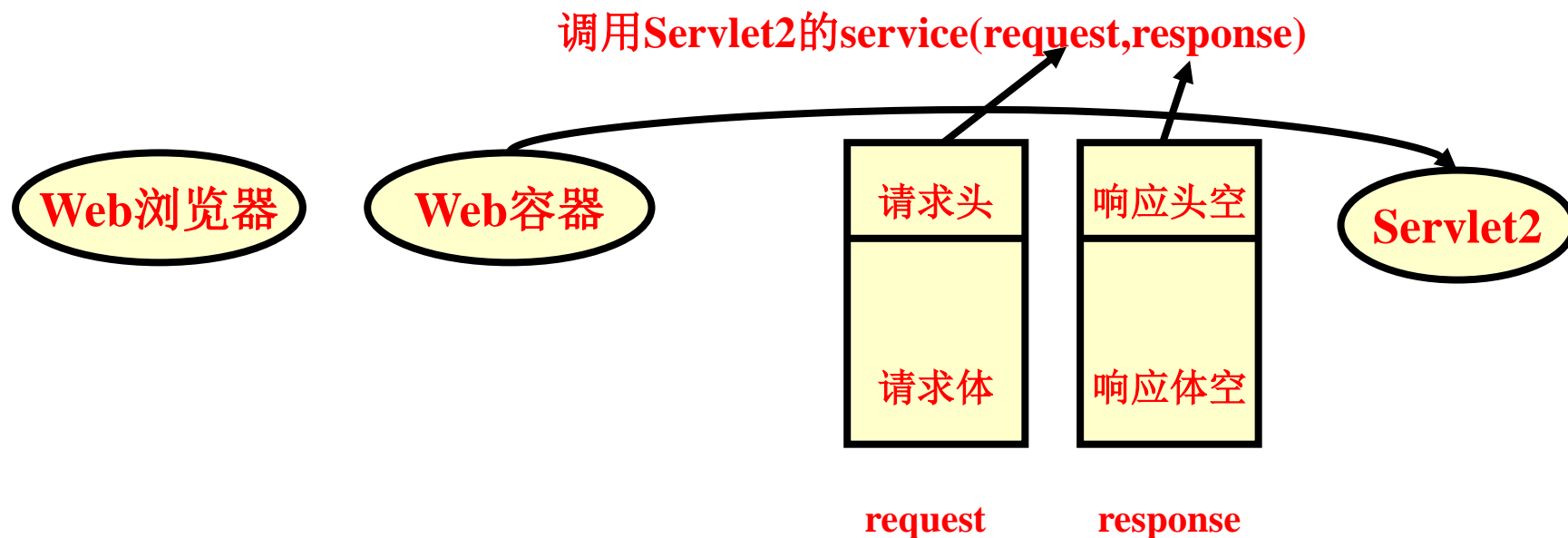
创建新的请求和响应对象



## 7.6 重定向与转发

请求重定向的运行流程

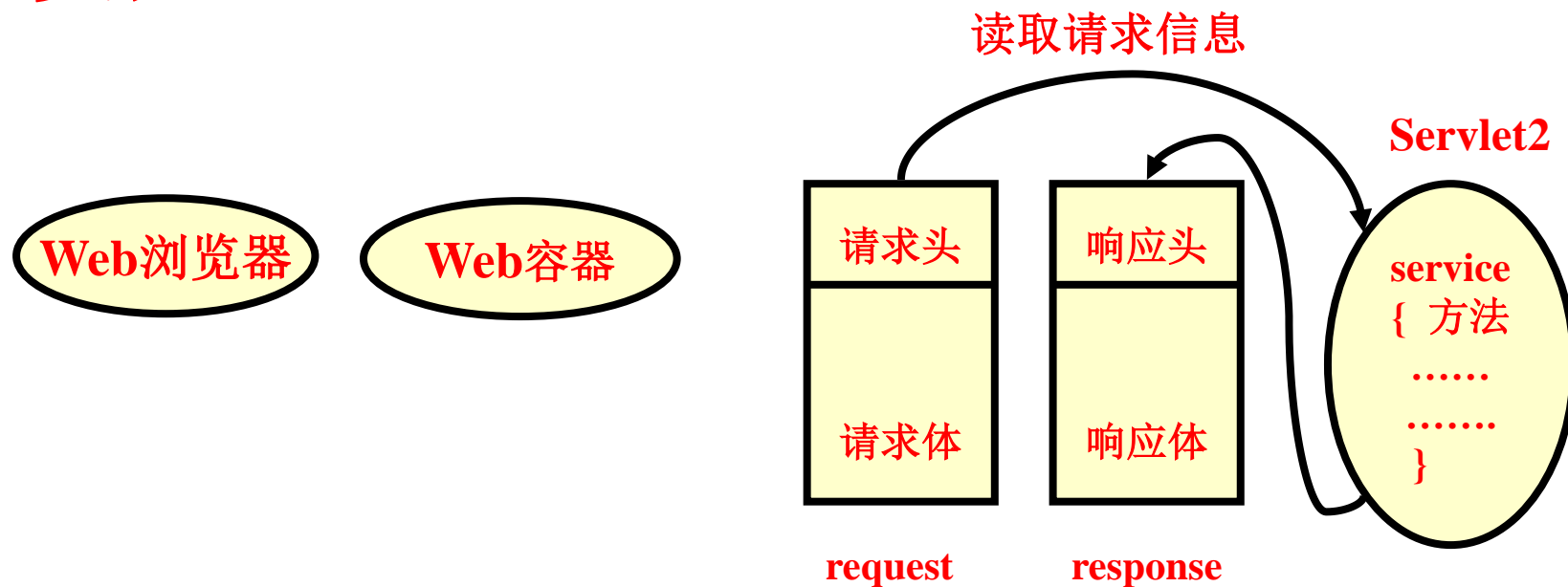
步骤12:



## 7.6 重定向与转发

请求重定向的运行流程

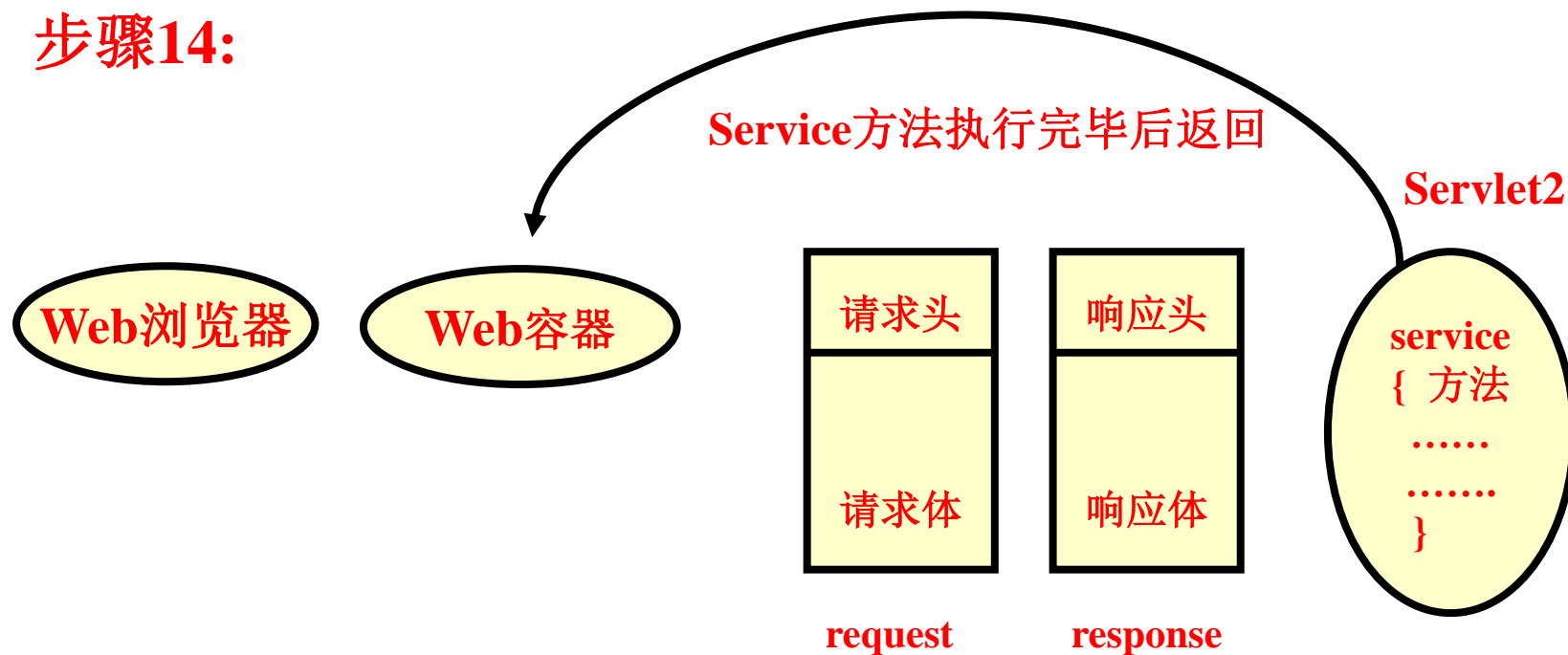
步骤13:



## 7.6 重定向与转发

请求重定向的运行流程

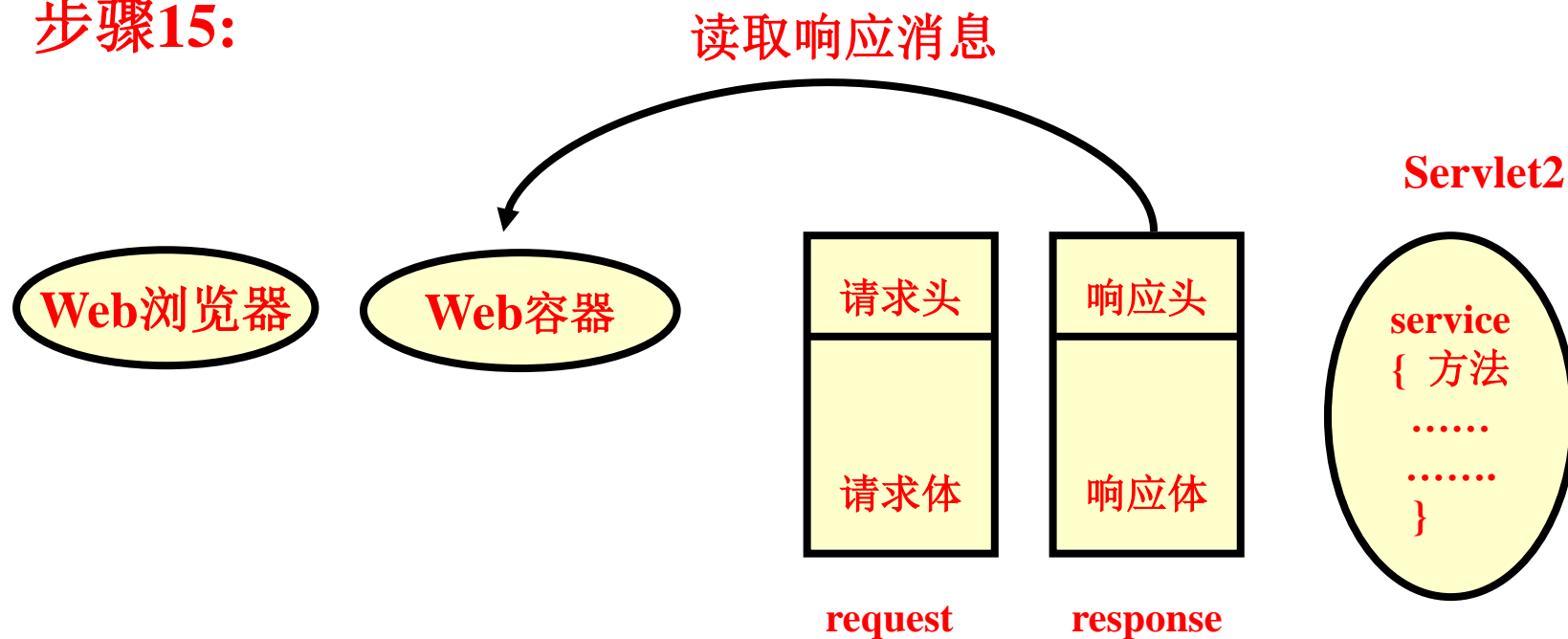
步骤14:



## 7.6 重定向与转发

请求重定向的运行流程

步骤15:

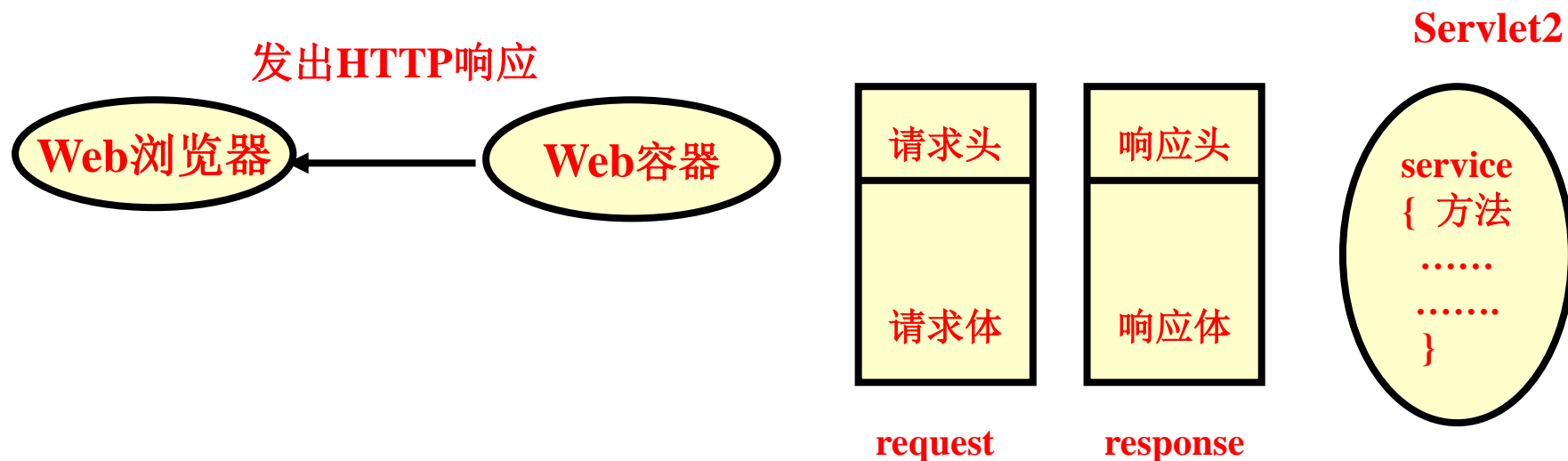




## 7.6 重定向与转发

请求重定向的运行流程

步骤16:



注意:

Web服务器(Web容器)把响应信息发送给浏览器, 重定向请求响应过程完全结束, `request`和`response`变成垃圾, 等待垃圾收集器将其彻底从内存中清除.

## 7.6 重定向与转发

### Example5\_6.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
```

```
<HTML><body bgcolor=cyan><font size=2>
```

```
<form action="twoPath" method=post>
```

```
  输入一个实数: <input type=text name=number>
```

```
<br><input type=submit value="提交">
```

```
</form></body></HTML>
```

## 7.6 重定向与转发

### web.xml

**<servlet>**

**<servlet-name> twoPath </servlet-name>**

**<servlet-class>myservlet.control.Example5\_6\_Servlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name> twoPath </servlet-name>**

**<url-pattern> / twoPath </url-pattern>**

**</servlet-mapping>**

## 7.6 重定向与转发

### Example5\_6\_Servlet.java

```
package myservlet.control;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Example5_6_Servlet extends HttpServlet{
    public void init(ServletConfig config) throws ServletException{
        super.init(config);
    }
    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException,IOException{
        String number=request.getParameter("number");
        try{ double n=Double.parseDouble(number);
            if(n<0)
                response.sendRedirect("example5_6_show.jsp"); //重定向
            else{
                RequestDispatcher dispatcher=
```

## 7.6 重定向与转发

### Example5\_6\_Servlet.java

```
public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{
    String number=request.getParameter("number");
    try{    double n=Double.parseDouble(number);
        if(n<0)
            response.sendRedirect("example5_6_show.jsp"); //重定向
        else{
            RequestDispatcher dispatcher=
            request.getRequestDispatcher("example5_6_show.jsp");
            dispatcher.forward(request, response); //请求转发转发
        }
    }
    catch(NumberFormatException e){
        response.sendRedirect("example5_6.jsp"); //重定向}
    }
    public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{
        doPost(request, response);
    }
}
```

## 7.6 重定向与转发

### example5\_6\_show.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
```

```
<HTML><body bgcolor=pink><font size=2>
```

尽管example5\_6.jsp它没有直接请求我<br>

我能获得example5\_6.jsp提交的非负数，

<br>但不能获得example5\_6.jsp提交的负数。

```
<%
```

```
String number= request.getParameter("number");
```

```
%>
```

```
<b><br>用户在example5_6.jsp输入的非负数是<%= number%>
```

```
</body></HTML>
```

## 7.7 会话和状态管理

### 1、什么是会话？

- 它是指一个客户端与Web服务器之间连续发生的一系列请求和响应过程(例如,一个用户在某网站上的整个购物过程就是一个会话).

### 2、什么是Web应用的会话状态？

- 指Web服务器与浏览器在会话过程中产生的状态信息,借助会话状态, Web服务器能够把属于同一个会话中的一系列的请求和响应过程关联起来,使得它们之间可以互相依赖和传递信息.

## 7.7 会话和状态管理

---

### 3、如何实现有状态的会话？

- ◆ 建立含有跟踪数据的隐藏表单字段
- ◆ 重写包含额外参数的URL
- ◆ 使用持久的Cookie(会话ID)



## 7.7 会话和状态管理

### Cookie

- HTTP cookies是最常用的会话跟踪机制，所有的servlet引擎都应该支持这种方法。
- 原理：引擎发送一个cookie到客户端，客户端就会在以后的请求中把这个cookie返回给服务器。用户会话跟踪的cookie的名字必须是JSESSIONID

## 7.7 会话和状态管理

### 隐藏表单

- 可以下列的方法来做隐藏表单字段的会话追踪。`<input type="hidden" name="userID" value="15">`
- **优点：** session数据传送到服务器端时，并不象GET的方法，会将session数据保露在URL之上。
- **缺点：** 一旦session数据储存在隐藏字段中，就仍然有暴露数据的危机，因为只要用户直接观看HTML的源文件，session数据将会暴露无疑，这将造成安全上的漏洞。

## 7.7 会话和状态管理

### URL重写

- URL重写是最低性能的通用会话跟踪方法。
- 当一个客户端不能接受cookie时，URL重写就会作为基本的会话跟踪方法；URL重写包括一个附加的数据，一个session id，这样的URL会被引擎解析和一个session相关联。session id是作为URL的的参数传输的，这个参数名字必须是jsessionid.

## 7.7 会话和状态管理

### 4、Servlet的会话机制

- 基于Cookie或URL重写技术，融合了这两种技术的优点。
- 当客户端允许使用Cookie时，内建session对象使用Cookie进行会话追踪。
- 如果客户端禁用Cookie，则选择使用URL重写。
- 使用Cookie和附加URL参数可以将上一次请求的状态信息传递到下一次请求中，但是如果传递的状态信息较多，将极大降低网络传输效率和增大服务器端程序处理的难度。

## 7.7 会话和状态管理

### 5、HttpSession对象

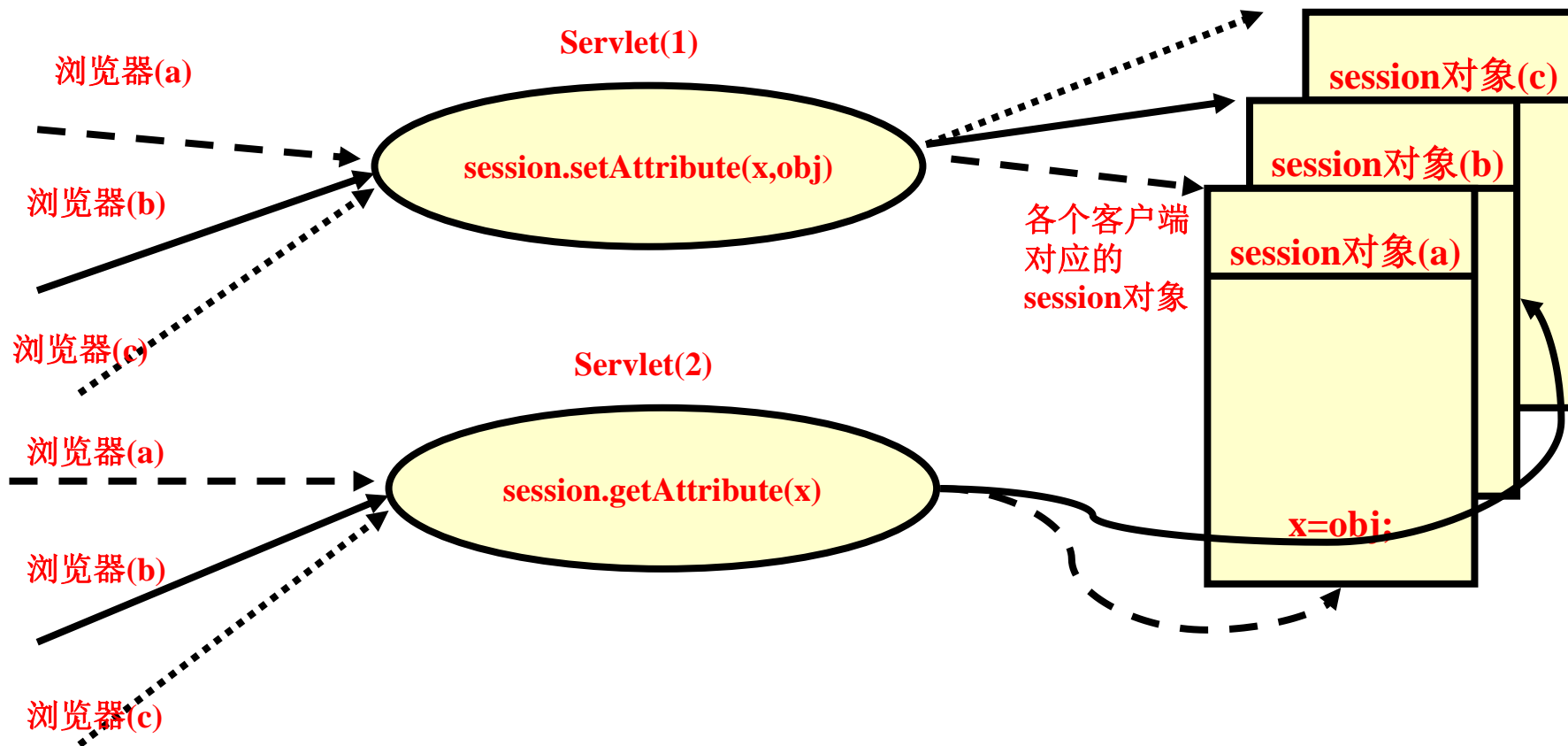
- HttpServletRequest对象request调用getSession方法获取用户的session对象:

```
HttpSession session=request.getSession(true)
```

- 存储在HttpSession对象中的属性也可以被所有的Servlet程序访问,但仅仅是来自同一个客户端的一组访问才能共享同一个HttpSession对象,来自不同客户端的各组访问共享的HttpSession对象是不同的.

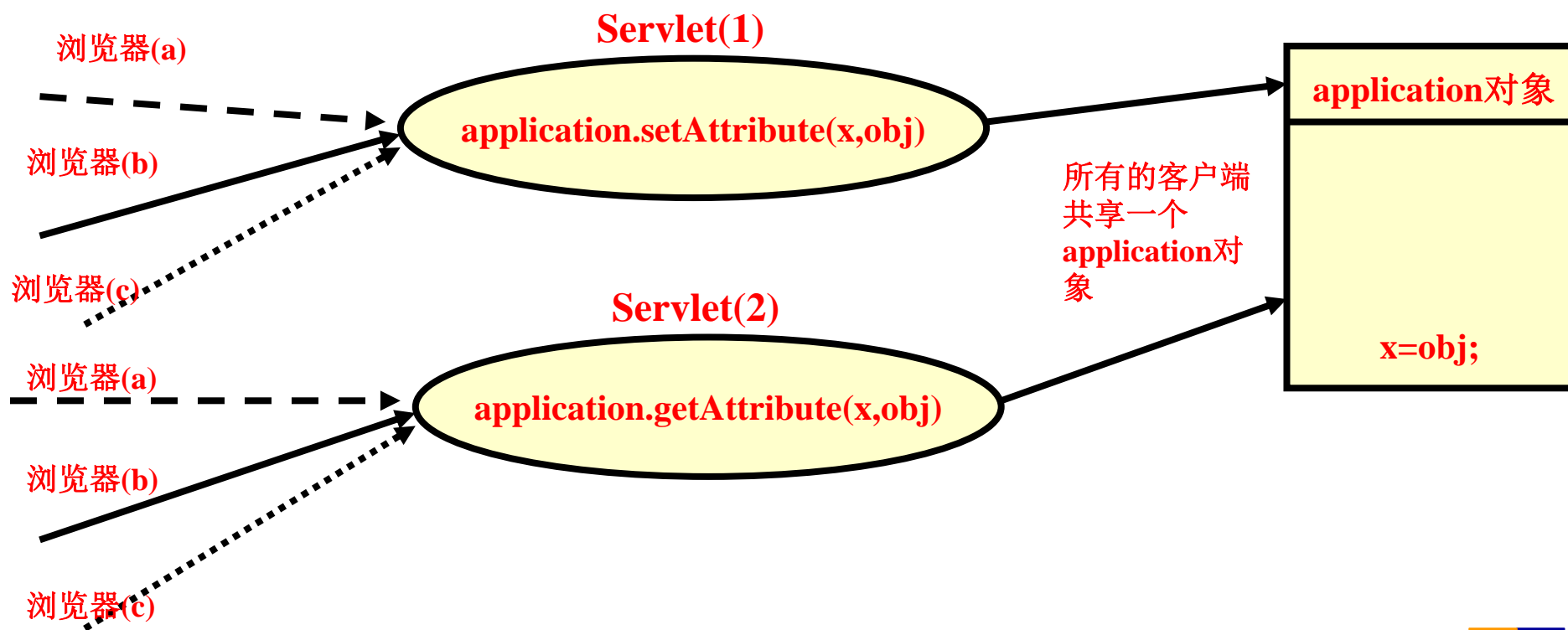
## 7.7 会话和状态管理

### 5、HttpSession对象



## 7.7 会话和状态管理

存储在**application**对象中的属性可以被该**Web**应用程序中的所有**Servlet**程序访问,而不管访问来自哪个客户端。



## 7.7 会话和状态管理

### example5\_7.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
<HTML><body bgcolor=#EEFFDD ><font size=3>
<% session.setAttribute("message","请您猜字母");
   char a[]=new char[26];
   int m=0;
   for(char c='a';c<='z';c++) {
       a[m]=c;
       m++;
   }
   int randomIndex=(int)(Math.random()*a.length);
   char ch=a[randomIndex];    //获取一个英文字母
   session.setAttribute("savedLetter",new Character(ch));//将该字母放入session中
%>
访问或刷新该页面可以随机得到一个英文字母.
<BR>单击超链接去猜出这个字母:<a href="example5_7_input.jsp">
   去猜字母</a>
</font></body></HTML>
```



## 7.7 会话和状态管理

### example5\_7\_input.jsp

```
<%@ page contentType="text/html;charset=gb2312" %>
<HTML><body ><font size=2>
<% String message=(String)session.getAttribute("message");
    %>
<table border=1>
<form action="guess" method=post>
  <tr><td> 输入您的猜测 (a~z之间的字母) :</td>
  <td><input Type=text name=clientGuessLetter size=4>
    <input Type=submit value="提交"></td>
</tr><td> 提示信息:</td>
  <td> <%= message%></td>
</form>
<form action="example5_7.jsp" method=post>
  <tr><td>单击按钮重新开始: </td>
    <td><Input Type=submit value="随机得到一个字母"></td>
  </tr>
</form>
</font></body></HTML>
```

## 7.7 会话和状态管理

### web.xml

**<servlet>**

**<servlet-name>** guess**</servlet-name>**

**<servlet-class>**myservlet.control.Example5\_7\_Servlet**</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>** guess**</servlet-name>**

**<url-pattern>**/ guess**</url-pattern>**

**</servlet-mapping>**

## 7.7 会话和状态管理

### **example5\_7\_Servlet.java**

```
package myservlet.control;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Example5_7_Servlet extends HttpServlet {
    public void init(ServletConfig config) throws ServletException {
        super.init(config); }
    public void doPost(HttpServletRequest request,HttpServletResponse
        response) throws ServletException,IOException {
        HttpSession session=request.getSession(true);
        String str=request.getParameter("clientGuessLetter");
        Character guessLetter=str.trim().charAt(0); //获取客户猜测所提
        交的字母
        //获得曾放入session中的字母:
        Character
        savedLetter=(Character)session.getAttribute("savedLetter");
```

## 7.7 会话和状态管理

### example5\_7\_Servlet.java

```
char realLetter=savedLetter.charValue();
    if(Character.isUpperCase(guessLetter)) {
        guessLetter=Character.toLowerCase(guessLetter);
    }
    if(guessLetter<realLetter) {
        session.setAttribute("message","您猜小了");
        response.sendRedirect("example5_7_input.jsp");
    }
    if(guessLetter>realLetter) {
        session.setAttribute("message","您猜大了");
        response.sendRedirect("example5_7_input.jsp");
    }
    if(guessLetter==realLetter){
        session.setAttribute("message","您猜对了");
        response.sendRedirect("example5_7_input.jsp");
    }
}
```

## 7.7 会话和状态管理

### example5\_7\_Servlet.java

```
char realLetter=savedLetter.charValue();
    if(Character.isUpperCase(guessLetter)) {
        guessLetter=Character.toLowerCase(guessLetter);
    }
    if(guessLetter<realLetter) {
        session.setAttribute("message","您猜小了");
        response.sendRedirect("example5_7_input.jsp");
    }
    if(guessLetter>realLetter) {
        session.setAttribute("message","您猜大了");
        response.sendRedirect("example5_7_input.jsp");
    }
    if(guessLetter==realLetter){
        session.setAttribute("message","您猜对了");
        response.sendRedirect("example5_7_input.jsp");
    }
}
```

# 小结

- Servlet的核心是在服务端创建响应用户请求的对象，即创建servlet对象。
- Servlet对象第一次被请求加载时，服务器创建一个servlet对象，这个对象调用init方法完成必要的初始化工作。**init方法只被调用一次**，当后续的客户请求该servlet对象服务时，服务器将启动一个**新的线程**，在该线程中servlet对象调用service方法响应客户的请求，**调用过程运行在不同的线程中，互不干扰**。

# 小结

➤Servlet类继承的service方法检查HTTP请求类型（Get、Post等），并在service方法中根据用户的请求方式，对应地再调用doGet或doPost方法。因此，Servlet类不必重写service方法，直接继承该方法即可，可以在Servlet类中重写doPost或doGet方法来响应用户的请求。

➤RequestDispatcher对象可以把用户对当前JSP页面或servlet的请求转发给另一个JSP页面或servlet，而且将用户对当前JSP页面或servlet的**请求和响应传递**给所转发的JSP页面或servlet。也就是说，当前页面所要转发的目标页面或servlet对象可以使用request获取用户提交的数据