Java Server Pages

Autori:
Goran Savić
Milan Segedinac

1. Servleti

U ranijim lekcijama smo videli da server može na zahtev klijenta da dinamički generiše sadržaj HTML dokumenta. Glavna tema ove lekcije je Java Server Pages (JSP) tehnologija kao najčešće korišćena Java tehnologija za dinamičko kreiranje veb dokumenata na serverskoj strani. Pre nego što pogledamo detalje JSP tehnologije, napravićemo kratak uvod kroz upoznavanje sa Java Servlet tehnologijom koja je bazična tehnologija za dinamičku obradu veb zahteva u programskom jeziku Java.

Glavna ideja Java Servleta je da veb server, kao posebna aplikacija, preuzme odgovornost za uspostavljanje mrežne komunikacije sa klijentom i parsiranje zahteva, a da se programer fokusira samo na sadržaj odgovora klijentu. Korisnička aplikacija se onda može postaviti unutar veb servera, pri čemu će veb server biti zadužen za mrežnu komunikaciju sa klijentom. U prošloj lekciji spominjali smo Apache Tomcat kao jedan veb server koji ovo omogućava.

Java Servleti predviđaju pisanje programskog koda koji definiše sadržaj odgovora u posebnim klasama nazvanim Servleti. Sintaksno, Servlet je klasa koja nasleđuje klasu HttpServlet i redefiniše metode doGet() i/ili doPost(), koje predstavljaju reakciju na GET i POST HTTP zahteve. Veb server unutar kojeg se Servlet klase postavljaju se naziva servlet kontejner. Kontejner omogućuje poziv metoda Servlet klase na osnovu URL-a zahteva. Da bi ovo bilo moguće, u konfiguraciji aplikacije je potrebno definisati mapiranje URL-ova zahteva na servlete. Dat je primer Java Servlet klase koja generiše HTML dokument sa spiskom država kao reakciju na GET zahtev klijenta.

```
public class CountryServlet extends HttpServlet {

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

      List<Country> countries = CountryStorage.findAll();

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();  // stream ka klijentu
      // zaglavlje tabele
      out.println("<table>");
      out.println("<tr>");
      out.println("<td>");
      out.println("Name");
      out.println("</td>");
      out.println("<td>");
      out.println("Population");
      out.println("</td>");
      out.println("</tr>");
      //novi red tabele za svaku drzavu
       for (Country c: countries) {
         out.println("<tr>");
         out.println("<td>");
         out.println(c.getName());
         out.println("</td>");
         out.println("<td>");
         out.println(c.getPopulation());
         out.println("</td>");
         out.println("</tr>");
       }
      out.println("</table>");
   }
```

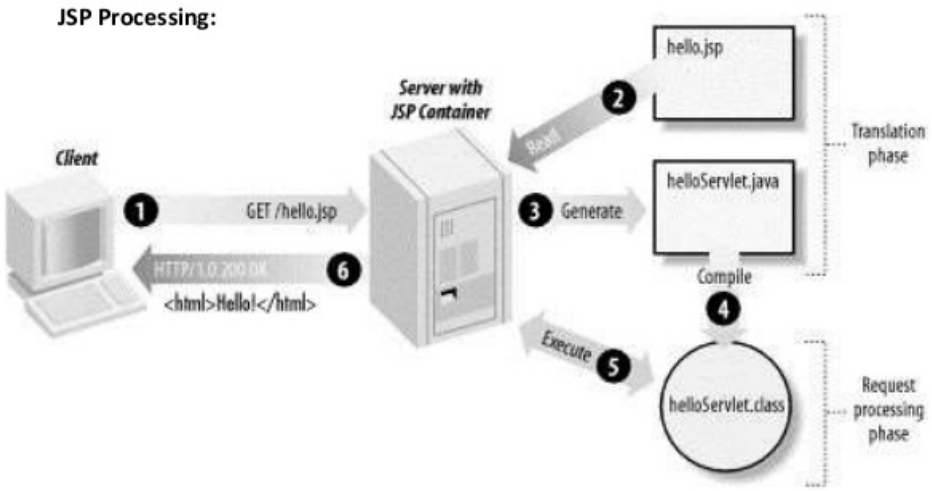
Vidimo da Servlet ima pristup objektima klase HttpServletRequest i HttpServletResponse koji predstavljaju HTTP zahtev, odnosno odgovor. Za instanciranje i inicijalizaciju ovih objekata zadužen je Servlet kontejner. Programer se može fokusirati na upotrebu ovih

objekata u programskom kodu. Zadatak servleta je da klijentu isporuči odgovor. Vidimo da sa ovim ciljem Servlet u izlazni tok podataka ka klijentu upisuje sadržaj HTML dokumenta.

2. JSP

Iako omogućuju dinamičko generisanje veb dokumenta, mana korišćenja servleta je to što su dizajn stranica (HTML) i programska obrada (Java) definisani zajednički unutar Java koda. Ovo dovodi do manje čitljivog koda gde je teže fokusirati se na sadržaj dokumenta, obzirom da je on definisan zajednički sa HTML tagovima koji specificiraju kako će sadržaj biti prikazan. Takođe, teže je razdvojiti poslove dizajnera i programera.

Kako bi se Ideja Java Server Pages (JSP) tehnologije je razdvajanje prikaza podataka na stranici od logike formiranja podataka. Naredna slika prikazuje osnovni princip JSP tehnologije.



* slika preuzeta sa https://www.slideshare.net/shagavikrant/jsp-java-server-pages

Izgled se definiše u okviru JSP veb stranice koja predstavlja šablon HTML dokumenta. Ovaj šablon sadrži standardne HTML elemente, ali i dodatne JSP tagove koji će biti isprocesirani na serveru od strane JSP kontejnera. JSP tagovi sadrže programski kod kojim se ranije definisani podaci inkorporiraju u HTML dokument. Naglasimo da se ovo spajanje podataka sa njihovim vizuelnim prikazom kod JSP tehnologije vrši na serverskoj strani. Iz tog razloga se ovakav tip tehnologija naziva *server side scripting*. JSP je tehnologija koja se zasniva na servletima. JSP kontejner će na osnovu šablona definisanog u JSP stranici programski izgenerisati servlet koji isporučuje HTML elemente i podatke koji se u njima prikazuju. Dakle, JSP tehnologija je dodatni sloj iznad Java Servleta koji omogućuje čitljiviji kod razdvajanjem logike formiranja podataka od njihovog prikaza. Konačni rezultat procesiranja JSP stranice je opet klasičan servlet. Razlika u odnosu na tradicionalan rad sa Servletima je to što programer kod JSP tehnologije ne mora eksplicitno da piše servlet koji će isporučiti HTML dokument.

Pogledajmo jedan jednostavan primer JSP stranice i Servleta koji će kontejner na osnovu nje izgenerisati.

```
<html>
 <body>
  <h1>
   <%= request.getAttribute("message") %>
  </h1>
 </body>
</html>
```

```
public class ServletXYZ extends .....
{
 .......
  out.write("<html>");
  out.write("<body>");
  out.write("<h1>");
  out.write(request.getAttribute(
"message"));
  out.write("</h1>");
  out.write("</body>");
  out.write("</html>");
 ........
```

Vidimo da JSP stranica sadrži standardne HTML tagove html, body i h1 koje smo do sada sretali. Pored toga, sadrži i specijalni JSP tag <%= koji treba dinamički da umetne sadržaj u stranicu na osnovu vrednosti koju vrati definisani Java kod. Konkretno, unutar h1 taga biće prikazana vrednost koju vrati metoda *getAttribute* objekta *request* za prosleđeni parametar *message*. Nešto kasnije pojasnićemo svrhu *request* objekta, ali je principijelno važno već sada primetiti da JSP omogućuje ugrađivanje vrednosti iz Java koda u HTML dokument.

Pogledajmo koje specijalne JSP elemente možemo dodati u JSP stranicu. Ovde ćemo pomenuti tri tipa ovakvih elemenata

- izrazi <%= %>
- skriptleti <% %>
- direktive <%@ %>

JSP izraz smo videli u prethodnom primeru. Na stranicu se postavljaju unutar tagova <%= %> i predstavljaju Java izraz čiji će rezultat biti kao tekst umetnut u stranicu. Za izraze koji ne vraćaju String, automatski se poziva toString() metoda.

Skriptleti predstavljaju proizvoljne fragmente Java koda. Njihova svrha nije kao kod izraza da vraćaju tekst koji će biti umetnut u stranicu već da izvršavaju određenu programsku logiku koja će omogućiti dinamičko generisanje kompleksnijih HTML stranica. Skriptleti se na stranicu postavljaju unutar elemenata <% %>.

Direktive se označavaju elementima <%@ %> i daju dodatne instrukcije kontejneru za procesiranje JSP stranice. Na primer, direktiva <%@ page import="java.util.List" %> označava da je klasa List koja se koristi u JSP stranici iz paketa java.util.

Pogledajmo primer JSP stranice koja uključuje sve pomenute tipove JSP elemenata. Stranica prikazuje spisak država i data je u nastavku.

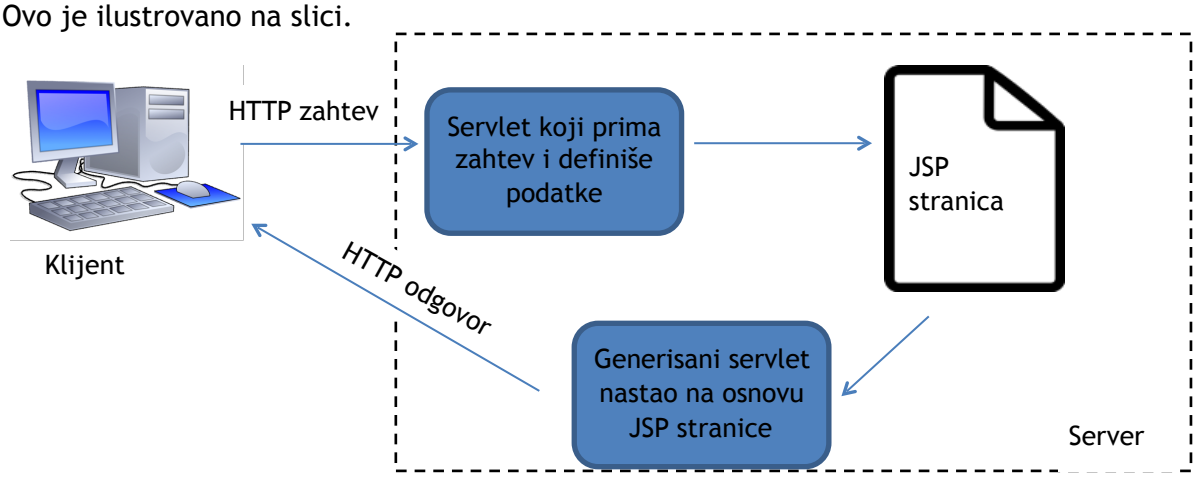
```
<%@ page import="vp.jsp.JSPCountry.model.Country" %>
<%@ page import="java.util.List" %>

<html>
  <body>
    <h1> Drzave </h1>
    <% List<Country> countries = (List<Country>) request.getAttribute("countries"); %>
    <table>
      <tr>
        <th>Naziv</th>
        <th>Populacija</th>
      </tr>
      <% for (Country country: countries) { %>
      <tr>
```

```
        <td> <%= country.getName() %> </td>
        <td> <%= country.getPopulation() %> </td>
      </tr>
      <% } %>
    </table>
  </body>
</html>
```

Deo sadržaja na stranici može biti definisan statički. To su zaglavlje stranice, tabela sa spiskom država i zaglavlje tabele. Međutim, tabela treba da ima onoliko redova koliko imamo evidentiranih država u aplikaciji i taj deo stranice ne može biti definisan statički. Umesto toga, dinamički se prolazi u petlji kroz listu država. Vidimo da je skriptletom definisana ova petlja. U telu petlje su elementi koji trebaju biti prikazani, a to su redovi i njihove ćelije. Svaki red u svojim ćelijama treba da prikaže podatke naziv i broj stanovnika jedne države, što je definisano JSP izrazom unutar <td> tagova.

Videli smo da JSP stranica može da prikazuje i podatke definisane u Java aplikaciji. U prethodnim primerima to su bile promenljive *message* i *countries*. Pogledajmo na koji način su ovi podaci definisani i postali dostupni JSP stranici. Vidimo da JSP stranica ove podatke preuzima iz *request* objekta. Objekat *request* je jedna od nekoliko predefinisanih promenljivi u JSP aplikaciji. Kontejner obezbeđuje kreiranje i inicijalizaciju ovih promenljivih. Konkretno za *request* objekat, on predstavlja klijentov zahtev i u toku obrade zahteva u ovaj objekat je moguće ubacivati nove podatke. Podaci se u objektu predstavljaju kao mapa parova {ključ, vrednost}.

Obzirom da je JSP tehnologija bazirana na servletima, prvi korak u obradi zahteva standardno je kod napisan u Java servletu. Na osnovu URL-a zahteva, poziva se odgovarajući servlet koji definiše podatke koje JSP stranica treba da prikaže. Servlet je zadužen da specificira na osnovu koje JSP stranica se formira konačan odgovor klijentu. Ovo je ilustrovano na slici.



Iako se za obradu zahteva u JSP tehnologiji standardno koriste servleti, za ilustraciju prikazanog koncepta mi ćemo prikazati primer obrade zahteva korišćenjem Spring kontrolera, obzirom da je to veb komponenta sa kojom smo se ranije susretali i da je uloga Spring kontrolera takođe reakcija na veb zahteve slično servletima. Sledeći primer prikazuje Spring kontroler koji prima korisnikov zahtev za prikazom stranice sa spiskom država.

```
@RequestMapping(value="/countries")
public String getAllCountries(Map<String, Object> model) {
      List<Country> countries = countryService.findAll();
      model.put("countries", countries);
```

```
        return "countries";
}
```

Vidimo da @RequestMapping anotacija specificira da će kontroler biti pozvan kada URL zahteva bude */countries*. Kontroler je zadužen da odredi JSP stranicu koja će prikazati odgovor. Ovo je definisano u poslednjem redu kontrolera gde je navedeno da se prikazuje stranica *countries.jsp* (Spring automatski doda ekstenziju jsp pa se ona ne navodi). Pre obrade JSP stranice, kontroler treba da pripremi podatke koji će se na stranici prikazivati u okviru HTML elemenata. U prikazanom primeru, listu država obezbeđuje objekat *countryService*. Ključno pitanje je na koji način da se dobavljena lista država učini dostupnom JSP stranici. Za ovo se standardno koristi *request* objekat kojem pristup imaju i servlet i JSP stranica. Ako kao u primeru koristimo Spring, parametar *model* ima ulogu mape objekata kojima i JSP stranica ima pristup. Potrebno je da željene objekte ubacimo u model i oni će biti dostupni JSP stranici putem *request* objekta. Za svaki objekat se definiše ključ preko kojeg može biti preuzet iz *request* objekta na JSP stranici. U prikazanom primeru, lista država se u model ubacuje sa ključem *countries*.

3. HTML forme

HTTP POST metoda

Kada je reč o HTTP zahtevima, do sada smo se susretali sa HTTP GET metodom. Ova metoda je namenjena preuzimanju podataka sa servera i treba da se koristi za operacije koje ne menjaju stanje resursa na serveru. Dodatni podaci koji se GET metodom šalju serveru se, umesto u telo zahteva, standardno ugrađuju u sam URL u obliku *url_resursa?podatak1=vrednost1&podatak2=vrednost2*.

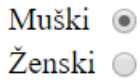
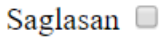
Kada je cilj HTTP zahteva da dostavi određene podatke serveru (najčešće da bi bili uskladišteni), koristi se POST metoda. Za razliku od GET metode, POST metoda je predviđena za operacije koje menjaju stanje podataka na serveru. Podaci poslani serveru POST metodom se ubacuju u telo HTTP zahteva. Dat je primer HTTP POST zahteva koji šalje podatke o jednoj državi serveru.

```
POST www.somesite.com/add HTTP/1.0
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0
Content-Type: application/x-www-form-urlencoded

name=England&population=53010000
```

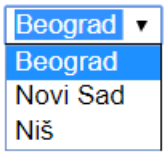
HTML polja za unos podataka

Ako podatke koji se sa klijentu šalju unosi korisnik na veb stranici, unos podataka se vrši u HTML elementima za unos podataka (eng. *input fields*). Postoji više tipova ovih polja. U nastavku su kroz primere predstavljena najčešće korišćena. Za *input* polja se standardno definiše i atribut *name*. Ovo je naziv ključa pod kojim će unešena vrednost biti prosleđena serveru, kako bi server znao na koji podatak se vrednost odnosi.

Tekst polje	
Primer	`<input type="text" name="firstName" value="Petar">`
Opis	Polje za unos teksta. Atribut value sadrži vrednost koja se prikazuje u polju

Izgled	
Polje za lozinku	
Primer	`<input type="password" name="pwd">`
Opis	Polje za unos lozinke. Ne prikazuje u polju karaktere koji se unose. Programski je iz atributa value moguće preuzeti vrednost upisanu u polje.
Izgled	
Radio dugme	
Primer	`<input type="radio" name="gender" value="male" checked="checked">` `<input type="radio" name="gender" value="female">`
Opis	Radio dugme koje omogućuje izbor jedne od više ponuđenih opcija. Međusobno će se isključivati izbor radio dugmadi koja imaju istu vrednost za atribut *name*. Svako dugme ima različitu vrednost. Polje *gender* imaće vrednost atributa *value* iz selektovanog radio dugmeta. Može se putem atributa *checked* podesiti da podrazumevano bude selektovano određeno dugme.
Izgled	
Checkbox	
Primer	`<input type="checkbox" name="agree" value="Yes">`
Opis	Polje za izbor opcije. Atribut *value* predstavlja vrednost polja u slučaju da je selektovano.
Izgled	
Padajuća lista	

Primer	`<select name="city" >`
	`  <option value="BG">Beograd</option>`
	`  <option value="NS" selected>Novi Sad</option>`
	`  <option value="NI">Niš</option>`
	`</select>`

Opis	Polje za izbor stavke iz padajuće liste. Svaka stavka se definiše *option* tagom. Podrazumevano je selektovana stavka označena atributom *selected*. Vrednost polja se preuzima iz atributa *value* selektovane stavke.
Izgled	
Skriveno polje	
Primer	`<input type="hidden" name="id" value="5" />`

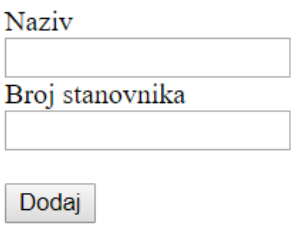
Opis	Skriveno polje za slanje određenog podatka sa forme, a da se podatak ne prikazuje korisniku.
Izgled	Polje se ne prikazuje na stranici.

HTML forme

Osnovni način slanja podataka iz *input* polja serveru je tako što se *input* polja postave u HTML formu. Forma grupiše *input* polja čiji će se podaci poslati servetu pri potvrdi forme. Ova operacija se naziva *submit* forme i vrši se klikom na *submit* dugme. *Submit* dugme šalje podatke sa forme u kojoj je definisano. Pogledajmo na primeru jednu formu sa ulaznim poljima i *submit* dugmetom.

```
<form action="/add" method="post">
	Naziv <br/>
	<input type="text" name="name">  <br/>
	Broj stanovnika <br />
	<input type="text" name="population"> </br> <br/>

	<input type="submit" value="Dodaj">
</form>
```

Form tag omogućuje grupisanje elemenata zbog slanja njihovih podataka i sam po sebi nema vizuelnu reprezentaciju na HTML stranici. Slika prikazuje kako forma iz primera izgleda na stranici.



Klikom na dugme *submit* izvršiće se akcija definisana u form tagu. Vidimo da akcija podrazumeva slanje zahteva na url */add* HTTP metodom POST. Za formu iz primera, HTTP zahtev će izgledati kao u ranijem prikazanom primeru HTTP POST zahteva sa početka ovog poglavlja.

Pre nego što pređemo na objašnjenje kako server može da prihvati ove podatke, pomenimo još da je osim *submit* dugmeta, moguće definisati i obično dugme na HTML stranici, koje je zaduženo da izvrši proizvoljnu akciju (najčešće poziv JavaScript funkcije). Primer ovakvog dugmeta je dat u nastavku.

```
<input type="button" onclick="verifyPayment()" value="Verify">
```

Obrada HTTP POST zahteva u Spring veb aplikaciji

Dakle, podaci sa HTML forme mogu POST metodom biti poslati veb serveru. Parsiranjem sadržaja zahteva, server može preuzeti ove podatke i upotrebiti ih pri obradi zahteva (npr. u evidenciju dodati državu sa naznačenim podacima). Kod Spring veb aplikacije posao parsiranja zahteva je odgovornost Spring kontejnera i programer se može fokusirati na upotrebu dobijenih podataka. Pogledajmo kako u Spring veb aplikaciji izgleda obrada POST zahteva. POST zahtev se obrađuje u metodi Spring kontrolera. U primeru je prikazana obrada zahteva za dodavanje nove države.

```
@RequestMapping(value="/countries/add", method=RequestMethod.POST)
public String addCountry(HttpServletRequest request) {
   String name = request.getParameter("name");
   String population = request.getParameter("population");

   Country country = new Country(-1, name, Integer.parseInt(population));
   countryService.save(country);

   return "redirect:/countries";
}
```

Metoda addCountry() će biti pozvana ako se uputi HTTP POST zahtev na URL */countries/add*. Podaci koje je klijent poslao u telu zahteva mogu se preuzeti iz objekta klase *HttpServletRequest*. Spring je zadužen za instanciranje ovog objekta na osnovu sadržaja zahteva, kao i za prosleđivanje ovog objekta metodi kontrolera. Podaci iz zahteva se koriste za formiranje nove države, koja se evidentira u skladištu putem objekta *countryService*. Poslednji red metode označava da se nakon dodavanja države vrši redirekcija na URL koji će obezbediti prikaz država.