

Spring ile Conversion ve Format İşlemleri



Conversion ve Data Binding

- Spring XML dosyaları içerisinde property değerleri **String** olarak tanımlanır
- Bu değerlerin **parse edilmesi** ve bean'ın property tipine dönüştürülmesi gerekir
- Bu işleme **conversion** adı verilir
- Dönüşüm sonrası değer property'ye set (**bind**) edilir
- Spring'in bu işlem için geleneksel yolu **PropertyEditor** kullanmaktır

PropertyEditor Nedir?

- **JavaBeans** spesifikasyonunun bir parçasıdır
- **Nesne - String dönüşümü** PropertyEditor vasıtası ile gerçekleştirilir
- Bean property'lerinin değerlerinin **set/get edilmesi** veya Spring MVC'de HTTP request parametrelerinin **parse edilerek Controller'a aktarılması** bu nesneler vasıtası ile gerçekleştirilir

Spring ve PropertyEditor

- Her Java tipine karşılık gelen bir **PropertyEditor** olmalıdır
- Spring **built-in pek çok PropertyEditor** implemantasyonuna sahiptir ve bunların bir kısmını default olarak **container'a register** eder



Built-in PropertyEditor Implementasyonları

- ByteArrayPropertyEditor
- ClassEditor
- CustomBooleanEditor
- CustomCollectionEditor
- CustomDateEditor (*)
- CustomNumberEditor
- FileEditor
- InputStreamEditor
- LocaleEditor
- PatternEditor
- PropertiesEditor
- StringTrimmerEditor (*)
- URLEditor

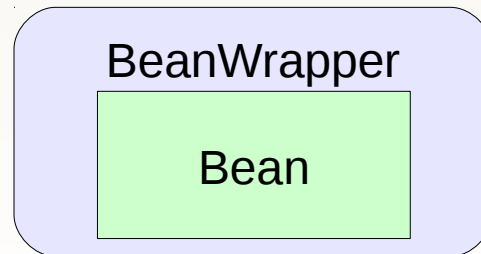
Buradaki PropertyEditor'ler Spring'in built-in implementasyonlarıdır

(*) ile işaretlenenler default olarak register edilmezler

Bunların dışında da register edilen Property editor'ler mevcuttur. Bütün hepsine **PropertyEditorRegistrySupport** sınıfından bakılabilir

BeanWrapper ve PropertyEditor

- Spring bean'ların property'lerini set etmek için PropertyEditor kullanır
- Ancak bu PropertyEditor'leri runtime'da kullanan **BeanWrapper** nesnesidir



- Bean'in property değerlerine set/get yapılmasını sağlar
- Property'lerin readable/writeable olup olmadıklarını tespit eder
- Property descriptor'ları döndürür
- Nested property'leri destekler

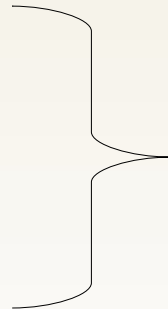
BeanWrapper ve PropertyEditor

- **BeanWrapper** Spring tarafından internal olarak kullanılır
- Daha çok **DataBinder** ve **BeanFactory** tarafından kullanılır
- Uygulama tarafında kullanılmaz
- PropertyEditor'lerin **register işlemi** de BeanWrapper veya Spring IoC Container düzeyinde yapılır

PropertyEditor Tespiti

- JVM, herhangi **bir tipe ait PropertyEditor**'ü o tipin sınıfı ile aynı paketin altında arar

Foo
FooEditor
FooBeanInfo



Örneğin com.example paketi altında Foo sınıfı olsun. Bu durumda Foo sınıfına ait PropertyEditor implemantasyonu sınıf ile aynı isimde ve Editor son ekine sahip olmalıdır.

BeanInfo implemantasyonu ile Foo sınıfı için custom biçimde PropertyEditor register edilebilir.

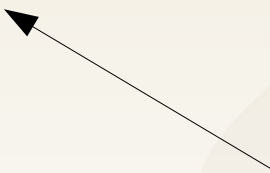
- java.beans.PropertyEditorManager** ile PropertyEditor'lerin tespiti için **search path** set edilebilir

PropertyEditor Tespiti

- JVM ayrıca default olarak **sun.bean.editors** paketi altını da tarar
- Bu paket altında awt Font, Color ve primitive Java tipleri için PropertyEditor implemantasyonları mevcuttur
- Spring BeanFactory düzeyinde **ConfigurableBeanFactory.registerCustomEditor()** metodu ile custom PropertyEditor'de register edilebilir

CustomEditorConfigurer

CustomEditorConfigurer bir
BeanFactoryPostProcessor'dür.



```
<bean
class="org.springframework.beans.factory.config.CustomEditor
Configurer">
    <property name="customEditors">
        <map>
            <entry key="com.example.model.Foo"
value="com.example.editors.CustomFooEditor"/>
        </map>
    </property>
</bean>
```

PropertyEditorRegistrar

```
public class CustomPropertyEditorRegistrar implements PropertyEditorRegistrar {
    public void registerCustomEditors(PropertyEditorRegistry registry) {
        registry.registerCustomEditor(Date.class, new CustomDateEditor(
            new SimpleDateFormat("dd/MM/yyyy"), true));
    }
}
```

PropertyEditorRegistrar üzerinden PropertyEditorRegistry nesnesine erişerek custom PropertyEditor nesnesi register edilebilir

```
<bean
class="org.springframework.beans.factory.config.CustomEditor
Configurer">
    <property name="propertyEditorRegistrars">
        <list>
            <bean
class="com.javaegitimleri.petclinic.web.CustomPropertyEditor
Registrar"/>
        </list>
    </property>
</bean>
```

Conversion API

- Spring 3 ile gelen bir kabiliyettir
- **PropertyEditor** kullanımına bir **alternatiftir**
- Type **conversion işlemleri** yapmayı sağlar
- Conversion **SPI** ile type conversion mantığı implement edilir
- Conversion **API** ile de runtime'da type conversion işlemleri gerçekleştirilir

Conversion SPI

```
package org.springframework.core.convert.converter;
```

```
public interface Converter<S, T> {
```

```
    T convert(S source);
```

```
}
```

Conversion SPI'nin arayüzüdür
Bir tipten başka bir tipe dönüşüm
bu arayüzü implement ederek
gerçekleştirilir

```
package org.springframework.core.convert.support;
```

```
final class StringToInteger implements Converter<String, Integer> {
```

```
    public Integer convert(String source) {  
        return Integer.valueOf(source);
```

```
    }
```

```
}
```

ConversionService

- Conversion mantığı runtime'da bu **façade arayüz** üzerinden erişilir
- ConversionService **tip dönüşüm işlemini** arka taraftaki register edilmiş **Converter nesnelerine** havale eder

```
package org.springframework.core.convert;
```

```
public interface ConversionService {

    boolean canConvert(Class<?> sourceType, Class<?> targetType);

    <T> T convert(Object source, Class<T> targetType);

    boolean canConvert(TypeDescriptor sourceType, TypeDescriptor targetType);

    Object convert(Object source, TypeDescriptor sourceType, TypeDescriptor targetType);

}
```

Default ConversionService

```
<bean id="conversionService"
class="org.springframework.context.support.ConversionServiceFactory
Bean" />
```

Default ConversionService implemantasyonu register edilir.
Eğer herhangi bir ConversionService register edilmez ise Spring
Geleneksel PropertyEditor mekanizmasını kullanacaktır

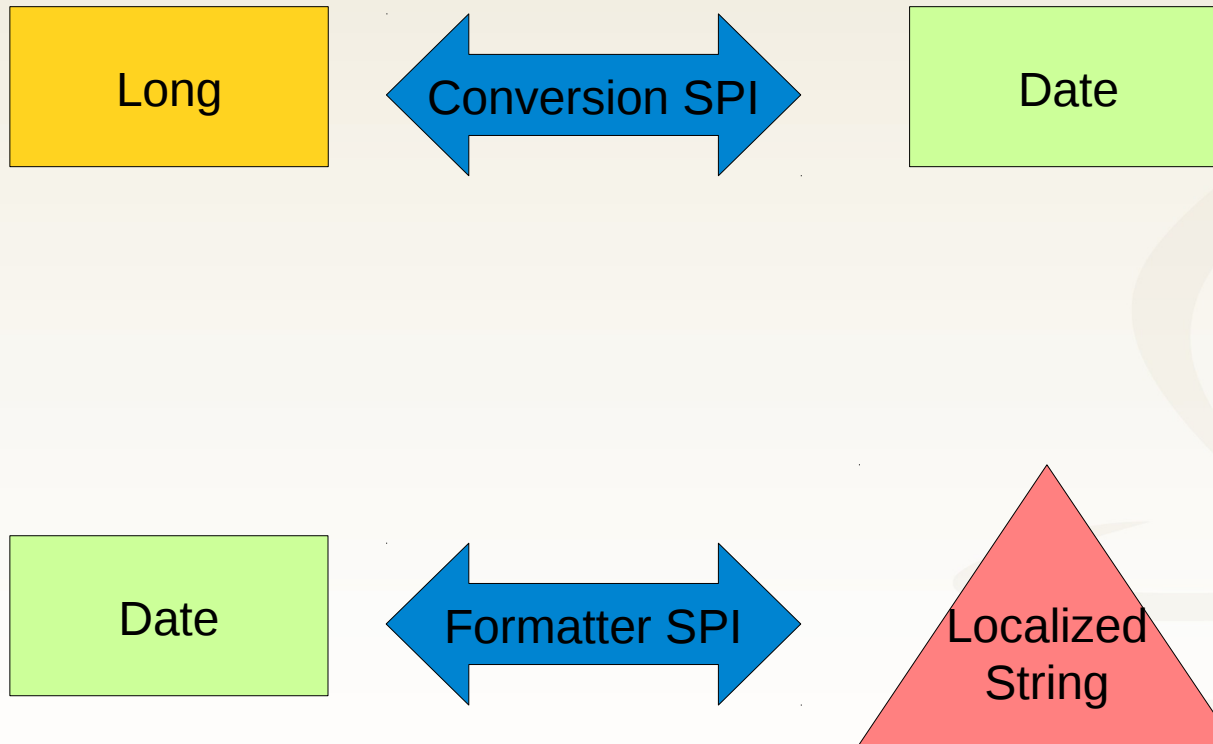
```
<bean id="conversionService"
class="org.springframework.context.support.ConversionServiceFactory
Bean">
  <property name="converters">
    <set>
      <bean class="com.example.CustomFooConverter" />
    </set>
  </property>
</bean>
```

İstenirse custom Converter implemantasyonları eklenebilir veya default
register edilenler override edilebilir

Format İşlemleri ve Format API

- Bir web uygulamasında kullanıcı verisi istemciden **String** formatında gönderilir
- String formatındaki bu verinin parse edilip **uygun Java tipine** dönüştürülmesi gerekir
- Yine bir Java nesnesinin kullanıcıya uygun formata dönüştürülerek **String** **represantasyonunun** istemciye gönderilmesi de söz konusudur
- **Formatter SPI** bu **parse** ve **print** işlemlerini gerçekleştirir

Conversion vs Format



Formatter SPI

```
import java.text.ParseException;

public interface Parser<T> {
    T parse(String clientValue, Locale locale) throws ParseException;
}

public interface Printer<T> {
    String print(T fieldValue, Locale locale);
}

package org.springframework.format;

public interface Formatter<T> extends Printer<T>, Parser<T> {
}
```

Built-in Formatters

- NumberFormatter
 - CurrencyFormatter
 - PercentFormatter
 - DateFormatter

}

NumberFormat ile formatlama yapılır
- DateFormatter

}

DateFormat ile formatlama yapılır
- Formatter nesneleri **FormatterRegistry** arayüzü ile register edilebilmektedir
- **FormattingConversionService** her iki SPI'ı da kapsamaktadır
- **FormattingConversionServiceFactoryBean** ile tanımlanabilir

Annotation Driven Format

- **@NumberFormat** ve **@DateTimeFormat** anotasyonları ile herhangi bir field'ın format konfigürasyonu belirlenebilir
- Spring MVC'de `<mvc:annotation-driven />` elemanı bu anotasyonları işleyen **Formatter** ve **Converter** nesnelerini **devreye sokmaktadır**
- Controller bean'lerinde handler metot parametreleri ve request/response body nesnelerinde bu anotasyonlar kullanılabilir

Annotation Driven Format

```
@RequestMapping(value = "/order")
@ResponseBody
public Order handle(
    @RequestParam("orderDate") @DateTimeFormat(pattern = "dd/MM/yyyy")
    Date orderDate,
    @RequestParam("price") @NumberFormat(style = Style.CURRENCY)
    BigDecimal price) {
    ...
}
```

```
public class Order {

    @NumberFormat(style=Style.CURRENCY)
    private BigDecimal price;

    @DateTimeFormat(iso=ISO.DATE)
    private Date orderDate;

    ...

}
```

İletişim

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

