Java jako język programowania serwerów WWW / aplikacji Webowych – servlety

Robert A. Kłopotek
r.klopotek@uksw.edu.pl

Wydział Matematyczno-Przyrodniczy. Szkoła Nauk Ścisłych, UKSW

25.05.2017

Java Servlet

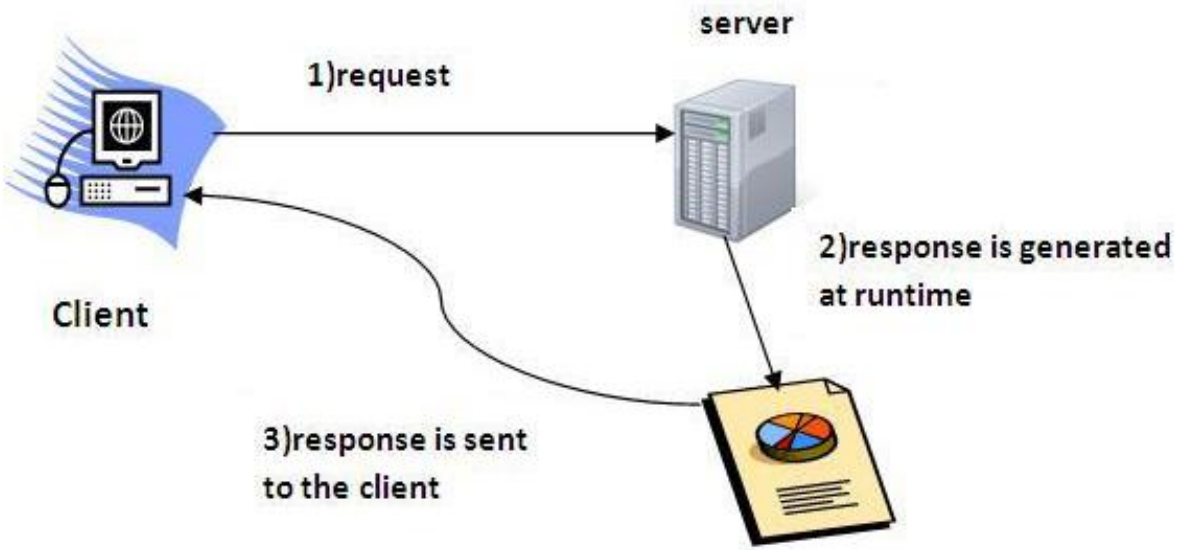
- Technologia serwletu służy do tworzenia aplikacji internetowych (znajduje się po stronie serwera i generuje dynamiczną stronę internetową).
- Technologia serwletu jest solidna i skalowalna z powodu języka java. Przed Servletami, język skryptowy CGI (Common Gateway Interface) był popularny jako język programowania po stronie serwera. Było jednak wiele wad tej technologii.
- W interfejsie API serwletu istnieje wiele klas i interfejsów, takich jak Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse itp.

Co to jest Servlet?

- Servlet jest technologią np. wykorzystywaną do tworzenia aplikacji sieci web.
- Servlet jest interfejsem API zawierającym wiele interfejsów i klas, w tym dokumentacji.
- Servlet jest interfejsem, który musi być zaimplementowany w celu utworzenia dowolnego serwletu.
- Servlet jest klasą, która rozszerza możliwości serwerów i odpowiada na przychodzące żądanie. Może odpowiadać na każdy rodzaj żądań.
- Servlet to składnik sieci Web, który jest instalowany na serwerze w celu utworzenia dynamicznej strony internetowej.

Co to jest aplikacja webowa/internetowa?

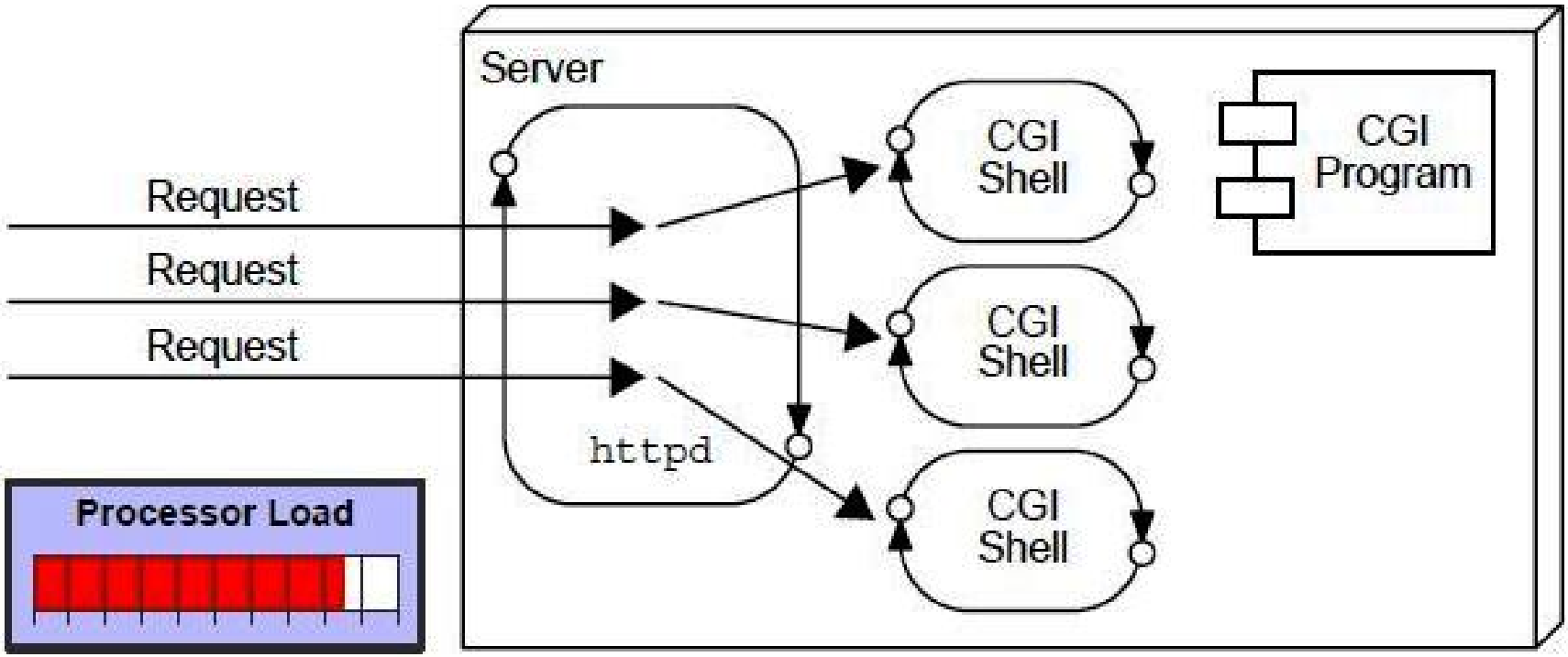
- Aplikacja internetowa to aplikacja dostępna w internecie.
- Aplikacja internetowa składa się z komponentów internetowych, takich jak Servlet, JSP, Filter itp. oraz innych komponentów takich jak HTML.
- Komponenty webowe zazwyczaj są uruchamiane w serwerze sieci Web i odpowiadają na żądanie HTTP.



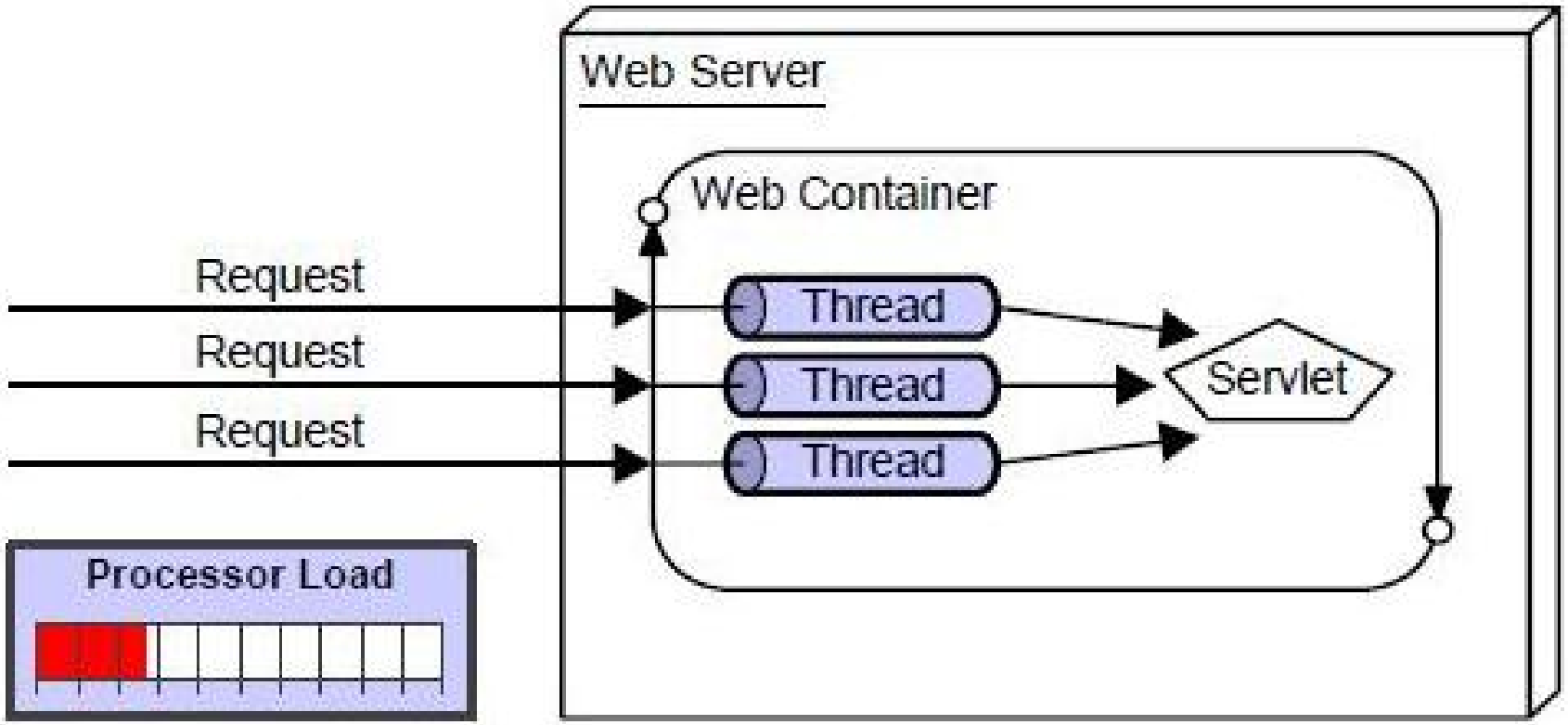
CGI (Commmon Gateway Interface)

- Technologia CGI umożliwia serwerowi WWW wywołanie zewnętrznego programu i przekazanie informacji o żądaniu HTTP do zewnętrznego programu w celu przetworzenia żądania.
- Dla każdego żądania rozpoczyna nowy proces.
- Wady:
  - Jeśli liczba klientów wzrasta, zajmuje więcej czasu na wysłanie odpowiedzi.
  - Dla każdego żądania rozpoczyna proces, a serwer sieci Web jest ograniczony do uruchamiania procesów.
  - Używa języka zależnego od platformy, np. C, C ++, perl

CGI - przetwarzanie żądań



Java Servlet - przetwarzanie żądań



Zalety servletu

- Istnieje wiele zalet serwletu nad CGI. Kontener internetowy tworzy wątki dla obsługi wielu żądań serwletu.
- Wątki mają wiele zalet w porównaniu z procesami, takie jak wspólne dzielenie pamięci, lekki, kosztu komunikacji między wątkami są niskie.
- Podstawowe zalety serwletu są następujące:
  - Lepsza wydajność: ponieważ tworzy wątek dla każdego żądania nie przetwarza
  - Przenośność: ponieważ używa języka java.
  - Solidne: serwlety są zarządzane przez JVM, więc nie musimy się martwić o wycieki pamięci, odśmiecanie itp.
  - Bezpieczne: ponieważ używa języka Java.

Strona statyczna vs dynamiczna

1. Zawartość wstępna jest taka sama za każdym razem, gdy strona jest załadowana.
2. Używa kodu HTML do opracowania witryny.
3. Wysyła dokładnie taką samą odpowiedź dla każdego żądania.
4. Zawartość zmienia się tylko wtedy, gdy ktoś publikuje i aktualizuje plik (wysyła go do serwera WWW).
5. Elastyczność jest główną zaletą statycznej witryny sieci Web.

1. Treść jest generowana szybko i regularnie się zmienia.
2. Do tworzenia witryny używa po stronie serwera języków takich jak PHP, SERVLET, JSP i ASP.NET itp.
3. Może generować inny kod HTML dla każdego z żądań.
4. Strona zawiera kod "po stronie serwera", który pozwala serwerowi generować niepowtarzalną treść po załadowaniu strony.
5. System zarządzania treścią (Content Management System - CMS) jest główną zaletą strony dynamicznej.

Podstawowe cechy HTTP (Hyper Text Transfer Protocol)

- HTTP jest niezależne od mediów: odnosi się do dowolnego typu treści multimedialnej przez HTTP, o ile zarówno serwer, jak i klient mogą obsługiwać zawartość danych.
- HTTP jest bezpołączeniowy (connectionless): jest to podejście bez połączenia, w którym klient HTTP, czyli przeglądarka inicjuje żądanie HTTP, a po wysłaniu żądania klient odłącza się od serwera i oczekuje na odpowiedź.
- HTTP jest bezstanowe: klient i serwer są świadomi siebie tylko w bieżącym żądaniu. Potem oboje zapominają się nawzajem. Ze względu na bezstanowość protokołu ani klient, ani serwer nie mogą przechowywać informacji o różnych żądaniach na stronach internetowych.

Żądania HTTP

- GET - Pozwala uzyskać zasób w żądanym adresie URL.
- POST - Pozwala serwerowi zaakceptować dołączone informacje o ciele. To jest podobne do żądania GET z dodatkowymi informacjami wysłanymi z żądaniem.
- HEAD - Zapytanie tylko o nagłówek, niezależnie od tego, czy GET wróci. To jest podobne do żądania GET, ale bez ciała.
- TRACE - Zapytanie o pętlę zwrotną komunikatu żądania, do testowania lub rozwiązywania problemów.
- PUT - mówi, aby umieścić załączone informacje (ciało) na żądany adres URL.
- DELETE - mówi, aby usunąć zasób pod żądanym adresem URL.
- OPTIONS - Pozwala wyświetlić listę metod HTTP, do których obiekt może odpowiadać na żądanie adresu URL

Idempotentność

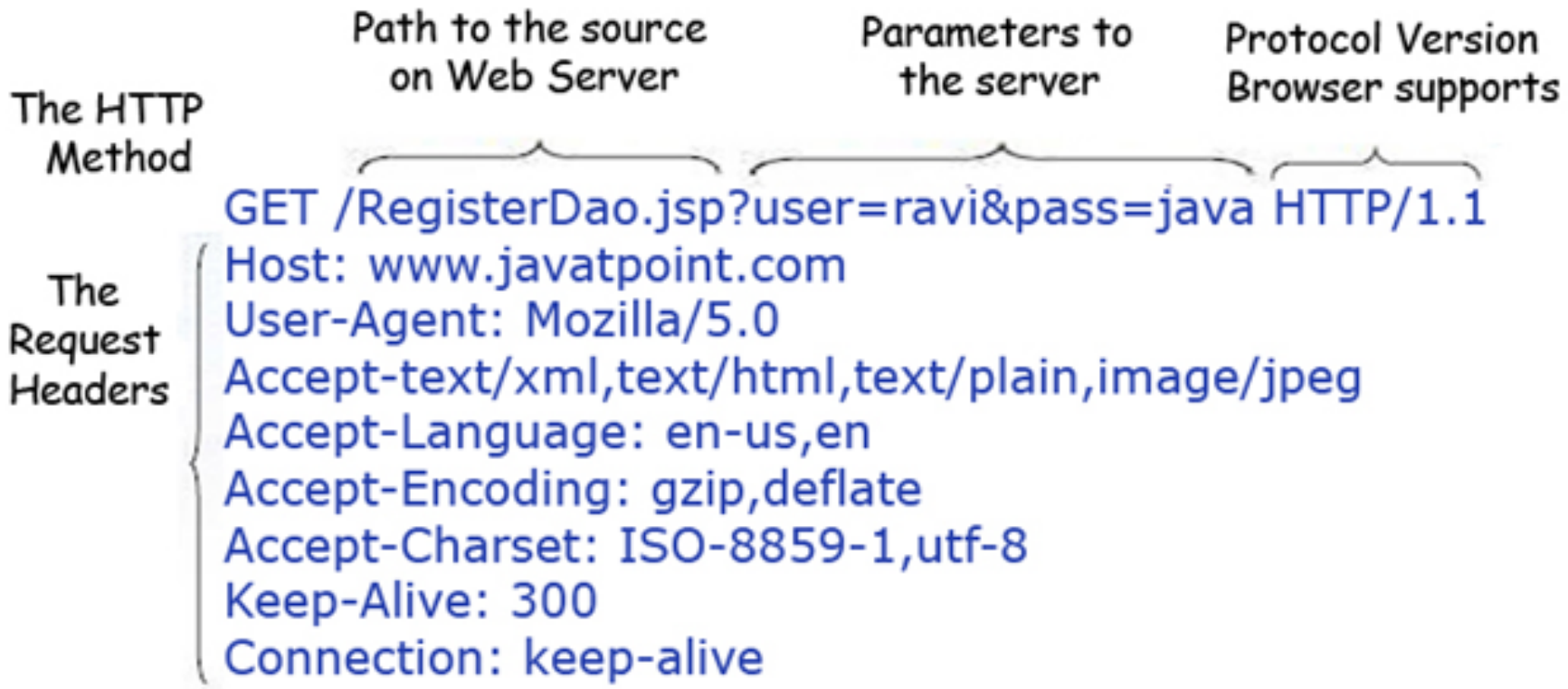
- W informatyce idempotentność jest własnością operacji pozwalającą na jej wielokrotne powtarzanie bez zmiany wyniku lub powodowania błędu
- Programista aplikacji internetowych powinien zadbać o idempotentność wykonywanych przez serwer operacji, nie dopuszczając np. do kolejnego zakupu identycznego wyrobu w sklepie internetowym po odświeżeniu strony (token synchronizujący)
- Standardowo uważa się metody GET i HEAD protokołu HTTP za idempotentne, więc przeglądarki internetowe nie wyświetlają żadnego ostrzeżenia w przypadku odświeżania strony za pomocą metody GET. Stąd poleca się implementację operacji zmieniających stan sesji klienta za pomocą metody POST.

GET vs POST

1. W przypadku żądania GET, jedynie ograniczona ilość danych może zostać wysłana, ponieważ dane są przesyłane w nagłówku.
2. Żądanie GET nie jest zabezpieczone, ponieważ dane są wyświetlane w pasku adresu URL.
3. Żądanie GET można zapisać w zakładce.
4. Żądanie GET jest idempotentne. Oznacza to, że drugie żądanie zostanie zignorowane do czasu dostarczenia odpowiedzi na pierwsze żądanie.
5. Żądanie GET jest bardziej wydajne i stosowane częściej niż POST.

1. W przypadku żądania POST, duża ilość danych może zostać wysłany, ponieważ dane są przesyłane w ciele.
2. Żądanie POST jest zabezpieczone, ponieważ dane nie są wyświetlane w pasku adresu URL.
3. Żądanie POST nie może być zakładką.
4. Żądanie POST nie jest idempotentne.
5. Żądanie POST jest mniej wydajne i rzadziej używane niż GET.

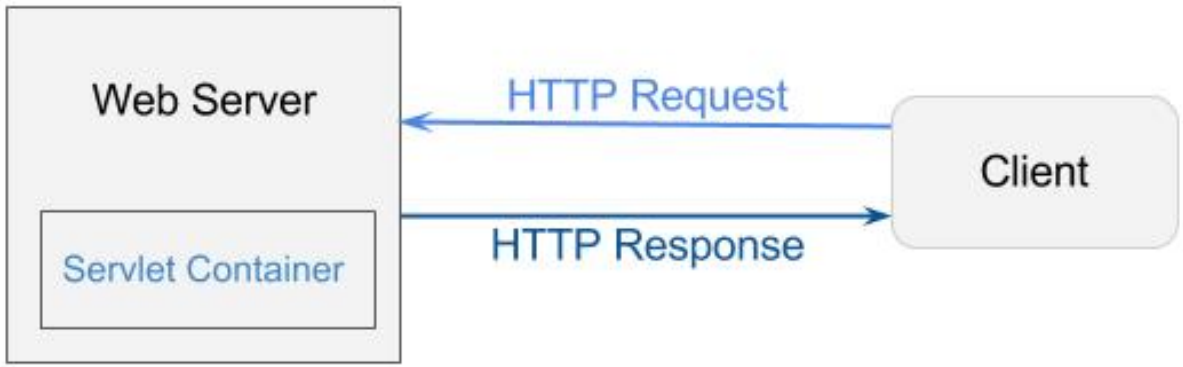
GET - przykład



POST - przykład

Kontener serwletów

- Zapewnia środowisko wykonawcze aplikacji JavaEE (j2ee)
- Kontener serwletu jest używany w Java do dynamicznego generowania stron internetowych po stronie serwera.
- Kontener serwletu jest częścią serwera sieciowego, który współdziała z serwletem w celu obsługi dynamicznych stron internetowych od klienta.



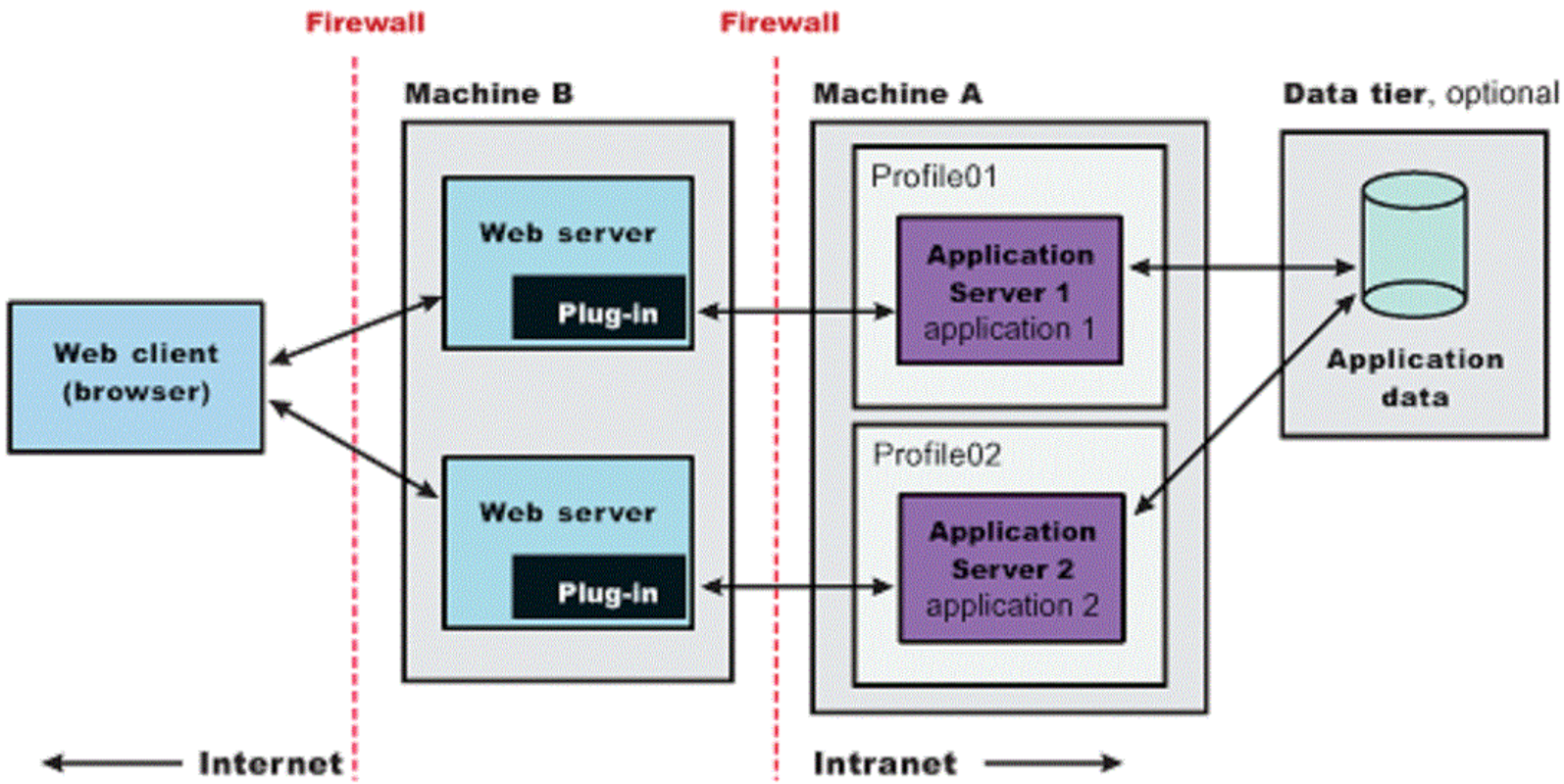
Stany kontenera serwletów

- Standalone: Typowe serwery Java, w których kontener serwletu i serwery WWW stanowią integralną część jednego programu. Na przykład: - Tomcat działa sam
- In-process: jest oddzielony od serwera WWW, ponieważ inny program działa w przestrzeni adresowej głównego serwera jako wtyczka. Na przykład: - Tomcat działający wewnątrz JBossa.
- Out-of-process: serwer WWW i kontenerem serwletów są różnymi programami, które są w różnych procesach. Aby komunikacja między nimi była możliwa, serwer WWW używa wtyczki dostarczonego przez kontener serwletów.

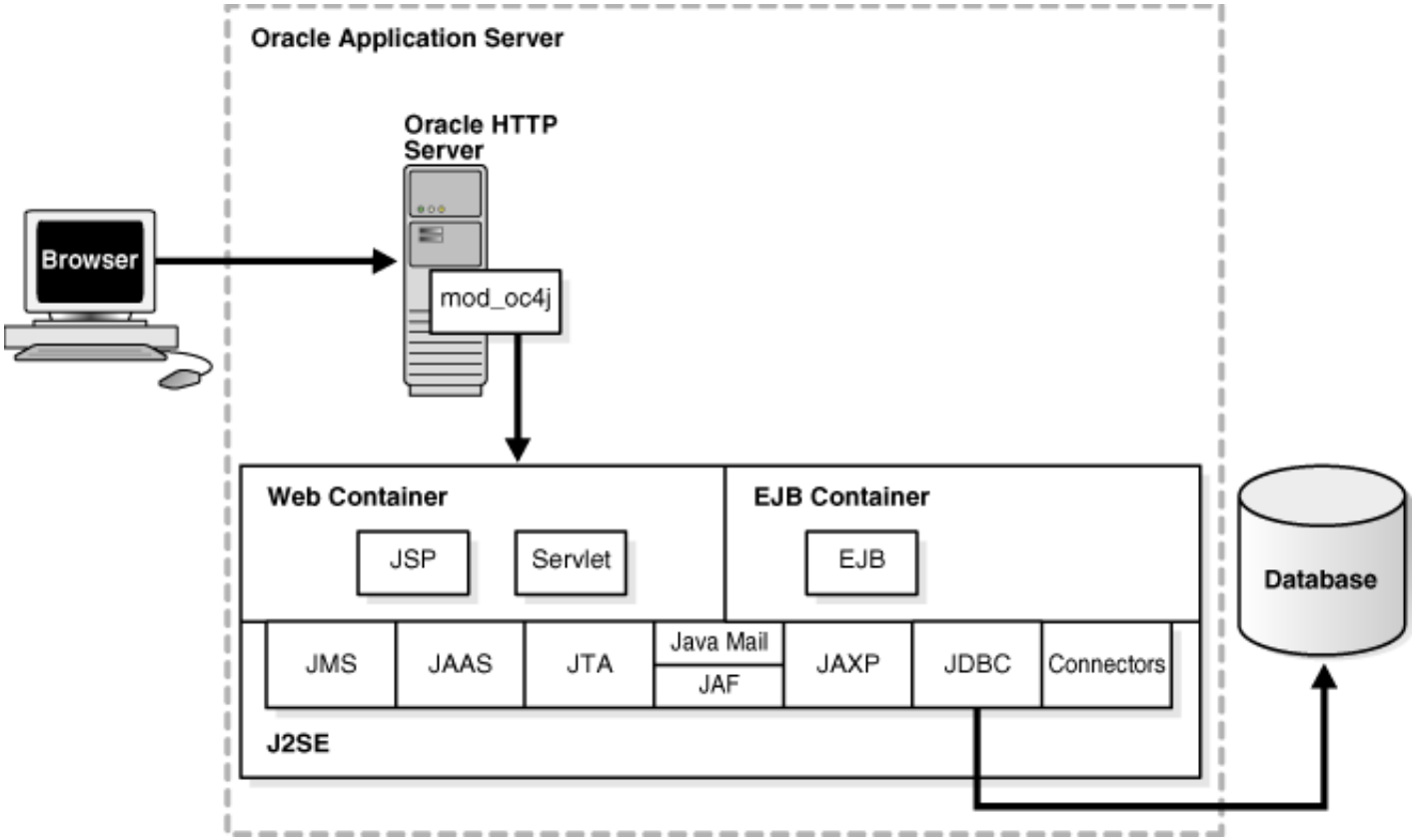
Serwer webowy vs serwer aplikacji

- Serwer sieci Web ma służyć do obsługi zawartości HTTP. Serwer aplikacji może również obsługiwać zawartość HTTP, ale nie jest ograniczony do tylko HTTP. Może być dostarczony z innym protokołem, takim jak RMI / RPC
- Serwer sieci Web jest przeznaczony głównie do obsługi zawartości statycznych, chociaż większość serwerów sieci Web ma wtyczki obsługujące języki skryptowe, takie jak Perl, PHP, ASP, JSP itp. Dzięki temu serwery mogą generować dynamiczną zawartość HTTP.
- Większość serwerów aplikacji ma serwer WWW jako integralną część, co oznacza, że serwer aplikacji może wykonywać dowolny serwer WWW. Większość serwerów aplikacji zawiera komponenty i funkcje umożliwiające obsługę usług na poziomie aplikacji, takich jak pulę połączeń, pulę obiektów, obsługa transakcji, usługi obsługi wiadomości itp.

Integracja serwerów



Oracle Application Server Containers

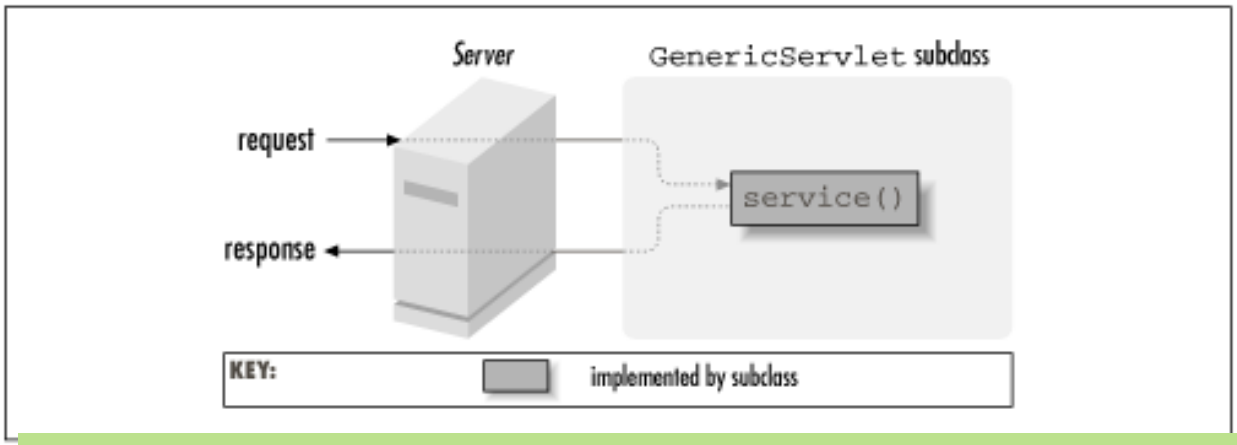


Servlet API

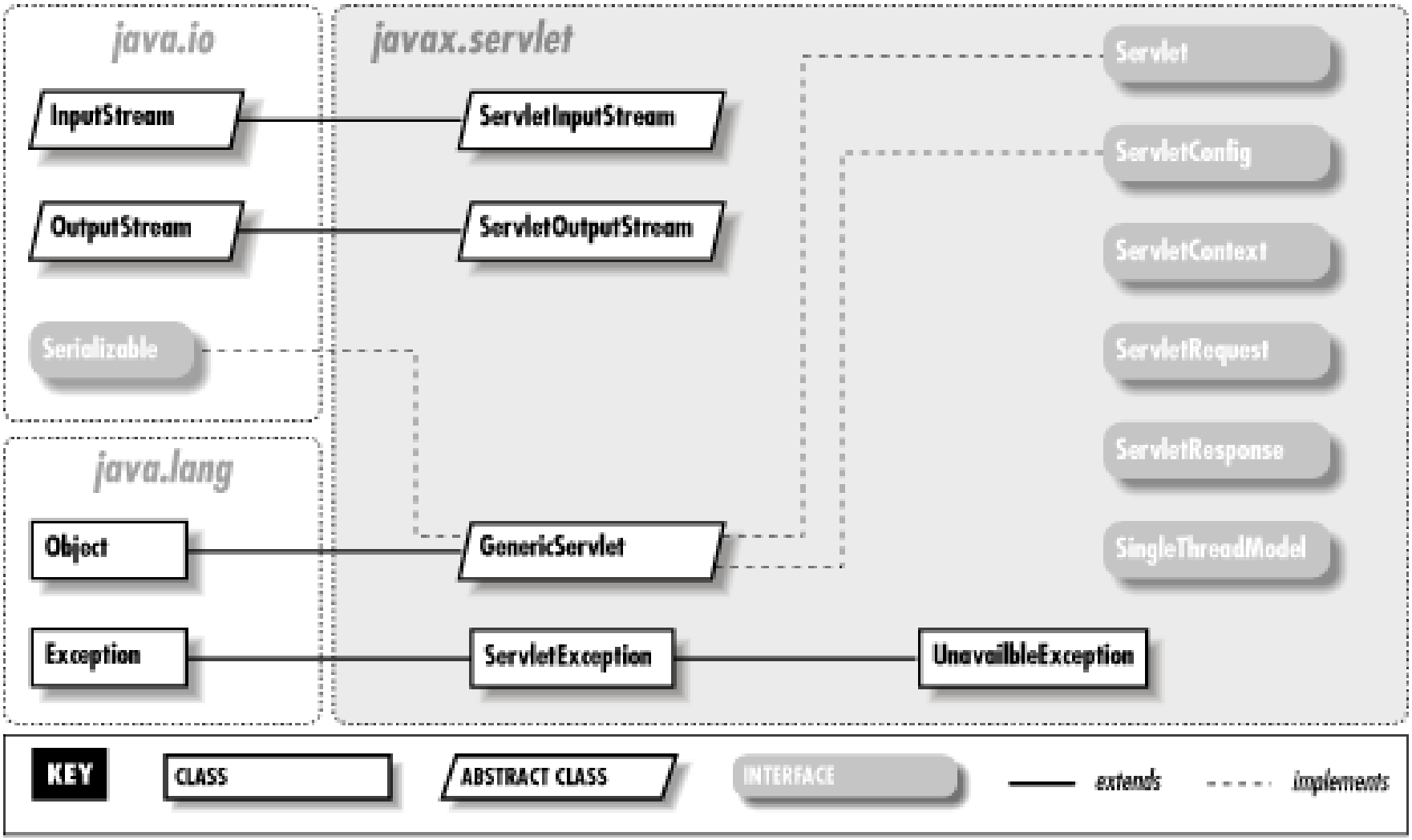
- Pakiety javax.servlet i javax.servlet.http reprezentują interfejsy i klasy dla api serwletu.
- Pakiet javax.servlet zawiera wiele interfejsów i klas używanych przez serwlet lub kontener WWW. Nie są specyficzne dla żadnego protokołu.
- Pakiet javax.servlet.http zawiera interfejsy i klasy, które są odpowiedzialne tylko za żądania http.

Pakiet javax.servlet

- Pakiet javax.servlet stanowi podstawę API Servlet.
- Zawiera podstawowy interfejs Servlet, który wszystkie serwlety muszą implementować w jednej lub innej formie, oraz abstrakcyjną klasę GenericServlet w celu opracowania podstawowych serwletów.
- Pakiet zawiera również klasy do komunikacji z serwerem hosta i klientem (ServletRequest i ServletResponse) i komunikowania się z klientem (ServletInputStream i ServletOutputStream).
- Serwlety powinny ograniczyć się do klas tego pakietu w sytuacjach, w których nieznany jest protokół bazowy.

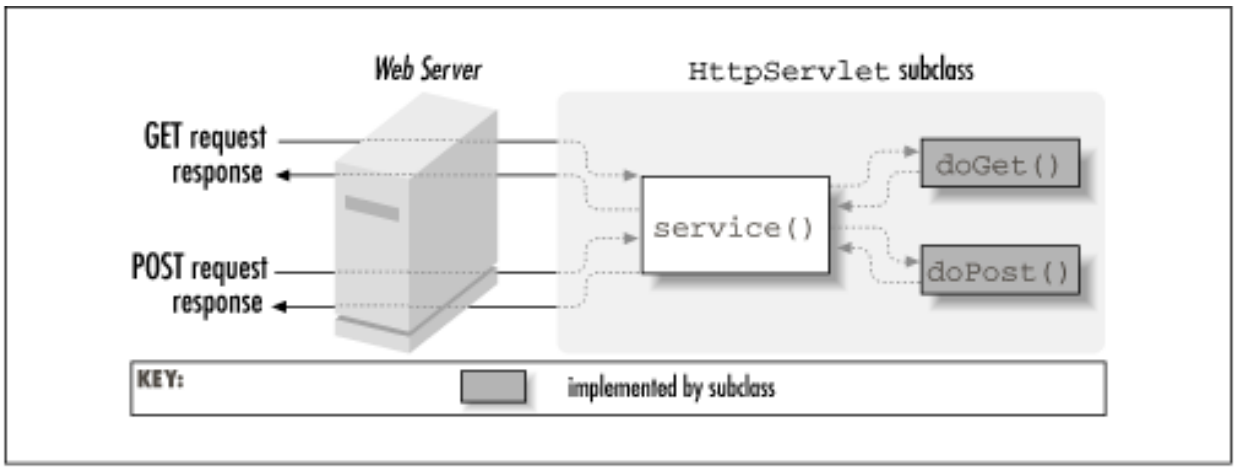


Hierarchia javax.servlet

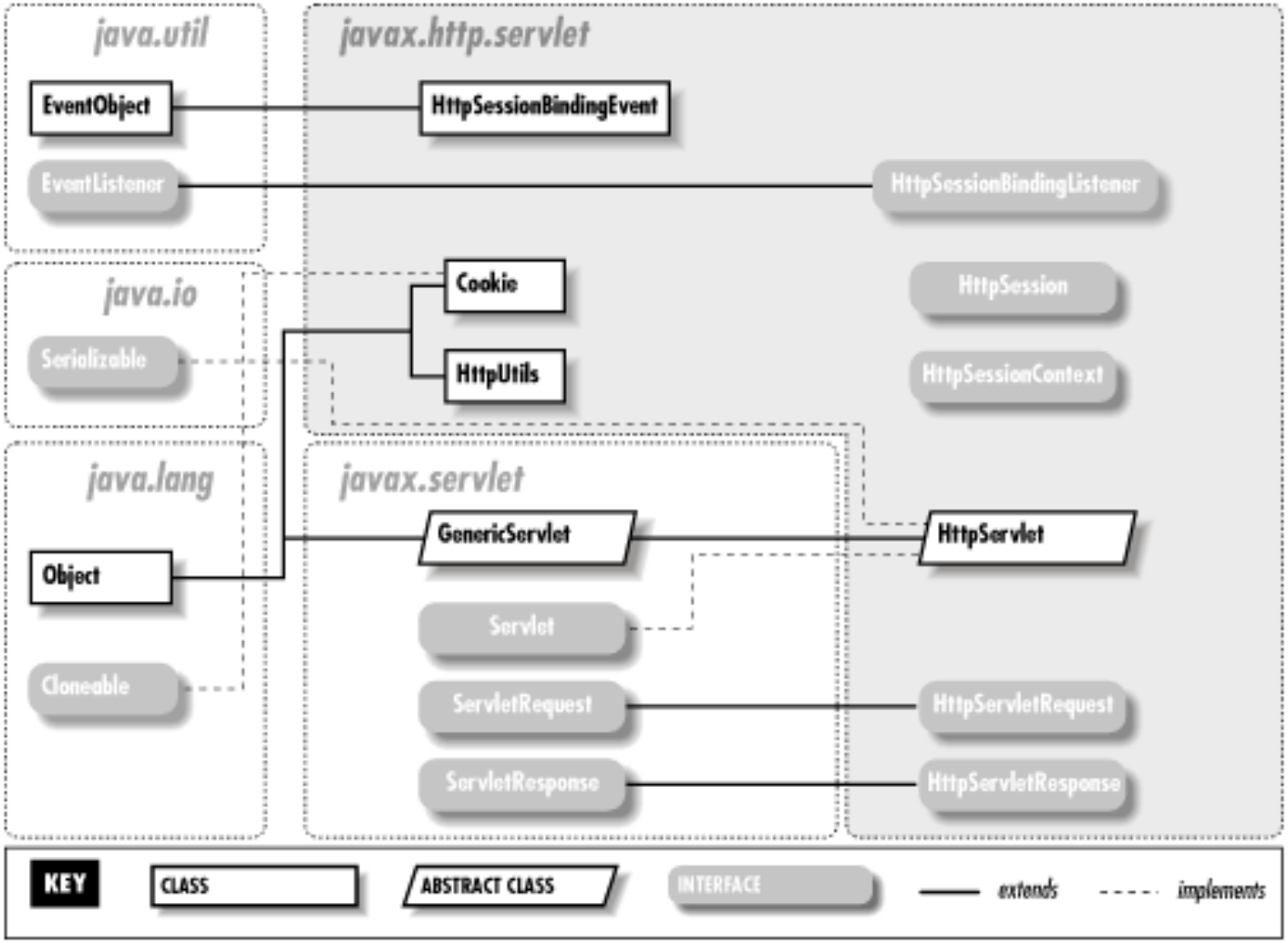


Pakiet javax.servlet.http

- Pakiet javax.servlet.http umożliwia tworzenie serwletów obsługujących protokół HTTP.
- Chociaż podstawowa funkcjonalność pakietu javax.servlet zapewnia wszystko, co niezbędne do tworzenia stron WWW, wyspecjalizowane klasy w tym pakiecie automatyzują wiele innych zadań.
- Abstrakcyjna klasa HttpServlet obejmuje obsługę różnych metod żądania HTTP i nagłówków.
- Interfejsy HttpServletRequest i HttpServletResponse umożliwiają dodatkową interakcję z serwerem internetowym, podczas gdy HttpSession zapewnia wbudowaną funkcję śledzenia sesji.
- Klasa Cookie umożliwia szybkie konfigurowanie i przetwarzanie plików cookie HTTP, a klasa HttpUtils robi to samo dla łańcuchów zapytań.



Hierarchia javax.servlet.http



Cykl życia serwletu

1. Klasa serwletu jest ładowana - Classloader jest odpowiedzialny, aby załadować klasę serwletu. Klasa serwletu jest ładowana, gdy pierwsze żądanie serwletu jest odbierane przez kontener internetowy.
2. Instancja serwletu jest tworzona - Kontener internetowy tworzy instancję serwletu po załadowaniu klasy serwletu. Instancja serwletu jest tworzona tylko raz w cyklu życia serwletu.
3. Metoda init() jest wywoływana - Kontener internetowy wywołuje metodę public void init(ServletConfig config) tylko raz po utworzeniu instancji serwletu. Metoda init jest używana do inicjowania serwletu.
4. Metoda service() jest wywoływana - Kontener internetowy wywołuje metodę public void service(ServletRequest request, ServletResponse response) przy każdym odebraniu żądania dotyczącego serwletu
5. Metoda destroy() jest wywoływana - Kontener internetowy wywołuje metodę public void destroy() przed usunięciem instancji serwletu z usługi. Daje serwletowi możliwość oczyszczenia dowolnego zasobu na przykład pamięci, wątku itp.

Jak działa serwlet?

- Serwer sprawdza, czy serwlet jest żądany po raz pierwszy.
- Jeśli tak, kontener internetowy wykonuje następujące zadania:
  - Ładuje klasę serwletu.
  - Instancjuje klasę serwletu.
  - Wywołuje metodę init przekazującą obiekt ServletConfig
- W przeciwnym wypadku:
  - Wywołuje metodę obsługi żądania i obiektów odpowiedzi
- Kontener internetowy wywołuje metodę destroy(), gdy jest potrzebuje usunąć serwlet, na przykład w momencie zatrzymania serwera lub rozwinięcia projektu.

Servlet - prosty przykład

```
import java.io.*;
import javax.servlet.*;
public class FirstServlet implements Servlet{
  ServletConfig config=null;

  public void init(ServletConfig config){
    this.config=config;
    System.out.println("servlet is initialized");
  }
  public void service(ServletRequest req,ServletResponse res)
                          throws IOException,ServletException{
    res.setContentType("text/html");
    PrintWriter out=res.getWriter();
    out.print("<html><body>");
    out.print("<b>hello simple servlet</b>");
    out.print("</body></html>");
  }
  public void destroy(){System.out.println("servlet is destroyed");}
  public ServletConfig getServletConfig(){return config;}
  public String getServletInfo(){return "copyright 2017";}
}
```

Servlet - przykład klasy uruchamiającej

```
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.servlet.ServletContextHandler;
import org.eclipse.jetty.servlet.ServletHolder;

public class FirstServletRunner {
  public static void main(String[] args) throws Exception{
    Server server = new Server(8080);

    ServletContextHandler context =
        new ServletContextHandler(ServletContextHandler.SESSIONS);

    context.setContextPath("/");
    server.setHandler(context);

    context.addServlet(new ServletHolder(new FirstServlet()),"/*");

    server.start();
    server.join();
 }
}
```

HttpServlet - prosty przykład

```
import java.io.*;
import javax.servlet.*;
public class HelloServlet extends HttpServlet
{
    private String greeting="Hello␣World";
    public HelloServlet(){}

    public HelloServlet(String greeting){
        this.greeting=greeting;
    }
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                  throws ServletException, IOException{
      response.setContentType("text/html");
      response.setStatus(HttpServletResponse.SC_OK);
      response.getWriter().println("<h1>"+greeting+"</h1>");
      response.getWriter().println("session="
                        + request.getSession(true).getId());
    }
}
```

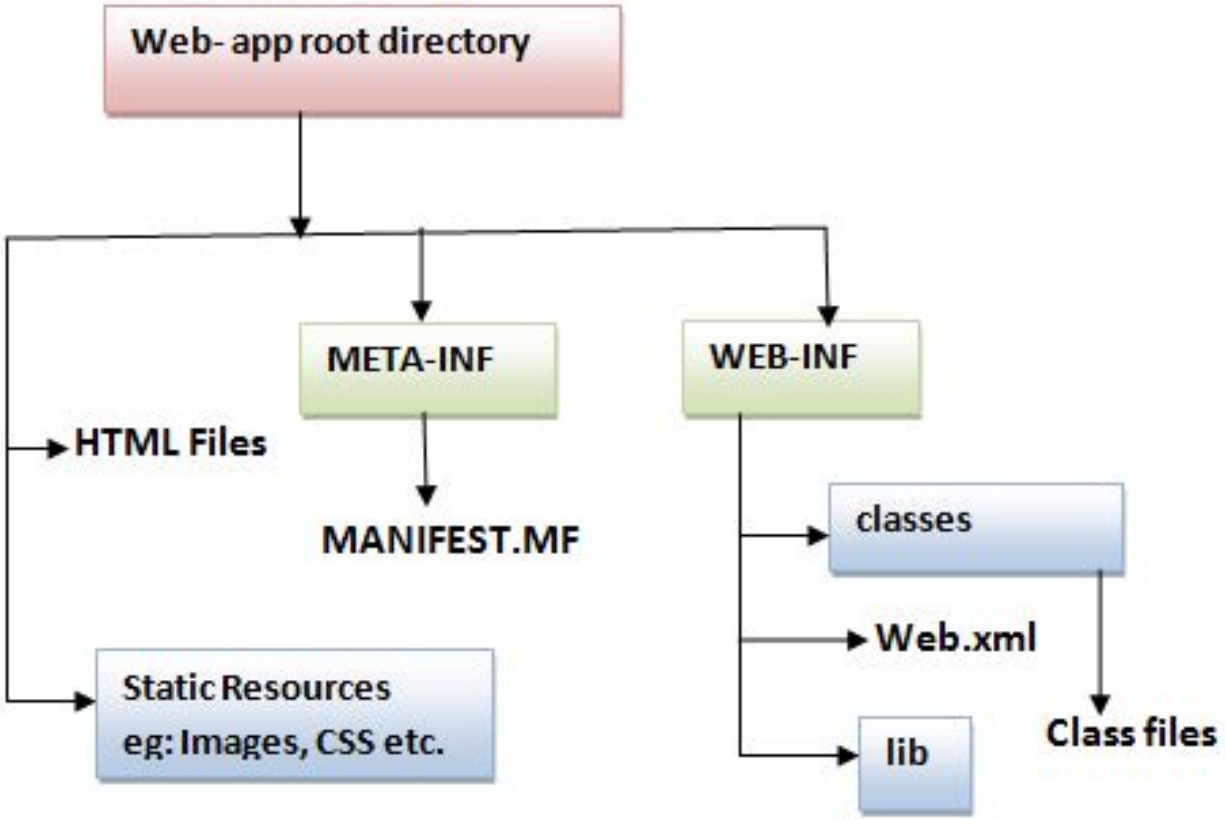
HttpServlet - przykład klasy uruchamiającej

```
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.servlet.ServletContextHandler;
import org.eclipse.jetty.servlet.ServletHolder;
public class OneServletContext
{
  public static void main(String[] args) throws Exception{
    Server server = new Server(8080);
    ServletContextHandler context =
        new ServletContextHandler(ServletContextHandler.SESSIONS);

    context.setContextPath("/");
    server.setHandler(context);

    context.addServlet(new ServletHolder(new HelloServlet()),"/*");
    context.addServlet(new ServletHolder(
                    new HelloServlet("Buongiorno␣Mondo")),"/it/*");
    context.addServlet(new ServletHolder(
                    new HelloServlet("Bonjour␣le␣Monde")),"/fr/*");
    server.start();
    server.join();
  }
}
```

Serwlety - tworzenie od podstaw

- Utwórz strukturę katalogów
- Utwórz serwlet
- Skompiluj serwlet
- Utwórz deskryptor wdrażania
- Uruchom serwer i wdróż projekt
- Uzyskaj dostęp do serwletu

Struktura katalogów



Tworzenie i kompilacja serwletu

- Istnieją trzy sposoby tworzenia serwletu.
  - Poprzez implementację interfejsu Servlet
  - Poprzez dziedziczenie klasy GenericServlet
  - Poprzez dziedziczenie klasy HttpServlet
- Klasa HttpServlet jest powszechnie używana do tworzenia serwletu, ponieważ udostępnia metody obsługiwania żądań HTTP, takich jak doGet (), doPost, doHead () itp.
- Aby skompilować serwlet należy załączyć plik JAR dla odpowiedniej platformy uruchomieniowej (serwera aplikacji), np. servlet-api.jar dla Apache Tomcat

HttpServlet - zaawansowany przykład

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

public class DemoServ extends HttpServlet{
  public void doGet(HttpServletRequest req,
                    HttpServletResponse res)
                    throws ServletException,IOException{

    res.setContentType("text/html");
    PrintWriter pw=res.getWriter();
    String name=req.getParameter("name");
    pw.println("Welcome␣"+name);
  }
}
```

index.html - przykład

```
<html><body>

<form action="welcome" method="get">
Enter your name<input type="text" name="name"><br>


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

Tworzenie deskryptora wdrażania (plik web.xml)

- Deskryptor wdrażania jest plikiem xml, z którego Web Container uzyskuje informacje dotyczące wywoływanego serwletu.
- Kontener internetowy używa parsera, aby uzyskać informacje z pliku web.xml. Istnieje wiele parserów XML, takich jak SAX, DOM i Pull.
- Elementy pliku web.xml:
  - `<web-app>` reprezentuje całą aplikację.
  - `<servlet>` to podczęść `<web-app>` i reprezentuje serwlet.
  - `<servlet-name>` jest podrzędnym elementem `<servlet>` reprezentuje nazwę serwletu.
  - `<servlet-class>` jest podrzędnym elementem `<servlet>` reprezentuje klasę serwletu.
  - `<servlet-mapping>` to podrzędny element `<web-app>`. Służy do mapowania serwletu.
  - `<url-pattern>` jest elementem podrzędnym `<servlet-mapping>`. Ten wzorzec jest używany po stronie klienta do wywołania serwletu.

Plik web.xml - przykład

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>

<servlet>
<servlet-name>MojeDemo</servlet-name>
<servlet-class>DemoServ</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>MojeDemo</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

Wdrażanie projektu - tworzenie pliku WAR

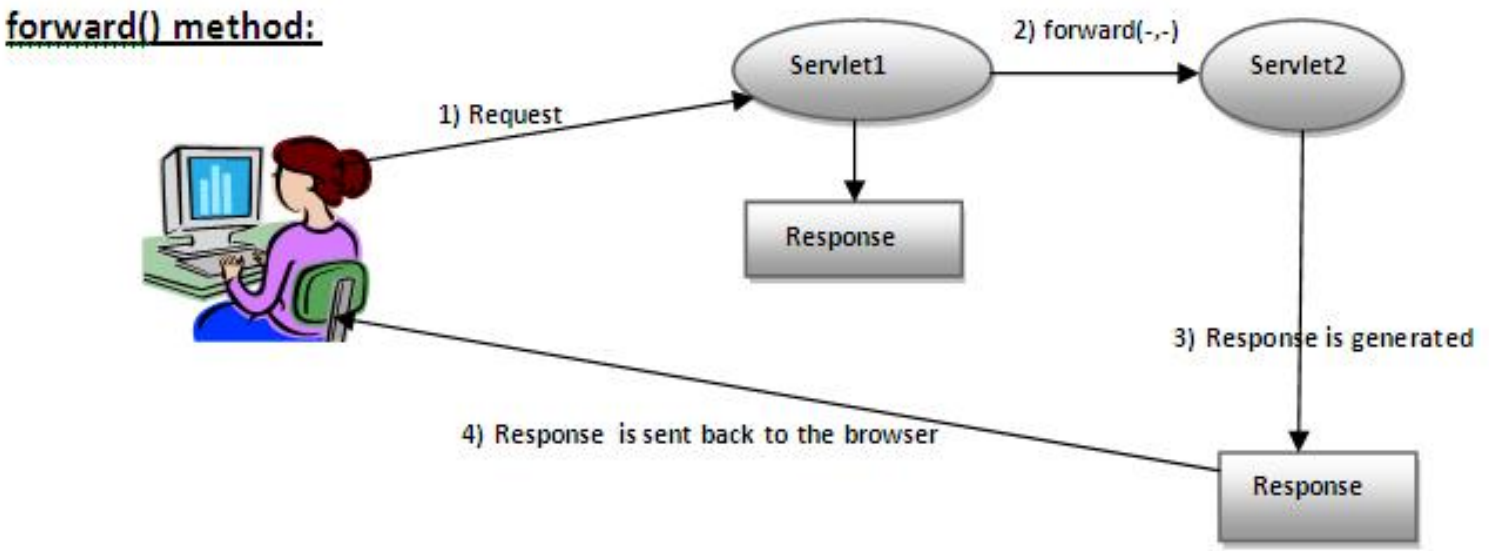
- Plik WAR (web archive) zawiera pliki projektu sieciowego. Może mieć pliki servletu, xml, jsp, obrazki, html, css, js itp.
- Oszczędza czas: plik WAR łączy wszystkie pliki w jedną jednostkę, przez co transfer plików z klienta na serwer zajmuje mniej czasu.
- Jak stworzyć plik WAR?
  - `jar -cvf projectname.war *`
  - za pomocą Eclipse EE (Export->War file)

Wdrażanie na serwer i jego uruchomienie

```
import org.eclipse.jetty.webapp.WebAppContext;
import org.eclipse.jetty.server.Server;

public class OneWebApp
{
  public static void main(String[] args) throws Exception{
    String jetty_home = System.getProperty("jetty.home",".");

    Server server = new Server(8080);

    WebAppContext webapp = new WebAppContext();
    webapp.setContextPath("/test");
    webapp.setWar(jetty_home+"/webapps/ServletDemo.war");

    server.setHandler(webapp);

    server.start();
    server.join();
  }
}
```

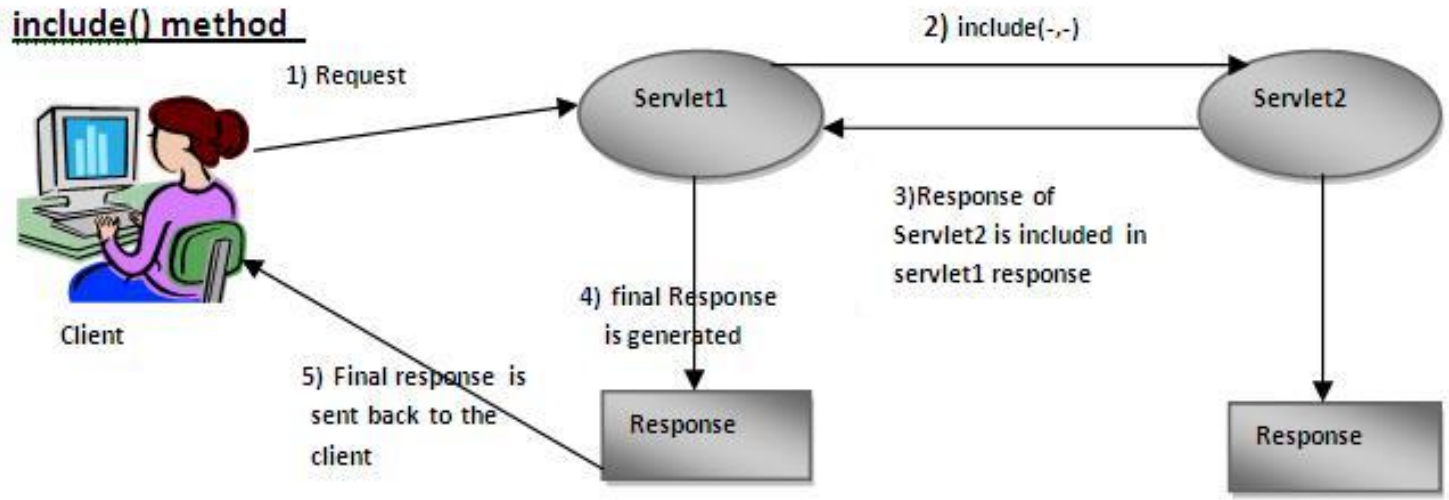
RequestDispatcher - forward()

public void forward(ServletRequest request,ServletResponse response) throws ServletException, java.io.IOException

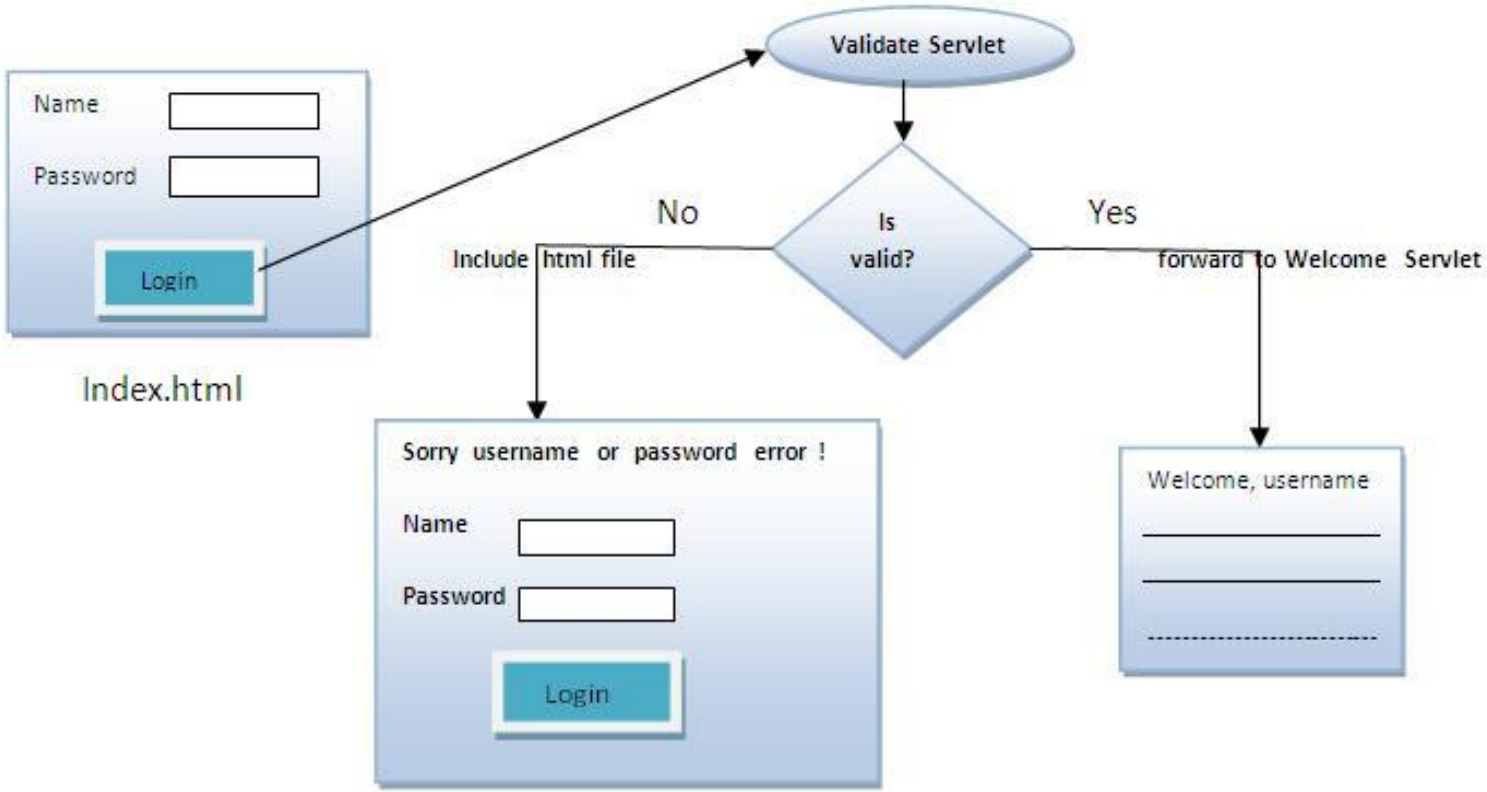


RequestDispatcher - include()

public void include(ServletRequest request,ServletResponse response) throws ServletException, java.io.IOException



RequestDispatcher - przykład



RequestDispatcher - przykład index.html

```
<form action="servlet1" method="post">
  Name:<input type="text" name="userName"/><br/>
  Password:<input type="password" name="userPass"/><br/>
  <input type="submit" value="login"/>
</form>
```

RequestDispatcher - przykład Login.java (1/2)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Login extends HttpServlet {

  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
        throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");
    String p=request.getParameter("userPass");

    ...
  }
}
```

RequestDispatcher - przykład Login.java (2/2)

```
public class Login extends HttpServlet {
  public void doPost(...){
    ...
    if(p.equals("servlet")){
        RequestDispatcher rd
            =request.getRequestDispatcher("servlet2");
        rd.forward(request, response);
    }
    else{
        out.print("Sorry UserName or Password Error!");
        RequestDispatcher rd
          =request.getRequestDispatcher("/index.html");
        rd.include(request, response);
    }
  }
}
```

RequestDispatcher - przykład WelcomeServlet.java

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

public class WelcomeServlet extends HttpServlet {

  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
        throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");
    out.print("Welcome␣"+n);
    }

}
```

RequestDispatcher - przykład web.xml (1/2)

```
<web-app>
  <servlet>
    <servlet-name>Login</servlet-name>
    <servlet-class>Login</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-class>WelcomeServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Login</servlet-name>
    <url-pattern>/servlet1</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/servlet2</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```

RequestDispatcher - przykład web.xml (2/2)

```
<web-app>
   ...
   <welcome-file-list>
     <welcome-file>index.html</welcome-file>
   </welcome-file-list>
</web-app>
```

Wdrażanie na serwer i jego uruchomienie

```
import org.eclipse.jetty.webapp.WebAppContext;
import org.eclipse.jetty.server.Server;

public class LoginRunner
{
  public static void main(String[] args) throws Exception{
    String jetty_home = System.getProperty("jetty.home",".");

    Server server = new Server(8080);

    WebAppContext webapp = new WebAppContext();
    webapp.setContextPath("/login");
    webapp.setWar(jetty_home+"/webapps/LoginServlet.war");

    server.setHandler(webapp);

    server.start();
    server.join();
  }
}
```

Przechowywanie i śledzenie sesji

- Sesja - to pewien przedział czasowy
- Śledzenie sesji to przechowywanie stanu (danych) użytkownika
- W świecie serwletów mamy managera sesji
- Protokół HTTP jest bezstanowy, więc potrzebujemy rożnych technik śledzenia, aby móc rozpoznawać użytkowników, np. zalogowanych
- Są 4 podstawowe techniki, które posiadają wsparcie w serwletach
  - Cookies
  - Hidden Form Field
  - URL Rewriting
  - HttpSession

Cookies

- Plik cookie to niewielka część informacji, która występuje między wieloma żądaniami klienta.
- Plik cookie zawiera nazwę, pojedynczą wartość i atrybuty opcjonalne, takie jak komentarze, ścieżki i domeny, maksymalny wiek i numer wersji.
- Domyślnie każde żądanie jest uważany za nowe żądanie.
- W technologii plików cookie dodajemy plik cookie z odpowiedzią z serwletu.
- Plik cookie jest przechowywany w pamięci podręcznej przeglądarki. Następnie, jeśli żądanie jest wysyłane przez użytkownika, plik cookie jest dodawany z żądaniem domyślnie.
- Możemy w ten sposób rozpoznać użytkownika.

Hidden Form Field

- Niewidziale pole tekstowe jest używane, aby przechowywać stan
  `<input type="hidden" name="uname" value="wartość_stanu">`
- W takim przypadku przechowujemy informacje w ukrytym polu i pobieramy je z innego serwletu. Takie podejście jest lepsze, jeśli musimy przesłać formularz na wszystkich stronach i nie chcemy polegać na przeglądarce.
- Zalety:
  - Zawsze będzie działać, niezależnie czy plik cookie jest wyłączony, czy nie.
- Wady:
  - Jest utrzymywany po stronie serwera.
  - Na każdej ze stron wymagane jest położenie dodatkowych pól ukrytych.
  - Można używać tylko informacji tekstowych.

URL Rewriting

- W przepisywaniu adresu URL dodajemy token lub identyfikator do adresu URL następnego serwletu lub następnego zasobu. Możemy wysyłać parę nazw / wartości parametru przy użyciu następującego formatu: `url?name1=value1&name2=value2&??`
- Zalety:
  - Zawsze będzie działać, niezależnie czy plik cookie jest wyłączony, czy nie.
  - Nie jest wymagane żadne dodatkowe ukryte pole
- Wady:
  - Będzie działać tylko z linkami.
  - Może przekazywać tylko informacje tekstowe.

HttpSession

- Kontener tworzy identyfikator sesji dla każdego użytkownika.
- Kontener używa tego identyfikatora, aby zidentyfikować danego użytkownika.
- Można użyć obiektu HttpSession do:
  - powiązania obiektów
  - przeglądania i manipulowania informacjami o sesji, np. identyfikator sesji, czas utworzenia i ostatni dostęp.
- Jak uzyskać dostęp do obiektu sesji:
  - public HttpSession getSession()
  - public HttpSession getSession(boolean create)

Pytania?