

Dinamičko generisanje HTML-a i servleti

Sadržaj www sajta

- HTML stranice
- multimedijalni elementi (slike, animacije, itd)
- drugi tipovi datoteka
- www server i klijent komuniciraju preko HTTP protokola
- Verzije
 - HTTP/1.0
 - HTTP/1.1 (permanent/persistent/**keep-alive** connection) -1997
 - HTTP/2.0 (performance improvements, header compression, usage of encryption, and prioritization of requests)- 2014

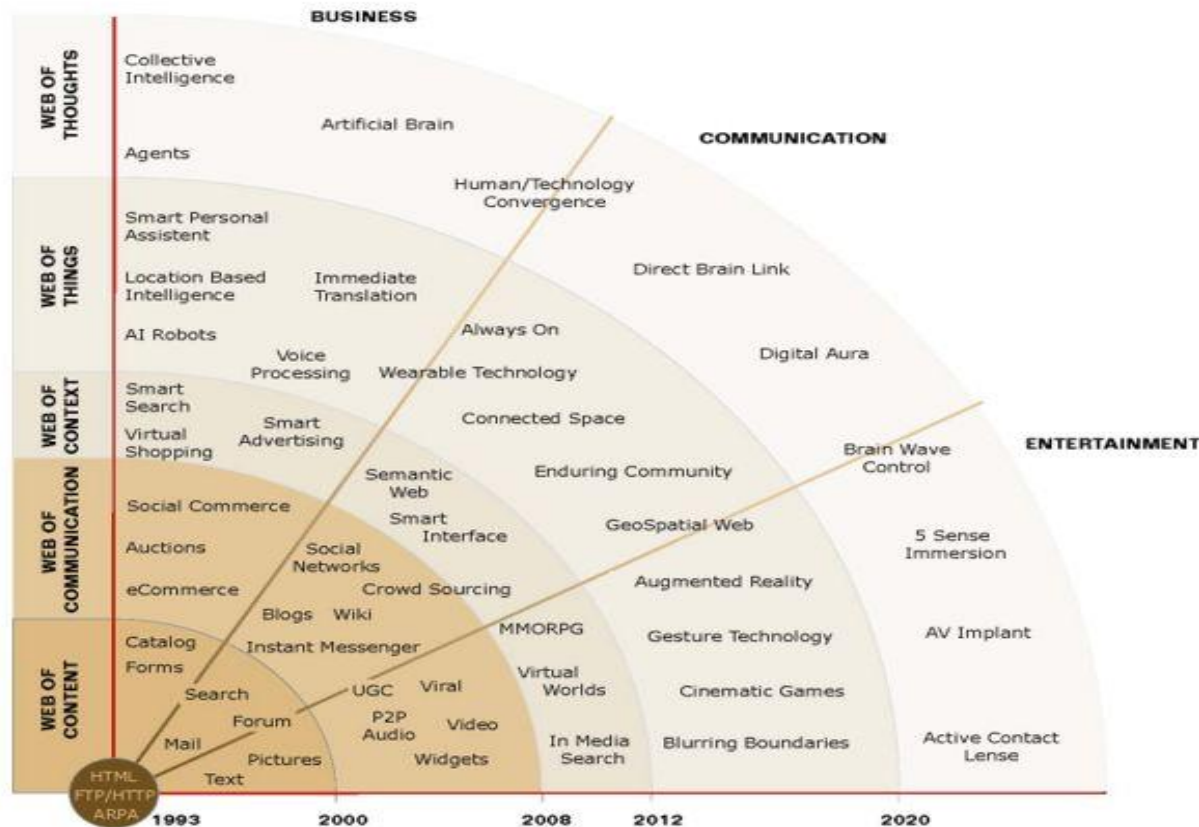
World Wide Web

- Prostor u kome se nalaze dokumenti kategorisani po Uniform Resource Locator (URL)
- Web 1.0 – Korisnici su puki “čitaoci” informacija (wikipedia). Statički sadržaj uglavnom. Server traži resurs u svom fajl sistemu
- Web 2.0 (1999) – Dinamički sadržaj. Koncept Web as a Platform. Nema više desktop aplikacija. Korisnici kreiraju sadržaj. Personalizovani korisnici (user account) WebShops, Social Networks. Rich User Experience, Cloud Computing, Software as a Service (SaaS).
- Web 3.0 (2006) - web of content where the meaning can be processed by machines
- Web 4.0, 5.0 – MobileOpen Linked, Iteligent, VR, AR

World Wide Web

trendone
NILS MÜLLER COMMUNICATIONS

THE WEB EXPANSION
FROM WEB OF THINGS TO WEB OF THOUGHTS



©TrendONE 2008 by Nils Müller
www.TrendONE.de All rights reserved

HTTP komunikacija

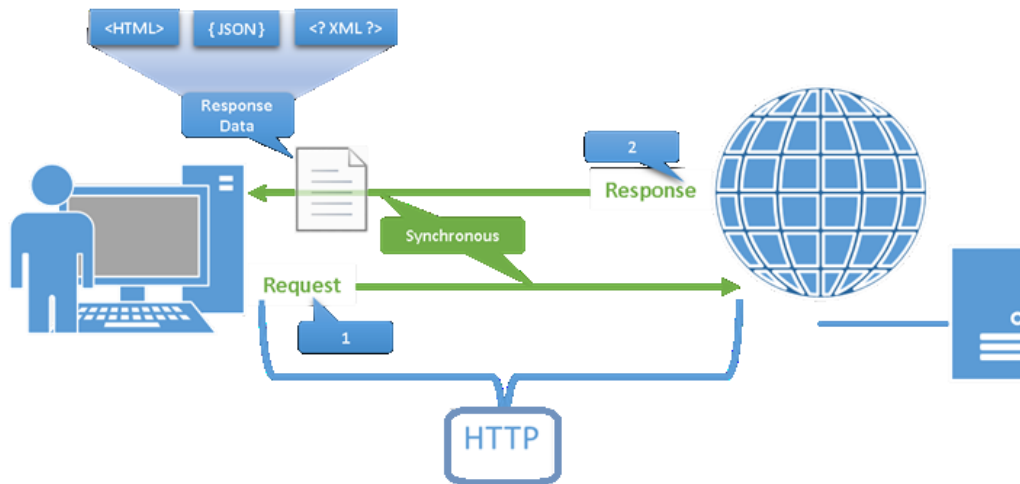
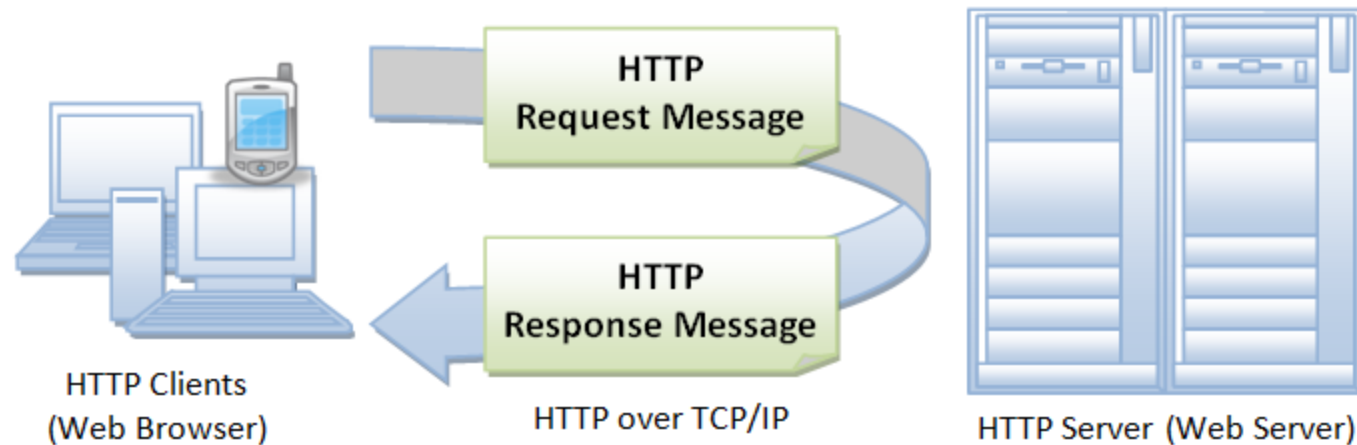
- zasnovana na zahtev/odgovor principu
- svaki par zahtev/odgovor se smatra nezavisnim od ostalih
- ne omogućava praćenje korisničke sesije, tj. niza zahteva upućenih od strane istog klijenta
- U verziji 1.0 po završetku isporuke odgovora klijentu konekcija se zatvara (za novu komunikaciju klijenta sa serverom opet treba da se uspostavi konekcija).
- U verziji 1.1 konekcija se ne zatvara tj. konekcija ostaje otvorena (keep-alive). Klijent će istu konekciju da koristi pri slanju novog zahteva ka serveru. Konekcija ostave otvorena sve dok neko od stana u komunikaciji (klijent ili server) ne odluči da je neophodno za završi komunikaciju sa drugom stranom, što će uraditi tako što će zatvoriti konekciju.

HTTP komunikacija

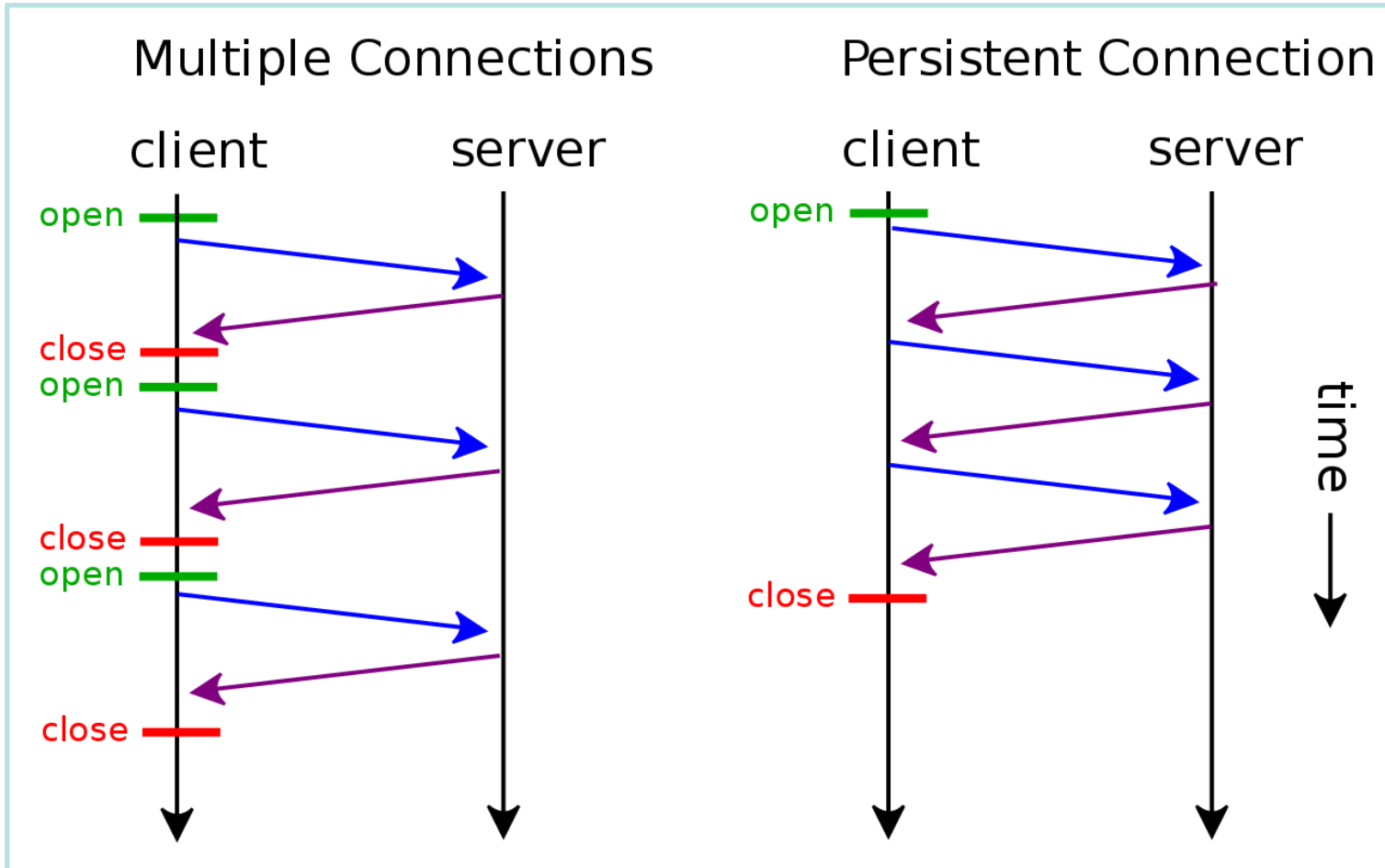
- Prednosti ver 1.1:
 - Smanjeno zauzeće CPU jer je smanjen broj poruka koje se kreiraju, obrađuju i šalju mrežom.
 - Smanjeno zagušene mreže (manje poruka za kreiranja TCP konekcija).
- Mana ver 1.1:
 - Situacija u kojoj je klijent preuzeo sve podatke od servera ali nije zatvorio konekciju je problem. U takvoj situaciji server nepotrebno troši resurse za otvorenu vezu, umesto da te resurse mogu da koriste drugi klijenti. Prethodno može da utiče na dostupnost servera da prima nove zahteve klijenta, ako je na serveru ograničen broj klijenta koje istovremeno server opslužuje. Server će izvršiti zatvaranje konekcije koja je idle u zavisnosti od konfiguracije.

HTTP komunikacija

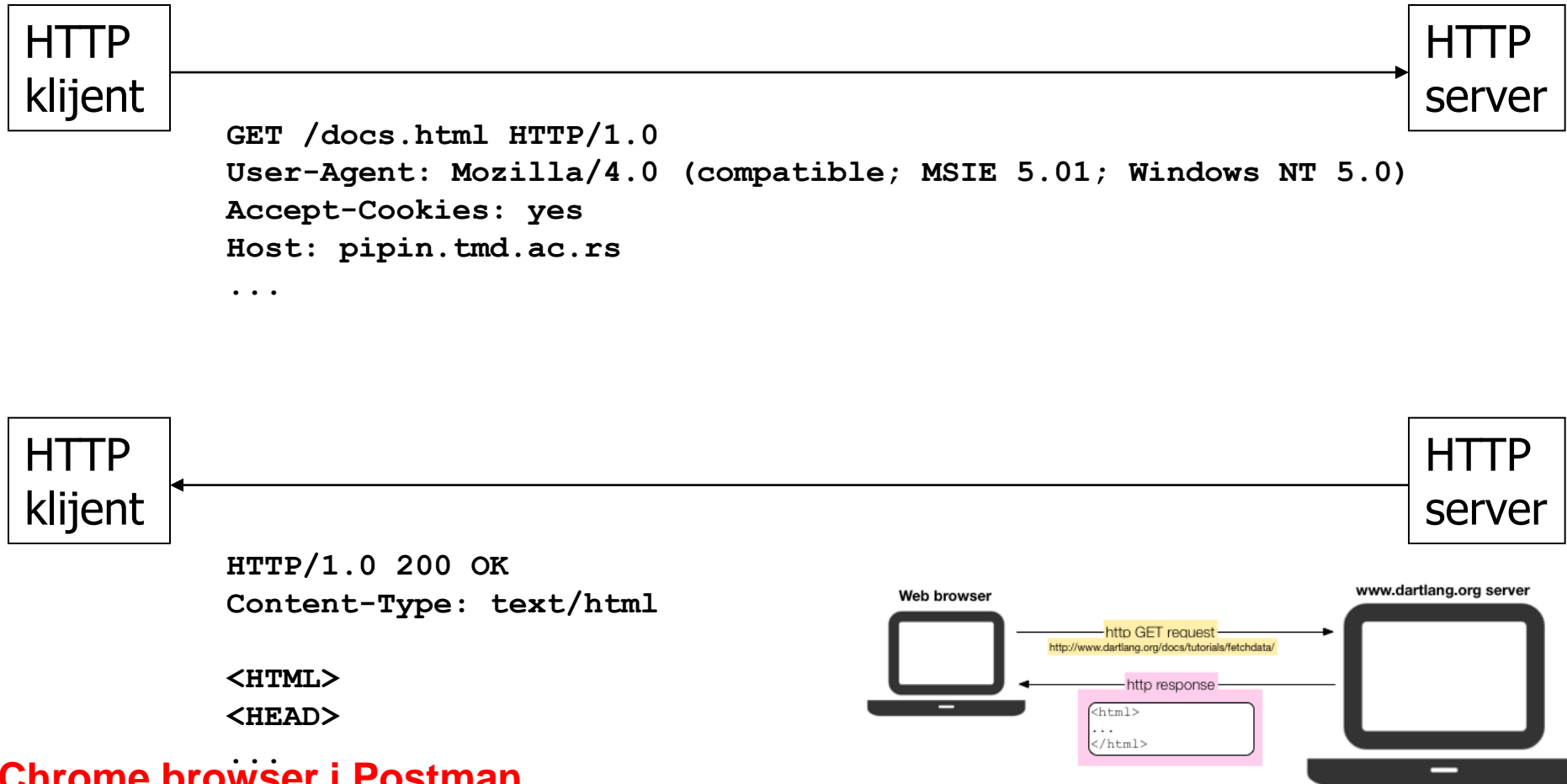
- zasnovana na zahtev/odgovor principu



HTTP 1.0 i HTTP 1.1 komunikacija



HTTP komunikacija

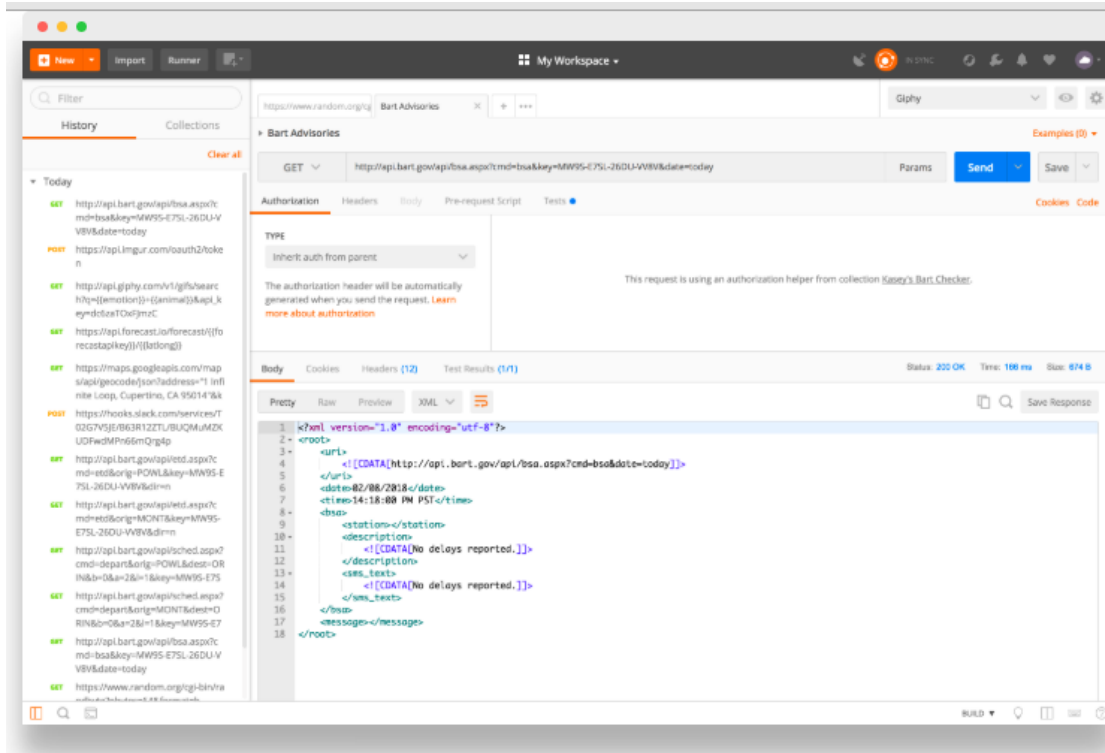


Chrome browser i Postman

<http://mondo.rs/Naslovna>

<https://www.youtube.com/watch?v=QV2iYFI5eSk>

Postman

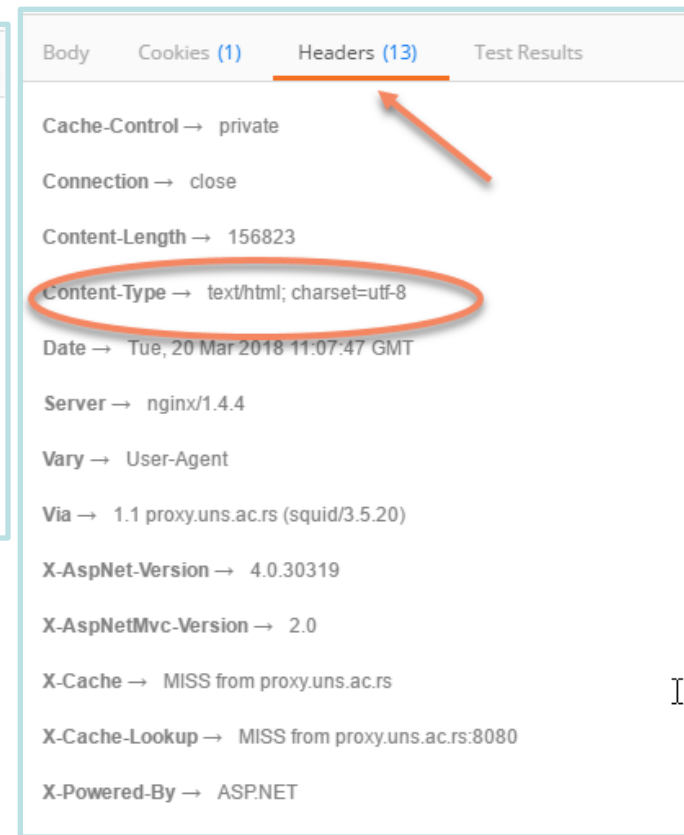
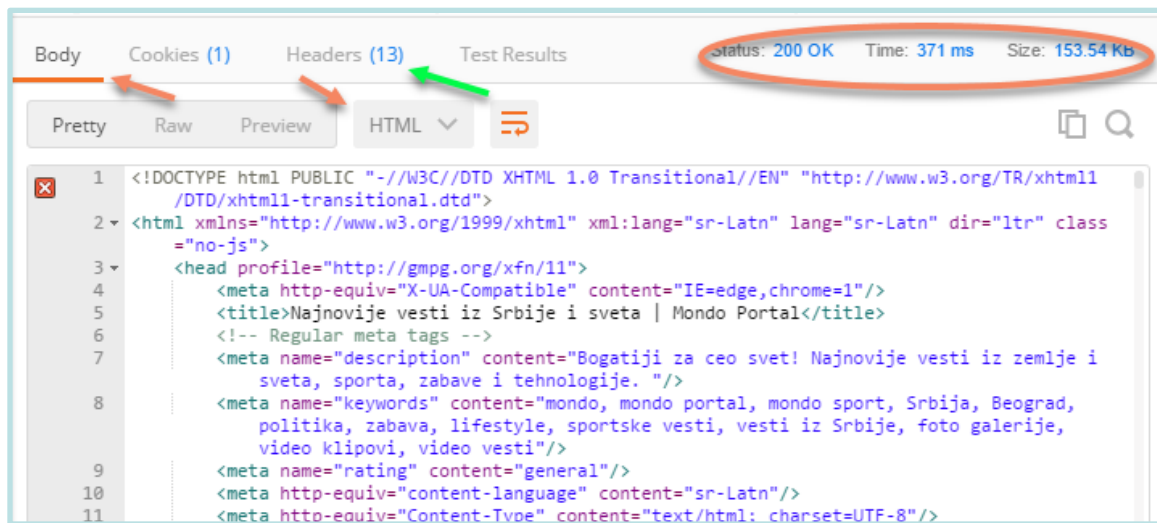
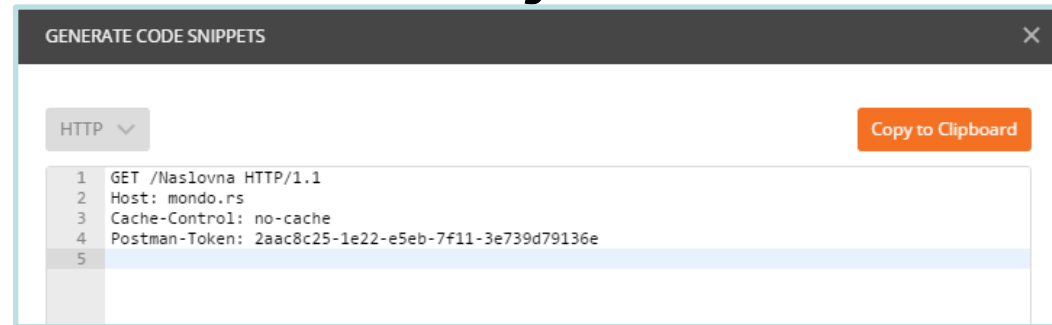
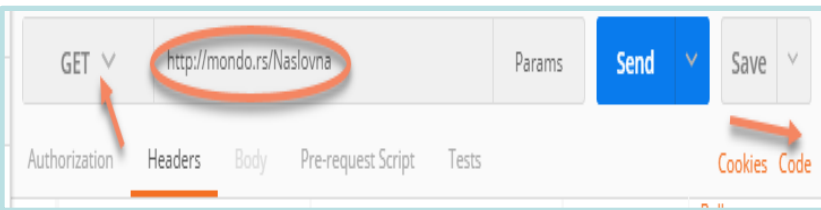


Postman is the most complete toolchain for API development

- The most-used REST client worldwide
- Designed from the ground up to support the API developer
- Intuitive user interface to send requests, save responses, add tests, and create workflows

[Read the docs](#)

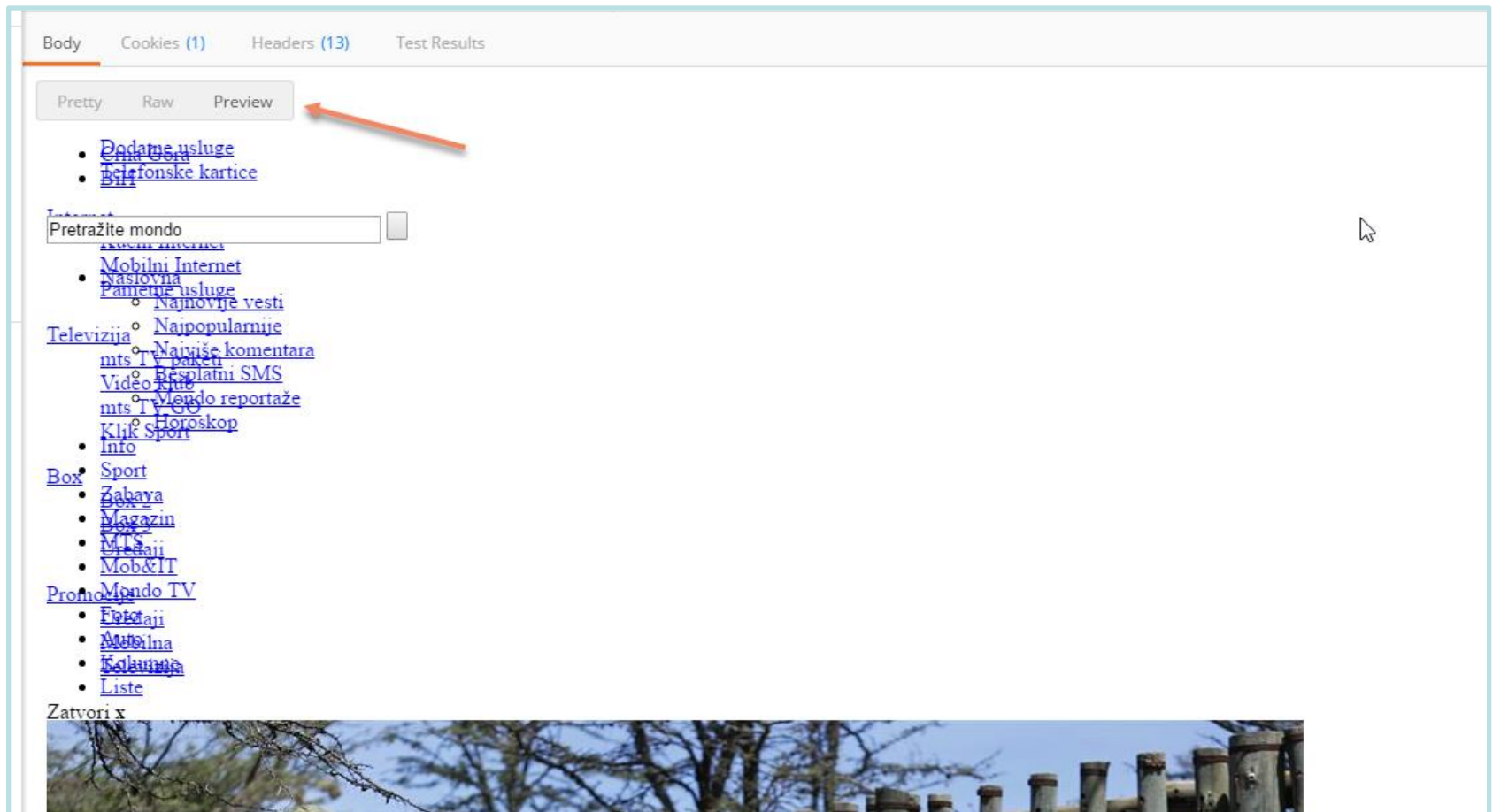
HTTP komunikacija



Postman

<http://mondo.rs/Naslovna>

HTTP komunikacija



Postman

<http://mondo.rs/Naslovna>

HTTP komunikacija

GET <https://www.youtube.com/watch?v=a76yzV0V1s...> Params Send Save

Authorization Headers Body Pre-request Script Tests Cookies Code

Key	Value	Description	Bulk Edit	Presets
New key	Value	Description		

Body Cookies (3) Headers (13) Test Results Status: 200 OK Time: 601 ms Size: 231.71 KB

Pretty Raw Preview HTML

```
1 <!DOCTYPE html>
2 <html lang="sr" data-cast-api-enabled="true">
3   <head>
4     <style name="www-roboto" >@font-face{font-family:'Roboto';font-style:italic;font
      -weight:400;src:local('Roboto Italic'),local('Roboto-Italic'),url(//fonts.gstatic
      .com/s/roboto/v18/KFOkCnqEu92Fr1Mu51xMIzc.ttf)format('truetype');}@font-face{font
      -family:'Roboto';font-style:italic;font-weight:500;src:local('Roboto Medium
      Italic'),local('Roboto-MediumItalic'),url(//fonts.gstatic.com/s/roboto/v18
      /KFOjCnqEu92Fr1Mu51S7ACc-CsE.ttf)format('truetype');}@font-face{font-family
      : 'Roboto';font-style:normal;font-weight:400;src:local('Roboto Regular'),local
      ('Roboto-Regular'),url(//fonts.gstatic.com/s/roboto/v18/KFOmCnqEu92Fr1Mu5mxP.ttf
      )format('truetype');}@font-face{font-family:'Roboto';font-style:normal;font
      -weight:500;src:local('Roboto Medium'),local('Roboto-Medium'),url(//fonts.gstatic
      .com/s/roboto/v18/KFOlCnqEu92Fr1MmEU9fABc9.ttf)format('truetype');}</style>
5     <script name="www-roboto" >if (document.fonts && document.fonts.load) {document.fonts
      .load("400 10pt Roboto", "C");document.fonts.load("500 10pt Roboto", "C");}
6     <script >var ytcsl = {gt: function(n) {n = (n || '') + 'data_';return ytcsl[n] ||
      (ytcsl[n] = {tick: {},info: {}})};now: window.performance && window.performance
```

GENERATE CODE SNIPPETS

HTTP

```
1 GET /watch?v=a76yzV0V1sk&list=RDa76yzV0V1sk HTTP/1.1
2 Host: www.youtube.com
3 Cache-Control: no-cache
4 Postman-Token: 749ccbc-4667-1202-ac0c-6a40057b006a
5
```

Body Cookies (3) Headers (13) Test Results Status: 200 OK Time: 601 ms Size: 231.71 KB

Alt-Svc → hq="443";ma=2592000;quic=51303431;quic=51303339;quic=51303335;quic="443";ma=2592000;v="41,39,35"

Cache-Control → no-cache

Connection → close

Content-Encoding → gzip

Content-Type → text/html; charset=utf-8

Date → Tue, 20 Mar 2018 11:19:17 GMT

Expires → Tue, 27 Apr 1971 19:44:06 EST

Server → YouTube Frontend Proxy

Strict-Transport-Security → max-age=31536000

Transfer-Encoding → chunked

X-Content-Type-Options → nosniff

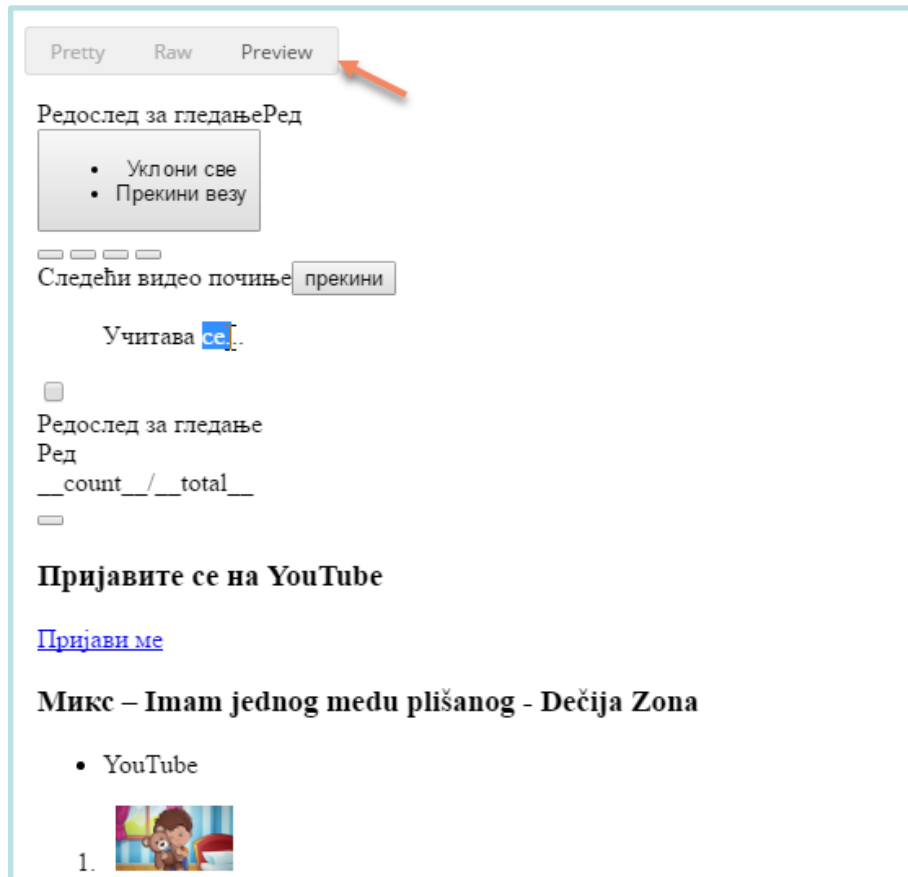
X-Frame-Options → SAMEORIGIN

X-XSS-Protection → 1; mode=block; report=https://www.google.com/appserve/security-bugs/log/youtube

Postman

<https://www.youtube.com/watch?v=QV2iYFI5eSk>

HTTP komunikacija



Postman

<https://www.youtube.com/watch?v=QV2iYFI5eSk>


HTTP komunikacija

GET Params **Send** Save

Authorization **Headers (1)** Body Pre-request Script Tests Cookies Code

Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/> my-sample-header	Lorem ipsum dolor sit amet				
New key	Value	Description			

Body **Cookies (1)** Headers (14) Test Results Status: 200 OK Time: 709 ms Size: 862 B

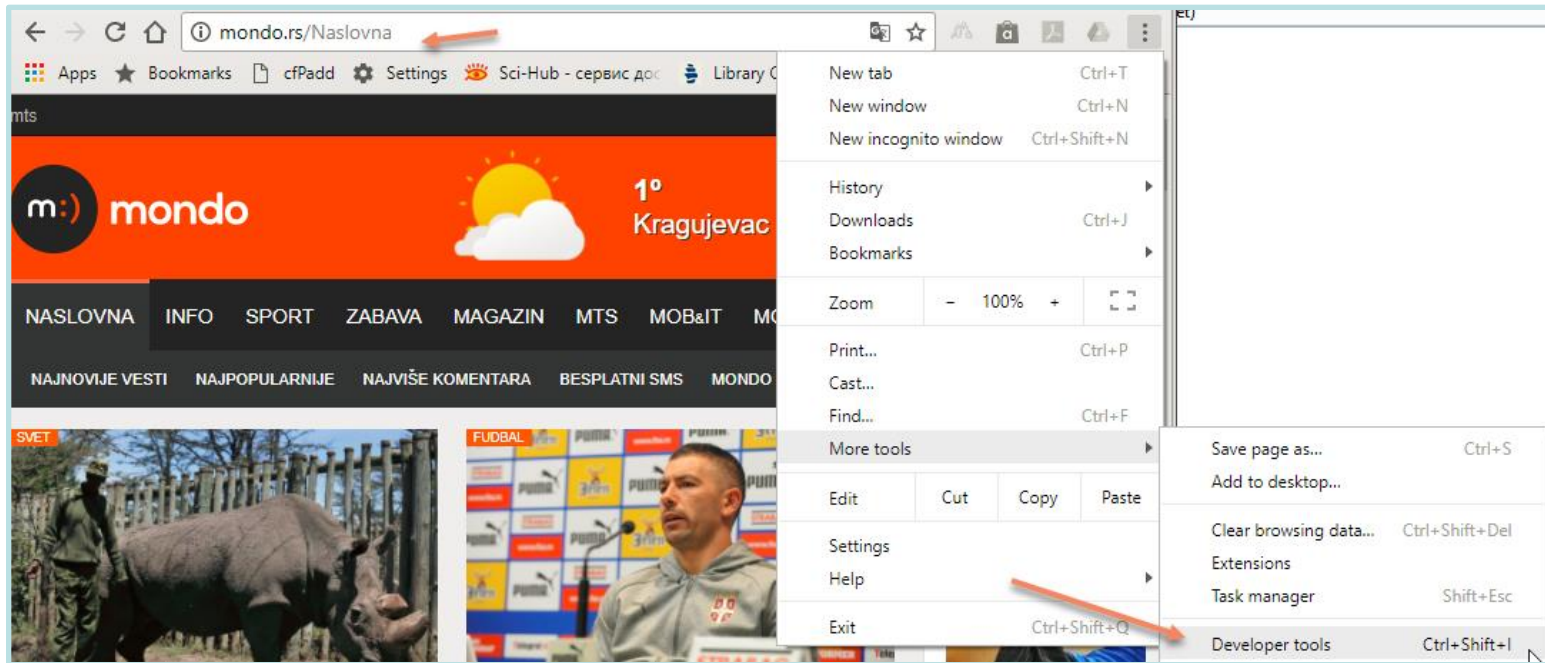
Pretty Raw Preview **JSON** 

```
1 {
2   "headers": {
3     "host": "postman-echo.com",
4     "accept": "*/*",
5     "accept-encoding": "gzip, deflate",
6     "cache-control": "no-cache",
7     "cookie": "sails.sid=s%3AUp918K51pBcP6v_osk1vp5WZ-v6wMIiy
8       .eYrmXgm%28DBJsQBWHnDhGQHEiPBQpFk8nsZ1BQN8aQQc",
9     "my-sample-header": "Lorem ipsum dolor sit amet",
10    "postman-token": "cfc30e75-3fdc-4422-8cbc-0e1e9e18d37a",
11    "user-agent": "PostmanRuntime/7.1.1",
12    "x-forwarded-port": "443",
13    "x-forwarded-proto": "https"
14  }
```

Postman

<https://postman-echo.com/headers>

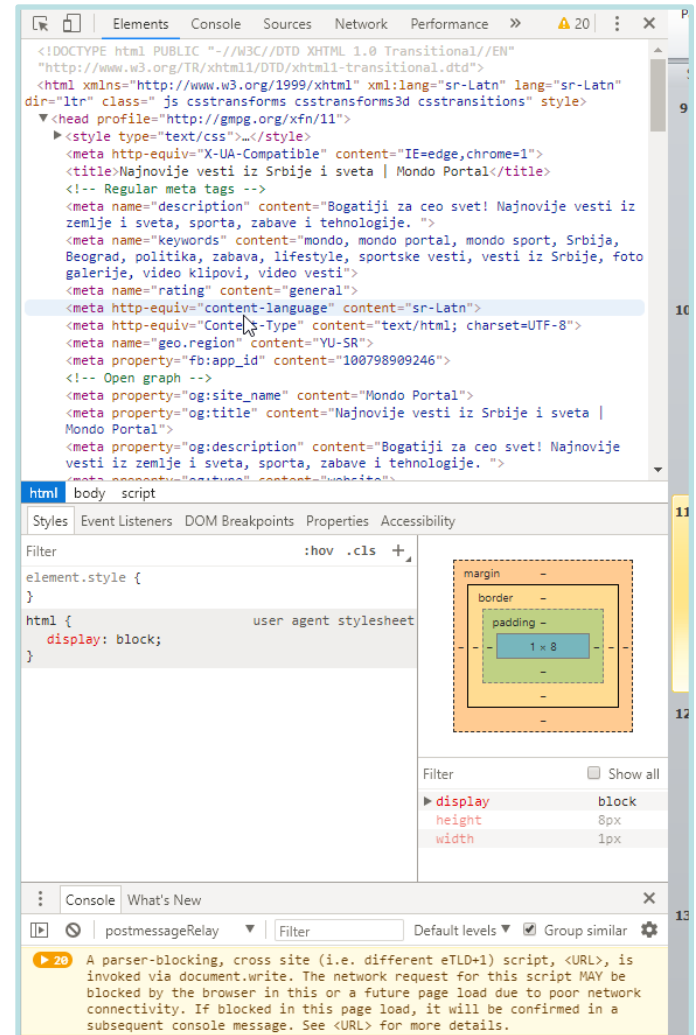
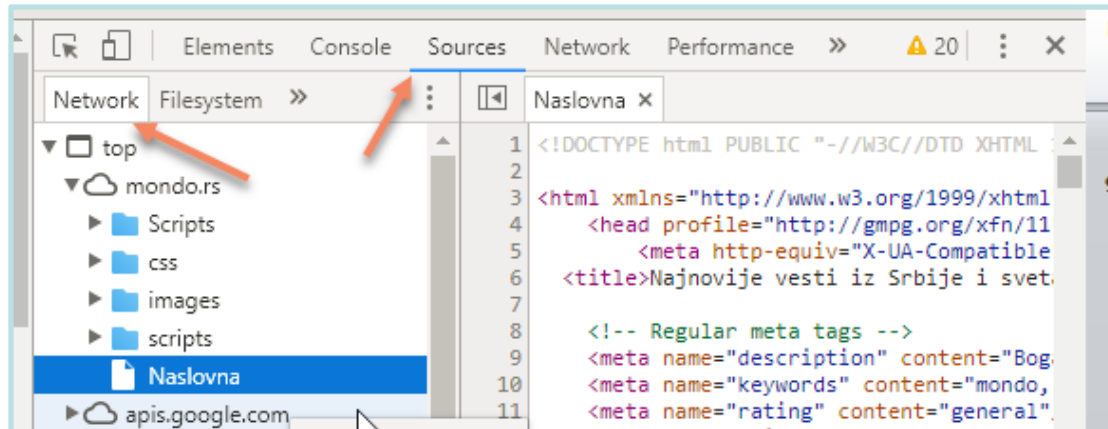
HTTP komunikacija



Chrome browser

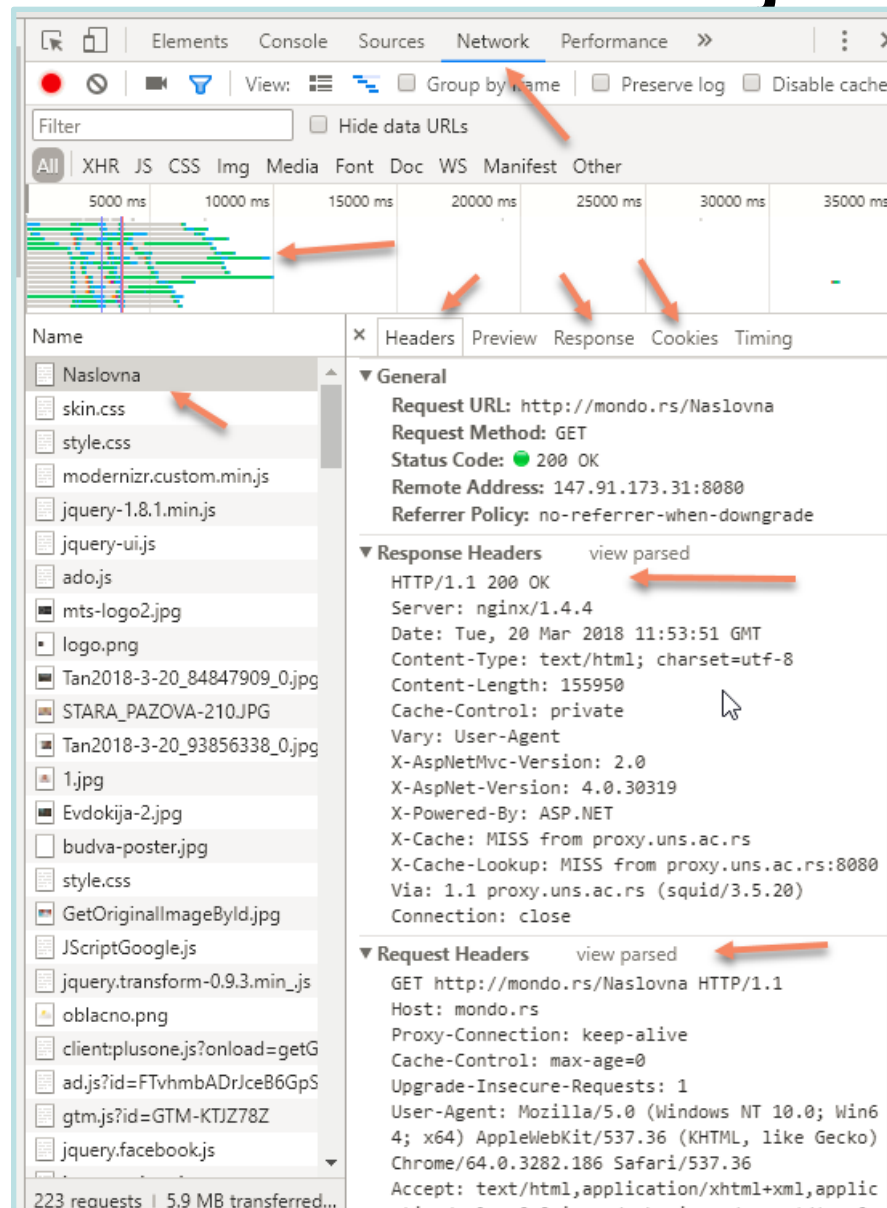
<http://mondo.rs/Naslovna>

HTTP komunikacija



Chrome browser
<http://mondo.rs/Naslovna>

HTTP komunikacija



Chrome browser
<http://mondo.rs/Naslovna>

HTTP komunikacija

- HTTP je stateless protokol koji ne zateva od servera čuvanje statusa klijenta ili korisničke sesije klijenta tj. niza zahteva upućenih od strane istog klijenta
 - HTTP serveri prevazilaze prethodno tako što implementiraju različite metode za održavanje i upravljanje sesijom, tipično se oslanjajući na jedinstveni identifikator *cookie* ili neki drugi parametar koji omogućava praćenje zahteva koji originiraju od istog klijenta (npr. URL Rewriting mehanizam), kreirajući stateful protokol iznad HTTP protokola.

HTTP zahtev

- Počinje redom:
METHOD /putanja HTTP/verzija
- METHOD je:
 - GET,
 - POST, i dr.
- dodatni redovi sadrže attribute oblika:
Ime: vrednost
- prazan red na kraju
 - Ako je POST zahtev posle praznog reda idu parametri forme

METHOD

- GET – zahteva resurs od web servera
- POST – šalje parametre forme i traži odgovor
- HEAD – zahteva samo HTTP odgovor (response), bez slanja samog resursa
- PUT – omogućava klijentu da pošalje datoteku na web server
- OPTIONS – od web servera se traži spisak metoda koje podržava
- DELETE – omogućava klijentu da obriše resurs sa web servera

Atributi u HTTP zahtevu

- User-Agent – identifikuje web browser

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.2) Gecko/20070219 Firefox/2.0.0.2

- Accept – definiše koje tipove resursa navigator prihvata kao odgovor na ovaj zahtev

Accept:

text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

- Accept-Language – definiše koji jezike očekuje kao odgovor

Accept-Language: en-us,en;q=0.5

- Accept-Encoding – definiše koje kodiranje očekuje kao odgovor

Accept-Encoding: gzip,deflate

Atributi u HTTP zahtevu

- Accept-Charset – definiše koju kodnu stranu očekuje

Accept-Charset: ISO-8859-1,utf-8;q=0.8,ASCII;q=0.7,*;q=0.6

- Cookie – definiše mehanizam praćenja sesije

Cookie: id1172566682241_1=1172566682241_1

- Referer – definiše URL sa kojeg se došlo na ovu stranicu
 - koristi se za statistiku
 - hotlinking

Referer: http://localhost/

- Connection – HTTP1.1 "kaže" serveru da ne zatvara konekciju po isporuci resursa

Connection: Keep-Alive

- q= broj definiše *qvalue*, a predstavlja **relative quality factor** odn. floating point vrednost "težine" parametra
 - Favorizovani Charset je ISO-8859-1 ili utf-8, ali ukoliko oni nisu podržani može i ASCII, a ako ništa od prethodnog nije podržano, prihvaću i * (bilo koji drugi)

Primer HTTP zahteva

GET / HTTP/1.1

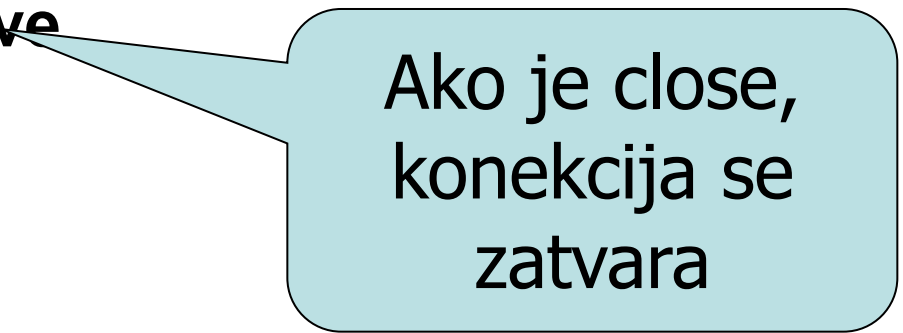
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, application/x-shockwave-flash, */*

Accept-Language: sr

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; .NET CLR 1.1.4322)

Host: localhost

Connection: Keep-Alive



Ako je close,
konekcija se
zatvara

Keep Alive

HTTP persistent connection, also called HTTP keep-alive, or HTTP connection reuse, is the idea of using a single TCP connection to send and receive multiple HTTP requests/responses, as opposed to opening a new connection for every single request/response pair.

The newer HTTP/2 protocol uses the same idea and takes it further to allow multiple concurrent requests/responses to be multiplexed over a single connection.

HTTP odgovor

- Počinje redom:
HTTP/verzija kod tekstualni_opis
- dodatni redovi sadrže attribute:
Ime: vrednost
- prazan red
- sledi sadržaj datoteke

"200" ; OK

"201" ; Created

"202" ; Accepted

"204" ; No Content

**"301" ; Moved
Permanently**

**"302" ; Moved
Temporarily**

"304" ; Not Modified

"400" ; Bad Request

"401" ; Unauthorized

"403" ; Forbidden

"404" ; Not Found

**"500" ; Internal Server
Error**

**"501" ; Not
Implemented**

"502" ; Bad Gateway

**"503" ; Service
Unavailable**

Atributi u HTTP odgovoru

- Content-type – definiše tip odgovora

Content-Type: text/html

- Cache-Control – definiše kako se keš na klijentu ažurira
 - koristi se i Pragma: no-cache

Cache-Control: no-cache

- Location – definiše novu adresu kod redirekcije

Location: new.html

- Connection – potvrda klijentu da li da zatvori konekciju ili da je ostavi otvorenu

Connection: Keep-Alive

Primer HTTP odgovora

HTTP/1.0 200 OK

Date: Tue, 04 May 02004 08:55:09 GMT

Status: 200

Servlet-Engine: Tomcat Web Server/3.1 (JSP 1.1; Servlet 2.2; Java 1.4.2_02; Windows XP 5.1 x86; java.vendor=Sun Microsystems Inc.)

Content-Type: text/html

Last-Modified: Fri, 24 Oct 02003 16:07:24 GMT

Content-Length: 2524

Content-Language: en

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
```

```
<html>
```

```
<head>
```

```
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

```
  <meta name="GENERATOR" content="Mozilla/4.72 [en] (WinNT; U) [Netscape]">
```

```
  <meta name="Author" content="Anil K. Vijendran">
```

```
  <title>Tomcat v3.1</title>
```

```
</head>
```

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

```
</html>
```

Primer HTTP odgovora sa redirekcijom

HTTP/1.1 302 Object moved
Server: Microsoft-IIS/5.1
Date: Mon, 26 Apr 2004 17:50:55 GMT
X-Powered-By: ASP.NET
Location: localstart.asp
Connection: Keep-Alive
Content-Length: 135
Content-Type: text/html
Set-Cookie: ASPSESSIONIDGGQQAQAEK=JKKPPFKCMNDNMEEHOHAADJKPM; path=/
Cache-control: private

Ako je close,
konekcija se
zatvara

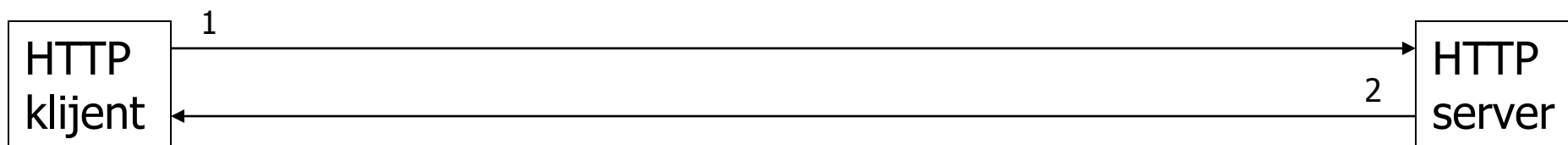
```
<html>
<head>
  <title>Object moved</title>
</head>
<body><h1>Object Moved</h1>This object may be found <a
  HREF="localstart.asp">here</a>.
</body>
</html>
```

Vrste WWW sadržaja

- statički (unapred uskladišteni)
- dinamički (generisani po zahtevu)

Isporuka statičkih sadržaja

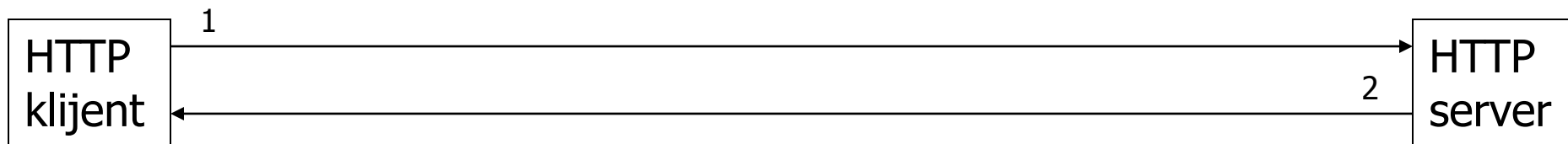
- statički sadržaji se nalaze u okviru datoteka WWW servera



1. kljent zahteva datoteku
2. server je učitava sa svog fajl-sistema i šalje je klijentu

Isporuka dinamičkih sadržaja

- traženi sadržaj se generiše po zahtevu i šalje klijentu



1. klijent zahteva "datoteku"
2. server je generiše i šalje klijentu; ne snima je u svoj fajl-sistem

SETUP

- Napravi novi workspace ImePrezime/Modul2Web
- Pokreni Eclipse EE i postavi putanju na novi workspace
 - Eclipse *New->Dynamic Web Project* (ne može, nedostaje server)
- Otpakujte materijale za ovo predavanje, u njima imate zip arhivu *apache-tomcat-6.0.39.zip*
- Napravi folder ImePrezime/Servers i u njemu otpakujte tomket
- Vraćamo se u Eclipse gde moramo u podešavanjima da postavimo putanju do otpakovanog servera
- Otvori datoteku *Razvoj Web Aplikacija - Eclipse Tomcat Servleti.pdf* i prati uputstva
- Poveži Eclipse sa Tomcat serverom

Prvi Projekat

- *Eclipse New->Dynamic Web Project*
- Odredimo ime projekta npr. *PrviWebProjekat* i odabiremo Runtime Environment *Apache Tomcat v6.0*, pa *Next-> Next*
- *ContextRoot* je ključno jer je to ime preko kojeg se aplikacija poziva iz web browser-a
- U *Content dirctory* folderu čuvamo slike, fajlovi, HTML stranice, CSS stilovi...
- Generate *web.xml* treba da je selektovano, sadrži opis naša web aplikacije za potrebe servera
 - Note: projekat ima ikonicu zemlje

Prvi Projekat

- Stavke projekta ne moraju predstavljati fizičke direktorijume ili datoteke na disku
 - Deployment deskriptor – je u stvarnosti fajl web.xml
 - Jax WS Web Services – ne postoji folder na disku
 - Java resuorces – ne postoji folder na disku
- Struktura foldera
 - src folder – idu java klase
 - WebContent - nalaze Web resursi kao što su slike, fajlovi, HTML stranice, CSS stilovi .
 - META-INF – ne dirati jer je to za jar fajlove
 - WEB-INF – je folder rad web aplikacije. Sadrži relevantne web resurse za funkcionisanje web aplikacije kao što su prekompajlirane java klase (podfolder classes), dodatne biblioteke koje koristimo (folder lib), opis rada web aplikacije (datoteka web.xml)

Servleti _{1/2}

- Tehnologija za generisanje dinamičkih sadržaja
- WWW server se proširuje podrškom za servlete
- Rezultat izvršenja servleta je dinamički kreiran sadržaj

Servleti 2/2

redefinisati
metodu:

doGet(...)
doPost(...)

- klasa koja nasleđuje klasu `HttpServlet`:

```
public abstract class HttpServlet {  
    protected void init(ServletConfig cnf) {}  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        {}  
    protected void doPost(HttpServletRequest request, HttpServletResponse  
        response) {}  
    protected void doPut(HttpServletRequest request,    HttpServletResponse  
        response) {}  
    protected void doHead(HttpServletRequest request, HttpServletResponse  
        response) {}  
    protected void delete(HttpServletRequest request, HttpServletResponse  
        response) {}  
    protected void doOptions(HttpServletRequest request, HttpServletResponse  
        response) {}  
    protected void doTrace(HttpServletRequest request, HttpServletResponse  
        response) {}  
    protected void destroy() {}  
    protected void service(HttpServletRequest request, HttpServletResponse  
        response) {  
        if (request.getMethod().equals("GET"))  
            doGet(request, response);  
        else if (request.getMethod().equals("POST"))  
            doPost(request, response);  
        else if ...  
    }  
}
```

Prvi Projekat

- Da ne bi kopirali kod najbolje je da koristimo wizard
- U projektu kreirati novi paket *webt3.zad01*
- *New->Servlet* dati ime klase npr. *ZdravoSvete*, pa -> *NEXT*
 - Servlet je najobičnija java klasa koja mora da nasledi klasu *HttpServlet* (roditeljska klasa je servlet) i implementira neke metode
- Opciono se može postaviti opis npr. *moj prvi servlet*
- *URL mapping* definiše putanju u web brauzeru preko koje će se preistupati kreiranom servletu (NOTE: ne mora biti isti kao naziv *ZdravoSvete* servlet java klase)
- U seldećem dijalogu selektujem metode koje želimo da implementiramo, *doGet*, *doPost*, pa *Finish*

Prvi Projekat

- Metodi doPost stavimo da poziva doGet metodu doGet(request, response);
- Metoda doGet iskucati kod

```
PrintWriter pw = response.getWriter();  
response.setContentType("text/html");  
pw.println("<html>");  
pw.println("<body>");  
pw.println("<h1>Zdravo svete</h1>");  
pw.println("</body>");  
pw.println("</html>");  
pw.flush();  
pw.close();
```

Prvi Projekat

- Potrebno je aplikaciju postaviti na web server i da taj server pokrenemo
- Prvo da vidimo izgled *apache tomcat* web servera
- Struktura foldera:
 - *bin* - sadrži skripte i exe fajlove koji omogućavaju upravljanje bazičnim radom samog servera (pokretanje i zaustavljanje).
 - Za Windows os iz cmd pokrećemo *startup.bat*
 - Za Linux os iz terminala pokrećemo *sh catalina.sh run*
 - *conf* – sadrži skripte za podešavanje konfiguracije web servera.
 - *lib* – sadrži biblioteke koje koristi tomcat prilikom rada
 - *servlet-api.jar* je ključna biblioteka koja se koristi u *Eclipse* za nasleđivanje *HttpServlet*
 - možemo ubaciti mysql connector jar

Prvi Projekat

- Struktura foldera:
 - *logs* - tomcat upisuje izveštaje tokom svog rada
 - *temp* – se koristi kao pomoćni folder u toku rada servera za web aplikacije, može da se briše sadržaj kada je tomcat isključen
 - *work* – se koristi kao pomoćni folder za mini keširanja, može da se briše sadržaj kada je tomcat isključen
 - *webapps* – najbitiniji folder, zadrži war arhive i web aplikacije
 - war arhiva je zip fajl koji sadrži *class* fajlove i *WebContent*
 - U njemu kopiramo war arhivu koja se raspakuje prilikom pokretanja tomcat servera u foldere koji predstavljaju postavljene web aplikacije na web server

Prvi Projekat

- U Eclipse desni klik na projekat pa *Export->War File*
- *Browse*, odabira se putanja do webapps foldera tomcat
- Selektujte *override existing file*, to se selektuje obavezno svaki naredni put kada se eksportuje war fajl
- Videti sadržaj webapps foldera
- Pokrenuti tomcat i opet videti sadržaj webapps foldera
- Startovati tomcat
- U browser kucate sledeću adresu
 - http://localhost:8080/Ime_Aplikacije/Resurs
 - <http://localhost:8080/PrviWebProjekat/ZdravoSvete>
- Otvori *web.xml* projekta iz Eclipse

HttpServlet.init()

- namenjena za inicijalizaciju prilikom pokretanja servleta

```
public void init() {  
    Connection conn = DriverManager.getConnection(...);  
    ...  
}
```

```
public void init(ServletConfig cnf) {  
    super.init(cnf);  
    Connection conn = DriverManager.getConnection(...);  
    ...  
}
```

HttpServlet.destroy()

- namenjena za clean-up zadatke neposredno pre uništenja servleta

```
public void destroy() {  
    conn.close();  
}
```

HttpServlet.doGet()

- Svaki poziv servleta se svodi na poziv ove metode
- Namenjena za obradu GET zahteva
- Tipičan scenario poziva:
 - postavi Content-type HTTP odgovora
 - uzmi PrintWriter ka klijentu
 - kroz PrintWriter šalji dinamički kreiran sadržaj

```
public void doGet(HttpServletRequest req, HttpServletResponse res) {  
    res.setContentType("text/html");  
    PrintWriter out = res.getWriter();  
    out.println("<HTML>");  
    out.println("<HEAD><TITLE>Test</TITLE></HEAD>");  
    out.println("<BODY>");  
    ...  
}
```

HTTP zahtev (klasa HttpServletRequest)

- Reprezentuje HTTP zahtev
- Izdvaja parametre forme prenete GET ili POST metodom i smešta ih u asocijativnu listu (naziv_polja_iz_forme, vrednost)
 - metode `getParameter(ime)`, `getParameterNames()`, `getParameterMap()`
- Prikuplja sve parametre zaglavlja HTTP zahteva i smešta ih u asocijativnu listu (naziv, vrednost)
 - metode `getHeader(ime)`, `getHeaderNames()` i `getHeaders(ime)`

HTTP odgovor (klasa HttpServletResponse) ^{1/4}

- Reprezentuje HTTP odgovor
- Čuva tip odgovora (atribut Content-Type)
 - metoda `setContentType(vrednost)`
- Čuva *cookie* (atribut SetCookie)
 - metoda `addCookie(cookie)`
- Omogućuje redirekciju (Location)
 - metoda `sendRedirect(nova_lokacija)`
- Podešava proizvoljan atribut zaglavlja
 - metoda `setHeader(naziv, vrednost)`
- Ugrađuje ID sesije ako cookies nisu uključeni
 - metode `encodeURL(url)` i `encodeRedirectURL(url)`
- Čuva izlazni tok podataka

Primer: elementarni servlet

```
public class TestServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
        HttpServletResponse response) {  
        response.setContentType("text/html");  
        PrintWriter pout = response.getWriter();  
        pout.println("<html>");  
        pout.println("<head>");  
        pout.println("</head>");  
        pout.println("<body>");  
        pout.println("Hello World!");  
        pout.println("<br>Klijent koji je pozvao ovaj  
servlet je: " + request.getHeader("User-Agent"));  
        pout.println("</body>");  
        pout.println("</html>");  
    }  
}
```


Preuzimanje podataka iz formi

- Parametri iz forme se za GET metodu smeštaju u zaglavlje GET zahteva.
- HTML kod na klijentu

accept-charset="utf-8"

```
<form method="get" action="FormServlet">  
  <input type="text" name="tekst_polje">  
  <input type="submit" value="Posalji">  
</form>
```

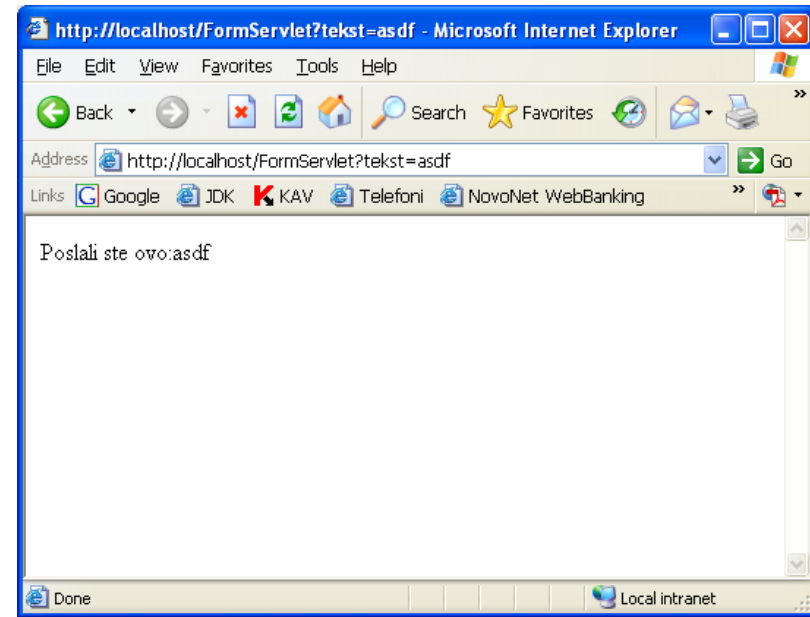
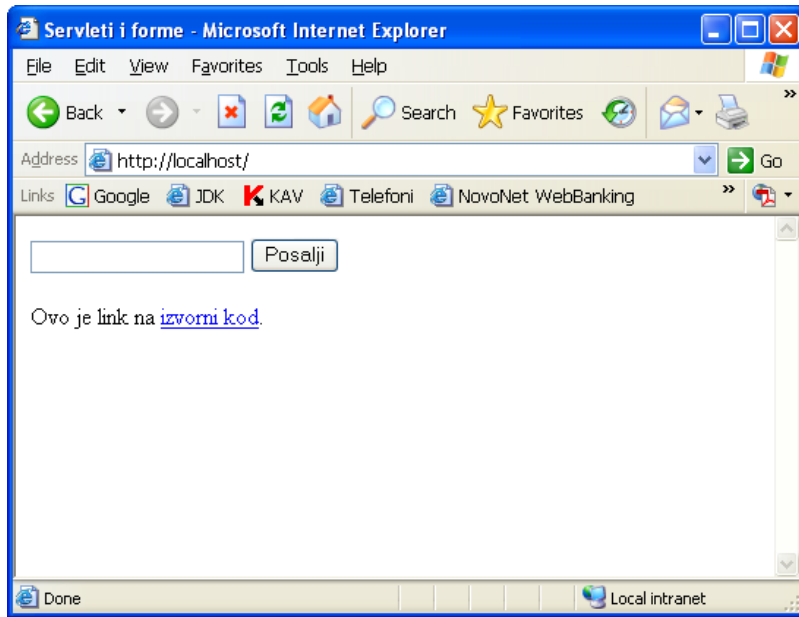
inputFormGet.html

- GET HTTP zahtev:

inputFormPost.html

GET /FormServlet?tekst_polje=asdf HTTP/1.1

Preuzimanje podataka sa formi



GET i POST zahtevi

Kod GET metode se parametri forme nalaze u heder delu HTTP request poruke

```
GET /FormServlet?tekst=asdf HTTP/1.1
```

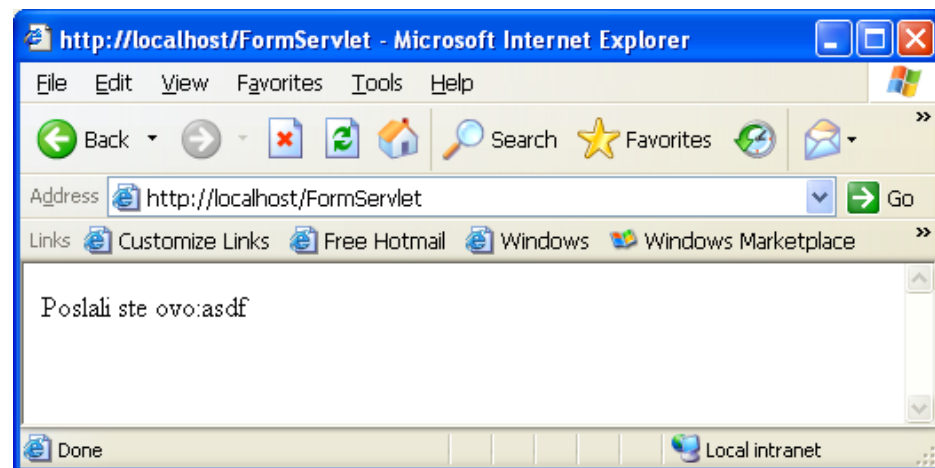
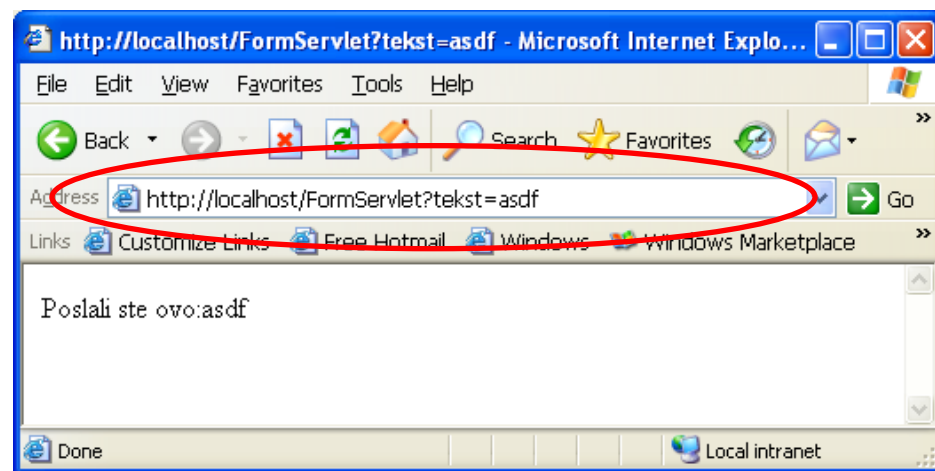
Kod HTTP **GET** metode, posle znaka ? zapisuju se vrednosti URL promenljivih tipa *ključ=vrednost&ključ=vrednost...*

```
POST /FormServlet HTTP/1.1
```

```
Content-length: 10
```

```
tekst=asdf&kljuc=vrednost
```

Kod HTTP **POST** metode se vrednosti parametara forme smeštaju na kraju HTTP zahteva (posle praznog reda), i posle vrednosti parametara nema “\r\n” karaktera na kraju reda



Kod POST metode se parametri forme nalaze u body delu HTTP request poruke

Primer: servlet koji ispisuje parametar unet u formi

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response) {
    response.setContentType("text/html");
    PrintWriter pout = response.getWriter();
    pout.println("<html>");
    pout.println("<head>");
    pout.println("</head>");
    pout.println("<body>");
    pout.println("Poslali ste ovo:" +
        request.getParameter("tekst_polje"));
    pout.println("</body>");
    pout.println("</html>");
    pout.flush();
}
```

Zadatak 2

- Sa 1 časa (HTML) kopiramo forma.html datoteku u folder *WebContent*
- action atribut forme treba da na nešto pokaže, na nekog ko će obraditi podatke
- Kreiraj servlet PrihvatanjePodataka
- U doGet metodi obrađujemo zahtev
request.getParameter("ime")
request.getParameter("prezime")
- U doGet metodi formirajte html odgovor tako da se vrati tekst koji ispisuje ime

Character Encoding

- Metodom `setContentType` se podešava i *character encoding*:

```
response.setContentType("text/html; charset=UTF-8");
```

- Parametar *charset* definiše kodnu stranu kojom će biti kodirani svi stringovi ka klijentu.

Primer: servlet sa UTF-8 encoding-om

```
public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws java.io.IOException {
    response.setContentType("text/html; charset=UTF-8");
    PrintWriter pout = response.getWriter();
    pout.println("<html>");
    pout.println("<head>");
    pout.println("<meta http-equiv=\"Content-Type\"
        content=\"text/html; charset=UTF-8\">");
    pout.println("</head>");
    pout.println("<body>");
    try {
        pout.println("Ovo je stranica sa UTF-8 karakterima: \u0428
            \u0429<br>");
    } catch (Exception ex) {
        pout.println(ex.getMessage());
    }
    pout.println("</body>");
    pout.println("</html>");
    pout.flush();
}
```

Character encoding i parametri forme

- **form** tag, atribut **accept-charset**
- Metoda `request.getParameter()` ne ume da "proceni" u kojoj kodnoj strani stižu podaci
 - informacija o kodnoj strani ne postoji u `HttpServletRequest` klasi
- Mora se eksplicitno podesiti:

```
response.setContentType("text/html;  
charset=utf-8");  
request.setCharacterEncoding(  
response.getCharacterEncoding());
```


Servlet Context

- Deljena memorija za sve servlete.
- Moramo upisati podatak u tu memoriju:

```
Objekat objekat;
```

```
getServletContext().setAttribute("ime_tributa", objekat);
```

Ime atributa mora biti jedinstveno

- Čitanje atributa iz Servlet Context:

```
Objekat temp=(Objekat)getServletContext().getAttribute("ime_tributa")
```

!!! ime atributa mora biti ono koje ste prethodno setovali

Zadatak 3

- Kreirati paket model i u njemu klasu Osoba sa poljima ime i prezime, konstruktorima, get i set metodama
- U servletu PrihvatanjePodataka svaki put kada očitavaš ime i prezime kreiraj objekat klase Osoba(ime,prezime)
- U doGet metodi formirajte html odgovor tako da se vrati tekst koji ispisuje podatke osobe

Zadatak 4

- U servletu *PrihvatanjePodataka* kreirajte listu osoba
- Svaki put kada kreiraš objekat osoba ubaci objekat u listu osoba
- Kreiraj novi servlet *IspisSvihOsoba* koji prikazuje prikazuje listu osoba
- Metoda doGet treba da HTML koji omogućuje prikaz svi osoba
- Ubaci anchor tag u HTML servleta *PrihvatanjePodataka* ka *IspisSvihOsoba*
- Koristi for petlju da bi ispisao listu osoba iz servleta *PrihvatanjePodataka*

Zadatak 4

- Izmeni *PrihvatanjePodataka* lista osoba nije statička
- Postavi listu osoba u servlet contex
`getServletContext().setAttribute("osobe", osobe);`
- U klasi *IspisSvihOsoba* očitaj podatke iz servlet contex
`ArrayList<Osoba> osobe = (ArrayList<Osoba>) getServletContext().getAttribute("osobe");`
- Ubaci anchor tag ka forma.html

Dodatni materijal

Uniform Resource Locator (URL)

- standard definisan u IETF RFC 1738 predstavlja podatke koji se mogu upotrebiti za dobavljanje resursa
- URL format - protokol:putanja-do-resursa
- Pokretanje web aplikacije u servletskoj tehnologiji se radi pomoću URL resurs.

Protokol://Adresa_računara:port/Dinamički ili statički resurs

Primeri

<http://example.com:80/pictures> - Simbolička adresa i dinamički adresa
NAPOMENA 80 je podrazumevani port za http protokol

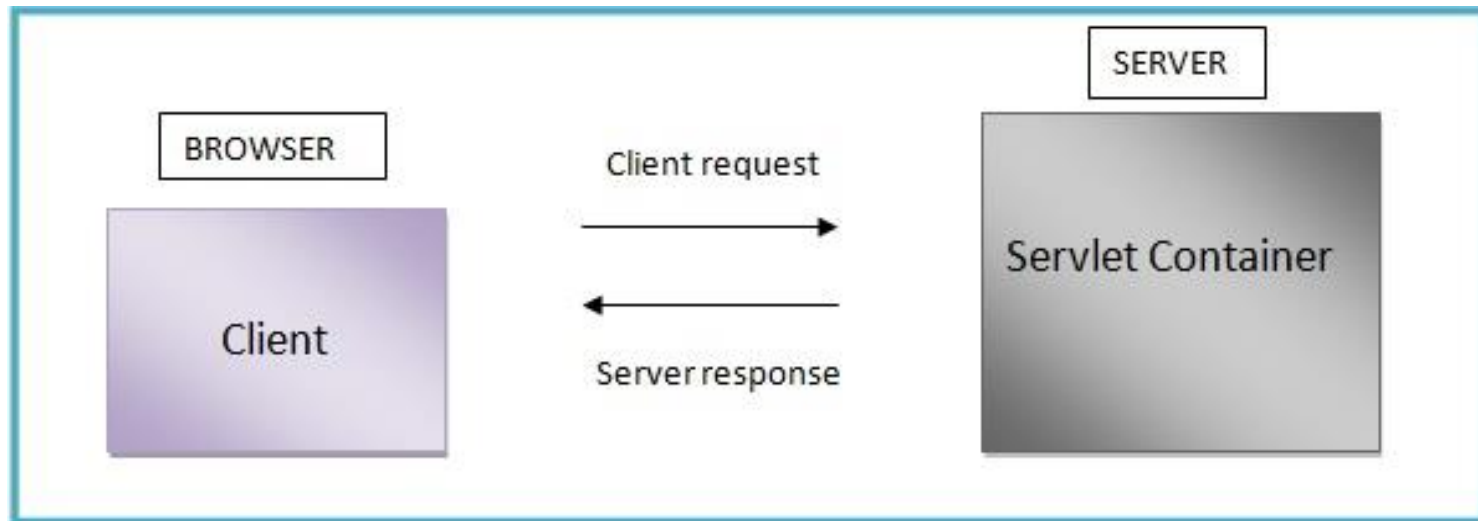
<http://192.168.0.1:4203/help.html> – Numerička adresa i statički resurs

Port

- U računarskoj mreži *port* je softverski zadat kanal kojim komuniciraju aplikacije putem računarskih mreža. Predstavlja broj u opsegu 0-63535. Neki od ovih brojeva su predifinisani (0-1023) dok ostale portove mogu da koriste korisničke aplikacije.
- Portom razlikujemo aplikacije na računarima u domenu mrežne komunikacije.

Servlet

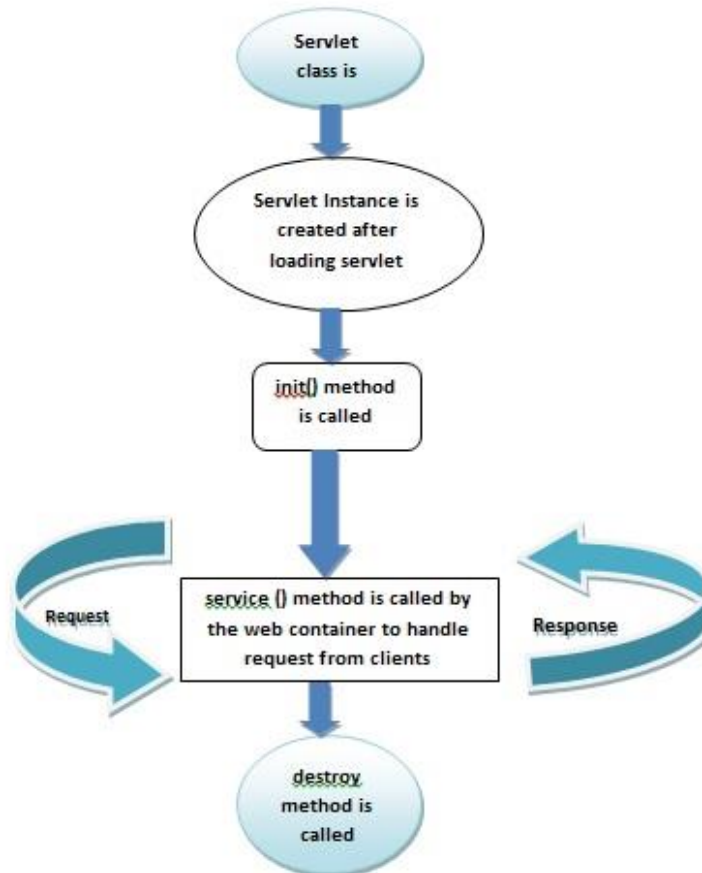
- Servlet is a Java programming language class, part of Java Enterprise Edition (Java EE). Sun Microsystems developed its first version 1.0 in the year 1997. Its current Version is Servlet 3.1.



Servleti

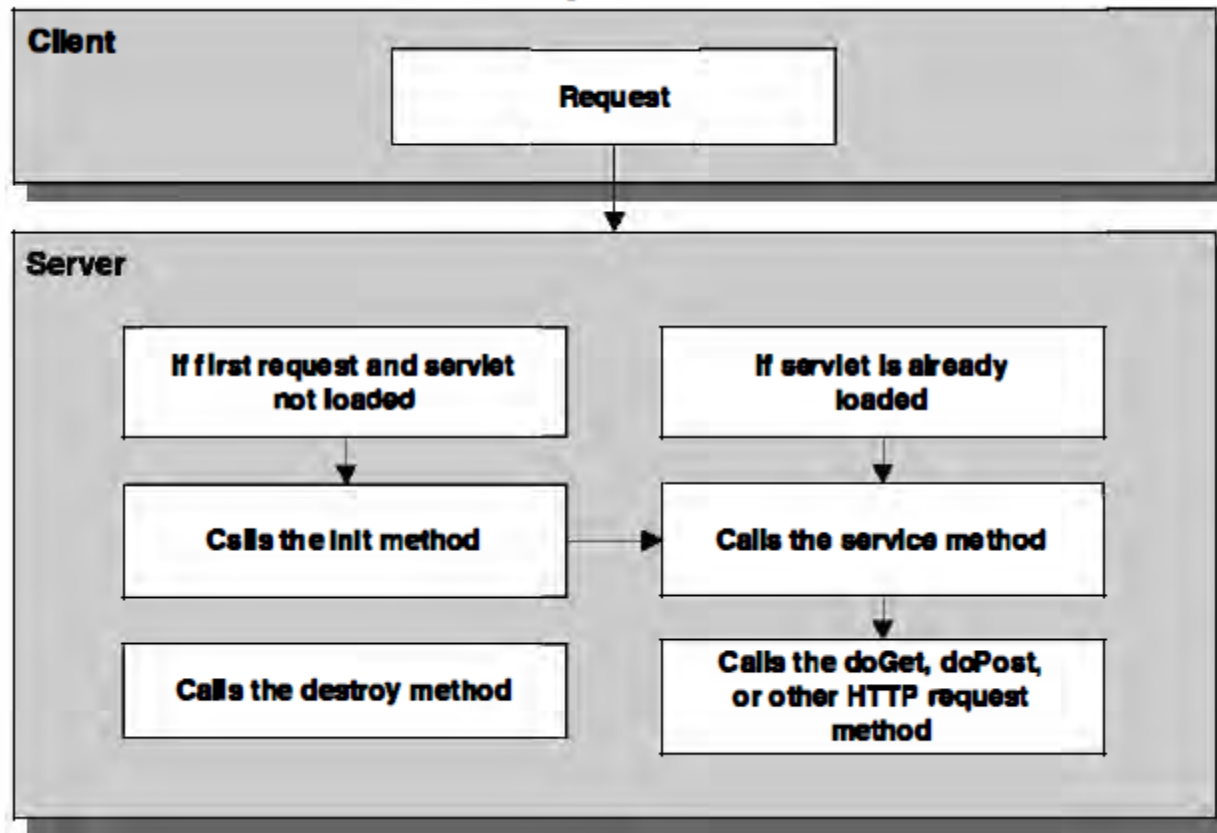
- A Request is sent by a client to a servlet container. The container acts as a Web server.
- The Web server searches for the servlet and initiates it.
- The client request is processed by the servlet and it sends the response back to the server.
- The Server response is then forwarded to the client

Servlet lifecycle

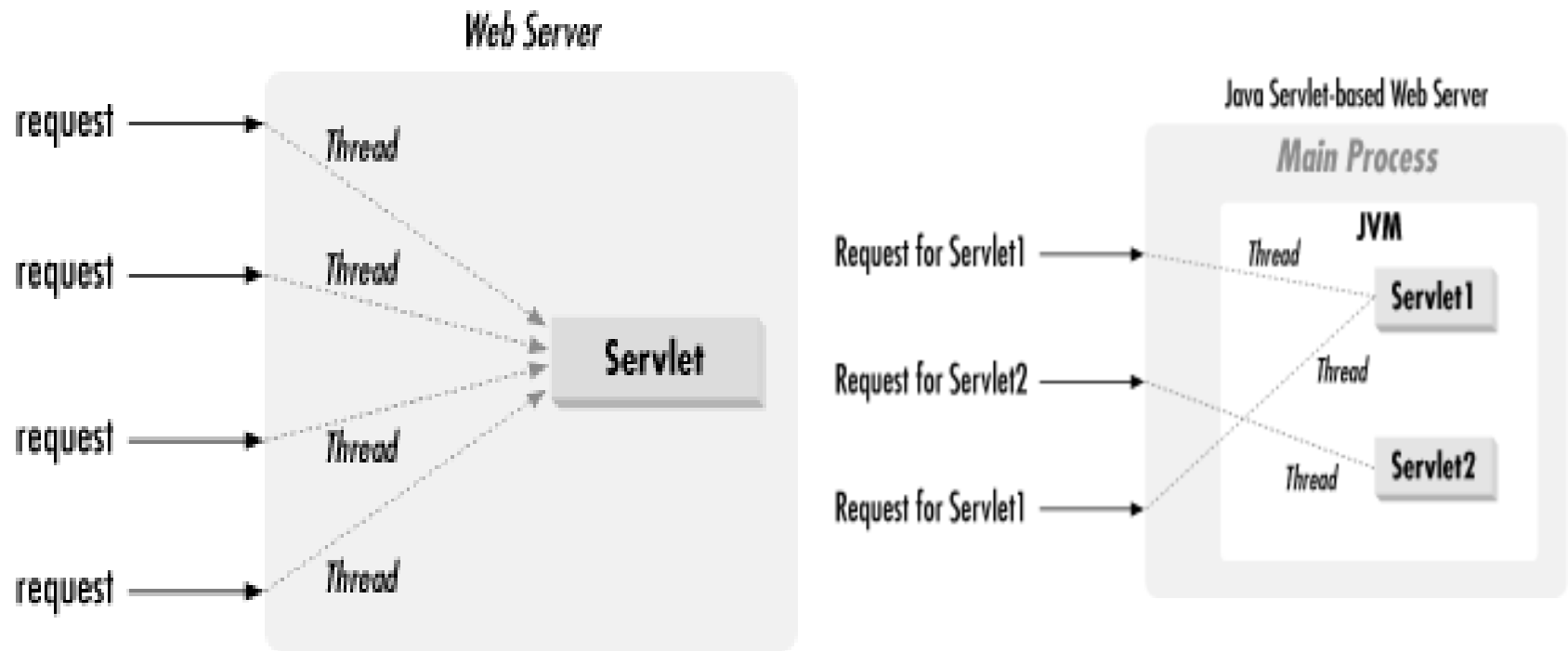


Servlet lifecycle

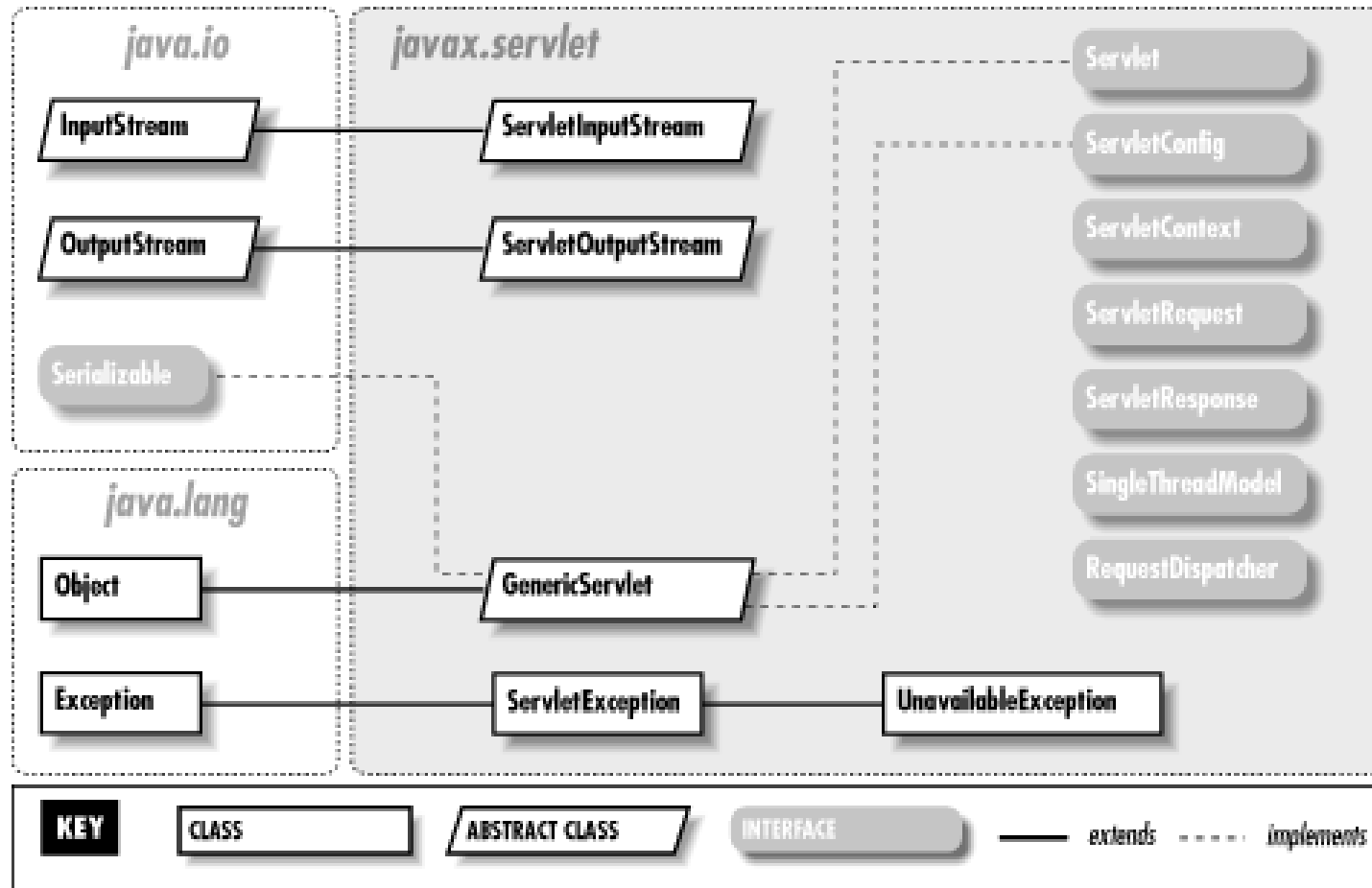
How the server handles a request for a servlet



Servlet lifecycle



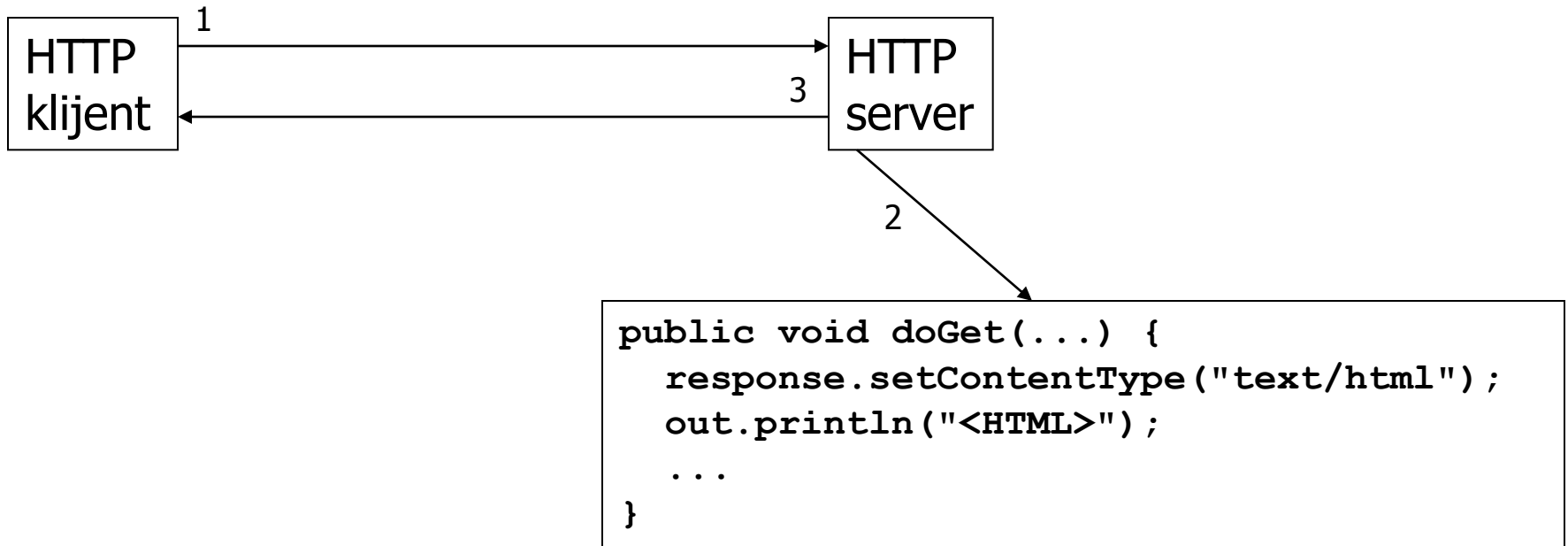
Servlet lifecycle



Konkurentni pristup servletu

- za svaku servlet klasu instancira se tačno jedan objekat koji opslužuje sve klijente
- njegove `doGet()` i `doPost()` metode mogu biti istovremeno pozvane iz više programskih niti Web servera
- atributi predstavljaju potencijalni problem, pošti ih dele niti
 - postoje "sigurni" repozitorijumi u koje će se smeštati deljene stvari: aplikacija, sesija, strana i zahtev

Generisanje dinamičkih sadržaja



Primer: elementarni servlet

```
public class TestServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
        HttpServletResponse response) {  
        response.setContentType("text/html");  
        PrintWriter pout = response.getWriter();  
        pout.println("<html>");  
        pout.println("<head>");  
        pout.println("</head>");  
        pout.println("<body>");  
        pout.println("Hello World!");  
        pout.println("<br>Klijent koji je pozvao ovaj  
servlet je: " + request.getHeader("User-Agent"));  
        pout.println("</body>");  
        pout.println("</html>");  
    }  
}
```


HTTP zahtev

- Počinje redom:
METHOD /putanja HTTP/verzija
- METHOD je:
 - GET,
 - POST,
 - HEAD,
 - PUT,
 - DELETE,
 - OPTIONS,
 - TRACE.
- dodatni redovi sadrže attribute oblika:
Ime: vrednost
- prazan red na kraju

HTTP zahtev (klasa HttpServletRequest) ^{2/3}

```
public HttpServletRequest(InputStream is) {  
    ...  
    BufferedReader rdr  
        = new BufferedReader(new InputStreamReader(is));  
    // pokupimo prvi red iz http zahteva  
    String s = rdr.readLine();  
    String[] tokens = s.split(" ");  
    // pokupimo METHOD  
    method = tokens[0];  
    // pokupili smo METHOD, pa je sledeći token putanja do resursa  
    String rsrc = tokens[1];  
    //izbacimo vodeći '/' znak  
    rsrc = rsrc.substring(1);  
    // izdvojimo parametre GET metode forme (ako ih ima),  
    // a ostatak je putanja do resursa  
    resource = extractGetParameters(rsrc);  
    // iščitamo zaglavlje http zahteva i popunimo asocijativnu  
    listu  
    // parametara iz zaglavlja  
    readHeader(rdr);  
    ...  
}
```

HTTP odgovor (klasa HttpServletResponse) 2/4

```
public HttpServletResponse(OutputStream out) {  
    outputStream = out;  
    writer = new PrintWriter(new OutputStreamWriter(out), true);  
}  
  
private PrintWriter writer = null;  
public PrintWriter getWriter() {  
    return writer;  
}  
  
private OutputStream outputStream = null;  
public OutputStream getOutputStream() {  
    return outputStream;  
}  
  
private String location;  
public void sendRedirect(String url) {  
    location = url;  
}
```

HTTP odgovor (klasa HttpServletResponse) 3/4

```
private String contentType = null;
private String getEncoding(String s) {
    String retVal = null;
    String[] tokens = s.split(";");
    if (tokens.length == 2) {
        String token = tokens[1].trim();
        int idx = token.indexOf("=");
        if (idx != -1 && token.substring(0,idx).equals("charset")) {
            retVal = token.substring(idx+1);
        }
    }
    return retVal;
}

public void setContentType(String c) {
    // podesi tip povratne datoteke i...
    contentType = c;
    if (c != null) {
        String encoding = getEncoding(c);
        if (encoding != null) {
            try {
                writer = new PrintWriter(new OutputStreamWriter(outputStream,
                                                                    encoding), true);
            } catch (Exception ex) {}
        }
    }
    // posalji zaglavlje HTTP protokola ka klijentu
    sendHeader();
}
```

HTTP odgovor (klasa HttpServletResponse) 4/4

```
private void sendHeader() {  
    // pošaljemo HTTP zaglavlje  
    if (location == null)  
        writer.print("HTTP/1.0 200 OK\r\n");  
    else {  
        writer.print("HTTP/1.0 302 Object moved\r\n");  
        writer.print("Location: " + location + "\r\n");  
    }  
    if (contentType != null)  
        writer.print("Content-type: " + contentType + "\r\n");  
    if (cookie != null)  
        writer.print("Set-Cookie: " + cookie + "\r\n");  
    writer.print("\r\n");  
    writer.flush();  
}
```

Izdvajanje parametara iz formi (klasa HttpServletRequest)

```
private String extractGetParameters(String rsrc) {  
    String[] tokens = rsrc.split("\\?");  
    // ako imamo parametre forme  
    if (tokens.length == 2) {  
        // zapamtimo prvi deo, tj. "putanju", jer cemo to  
        // vratiti  
        String retVal = tokens[0];  
        // uzmemo parametre  
        String s = tokens[1];  
        paramMap.clear();  
        putInParamMap(s);  
        return retVal;  
    } else  
        return rsrc;  
}
```

Izdvajanje parametara iz formi (klasa HttpServletRequest)

```
private void putInParamMap(String params) {  
    // izdelimo ih na pojedinačne parove "ime=vrednost"  
    String[] tokens = params.split("&");  
    for (String s : tokens) {  
        int idx = s.indexOf("=");  
        // levo od '=' je ime  
        String pName = s.substring(0, idx).trim();  
        // desno od '=' je vrednost  
        String pValue = s.substring(idx + 1).trim();  
        paramMap.put(pName, pValue);  
    }  
}
```

Pristup parametrima forme (klasa `HttpServletRequest`)

```
/** Svi parametri iz forme se smeštaju u
 * asocijativnu mapu.
 */
private HashMap<String, String> paramMap = new
    HashMap<String, String>();
public String getParameter(String name) {
    return paramMap.get(name);
}
```


Primer: servlet sa UTF-8 encoding-om

```
public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws java.io.IOException {
    response.setContentType("text/html; charset=UTF-8");
    PrintWriter pout = response.getWriter();
    pout.println("<html>");
    pout.println("<head>");
    pout.println("<meta http-equiv=\"Content-Type\"
        content=\"text/html; charset=UTF-8\">");
    pout.println("</head>");
    pout.println("<body>");
    try {
        pout.println("Ovo je stranica sa UTF-8 karakterima: \u0428
            \u0429<br>");
    } catch (Exception ex) {
        pout.println(ex.getMessage());
    }
    pout.println("</body>");
    pout.println("</html>");
    pout.flush();
}
```