

Spring Container Extension Noktaları



Container Extension Noktaları

- ApplicationContext'e ve içindeki bean'lara dinamik olarak **yeni özellikler eklemek** mümkündür
- Bunun için “**pluggable extension point**”ler vardır
- Bu extension point'ler **iki tür**lüdür
 - Post-processor bean'lar
 - Namespace handler'lar

Post-Processor Bean'lar

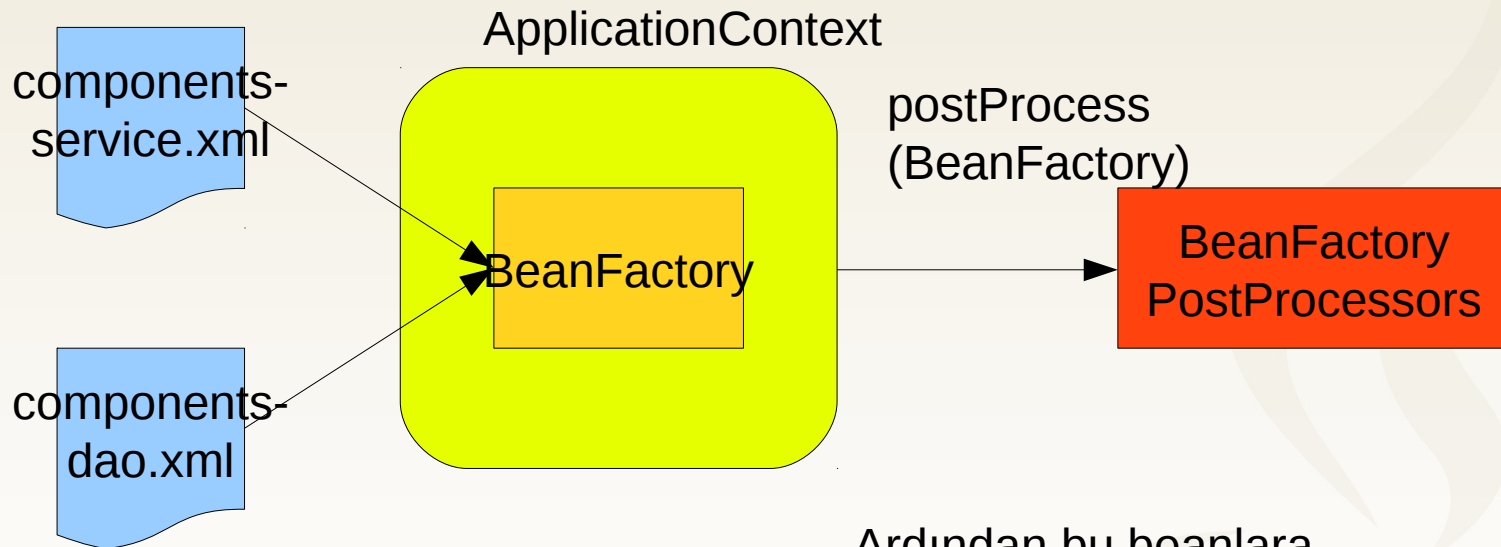
- ApplicationContext'e ve içindeki bean'lara dinamik olarak **yeni özellikler eklemek** sağlayan özel bean'lardır
- Kendi içinde ikiye ayrılırlar
 - **BeanFactoryPostProcessor**
 - **BeanPostProcessor**

BeanFactory

PostProcessor

- Bean tanımları (**BeanDefinition**) üzerinde işlem yaparlar
- Özel bean'lerdir
- Container tarafından otomatik tanınırlar
- Diğer **bütün bean'lerden önce** yaratılırlar
- Normal bean tanımları application context tarafından işlenip bean'lar yaratılmadan önce (!) devreye girerek **bean tanımlarını değiştirmeye yararlar**

PostProcessor Nasıl Çalışır?



ApplicationContext,
BeanFactoryPostProcessor
arayüzüne sahip bean'ları **startup**
aşamasında
tespit eder

Ardından bu beanlara
konfigürasyon metadata'yı
okuma ve değiştirme izini verir

Container diğer bean'ları yaratmaya
bu aşamadan sonra başlar

Örnek: Property Placeholder Kabiliyeti

```
<beans...>
```

```
    <context:property-placeholder  
location="classpath:application.properties"/>
```

```
</beans>
```



Bu namespace tanımı sayesinde ApplicationContext'e PropertySourcesPlaceholder isimli BeanFactoryPostProcessor'ü eklenir.

Bu post-processor'de diğer bean tanımları üzerindeki placeholder değişkenlerini resolve eder.

Örnek: Property Placeholder Kabiliyeti

```
<bean id="dataSource"  
class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
    <property name="driverClassName" value="${jdbc.driverClassName}"/>  
    <property name="url" value="${jdbc.url}"/>  
    <property name="username" value="${jdbc.username}"/>  
    <property name="password" value="${jdbc.password}"/>  
</bean>
```

ApplicationContext

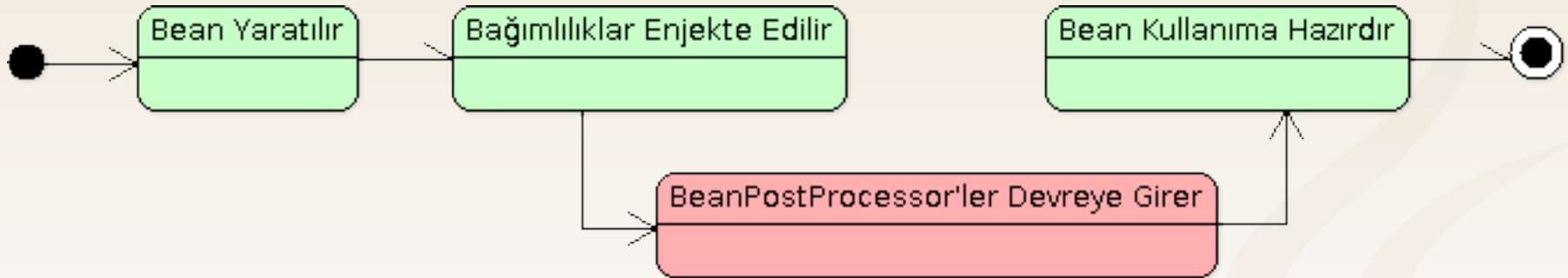
PropertySources
PlaceholderConfigurer

```
<bean id="dataSource"  
class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
    <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>  
    <property name="url" value="jdbc:hsqldb:hsqldb://production:9002"/>  
    <property name="username" value="sa"/>  
    <property name="password" value="secret"/>  
</bean>
```

BeanPostProcessor

- Normal **bean instance**'ların üzerinde değişiklik yapmayı sağlarlar
- Bunlarda özel bean'lardır
- Container tarafından otomatik olarak tanınırlar
- Diğer bean'lardan önce yaratılırlar

BeanPostProcessor Nasıl Çalışır?




Her bean instance'ı için post processor'ler ayrı ayrı devreye girer

Devreye girme ilk olarak bağımlılıklar enjekte edildikten sonra, fakat initialization metotları çağrılmadan önce, ikinci olarak da initialization metotları çağrıldıktan sonra olmak üzere iki defa gerçekleşir

Post processor'ler orijinal bean instance'ı üzerinde değişiklik yapabilirler, onu wrap edebilirler. Örneğin bir proxy yaratabilirler ve bu proxy'yi asıl bean olarak dönebilirler

Örnek: Caching Kabiliyeti

```
<beans...>  
  <cache:annotation-driven/>  
</beans>
```

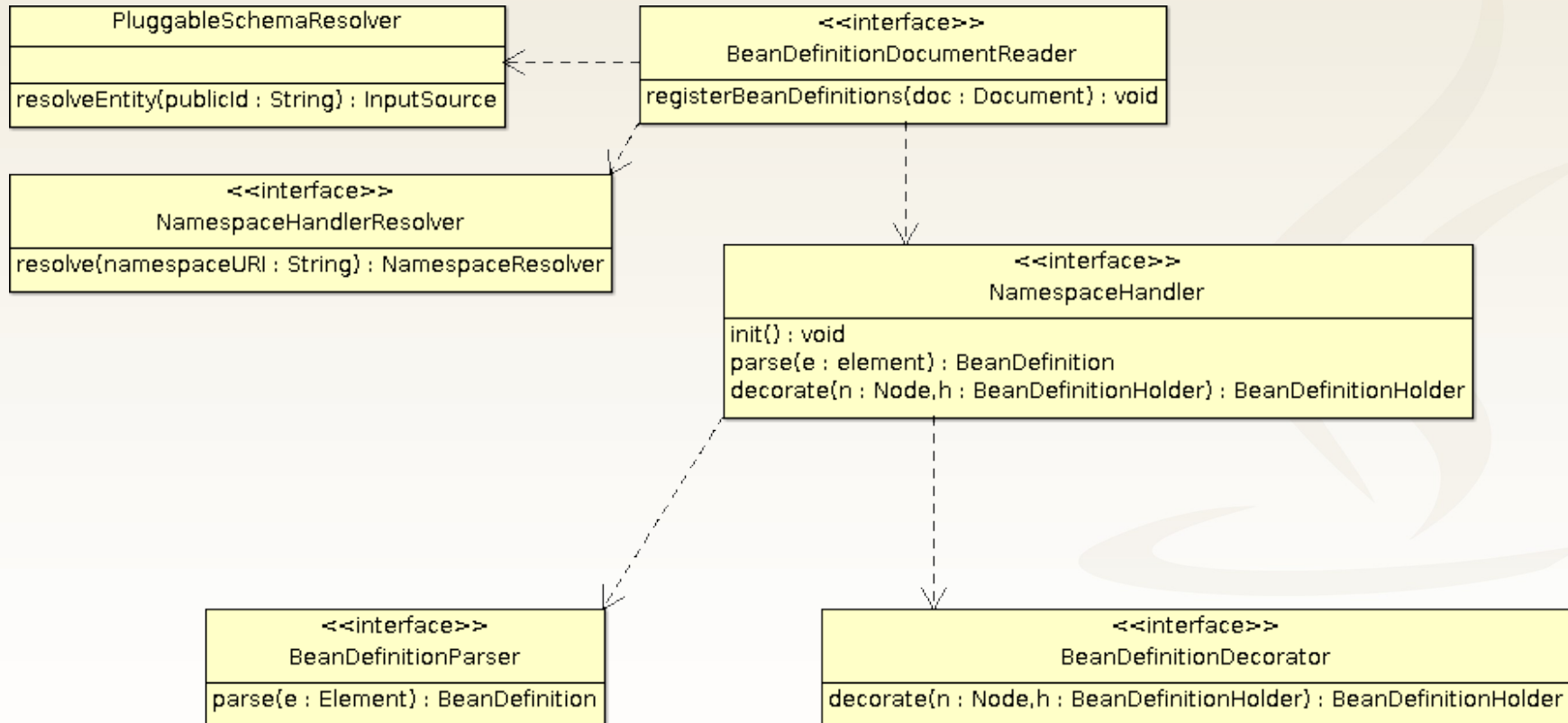


Bu namespace tanımı sayesinde ApplicationContext'e özel bir BeanPostProcessor tanımı eklenir.

Bu post-processor'de diğer bean tanımlarının sınıf veya metotları üzerinde @Cacheable anotasyonlarını tespit ederek bu bean instance'larında cache kabiliyetini devreye sokar.

NamespaceHandler Kabiliyeti Nasıl Çalışır?

/META-INF/spring.schemas: namespaceURI=XSD location path
/META-INF/spring.handlers: namespaceURI=NamespaceHandler FQN



```

<beans...>
  <jee:jndi-lookup id="dataSource"
    jndi-name= "java:comp/env/jdbc/DS"/>
</beans>
  
```

```

<beans...>
  <bean id="userPreferences"
    class="x.y.UserPreferencesImpl" scope="session">
    <aop:scoped-proxy/>
  </bean>
</beans>
  
```

İletişim

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

