So far, we learned how to install Apache Tomcat server and how to deploy Java web application to it. We briefly discussed the concept of Java web applications components, including servlets and JSPs.

Before we go into more depth explaining Tomcat features, we need to introduce the concepts, components, and syntax of main Java web application components—servlets, JSPs, and their relationship to ServletContext. We won’t cover all the details about these broad subjects, but you should gain enough understanding to follow the examples in the chapters that follow, in which we explore the Tomcat server specific functionalities.

If you would like to read more about these technologies, visit the Java web site (www.oracle.com/technetwork/java/index.html).

In this chapter we:

- Describe the Java servlet architecture and its core components
- Describe the JSP architecture with syntax introduction
- Define the ServletContext and its relationship to web applications

**Servlets**

A Java servlet is a platform-independent web application component that is hosted in a servlet container. Servlets communicate with web clients using a request/response model managed by a servlet container, such as Apache Tomcat. When the user submits a request to the server (i.e., clicks on the link in the browser), the servlet container will accept the request. The servlet container then checks the servlets deployed on it, locates the one that should handle the incoming request (based on the request URL and servlet mapping), and sends the request to it, wrapped as an instance of the ServletRequest class. After processing the request, the servlet produces a response in the form of a ServletResponse instance, and sends it back to the servlet container. Finally, the servlet container sends the response to the calling client, resulting in the web page rendered in the client browser. Figure 3-1 depicts the execution of a Java servlet.
The servlet architecture comprises two Java packages: `javax.servlet` and `javax.servlet.http`. The `javax.servlet` package contains the generic interfaces and classes that are implemented and extended by all servlets. The second package is the `javax.servlet.http` package, which contains all the servlet classes that are specific to HTTP, such as a simple servlet that responds using HTML.

At the heart of this architecture is the interface `javax.servlet.Servlet`. The base class for all servlets, the `Servlet` interface, defines five methods. The three most important of these methods and their functions are:

- the `init()` method, which initializes the servlet;
- the `service()` method, which services client requests; and
- the `destroy()` method, which performs cleanup.

These methods make up the servlet lifecycle methods. (We describe these lifecycle methods in the next section.) All servlets must implement this interface, either directly or through inheritance.

The Lifecycle of a Servlet

The lifecycle of a Java servlet follows a very logical sequence. The interface that declares the lifecycle methods is the `javax.servlet.Servlet` interface. These methods are the `init()`, `service()`, and `destroy()` methods. This sequence can be described in a simple three-step process:

1. A servlet is loaded and initialized using the `init()` method. This method is called when the servlet is reloaded or upon the first request to this servlet.
2. The servlet then services zero or more requests. The servlet services the request using the `service()` method.
3. The servlet is then destroyed and garbage-collected when the web application containing the servlet shuts down. The method that is called upon shutdown is the `destroy()` method.

Now, let's look at each of these methods.

**init**

The `init()` method is where the servlet begins its life. This method is called immediately after the servlet is instantiated, and it is called only once. The `init()` method should be used to create and initialize the