

Web Beans ve Ajax

Burak Bayramlı

Bu belge, \LaTeX ile üretilmiştir

Javascript

- Client tarafında, çabuk bir şekilde yapılması gereken işlemler Web dünyasında Javascript ile yapılır.
- Javascript, client (tarayıcı) tarafında çalışan, servis tarafına gidilmesini gerektirmeyen bir teknolojidir.
- Form giriş hataları (validation), çabuk bazı hesaplar ve dinamik olarak sayfa görüntüsünü değiştirmek Javascript kullanılabilir.

- Javascript, sayfanızın her görsel birimine dinamik olarak erişme özelliğine sahiptir. Mesela hiç servis tarafına gitmeden bir `<div>` içindeki HTML'i bile Javascript ile değiştirebilirsiniz.

`<div id="element1">..</div>` olduğunu düşünün. Javascript fonksiyonunuz içinde `document.getElementById('element1').innerHTML = ..` ile içerik *servis tarafına gitmeden* değiştirilmiş olur.

Javascript

Javascript kodları genelde XHTML/HTML sayfasının en üstünde

```
<script language="JavaScript">  
  function doSomething() {  
    ...  
  }  
</script>
```

şeklinde tanımlanır. Bu fonksiyonları çağırmak için,

sayfanızda olan herhangi bir kullanıcı *hareketine* bu fonksiyonları bağlayabilirsiniz.

Bu bağlama, herhangi bir XHTML elementi üzerinden olabilir. `<a ..>`, ``, `<div ..>` gibi etiketlerin her birine bir kullanıcı hareketiyle tetiklenen bir Javascript fonksiyonu bağlanabilir.

Javascript

Hangi kullanıcı hareketleri?

- `onMouseOver`: Mouse işaretinin etiket üzerine getirilmesi
- `onMouseOut`: Mouse işaretinin etiket üzerini terketmesi
- `onClick`: Bir etiketin tıklanması
- `onChange`: Bir etiketin içeriğinin değiştirilmesi

- `onKeyDown`: Bir etiket üzerindeyken bir tuşa basılmış olması
- ...

Servis Tarafı

Servis Tarafı

Fakat bir Web sitesinin tamamı tabii ki sadece client tarafında yapılamaz. Servis tarafı çetrefil, veri tabanı etkileşimi gerektiren ağır operasyonları yapabilme özelliğine sahiptir.

Öte yandan Web uygulamalarında servise gitmek demek, tüm sayfanın yeniden işlenmesi anlamına gelir. Bunu da her zaman tercih etmeyebiliriz.

İkisi de Olsa?

Her ikisine de sahip olamaz mıyız? Hem dinamik, esnek ve sadece istediğimiz sayfa birimlerinin daha geniş kullanıcı hareketleriyle tetiklenmesi, hem de, sadece bu ufak hareketlerin *gerektirdiği kadar* çetrefil işlem için servis tarafına gidilmesi.

İkisi de Olsa?

Javascript + servis tarafı = Ajax

Ajax

Asynchronous **J**avascript and **X**ML kelimelerinin kısaltılmışıdır.

Ajax

Aslında yanlış bir kısaltma, çünkü Ajax'ın

- Asenkron olması
- XML içermesi

gerekmez. Javascript içermesi kesin.

Ajax Altyapıları

- Ajax'i elle kodlamak oldukça külfetli.
- Bunun için birçok yardımcı paket çıktı
- Bunların arasından kullanımı en rahat, “objesel” ve EJB3'e entegre olanı: Web Beans Ajax yöntemidir.

Web Beans Ajax

- BU teknoloji ile kullanıcı hareketleri uzaktaki bir metodu sanki yerel bir fonksiyonu çağırıyormuş gibi çağırabilirler.
- Servis tarafında çağırılan objeler EJB3 servisleri olabilir.

Ayarlar

Web Beans Ajax kullanmak için web.xml dosyasına

```
<servlet>
  <servlet-name>Seam Resource Servlet</servlet-name>
  <servlet-class>
    org.jboss.seam.servlet.ResourceServlet
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Seam Resource Servlet</servlet-name>
  <url-pattern>/seam/resource/*</url-pattern>
```

```
</servlet-mapping>
```

tanımı eklenmelidir.

Ayarlar

Kullanan her sayfanın başına ise

```
<script type="text/javascript"
        src="seam/resource/remoting/resource/remote.js">
</script>
<script type="text/javascript"
        src="seam/resource/remoting/interface.js?[x]">
</script>
```

bulunmalıdır. Burada [x] için, servis tarafında çağırmak istediğimiz servis bean'e @Name ile verdiğimiz aynı ismi

kullanmalıyız.

Servis Tarafı - Interface

```
@Local
public interface ABC {

    @WebRemote
    public int getCount() ;

}
```

Servis Tarafı - Bean

```
@Stateless
@Name("isim1")
public class AbcBean implements ABC {

    public int metot1() {
        ...
        return x;
    }
}
```

Servis Tarafı

- Çağırılmak istenen metot interface'de `@WebRemote` ile işaretleniyor.
- Bean ise `@Name` ile isimlendiriliyor.

Çağırma İçin

```
<script language="JavaScript">
  function jsMetot3() {
    Seam.Component.getInstance("isim1").metot1(putResult)
  }
  function putResult(result) {
    document.getElementById("id2").innerHTML = ..;
  }
</script>
```

Çağırma İçin

Çağırma metodu normal. Fakat cevabın nasıl alındığına dikkat edelim: Burada *callback* usulü cevap alma tekniğini kullanıyoruz. `metot1`'den gelen cevap hiçbir değişkene atanmıyor - fakat çağrıyı yaparken biz cevap gelince *hangi lokal metodun* çağırılacağını belirtiyoruz, burada bu metod `putResult` metodudur.

Çağırma İçin

Her Ajax metot çağrısındaki parametrelerdeki *en son* parametre her zaman callback metot ismidir. Eğer hiçbir geri sonuç yok ise (void dönüş tipi için) o zaman bu son parametre `false` olabilir.

Çağırma İçin

Üç parametre alan ve tek bir sonuç döndüren metod çağırısı şöyle olurdu:

```
function jsMetot3() {  
    Seam.Component.getInstance("isim1").metot2  
        (param1, param2, param3, putResult);  
}  
function putResult(result) {  
    document.getElementById("id2").innerHTML = result;  
}
```

Etikete Bağlamak

Artık Ajax çağrısını bir etikete bağlamak için şunu yapmak yeterli:

```
<h:outputText value="..." rendered="." ...  
    onclick="javascript:jsMetot3()" />
```

ya da

```
<a href="..." onclick="javascript:jsMetot3()" />
```

Görevler

- Bir önceki Web Beans projenizi baz alın. Bu projede `build.xml` dosyanızda `example.war.webinf.lib` listesine `jboss-seam-remoting.jar` ismin ekleyin. Bu jar uzak Ajax çağruları yapabilmek için gerekli.
- `CarCounter` ve `CarCounterService` dosyalarıyla yeni bir bean yaratın. Bu bean veri tabanındaki tüm arabaların *sayısını* veri tabanından alarak döndürebilen

bir bean olsun. Bunu yapan metot ismi ise `int getCount()` olsun.

- Şimdi bu metodu `home.xhtml` sayfamızdan Ajax ile çağıralım. Bu çağrıyı tetiklemek ve sonucunu göstermek için form alanlarına yeni bir *Araba Sayısı* alanı ekleyelim. Bu alanın ilk değeri `--` olsun.
- Kullanıcı `mouse pointer`'i bu alan üzerine “getirince” yani `onMouseOver` hareketi ile servis tarafında `CarCounterBean` çağrısı yapılsın. Bu servis veri tabanındaki tüm arabaların sayısını sorgulayıp geriye bir

`int` döndürmeli.

- Bu `int` değerini alıp, sayfadaki `--` değeri yerine `innerHTML` ile atayın.
- Yani, sonuç olarak mouse `--` üzerine geldiği an, hiç bir tıklama, submit, vs. gerektirmeden servis tarafında yapılan bir işlemin sonucunu sayfamızın tek bir dinamik olarak görebilmeliyiz.

Import'lar

```
import org.bilgidata.kitapdemo.pojo.Car;  
import javax.ejb.Local;  
import java.util.List;  
import org.jboss.seam.annotations.WebRemote;
```

Import'lar

```
import org.jboss.seam.annotations.Name;  
import javax.ejb.Stateless;  
import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;
```

Son