Vaadin Push

Vaadin Push

- Normalde kullanıcı arayüzü **istemciden gelen istekler sonucu**nda güncellenir
- Buna “**pull based**” yaklaşım denir
- Bazı durumlarda kullanıcı ekranına sunucu tarafından **anlık güncellemeler**de bulunmak gerekir
- Bu durumda **sunucudan istemciye doğru** bir iletişim kurulması söz konusudur
- Buna da “**push based**” yaklaşım adı verilir

Vaadin Push

- Bu iletişim **WebSocket** bağlantısı üzerinden gerçekleşir
- Tarayıcı WebSocket desteklemiyorsa Vaadin **tarayıcıya uygun yöntem**i kullanır
- Vaadin bunun için **Atmosphere Framework**'ün özelleşmiş bir halini kullanmaktadır
- **vaadin-push.jar**'ına ihtiyaç vardır

Vaadin Push

- Server push yönteminin **iki modu** vardır
  - **automatic** (default)
  - **manual:** push işlemi explicit olarak push() metodunun invoke edilmesi ile gerçekleşir
- Push modunun öncelikle **aktive edilmesi** gerekir
  - Ya web.xml'de **pushmode** init-param ile
  - Ya da UI üzerinde @**Push** annotasyonu ile

Vaadin Push Ayarları

```
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <servlet>
        <servlet-name>PetClinicApplication</servlet-name>
        <servlet-class>
        com.vaadin.server.VaadinServlet
        </servlet-class>
        <init-param>
            <param-name>UI</param-name>
            <param-value>
            com.javaegitimleri.petclinic.view.PetClinicUI
            </param-value>
        </init-param>
        <init-param>
            <param-name>pushmode</param-name>
            <param-value>automatic</param-value>
        </init-param>
        <async-supported>true</async-supported>
    </servlet>
    ...
</web-app>
```

Servlet 3.0 API ile çalışılıyor ise async kabiliyeti de aktive edilmelidir

Birden Çok Thread ile UI'a Erişim

- Eğer **birkaç thread** birden **UI üzerinde güncelleme** yapacak ise bunların birbirleri ile **senkron çalışma**sı gerekir
- Aksi takdirde veri kaybı, günceleme problemleri veya deadlock durumları söz konusu olabilir
- **UI.access()** metodu senkronizasyon işlemini yürütür
- Senkronizasyon sırasında "**user session**" **lock**'lanır ve sadece tek bir thread aktif olur

Vaadin Push Kullanımı

```
UI.getCurrent().access(new Runnable() {
                public void run() {
                    labelCurrentTime.setValue(
                            new Date().toString());
                    UI.getCurrent().push();
                }
            });
```

automatic modda push() metodunu çağırmaya gerek yoktur

Access metodu Runnable bir nesneyi argüman olarak kabul eder

Bu nesnenin run() metodu içerisinde UI üzerinde güncelleme yapılabilir

Vaadin Push ile Broadcast

- Bir kullanıcıdan o anda aktif diğer kullanıcılara **anlık mesaj gönderme** ihtiyacı söz konusu olabilir
- Bunun için UI nesnelerinin takip edilmesi ve bir mesaj geldiğinde hepsinin sıra ile haberdar edilmesi gerekir
- Mesaj gönderme işi için "**singleton broadcaster örüntüsü**" kullanılabilir

Vaadin Push ile Broadcast

- Mesajları **ayrı bir thread** ile göndermek en sağlıklı yaklaşımdır
- Bunun için de Java'nın **ExecutorService**'inden yararlanılabilir
- Mesaj gönderimi ve alımı için bir **BroadcastListener** arayüzü oluşturulabilir
- UI sınıfları bu arayüzü implement ederler ve **attach/detach** sırasında kendilerini broadcaster'a register/deregister ederler

BroadcastListener & Singleton Broadcaster

```
public interface BroadcastListener {
    public void receive(String message);
}
```

```
public enum Broadcaster {
    INSTANCE;

    private Set<BroadcastListener> listeners = new LinkedHashSet<BroadcastListener>();
    private ExecutorService executorService = Executors.newSingleThreadExecutor();

    public synchronized void addListener(BroadcastListener listener) {
        listeners.add(listener);
    }
    public synchronized void removeListener(BroadcastListener listener) {
        listeners.remove(listener);
    }

    public synchronized void send(final String message) {
        for(final BroadcastListener listener:listeners) {
            executorService.execute(new Runnable() {
                @Override
                public void run() {
                    listener.receive(message);
                }
            });
        }
    }
}
```

Gönderici & Alıcı Bölümleri

JAVA Eğitimleri

Gönderici bölümü

```
Button btn = new Button("send message");
btn.addClickListener(new ClickListener() {

@Override
public void buttonClick(ClickEvent event)
{
     Broadcaster.INSTANCE.send("hello
there!");
}
});
```

Alıcı bölümü

```
@Override
public void attach() {
     super.attach();
     Broadcaster.INSTANCE
               .addListener(this);
}

@Override
public void detach() {
     Broadcaster.INSTANCE
               .removeListener(this);
     super.detach();
}

@Override
public void receive(final String
message) {
     this.access(new Runnable() {
     @Override
     public void run() {
          Notification.show(message);
     }
     });
}
```

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com