
# Struts keretrendszer

- Már a megjelenésük kezdetén bebizonyosodott a **szervletek** hasznos volta.
- A CGI-vel szemben gyorsabbak voltak, hatékonyabbak, hordozhatók és bővíthetők.
- A HTML kód beágyazása `println()` metódusokon keresztül fárasztó volt és problematikus.
- A választ erre a **JSP** adta meg, mely a szervlet írást "kifordította".
- A fejlesztők simán keverhették a HTML kódot java kóddal, megtartva a szervlet összes előnyét.
- A java alapú webalkalmazások először "JSP"-centrikusak lettek, azaz keveset tettek a folyamatvezérlés megoldására. Más modellre volt szükség.

- Rájöttek, hogy a JSP-k és szerveltek együtt jól használhatók a web-alkalmazásokban: a szerveletek gondoskodnak a folyamatvezérlésről a JSP-k pedig a HTML kód létrehozásáról.
- Ezt a modellt nevezték el `Model2`-nek (A jsp-k kizárólagos alkalmazása volt a `Model1`).
- Ez a `Model2` nagyon hasonlít a klasszikus **MVC** modellhez (*Model-View-Controler*), és ma már ugyanarra a modellre mindkét nevet használják.

A **Struts** keretrendszer megvalósítja az MVC tervezési mintát.

## Model1 Architektúra

Ebben az arhitektúrában minden erőforrás foglalkozhat midennel: megjelenítés, vezérlés és modell (üzleti logika).

## Modell 2 Architektúra (vagy MVC)

Célja, hogy a logikailag elkülöníthető részek ne keveredjenek az alkalmazásban. Ez fontos például a későbbi kódmódosítások valamint az újrafelhasználás miatt. A JSP önmagában kevés ahhoz, hogy MVC alkalmazásokat hozzunk létre.

## Az MVC részei:

- M - modell vagy üzleti logika. Elvégez valamit, de nem mondja meg hogy az adatokat hogyan kell megjeleníteni.
- V - megjelenítés vagy nézet (view): a modelltől kapott adatokat megformázza és megjeleníti, valamint a felhasználó által bevitt adatokat átadja a modellnek, azok értelmezése nélkül. A megjelenítés tipikusan JSP (szöveges output) vagy szervletek (bináris output) segítségével történik, de bármi lehet. Az MVC szempontjából a JSP csak egy megjelenítő réteg a sok között.
- C - vezérlés (controller): tipikusan egy szervlet, amelyiket a `web.xml`-ben kell megadni, és amely ellátja a vezérlési teendőket:
  - feldogozza a HTTP kérést, és annak alapján eldönti, hogy milyen üzleti folyamatot kell elvégezni, majd segít kiválasztani a következő nézetet.

Az MVC-ben a struts a vezérlés réteget valósítja meg és a megjelenítés réteget saját elemkönyvtárakkal segíti. A modell réteg megvalósítása a struts szempontjából lényegtelen, lehet az `Action` osztályokba megvalósított logika vagy akár EJB-k segítségével.

# MVC tervezési minta

- A vezérlés egy központi kontrolleren keresztül történik. A kontroller a kéréseket (request) a megfelelő kezelőhöz (handler) irányítja.
- A kezelők a modellhez kötődnek, és mindegyik kezelő egy adapterként viselkedik a kérés és a modell között.
- A modell az alkalmazás üzleti logikáját valósítja meg.
- Az üzleti logika elvégzése után a vezérlés a kontrolleren keresztül visszakerül a megfelelő nézetre (JSP).
- Ez a bizonyos nézet map-elések alapján lesz kiválasztva, amely konfigurációs fájlban van megadva.
- Ez egy igen laza kapcsolatot biztosít a nézet és a modell között, mely megkönnyíti az alkalmazások létrehozását és karbantartását.

# Modell: rendszer-állapot és logikai bean-ek

- Az üzleti logikát megvalósító részt OOP modellezéssel interfészekbe és osztályokba (metódusok és mezők) foglaljuk, illetve meghatározzuk a kapcsolatokat közöttük.
- A modell rész gyakran két részre osztható:
  - a modell belső állapota
  - az állapot megváltoztatását célzó műveletek

## 1. belső állapot

Sok alkalmazás a belső állapotát `JavaBean`-ekben tárolja, melyek

- tudják, hogy hogyan mentsék le saját állapotukat, vagy
- façade-ok, melyek tudják, hogyan nyerjék ki az állapotukat más komponensekből.
  A másik komponens lehet adatbázis, kereső motor, Entity EJB, vagy bármi más.

## 2. műveletek

- Nagyobb alkalmazások a lehetséges üzleti műveleteket metósdusokként jelenítik meg, melyek meghívhatók a bean-ekre.
- Ezek lehetnek például *Session EJB*-k.
- Kisebb alkalmazások esetén ezek a műveletek bele lehetnek ágyazva az `Action` osztályokba, melyek részei a `Struts` kontroller rétegnek. Ez egészen kicsi alkalmazások esetében használatos, ahol az üzleti logika újrafelhasználása más környezetben nem követelmény.

# A nézet: JSP-k és más komponensek

- A nézet részt általában JSP-ken keresztül valósítjuk meg.
- A struts tartalmaz egy pár saját elemkönyvtárat, melyek együttműködnek az `ActionBean`-ekkel

# Vezérlés (kontroller)

## A kontroller rész

- fogadja a bögészőből jövő hívásokat
- eldönti, hogy melyik üzleti logika rész kell lefusson,
- az üzleti logika lefutása után segít kiválasztani a következő nézetet

# Struts folyamatvezérlés

A Struts több komponenst biztosít a kontroll réteg megvalósítására:

- egy kontroller szervletet, a fejlesztő által megírt kérés kezelőket (request handler) és sok támogató objektumot.

- A Struts-os saját elemkönyvtárak közvetlen módon támogatják a nézet réteget.
- Természetesen más elemkönyvtárakat (pl. JSTL) is használhatunk a Struts-al. A JSP-n kívül más nézet-tehnológiák is használhatók Struts-al, (pl. `Velocity Template`-ek valamint `XSLT`).
- A modell réteg mindig projekt-specifikus.
- A Struts megkönnyíti az üzleti logika elérését, de azt a részt másokra hagyja: pl. JDBC, EJB, perszisztens keretrendszerek stb.

## Hogyan is működik ez az egész együtt:

- Inicializáláskor a kontroller elemzi (parse) a konfigurációs fájlt (`struts-config.xml`) és felhasználja a kontroll réteghez tartozó objektumok létrehozására (**pl.** `ActionMappings`).
- A kontroller szervlet ezen `ActionMappings` objektumok alapján továbbítja a HTTP kéréseket a keretrendszer más komponenseihez.
- A kérés továbbítható közvetlenül egy JSP-hez vagy egy `Action` osztályhoz, amit a Struts fejlesztő ír meg.
- Leggyakrabban a kérés egy `Action`-hoz továbbítódik, majd innen egy JSP-hez.
- A map-elések segítségével fordítja át a kontroller a HTTP-kéréseket Action-okra.

## Egy `ActionMapping` az alábbiakból áll:

- Kérés út (Request path)
- `Action` osztály, amelyik a kérést (request) feldolgozza
- Más tulajdonságok.

- Egy `Action` objektum feldolgozza a kérést és válaszol a kliensnek (böngésző), vagy átirányít egy másik oldalra.
  **Pl.** ha a bejelentkezés sikerül, akkor atirányít a főoldalra.
- Átirányításkor betehet objektumokat (`JavaBean`-eket) a standard névterekbe (hatókör), melyek más `Action` osztályokból, jsp-kből stb. elérhetők.
- Egy Struts alkalmazásban az üzleti logika nagy részét JavaBean-ek segítségével jeleníthetjük meg.
- Egy `Action` osztály meghívja a JavaBean-ek metódusait/tulajdonságait anélkül, hogy tudná, hogy az hogy működik (az üzleti logika nincs benne az `Action` osztályban).
- A bean tartalmazza az üzleti logikát és az `Action` a hibakezelésre és a kontroll-továbbításra koncentrálhat.

- A JavaBeans-ek használatosak a beviteli form-ok kezelésére is.
- Web-alkalmazások esetén fontos a felhasználó által bevitt adatok megtartása és ellenőrzése (validate).
- Struts-ban saját beviteli bean osztályokat definiálhatunk, az `org.apache.struts.action.ActionForm` osztály kibővítése által.
- Ez megkönnyíti a form-ba bevitt adatok tárolását és ellenőrzését.
- Az `ActionForm` automatikusan a `request` vagy a `session` névtérbe lesz lementve, és elérhető lesz az `Action` osztályokból vagy a JSP-kből.
- Egy form bean-t használhat:
  - egy JSP, hogy adatokat gyűjtsön be a felhasználótól,
  - egy `Action` osztály pl., hogy ellenőrizze a bevitt adatokat, lementse őket,
  - majd ismét egy JSP pl., hogy újratöltse a form mezőket.
- Ellenőrzési hiba esetén a Strutsnak van egy mechanizmusa a megfelelő hibagenerálásra ill. annak a megmutatására a JSP-ben.

Amikor a kérés egy `ActionForm`-ot tartalmazó map-elésnek felel meg, az események sorozata a következő:

- a kontroller szervlet kinyeri vagy létrehozza az `ActionForm` instanciát
- a kontroller szervlet továbbadja ezt az `Action` objektumnak.
- Ha a kérés egy `submit` eredménye az `Action` objektum további ellenőrzéseket végezhet az adatokra.

- Ha kell, az adatok visszaküldhetők a beviteli form-ra egy üzenetlistával együtt, amit megmutatunk az oldalon.
- Ha az adatok jók, továbbadhatjuk őket az üzleti logika rétegnek.
- Az `ActionForm` objektumot felhasználhatjuk arra is, hogy a HTML form-ot előre feltöltsük megfelelő adatokkal.

- A struts keretrendszer saját elemeket tartalmaz, melyek `JavaBean`-ek alapján feltöltik a HTML form mezőket.
- A JSP-k a keretrendszer további részeiről általában csak a mezők nevét kell tudják és, hogy hova legyen elküldve a form.
- Más struts elemek az `Action` és `ActionForm`-ban elmentett üzeneteket jelenítenek meg, és ezek az üzenetek lokalizálva vannak.

- Az egyszerűbb alkalmazásoknál az `Action` implementálhatja a kéréssel kapcsolatos üzleti logikát.
- Komplexebb alkalmazások esetén azonban az `Action` objektum más objetumokat, pl. JavaBean-eket kell meghívjon, melyek az üzleti logikát elvégzik.
- A maximális újrafelhasználhatóság érdekében az üzleti logika bean-ek nem hivatkozhatnak web-alkalmazás objektumokra.
- Az `Action` objektum le kell "fordítsa" a HTTP kérést és ezt továbbítania kell az üzleti logika réteg felé java bean-ekként.

## Egy tipikus adtabázis-alkamazás esetében pl.:

- Egy üzleti logika bean kapcsolódik az adatbázishoz és lekérdezi azt.
- Egy üzleti logika bean az eredményt átadja az `Action` objektumnak
- Az `Action` lementi ezt egy `FormBean`-be, request vagy session hatókörben
- A JSP megmutatja az eredményt egy HTML form formájában.

Sem az `Action` sem a JSP nem kell tudja, hogy az adatok honnan jönnek, csak azt kell tudják, hogy hogyan csomagolják illetve mutassák meg azokat.

# Modell komponensek

## Renszerállapot bean-ek

- A rendszer aktuális állapotát JavaBean-s osztályok írják le, melyek tulajdonságainak összessége megadja a rendszer aktuális állapotát. **Pl.** bevásárlókosár tartalma, felhasználó profilja (hitelkártya, kézbesítési cím, stb.), a megvásárolható áruk katalógusa és a készlet az egyes árukból.
- Ezeket a bean-eket adatbázisban tároljuk, nagyobb rendszerek esetében pl. Entity Bean-ek segítségével.

## Üzleti logika bean-ek

- Az alkalmazás funkcionális logikája java bean-ekre meghívott metódus-hívásokban testesül meg.
- A kétfajta alkotórész szerepelhet akár ugyanazon osztályon belül is, vagy lehetnek külön osztályokban (rendszerállapotot tartalmazó-, illetve az üzleti logikát megvalósító beanek).
- Utóbbi esetben a rendszerállapot bean-eket paraméterekként kell megadjuk az üzleti logikát implementáló metódusoknak.
- A maximális újrafelhasználás érdekében úgy kell az üzleti logika bean-eket megtervezni, hogy "ne tudják", hogy web-alkalmazásban lesznek meghívva.
  Ezáltal az üzleti logika más környezetben is felhasználható lesz.
- Egyszerűbb alkalmazások esetén az üzleti logika bean-ek lehetnek egyszerű `JavaBean`-ek, melyek a paramétéként kapott állapot bean-ekkel együttműködnek, vagy komplexebb rendszerek esetén lehetnek `Session Bean`-ek.

# Nézet komponensek

## Lokalizált üzenetek

A Struts a lokalizálást a java platfom standard lokalizáló osztályainak segítségével valósítja meg:

- **Locale**: minden Locale objektum egy országot és nyelvet azonosít, és ehhez kapcsolódó szám, dátum, stb. formázásokat.
- **ResourceBoundle**: ez az osztály támogatja pl. a különböző nyelvű erőforrásokat.
- **PropertyResourceBoundle**: a `ResourceBundle` egy standard implementációja, mely az üzeneteket java `property` fájlokban `name=value` formában tárolja
- **MessageFormat**: szöveges üzenet bizonyos részeit futásidőben adhatjuk meg. **Pl.** ha egy mondat szórendje különböző nyelveken más és más lehet, a $\{0\}$ string fel lesz cserélve az első paraméterrel, az $\{1\}$ a másodikkal stb.
- **MessageResources**: Struts osztály egy bizonyos konkrét üzenetnek egy bizonyos nyelven való megjelenítésére.

## Üzenet-erőforrás fájlok:

`MyApplication.properties` - az üzenetek az alapértelmezett esetben
`MyApplication_xx.properties` - az üzenetek az egyes nyelveken, ahol *xx* az egyes nyelvek ISO kódja.

- A kontroller szervlet konfigurálásánál kell megadjuk az üzeneterőforrás alapnevét (*_xx* nélkül).
- Ezek az erőforrások lehetnek egy csomagban vagy közvetlenül a `classes` alkatalógusban:
  `<message-resources parameter="com.mycompany.mypackage.MyApplication"/>`

## Form és form-bean együttműködések

Könnyen építhetünk HTML form-okat Struts saját elemkönyvtárak segítségével:

## JSP-ben egy textbox:

```
<input type="text" name="username"
value="<%= loginBean.getUsername() >"/>
```

## Struts segítségével:

```
<html:text property="username"/>
```

anélkül, hogy egyáltalán hivatkozna a form-bean-re.

A Struts többek között a következő beviteli elemeket támogatja:

- checkbox, hidden, password, radio, reset, select, option, options, submit, text, textarea.

Ezek mindig egy form elembe kell beágyazva legyenek.

# Kontroller komponensek

- A Struts esetében az elsődleges kontroller komponens egy `ActionServlet` osztályú szervlet.
- Ez `ActionMappings`-ek segítségével van konfigurálva.
- Egy `ActionMapping` definiál egy utat, amely egy kérés URI-nak lesz megfeleltetve és megadja az `Action` osztály teljes nevét.
- Minden egyes `Action` az `org.apache.struts.action.Action` osztályt bővíti ki.
- Az `Action` osztályok –melyek tartalmazzák az üzleti logikát vagy (jobb esetben) kapcsolódnak hozzá–, elvégzik/meghívják az üzleti logika részt, értelmezik az eredményt és átirányítanak a megfelelő nézethez.
- Logikai neveket definiálhatunk, ahova a kontroll továbbítva lesz azaz egy `Action` továbbíthat a "Main Menu" oldalra anélkül, hogy tudná a konkrét JSP nevét.
- Ezáltal a vezérlési logika elhatárolható a nézet logikától.

## ActionServlet

A Struts egy szervletet (`ActionServlet`) tartalmaz, amely implemantálja a kérés URI-Action map-eléseket.

## A kontroller elsődleges feladatai:

- `ActionForm`-ok írása, melyek közvetítenek a modell és a nézet között
- `Action` osztályok írása minden egyes logikai kérésre
- A map-elések konfigurálása minden egyes logikai kérésre.
  A konfigurációs fájl neve rendszerint `struts-config.xml`

## Az `ActionServlet`

- feldolgozza a kérést,
- meghatározza, hogy mit akar a felhasználó a kérés által,
- kinyeri az adatot a modellből, amit majd a nézet megkap,
- majd kiválasztja a megfelelő nézetet.

A munka nagyrészét az `Action` osztályokkal végezteti el.

## ActionForm bean-ek:

- Tulajdonságai gyakran megfelelnek a `model bean`-ek tulajdonságainak, de a `FormBean-eket` kontroller komponensnek kell tekintenünk, melyek adatokat cserélnek a modell és a nézet rétegek között.
- Minden egyes input form-hoz `form bean`-t rendelünk hozzá.
- Minden egyes formhoz saját form bean-t rendelhetünk, esetleg ugyanazt több form-hoz vagy ugyanazt az egész alkalmazáshoz.

Ha a struts konfigurációs fájlban ilyen bean-eket deklarálunk, akkor a struts kontroller szervlet a következő szolgáltatásokat nyújtja nekünk mielőtt a megfelelő Action metódust meghívná:

- megnézi, hogy van-e már egy bean arra a kulcsra a megfelelő hatókörben (request vagy session)
- ha nincs, akkor automatikusan létrehoz egyet a megfelelő hatókörben
- minden kérés paraméterre, melynek neve megfelel a bean egyik tulajdonság-nevének, meghívódik a megfelelő `set` metódus (hasonlóan a `<jsp:setProperty>`-hez a `*`-al)
- az aktualizált `ActionForm` át lesz adva az `Action` osztály `execute` metódusának, így a bean elérhertő lesz a rendszerállapot- valamint üzleti logika bean-ek számára.

Egy form nem feltétlenül a felhasználói felület egyetlenegy JSP oldalára vonatkozik. Akár több JSP oldalra is vonatkozhat (pl. varázslók, tab-ok)

# Metódusok

Az `ActionForm` osztályban nem kell metódusokat implemtálni, csak a megfelelő `get/set` metódusokat és esetleg a `validate` valamint `reset` metódusokat.

**Tehát semmi üzleti logika!**

Tulajdonságot (`get/set`) kell definiálni a HTML form minden egyes mezőjére:

- a mező neve meg kell egyezzen az egyes tulajdonságok nevével.

## Automatikus form-bean ellenőrzés (validation):

a `validate(ActionMapping, HttpServletRequest request)` metódus segítségével történik.

- Ez akkor hívódik, miután a bean tulajdonságai be lettek állítva, de mielőtt az `Action` osztály execute metódusa lefut.
- Ha nem találunk hibát, akkor null-t vagy egy üres `ActionErrors` listát adunk vissza, és az `execute` metódus lesz meghívva.
- Ha találunk hibát, a megfelelő `ActionError`-okat tartalmazó `ActionErrors` listát adunk vissza.
- Az `ActionError` osztályok a `MessageResource`-ból a megfelelő hibakulcsokat tartalmazzák.
- A kontroller szervlet lementi ezeket a hibakulcsokat a request hatókörben, visszairányít a bemeneti oldalra és megmutatja a hibákat a `<html:errors>` saját elem segítségével.

Ha szesszió hatókörbe hozzuk létre az `ActionBean`-t, akkor fontos a `reset` metódus implementálása, hogy mindegyik használat előtt inicializáljuk az értéket (checkbox, select)

# DynaActionForm

- Minden egyes html-formhoz saját `ActionForm` osztályt rendelni elég nagy ráfordítást jelent.
- Ehelyett használhatunk `DynaActionClass` osztályokat úgy, hogy a bean tulajdonságait felsoroljuk a struts konfigurációs fájljában.

## Pl.

```
<form-bean name="UserForm"
   type="org.apache.struts.action.DynaActionForm">
   <form-property
      name="givenName"
      type="java.lang.String"
      initial="John"/>
   <form-property
      name="familyName"
      type="java.lang.String"
      initial="Smith"/>
</form-bean>
```

- A JSP-ben ugyanúgy használhatjuk, mint a standard `ActionForm`-ok tulajdonságait.
- A tulajdonságait ugyanúgy definiálhatjuk `<bean:define>` segítségével
  (viszont a `DynaActionBean`-t magát nem hozhatjuk létre `<bean:define>` segítségével, mivel be kell állítni a megfelelő dinamikus tulajdonságokat).
- JSTL EL esetében is van különbség a hivatkozásban:
  - `ActionForm` esetében ${formbean.prop},
  - `DynaActionForm` esetében: ${formbean.map.prop}

  A map a `DynaActioForm` egy tulajdonsága, mely egy `HashMap`, és amely tartalmazza a többi tulajdonságokat.

- Az `Action` osztályból csak map-alapú szintaxissal férhetünk hozzá a tulajdonságaihoz: `myForm.get("name");`
- Nem hozhatók létre paraméter nélküli konstruktorral, a Struts hozza létre őket a háttérben.
- Ha szükség van rá, származtathatunk a `DynaActionForm` osztályból, hogy felüldefiniáljuk a `validate` és `reset` metódusokat. Ekkor a Struts konfigurációs fájlban a származtatott osztályt kell megadni.

# Map alapú ActionForm

Akkor használhatjuk, ha nem ismerjük előre a form tulajdonságait.

## Az ActionForm-nak lesz egy Map alapú tulajdonsága:

```
public FooForm extends ActionForm {

   private final Map values = new HashMap();

   public void setValue(String key, Object value) {
      values.put(key, value); }

   public Object getValue(String key) {
      return values.get(key);
   }

}
```

A megfelelő JSP oldal egy speciális jelöléssel fér hozzá a `Map`-ben tárolt értékekhez:
`mapNev(kulcsNev)`
A kerek zárójel azt jelenti, hogy a `mapNev` egy `Map` alapú tulajdonság, melynek tartalmához string alapú kulcsokkal férünk hozzá.

**Pl.** `<html:text property="value(foo)"/>`
meghívja a `FooForm` `getValue` metódusát a `"foo"` kulcsra.

**Ha pl. egy formot akarunk létrehozni, dinamikus mezőnevekkel:**

```
<%
   for (int i = 0; i < 10; i++) {
      String name = "value(foo-" + i + ")";
%>
      <html:text property="<%= name %>"/>
      <br/>
<%
   }
%>
```

## Hasonlóan létrehozhatunk Lista típusú tulajdonságokat is:

```
public FooForm extends ActionForm {

   private final List values = new ArrayList();

   public void setValue(int key, Object value) {
      values.set(key, value);
   }

   public Object getValue(int key) {
      return values.get(key);
   }
}
```

A JSP-ből a lista egyes elemeihez a következő szintaxissal férünk hozzá: `listaNev[index]`
A szögletes zárójel azt jelenti, hogy a `listaNev` egy `Lista` alapú tulajdonság, melynek tartalmához az egyes indexeken keresztül férünk hozzá.

# Action osztály

## Az `Action` osztály fő metódusa az `execute`:

```
public ActionForward execute(ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
    throws Exception;
```

Az `Action` osztály célja, hogy az `execute` metódusban feldolgozza a kérést és egy `ActionForward` objektumot adjon vissza, amely megmutatja, hogy a kontrollt hova kell továbbítani:

- JSP, Tile definíció, Velocity sablon vagy más Action.

## A logika bővebben:

- Leellenőrzi a felhasználói szesszió állapotát: be van-e jelentkezve a felhasználó.
  Ha nem, akkor átirányítunk a login oldalra (**pl.** közvetlen URL beírás vagy szesszió lejárás esetén)
- Ha az adatok ellenőrzése nem történt meg teljesen a `FormBean`-ben, itt megtehetjük.
  Ha probléma van, a megfelelő hibakulcsokat egy kérés attribútum listában tároljuk, majd visszairányítunk a bemeneti oldalra, ahol megmutatjuk a hibás részeket és kijavíthatjuk a hibás/hiányos adatokat.

- A kéréshez tartozó adatokat feldolgozzuk (**pl.** lementünk egy sort az adatbázisba).
  Ez a feldolgozás történhet az `Action` osztályba beágyazott kóddal, de sokkal ajánlottab egy üzleti logika bean-el elvégeztetni, átadva neki a megfelelő adatokat paraméterként.
- Feltöltjük adatokkal azokat a bean-eket, amelyek a következő oldal megmutatásához kellenek.
  Ezeket a bean-eket tipikusan kérés vagy szesszió hatókörben tároljuk.
- a megfelelő `ActionForward` objektumot adjuk vissza, amelyik azonosítja a válasz oldalt.

# ActionMapping

- A kontroller szervletnek tudnia kell több dolgot arra vonatkozólag, hogy hogyan feleltessünk meg minden egyes kérés URI-nak egy `Action` osztályt.
- Ezek az információk egy `ActionMapping` osztályban lesznek tárolva.

## Fontosabb tulajdonságai:

- **type** - az `Action`-t implementáló Java osztály teljes neve.
- **name** - Az `Action` által használt **FormBean** neve.
- **path** - a kérés URI, amelyik megfelel ennek a map-elésnek
- **unknown** - `true` ha ez az alapértelmezett `Action`.
  Ez fog lefutni az explicit be nem konfigurált kérések esetében.
  Csak egyetlenegy `Action` esetében lehet ez a paraméter `true`.
- **validate** - `true`, ha a `validate()` metódust meg akarjuk hívni

A kontroller szervlet ezeket az `ActionMappings` konfigurációkat egy `struts-config.xml` fájlból olvassa ki. A legkülső elem a `<struts-config>`.

**Három fő elem tartalmazza a konfigurációkat:**

- `<form-beans>`
- `<global-forwards>`
- `<action-mappings>`

## <form-beans>

- Ez az elem tartalmazza a form bean definíciókat.
- Ezek segítségével hozzuk létre futásidőben az `ActionForm` objektumokat.
- Minden egyes `ActionForm` objektumra definiálunk egy `<form-bean>` elemet, melynek a következő attribútumai vannak:
  - **name**: Egyedi azonosító, melyre az `Action` map-elésben hivatkozhatunk.
    Általában ez a neve a kérés vagy szesszió attribútumnak, amelyben a bean elmentődik
  - **type**: `ActionForm`-ot implementáló Java osztály teljes neve.

## <global-forwards>

- Ez a rész a globálisan definiált átirányításokat (forwards) tartalmazza.
- Az átirányítások `ActionForward` típusú objektumok lesznek, melyeket a `Action` osztály `execute` metódusa térít vissza.
- Segítségükkel logikai neveket map-elünk konkrét erőforrásokhoz (**pl.** JSP).
  Ezáltal az erőforrás lecserélhető anélkül, hogy a rá vonatkozó hivatkozásokat átírjuk az alkalmazásban.
- Minden egyes globális átirányításra definiálunk egy `<forward>` elemet, melynek a következő attribútumai vannak:
  - **name**: a `forward` logikai neve. Ezt használjuk majd az `execute` metódusban, hogy a megfelelő erőforráshoz továbbítsunk.
    **Pl**: loginpage, homepage
  - **path**: a kontextus relatív út az erőforráshoz.
    **Pl.**: `/index.jsp` vagy `/index.do`
  - **redirect**: `true` vagy `false` (alapértelmezett).
    `redirect`-et vagy `forward`-ot használjunk: a `forward` csak szerver oldalon történik, a `redirect` esetén aktualizálódik az URL a böngészőben.

## <action-mappings>

- Ez a rész az `Action` definíciókat tartalmazza.
- Minden egyes URL map-elésre definiálunk egy `<action>` elemet.
- A következő főbb attribútumai vannak:
  - **path**: a relatív út ehhez az Action-hoz (a web-alkalmazás kontextusához képest)
  - **type**: az `Action` osztály teljes neve.
  - **name**: a form-bean neve, amit egy `<form-bean>` elemben definiáltunk.

# struts-config.xml példa

```
<struts-config>
  <form-beans>
    <form-bean
      name="logonForm"
      type="org.apache.struts.webapp.example.LogonForm" />
  </form-beans>
  <global-forwards
    type="org.apache.struts.action.ActionForward">
    <forward
      name="logon"
      path="/logon.jsp"
      redirect="false" />
  </global-forwards>
```

```
  <action-mappings>
    <action
      path="/logon"
      type="org.apache.struts.webapp.example.LogonAction"
      name="logonForm"
      scope="request"
      input="/logon.jsp"
      unknown="false"
      validate="true" />
  </action-mappings>
</struts-config>
```

## A globális `forward`-okon kívül lokális `forward`-okat is definiálhatunk, **pl.**:

```
<action
  path="/editSubscription"
  type="org.apache.struts.webapp.EditSubscriptionAction"
  name="subscriptionForm"
  scope="request"
  validate="false">
  <forward
    name="failure"
    path="/mainMenu.jsp"/>
  <forward
    name="success"
    path="/subscription.jsp"/>
</action>
```