Vaadin – Spring Entegrasyonu

Spring Addon

- Spring addon ile **UI ve View instance'ları**nı Spring managed bean yapmak mümkündür
- Böylece Vaadin uygulaması içerisinde **Spring bean'lerine erişim** daha kolay olur
- Üç farklı **custom scope** sunar
  - UIScope
  - ViewScope
  - VaadinSessionScope

Spring Konfigürasyonu

- Web uygulaması için XML veya Java tabanlı **Spring konfigürasyonu** yapılarak başlanır

web.xml

```
<listener>
     <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<context-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>classpath*:/appcontext/applicationContext.xml</param-value>
</context-param>
```

applicationContext.xml

```
<beans...>

     <context:component-scan base-package="com.javaegitimleri.petclinic"/>

     <bean class="com.vaadin.spring.VaadinConfiguration"/>

</beans>
```

Spring Konfigürasyonu (Java Tabanlı)

web.xml

```
<listener>
     <listener-class>
          org.springframework.web.context.ContextLoaderListener
     </listener-class>
</listener>

<context-param>
     <param-name>contextClass</param-name>
     <param-value>
org.springframework.web.context.support.AnnotationConfigWebApplicationContext
     </param-value>
</context-param>

<context-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>com.javaegitimleri.petclinic.AppContextConfig</param-value>
</context-param>
```

```
@Configuration
@ComponentScan(basePackages="com.javaegitimleri.petclinic")
@EnableVaadin
public class AppContextConfig {

}
```

Vaadin Konfigürasyonu

- web.xml'de VaadinServlet yerine **SpringVaadinServlet** tanımlanmalıdır
- Bu servlet yaratılan her bir Vaadin session'ına **SpringUIProvider** ekler
- SpringUIProvider UI sınıflarına karşılık gelen nesneleri Spring ApplicationContext'den **bean olarak çözümleme**ye çalışır

@SpringUI ve @UIScope

UI sınıfında @SpringUI anotasyonu kullanılır. Bu şekilde Spring managed UI bean'i yaratılmış olunur. Anotasyonda path belirtmek opsiyoneldir. UI'a belirtilen path ile erişim sağlanır. Path belirtilmez ise servlet path'i UI'a erişim sağlar.

```
@SpringUI(path="/pcui")
public class PetClinicUI extends UI {
...
    @Autowired
    private PetClinicService petClinicService;
...
}
```

Ayrıca **@UIScope** anotasyonunu kullanmaya gerek yoktur. @SpringUI anotasyonu bu anotasyonu inherit eder. Böylece yaratılan UI bean'i custom UI scope'a sahip olur.

UI bean'i içerisinde istenilen herhangi bir Spring bean enjekte edilebilir.

Navigator Views

- Navigator tarafından oluşturulan View nesnelerinin de yönetimi Spring'e havale edilebilir
- **@SpringView** anotasyonu ile view scoped bir Spring managed bean tanımlanabilir

```
@SpringView(name="vets")
public class VetListAndDetailView extends CustomComponent implements View {

    @Autowired
    private PetClinicService petClinicService;

}
```

Name attribute ile hangi URI fragman'da erişilebileceği belirtilir. Kullanılmaz ise sınıf isminden türetilir. (vet-list-and-detail)
Ayrıca **@ViewScope** anotasyonunu kullanmaya gerek yoktur.

Navigator ve ViewProvider Konfigürasyonu

```
@SpringUI
public class PetClinicUI extends UI {

    @Autowired
    private ViewProvider
    viewProvider;

    @Override
    protected void init(VaadinRequest
    request) {
    ...
    Navigator navigator = new
    Navigator(this, new
    VerticalLayout());

    navigator.addProvider(viewProvider);

    setNavigator(navigator);
    ...
    }
```

- Navigator nesnesinin artık View nesnelerini Spring üzerinden çözümlemesi gerekir
- Bunu sağlayan **SpringViewProvider**'dır
- UI içerisine enjekte edilerek Navigator'a eklenebilir

Erişim Denetimi ve Error Views

- İstenirse @SpringView ile belirtilen view'ların sadece **spesifik UI instance**'ları içerisinden kullanılması sağlanabilir

```
@SpringView(name="vets",ui={PetClinicUI.class})
public class VetListAndDetailView extends CustomComponent implements View {
...
}
```

- Farklı bir UI'dan erişilmeye çalışıldığı vakit **errorView** gösterilecektir

```
navigator.setErrorProvider(...);
navigator.setErrorView(...);
```

Erişim Denetimi ve Error Views

- SpringViewProvider herhangi bir View'a erişim öncesi **ViewAccessControl** ve **ViewInstanceAccessControl** tipinde Spring bean'leri ile erişim denetimi yapmayı da sağlar
- ViewAccessControl, view nesnesi elde edilmeden önce **isminden kontrol** yapar
- ViewInstanceAccessControl ise view nesnesi elde edildikten sonra **instance üzerinden kontrol** yapar

Erişim Denetimi ve Error Views

- Bu bean'lerden herhangi birisi erişim izni vermez ise yine SpringViewProvider'da tanımlı **accessDeniedViewClass** a karşılık gelen View nesnesi görüntülenir

```
springViewProvider.setAccessDeniedViewClass(AccessDeniedView.class);
```

- Bu bölüm üzerinden **Spring Security** ile de rahatlıkla bir entegrasyon kurulabilir

İletişim

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