**Chapter 15  
Java Servlet Technology**

Shortly after the Web began to be used for delivering services, service providers recognized the need for dynamic content. Applets, one of the earliest attempts toward this goal, focused on using the client platform to deliver dynamic user experiences. At the same time, developers also investigated using the server platform for the same purpose. Initially, Common Gateway Interface (CGI) server-side scripts were the main technology used to generate dynamic content. Although widely used, CGI scripting technology had many shortcomings, including platform dependence and lack of scalability. To address these limitations, Java Servlet technology was created as a portable way to provide dynamic, user-oriented content.

Servlet是随着网络多动态内容而兴起的，开始是applet的一些早期的尝试，这些尝试使用client platform以达到目的。同时一些开发者投身到server playform中。

Server端的通用网管接口（CGI）脚本是提供动态内容的主要技术，尽管这项技术被很广泛的使用，但是他包含了很多的缺点。为了提供动态的、用户导向的内容，Java servlet 技术y应运而生。

The following topics are addressed here:

* [What Is a Servlet?](http://docs.oracle.com/javaee/6/tutorial/doc/bnafe.html)
* [Servlet Lifecycle](http://docs.oracle.com/javaee/6/tutorial/doc/bnafi.html)
* [Sharing Information](http://docs.oracle.com/javaee/6/tutorial/doc/bnafo.html)
* [Creating and Initializing a Servlet](http://docs.oracle.com/javaee/6/tutorial/doc/bnafu.html)
* [Writing Service Methods](http://docs.oracle.com/javaee/6/tutorial/doc/bnafv.html)
* [Filtering Requests and Responses](http://docs.oracle.com/javaee/6/tutorial/doc/bnagb.html)
* [Invoking Other Web Resources](http://docs.oracle.com/javaee/6/tutorial/doc/bnagi.html)
* [Accessing the Web Context](http://docs.oracle.com/javaee/6/tutorial/doc/bnagl.html)
* [Maintaining Client State](http://docs.oracle.com/javaee/6/tutorial/doc/bnagm.html)
* [Finalizing a Servlet](http://docs.oracle.com/javaee/6/tutorial/doc/bnags.html)
* [The mood Example Application](http://docs.oracle.com/javaee/6/tutorial/doc/gkcpg.html)
* [Further Information about Java Servlet Technology](http://docs.oracle.com/javaee/6/tutorial/doc/bnagw.html)

What Is a Servlet?

A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.

The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines lifecycle methods. When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

Servlet是一个为了扩展servers的功能的java语言类，host application 通过请求-响应模型（request-response model）来接收访问。尽管servlets能够响应几乎任何的请求，但是他们被广泛的使用在扩展web服务器上的应用。为了这些web应用，java servlet技术定义了单独为http使用的类。

Javax.servlet和javax.servlet.http包为写servlet提供了接口和类。所有的servlet必须实现servlet接口，在接口中定义了存在于servlet的生命周期的方法。当要实现一个一般的service，可以使用或者扩展java servlet api中提供的一般的servlet类。例如，HttpServlet类提供了方法，如doGet和doPost来处理特定的http请求。

## Servlet Lifecycle

The lifecycle of a servlet is controlled by the container in which the servlet has been deployed. When a request is mapped to a servlet, the container performs the following steps.

1. If an instance of the servlet does not exist, the web container
   1. Loads the servlet class.
   2. Creates an instance of the servlet class.
   3. Initializes the servlet instance by calling the init method. Initialization is covered in [Creating and Initializing a Servlet](http://docs.oracle.com/javaee/6/tutorial/doc/bnafu.html).
2. Invokes the service method, passing request and response objects. Service methods are discussed in [Writing Service Methods](http://docs.oracle.com/javaee/6/tutorial/doc/bnafv.html).

If it needs to remove the servlet, the container finalizes the servlet by calling the servlet’s destroy method. For more information, see [Finalizing a Servlet](http://docs.oracle.com/javaee/6/tutorial/doc/bnags.html).

一个servlet的生命周期是由该servlet被部署的容器控制的。当一个请求被映射到一个servlet处理后，这个容器就会执行以下步骤：

1. 如果该servlet实例不存在，那么web容器将会：
   1. 载入此servlet的类文件
   2. 创建这个servlet的实例
   3. 调用servlet中的init方法初始化这个servlet实例，初始化包含串讲和初始化一个servlet
2. 调用service方法，传递参数requet和response对象。

如果需要移除某个servlet，容器会通过调用servlet的destroy方法来终止该servlet。

### Handling Servlet Lifecycle Events

You can monitor and react to events in a servlet’s lifecycle by defining listener objects whose methods get invoked when lifecycle events occur. To use these listener objects, you must define and specify the listener class.

出来servlet生命周期中的事件：

通过定义listener对象，你可以监控和响应servlet生命周期中的事件，当生命周期中的事件发生时会激活这些方法。为了使用这些listener，你必须定义并指定这些监听类。

#### Defining the Listener Class

You define a listener class as an implementation of a listener interface. [Table 15-1](http://docs.oracle.com/javaee/6/tutorial/doc/bnafi.html#bnafl) lists the events that can be monitored and the corresponding interface that must be implemented. When a listener method is invoked, it is passed an event that contains information appropriate to the event. For example, the methods in the HttpSessionListener interface are passed an HttpSessionEvent, which contains an HttpSession.

定义监听类-listener class

通过实现一个监听接口-listener interface来定义一个监听类。下表列出了哪些事件可以被监听和相应的需要实现的接口。当一个监听方法被调用了之后，会传递一个包含了该事件信息的参数。

**Table 15-1 Servlet Lifecycle Events**

|  |  |  |
| --- | --- | --- |
| **Object** | **Event** | **Listener Interface and Event Class** |
| Web context | Initialization and destruction | javax.servlet.ServletContextListener and ServletContextEvent |
| Web context | Attribute added, removed, or replaced  添加、移除和替换属性事件 | javax.servlet.ServletContextAttributeListener and ServletContextAttributeEvent |
| Session | Creation, invalidation, activation, passivation, and timeout  创建、验证、激活和timeout事件 | javax.servlet.http.HttpSessionListener, javax.servlet.http.HttpSessionActivationListener, and HttpSessionEvent |
| Session | Attribute added, removed, or replaced | javax.servlet.http.HttpSessionAttributeListener and HttpSessionBindingEvent |
| Request | A servlet request has started being processed by web components  被web组件开始处理一个servlet请求 | javax.servlet.ServletRequestListener and ServletRequestEvent |
| Request | Attribute added, removed, or replaced | javax.servlet.ServletRequestAttributeListener and ServletRequestAttributeEvent |

Use the @WebListener annotation to define a listener to get events for various operations on the particular web application context. Classes annotated with @WebListener must implement one of the following interfaces:

使用 @WebListener注解来定义一个监听器以监控web应用的各种事件，被此注解修饰的类必须实现以下接口中的至少一个。

javax.servlet.ServletContextListener

javax.servlet.ServletContextAttributeListener

javax.servlet.ServletRequestListener

javax.servlet.ServletRequestAttributeListener

javax.servlet..http.HttpSessionListener

javax.servlet..http.HttpSessionAttributeListener

For example, the following code snippet defines a listener that implements two of these interfaces:

import javax.servlet.ServletContextAttributeListener;

import javax.servlet.ServletContextListener;

import javax.servlet.annotation.WebListener;

@WebListener()

public class SimpleServletListener implements ServletContextListener,

ServletContextAttributeListener {

...

### Handling Servlet Errors

Any number of exceptions can occur when a servlet executes. When an exception occurs, the web container generates a default page containing the following message:

A Servlet Exception Has Occurred

But you can also specify that the container should return a specific error page for a given exception.

处理servlet错误

当异常发生时，web容器会产生一个包含以下内容的默认页面：

但是你可以针对特定的异常指定容器返回特定的错误页面。

# 17.3 Sharing Information

Web components, like most objects, usually work with other objects to accomplish their tasks. Web components can do so by doing the following.

* Using private helper objects (for example, JavaBeans components).
* Sharing objects that are attributes of a public scope.
* Using a database.
* Invoking other web resources. The Java Servlet technology mechanisms that allow a web component to invoke other web resources are described in [Invoking Other Web Resources](https://docs.oracle.com/javaee/7/tutorial/servlets007.htm#BNAGI).

Web组件之间也会共同工作已完成他们的任务，他们可能通过以下操作：

使用私有的帮助对象（例如JavaBean组件）

共享public修饰的对象

使用数据库

使用其他的web资源，具体信息参考 [Invoking Other Web Resources](https://docs.oracle.com/javaee/7/tutorial/servlets007.htm#BNAGI)

## 17.3.1 Using Scope Objects

Collaborating web components share information by means of objects that are maintained as attributes of four scope objects. You access these attributes by using the getAttribute and setAttribute methods of the class representing the scope. [Table 17-2](https://docs.oracle.com/javaee/7/tutorial/servlets003.htm#BNAFQ) lists the scope objects.

***Table 17-2 Scope Objects***

| **Scope Object对象空间** | **Class** | **Accessible From** |
| --- | --- | --- |
| Web context | javax.servlet.ServletContext | Web components within a web context. See [Accessing the Web Context](https://docs.oracle.com/javaee/7/tutorial/servlets008.htm#BNAGL). |
| Session | javax.servlet.http.HttpSession | Web components handling a request that belongs to the session. See [Maintaining Client State](https://docs.oracle.com/javaee/7/tutorial/servlets009.htm#BNAGM). |
| Request | Subtype of javax.servlet.ServletRequest | Web components handling the request. |
| Page | javax.servlet.jsp.JspContext | The JSP page that creates the object. |

使用空间对象：

协作的web组件通过对象（这些对象作为四个空间对象的属性）来共享信息（或者可以传递信息）。你可以使用这写对象提供的getAttribute和setAttribute方法操作这些属性。

这些对象如下表所示，包括：ServletContext、httpSession、ServletRequest和JspContext

## 17.3.2 Controlling Concurrent Access to Shared Resources

In a multithreaded server, shared resources can be accessed concurrently. In addition to scope object attributes, shared resources include in-memory data, such as instance or class variables, and external objects, such as files, database connections, and network connections.

对并发访问共享资源的控制：

在多线程的服务器中，共享资源可能被并发访问。这些共享资源除了空间对象属性之外，还包括内存数据例如实例和类的变量，外部对象如文件、数据库连接和网络连接等。

Concurrent access can arise in several situations.

* Multiple web components accessing objects stored in the web context.
* Multiple web components accessing objects stored in a session.
* Multiple threads within a web component accessing instance variables. A web container will typically create a thread to handle each request. To ensure that a servlet instance handles only one request at a time, a servlet can implement the SingleThreadModel interface. If a servlet implements this interface, no two threads will execute concurrently in the servlet's service method. A web container can implement this guarantee by synchronizing access to a single instance of the servlet or by maintaining a pool of web component instances and dispatching each new request to a free instance. This interface does not prevent synchronization problems that result from web components' accessing shared resources, such as static class variables or external objects.

When resources can be accessed concurrently, they can be used in an inconsistent fashion. You prevent this by controlling the access using the synchronization techniques described in the Threads lesson at <http://docs.oracle.com/javase/tutorial/essential/concurrency/>.

一些情况会导致并发访问：

多web组件访问存储在web context中的对象

多web组件访问存储在session中的对象

在一个web组件内的多线程访问实例变量：

通常，一个web容器会为每个请求创建一个线程。一个servlet实例每次只处理一个request，servlet可以实现SingleThreadModel接口。如果一个servlet实现了这个接口，不会有两个线程同时执行servlet中的方法。Web容器通过同步访问一个servlet的单实例，或者维护一个web组件的线程池然后将每个请求分发给空闲的实例来请求。实现了这个接口也不会阻止web组件对共享资源的同步访问，web组件仍然可以访问共享资源如类变量或者外部的对象(这个也间接说明了web的容器可以实现单例而php的cgi模式不能实现单例)。

# 17.4 Creating and Initializing a Servlet

Use the @WebServlet annotation to define a servlet component in a web application. This annotation is specified on a class and contains metadata about the servlet being declared. The annotated servlet must specify at least one URL pattern. This is done by using the urlPatterns or value attribute on the annotation. All other attributes are optional, with default settings. Use the value attribute when the only attribute on the annotation is the URL pattern; otherwise, use the urlPatterns attribute when other attributes are also used.

通过使用@WebServlet注解可以定义一个web应用的servlet组件。这个注解定义在一个类上，包含了定义一个servlet的metadata.这个被注解修饰的servlet必须包含一个指定的url模式。可以使用url格式的字符串或者值属性在这个注解上。除此之外，所有的其他的属性是可选的，如果没有指定则使用默认的配置。

Classes annotated with @WebServlet must extend the javax.servlet.http.HttpServlet class. For example, the following code snippet defines a servlet with the URL pattern /report:

被@WebServlet注解修饰的类必须继承自HttpServlet类

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

@WebServlet("/report")

public class MoodServlet extends HttpServlet {

...

The web container initializes a servlet after loading and instantiating the servlet class and before delivering requests from clients. To customize this process to allow the servlet to read persistent configuration data, initialize resources, and perform any other one-time activities, you can either override the init method of the Servlet interface or specify the initParams attribute of the @WebServlet annotation. The initParams attribute contains a @WebInitParam annotation. If it cannot complete its initialization process, a servlet throws an UnavailableException.

Use an initialization parameter to provide data needed by a particular servlet. By contrast, a context parameter provides data that is available to all components of a web application.

当web容器加载和初始化一个servlet类后，会调用init方法初始化一个servlet，然后才会分发处理客户端的请求。为了自定义处理，如允许servlet读取持久性的配置数据，初始化资源和任何其他的一次性的活动，开发者可以重写Servlet接口中的init方法，或者在@WebServlet注解中指定initParms属性。InitParams属性包含一个 @WebInitParam属性。如果在初始化阶段没有完成此初始化过程，servlet会抛出一个UnavailableException异常。

# 17.5 Writing Service Methods

The service provided by a servlet is implemented in the service method of a GenericServlet, in the do*Method* methods (where *Method* can take the value Get, Delete, Options, Post, Put, or Trace) of an HttpServlet object, or in any other protocol-specific methods defined by a class that implements the Servlet interface. The term **service method** is used for any method in a servlet class that provides a service to a client.

The general pattern for a service method is to extract information from the request, access external resources, and then populate the response, based on that information. For HTTP servlets, the correct procedure for populating the response is to do the following:

1. Retrieve an output stream from the response.
2. Fill in the response headers.
3. Write any body content to the output stream.

Response headers must always be set before the response has been committed. The web container will ignore any attempt to set or add headers after the response has been committed. The next two sections describe how to get information from requests and generate responses.

一个servlet提供服务实现了GenericServlet类中的服务方法（HttpServlet类继承了GenericServlet类）。在HttpServlet对象中doMethod方法中（method可以是Get,Delete,Options,Post,Put或者Trace），或者在其他协议(这里指的是http，tcp等协议，以为servlet可以处理几乎任何的请求)中定义的方法，这些方法从实现Servlet接口继承而来。术语**service method**指的是Servlet类中的任意方法，由这些方法来为client提供服务。

服务的一般编程模式是从requesth中获取信息，然后基于这些请求信息访问外部的资源然后做出响应。对于HTTP的Servlet来说，正确的响应请求是遵循以下流程：

1. 打开一个此响应的输出流
2. 填充响应头部
3. 先输出流中写入任何数据

注意：在响应提交之前必须设置好响应头。当响应提交之后，web容器会忽略一切响应头的添加和修改。

下面的两部分描述了如何从request获取信息以及如何产生response。

## 17.5.1 Getting Information from Requests

A request contains data passed between a client and the servlet. All requests implement the ServletRequest interface. This interface defines methods for accessing the following information:

* Parameters, which are typically used to convey information between clients and servlets
* Object-valued attributes, which are typically used to pass information between the web container and a servlet or between collaborating servlets
* Information about the protocol used to communicate the request and about the client and server involved in the request
* Information relevant to localization

You can also retrieve an input stream from the request and manually parse the data. To read character data, use the BufferedReader object returned by the request's getReader method. To read binary data, use the ServletInputStream returned by getInputStream.

从request获取请求信息：

一个request携带者client和servlet之前通信的数据，所有的request实现了ServletRequest接口，通过此接口中的方法可以访问以下信息：

1. 参数-典型的用于client和servlet之间转换数据
2. 对象值属性-典型的用于在web容器和servlet之间或者协同的servlets之间传递数据。
3. 协议信息-request的client和servlet之间交互使用的协议信息
4. 本地的某些信息

除了使用接口提供的方法外，你也可以通过解析request的输入流来手动获取这些相关信息。可以使用BufferedReader对象来读取字节流，或者通过ServletInputStream来获取二进制流。

HTTP servlets are passed an HTTP request object, HttpServletRequest, which contains the request URL, HTTP headers, query string, and so on. An HTTP request URL contains the following parts:

http://[*host*]:[*port*][*request-path*]?[*query-string*]

http servlet传递一个HTTP request对象给方法，HttpServletRequest包含了请求的url、请求头、请求参数等等。一个Http的请求的url包含以下部分：

[http://[*host*]:[*port*][*request-path*]?[*query-string*](http://[host]:[port][request-path]?[query-string)]、

The request path is further composed of the following elements.

* **Context path**: A concatenation of a forward slash (/) with the context root of the servlet's web application.
* **Servlet path**: The path section that corresponds to the component alias that activated this request. This path starts with a forward slash (/).
* **Path info**: The part of the request path that is not part of the context path or the servlet path.

You can use the getContextPath, getServletPath, and getPathInfo methods of the HttpServletRequest interface to access this information. Except for URL encoding differences between the request URI and the path parts, the request URI is always comprised of the context path plus the servlet path plus the path info.

Query strings are composed of a set of parameters and values. Individual parameters are retrieved from a request by using the getParameter method. There are two ways to generate query strings.

* A query string can explicitly appear in a web page.
* A query string is appended to a URL when a form with a GET HTTP method is submitted.

一个请求的path包含以下部分：

1. Context path：
2. Servlet path
3. Path info
4. 请求的字符串，显示的暴露在web页面中

## 17.5.2 Constructing Responses

A response contains data passed between a server and the client. All responses implement the ServletResponse interface. This interface defines methods that allow you to do the following.

* Retrieve an output stream to use to send data to the client. To send character data, use the PrintWriter returned by the response's getWriter method. To send binary data in a Multipurpose Internet Mail Extensions (MIME) body response, use the ServletOutputStream returned by getOutputStream. To mix binary and text data, as in a multipart response, use a ServletOutputStream and manage the character sections manually.
* Indicate the content type (for example, text/html) being returned by the response with the setContentType(String) method. This method must be called before the response is committed. A registry of content type names is kept by the Internet Assigned Numbers Authority (IANA) at <http://www.iana.org/assignments/media-types/>.
* Indicate whether to buffer output with the setBufferSize(int) method. By default, any content written to the output stream is immediately sent to the client. Buffering allows content to be written before anything is sent back to the client, thus providing the servlet with more time to set appropriate status codes and headers or forward to another web resource. The method must be called before any content is written or before the response is committed.
* Set localization information, such as locale and character encoding. See [Chapter 20, "Internationalizing and Localizing Web Applications"](https://docs.oracle.com/javaee/7/tutorial/webi18n.htm#BNAXU) for details.

HTTP response objects, javax.servlet.http.HttpServletResponse, have fields representing HTTP headers, such as the following.

* Status codes, which are used to indicate the reason a request is not satisfied or that a request has been redirected.
* Cookies, which are used to store application-specific information at the client. Sometimes, cookies are used to maintain an identifier for tracking a user's session (see [Session Tracking](https://docs.oracle.com/javaee/7/tutorial/servlets009.htm#BNAGR)).