Java és web programozás

Kovács Kristóf, Rimay Zoé

Budapesti Műszaki Egyetem

2013. november 6.

8. Előadás

Grizzly Szerver

- A *Project Grizzly* HTTP szerver keretrendszerét fogjuk haszálni.
- Dokumentácó és egyéb információk elérhetők a *https://grizzly.java.net/* oldalon.
- Sokkal többre képes ez a rendszer, mint amennyit mi fogunk belőle használni.

Grizzly Szerver

- A *Project Grizzly* HTTP szerver keretrendszerét fogjuk haszálni.
- Dokumentácó és egyéb információk elérhetők a *https://grizzly.java.net/* oldalon.
- Sokkal többre képes ez a rendszer, mint amennyit mi fogunk belőle használni.
- Könnyen átlátható, egyszerű módon működik. Minden oldalhoz egy *HttpHandler*t rendelünk majd hozzá, ami a kapott információk alapján eldönti, hogy milyen választ küldjön a felhasználónak (böngészőnek).

Üres példa

```
HttpServer server = HttpServer.createSimpleServer();
try {
    server.start();
    System.out.println("Press any key to stop the server");
    System.in.read();
} catch (Exception e) {
    System.err.println(e);
}
```

Üres példa

```
HttpServer server = HttpServer.createSimpleServer();
try {
    server.start();
    System.out.println("Press any key to stop the server");
    System.in.read();
} catch (Exception e) {
    System.err.println(e);
}
```

Ebben a példában nem történik semmi, nem lesz semmi az oldalunkon, csak elindítjuk a webszervert.
Feltételezzük, hogy ez a kód (és a továbbiak) valahol egy *main*ben vagy valami olyan helyen van ahol biztosan lefutnak.

Üres példa magyarázata

- A *createSimpleServer* metódus egy előre konfigurált szervert hoz létre.
- Ehelyett mást fogunk használni, hogy személyre szabottabb legyen. De a példa erejéig ez jó lesz.
- *start*al indítjuk a szervert.
- Ez a *HttpServer* mindent tartalmaz a webszerverünkkel kapcsolatban. Neki fogjuk megmondani, hogy melyik oldal mit es hogyan csináljon.

Töltelék

```
...
HttpServer server = HttpServer.createSimpleServer();
server.getServerConfiguration().addHttpHandler(
    new HttpHandler() {
        public void service(Request request,
                Response response) throws Exception {
            final String respString = "Elso oldal";
            response.setContentType("text/plain");
            response.setContentLength(respString.length());
            response.getWriter().write(respString);
        }
    },
    "/time");
try {
...
```

Ez az előző üres példa kiegészitve.

Töltelék magyarázata

- A *server.getServerConfiguration().addHttpHandler(...)* parancssorral megadhatunk egy *HttpHandler* objektumot, aminek egy metódust kell igazán tartalmaznia, a *service*-t.
- A *service* kap egy *Request* és egy *Response* objektumot.
- A *request*ben vannak az információk amiket a böngésző küldött a webszerverünknek. Míg a *response*ba kell beleírnunk, hogy milyen választ akarunk neki küldeni.
- A *setContentType*-al tudjuk megmondani, hogy milyen tartalmat küldünk, jelen esetben nem htmlt, hanem egy egyszerű szöveget.
- A *setContentLength*el meg kell adnunk a küldött tartalom méretét.
- Végül a *response.getWriter().write(respString)* parancssor beleírja a válaszba (response-ba) a kívánt szöveget.
- Legvégül, miután megadtuk a *HttpHandler*t, meg kell még adnunk, hogy hova *kösse be* ezt az oldalt, jelen esetben a /time címre.

Még egy kis magyarázat

- Tehát így működik a válasz oldal. Fogjuk a *Response* objektumot és feltöltjük tartalommal.
- A *text/plain* helyett a *setContentType*-be majd *text/html*-t fogunk írni amikor htmlt küldünk a böngészőnek.
- Sütiket is ilyen egyszerűen tudunk majd küldeni.
- A *setContentLength*el vigyázzunk, mert ha rosszul adjuk meg a méretet, akkor a megadott méretű részét küldi csak el az oldalnak, és ez html kód esetén elég végzetes lehet.

Request példa

Tegyük fel, hogy a következő *form* található egy html oldalunkban:

```
<form action="tovabb" method="get">
  <label>User: <input type="text"
                      name="username" value=""></label>
  <label>Password: <input type="password"
                      name="password"></label>
  <button type="submit">Login</button>
</form>
```

Request példa

Tegyük fel, hogy a következő *form* található egy html oldalunkban:

```
<form action="tovabb" method="get">
  <label>User: <input type="text"
                      name="username" value=""></label>
  <label>Password: <input type="password"
                      name="password"></label>
  <button type="submit">Login</button>
</form>
```

- Ez annyit tesz, hogy ha beírunk valamit az *input* mezőkbe, majd megnyomjuk a Login gombot, akkor ugrik a *tovabb* oldalra, amit a *form action* attribútumában adtunk meg.
- Valamint ezen az oldalon ahova ugrottunk láthatók lesznek a beírt adatok, valahogy igy:

```
/tovabb?username=Tofi&password=kutya
```

Request példa tovább

- Tehát a *tovabb* oldal megkapja az előző oldalba beírt adatokat.
- Így ha kapcsolunk egy *HttpHandler*t a *tovabb*hoz, akkor az ő *request* objektumában benne lesz ez a két paraméter.
- Ezeket a következő módon tudjuk lekérni:

```
request.getParameter("parameterNeve");
```

- Itt a *parameterNeve* az adott paraméter neve, jelen esetben a *username* és *password*, amiket az input *name* attribútumában adtunk meg.
- Tehát a mostani esetünkben ezt irhatnánk pl:

```
String user = request.getParameter("username");
String pass = request.getParameter("password");
```

Még pár megjegyzés a requesthez

- A paraméterek értékét mindig *String*ként kapjuk meg. Még akkor is ha számokat írtunk az adott mezőbe.
- Ha ugyanazzal a névvel két paraméter is van az oldalon, akkor a *getParameter(...)* a legelsőt kéri le.
- A *request* objektumon belül a paramétereken kívül rengeteg más dolog is van. Például az oldalunkhoz tartozó sütik is benne vannak. Valamint a böngésző adatai ami a kérést küldte.
- Le lehet kérni az összes paramétert a *request.getParameterMap()* metódussal, mely egy *Map*et ad vissza. Ebben a duplikált elemek is benne vannak.
- Lekérhetjük a teljes query Stringet (? utáni részt) a *request.getQueryString()* metódussal.