

# Annotasyon Tabanlı ApplicationContext Konfigürasyonu



# Annotasyon Tabanlı Container Konfigürasyonu

- Spring Container'da **bean tanımları** ve **bağımlılıkların belirtilmesi** işlemleri **Java anotasyonları** kullanılarak da yapılabilir
- Anotasyon tabanlı konfigürasyon **iki kısımda** incelenebilir
  - Bean tanımları XML'de yapılmaya devam ederken, bağımlılıkların enjekte edilmesi anotasyonlar ile gerçekleştirilebilir
  - Ya da hem bean tanımları, hem de bağımlılıkların enjeksiyonu tamamen anotasyonlar ile gerçekleştirilir

# XML Tabanlı Bean Tanımlarında Anotasyon Kullanımı

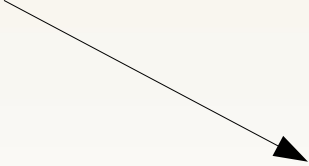
- Spring ilk çıktığında **sadece XML tabanlı bean konfigürasyon kabiliyeti** vardı
- Java 5 ile birlikte anotasyon desteği gelince Spring ekibi XML tabanlı bean tanımlarının **bağımlılıklarını anotasyonlarla yönetmeye** başladılar
- Bu aşamada **bean tanımları yine XML'de yapılmaktaydı**

# XML Tabanlı Tanımlarda Java Anotasyonlarını Devreye Alma

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans ...>
```

```
<context:annotation-config/>
```

```
</beans>
```

 @PostConstruct, @PreDestroy, @Required, @Autowired gibi anotasyonları devreye sokar

**Bu aşamada henüz bean tanımları XML konfigürasyon dosyalarında yapılmaya devam etmektedir!**

# @Required

- Bean tanımı ve bağımlılıkların enjeksiyonu **XML ile** yapılır
- @Required anotasyonu Sınıfların **setter metotları** üzerinde kullanılır
- Spring Container startup sırasında bean tanımında bu property'ler için **setter injection** yapılıp yapılmadığını **kontrol** eder
- Eğer setter injection yapılmayan bir property **@Required** ile işaretlenmiş ise container **hata** verir

# @Required

```
public class Foo {
    private Bar bar;

    @Required
    public void setBar(Bar bar) {
        this.bar = bar;
    }

    // ...
}
```

XML bean tanımında bar property'sine enjeksiyon yapılmıyor olsaydı hata ortaya çıkacaktı

```
<bean id="bar" class="x.y.Bar"/>
```

```
<bean id="foo" class="x.y.Foo">
```

```
    <property name="bar" ref="bar"/>
```

```
</bean>
```

# @Autowired

- Bean tanımı yine **XML** tarafında yapılır
- **@Autowired** anotasyonu ile **sınıf içerisinde hangi property'lere** bağımlılık enjeksiyonu yapılacağı belirtilir
- Spring Container mevcut bean'lardan uygun olanlarını **@Autowired anotasyonu ile işaretli property'lere** enjekte eder
- XML bean tanımında **bağımlılık tanımlarını yapmaya gerek yoktur**

# @Autowired

```
public class Foo {  
    private Bar bar;  
  
    @Autowired  
    public void setBar(Bar bar) {  
        this.bar = bar;  
    }  
  
    // ...  
}
```

ApplicationContext'de tanımlı Bar tipin'deki bean'ı enjekte eder

```
<beans...>  
    <bean id="bar" class="x.y.Bar"/>  
  
    <bean id="foo" class="x.y.Foo"/>  
</beans>
```



# @Autowired

```
public class Foo {  
    @Autowired  
    private Bar bar;  
  
    private Baz baz;  
  
    @Autowired  
    public Foo(Baz baz) {  
        this.baz = baz;  
    }  
  
    // ...  
}
```

Field, setter ve constructor'a uygulanabilir

Default olarak **byType** modunda çalışır

Birden fazla aynı tipte bean olması ve bu bean'lardan herhangi birisinin ismi property ile eşleşmediği durumda hata verir

# Qualifier ile Aday Bean'ların Sınırlandırılması

```
public class Foo {
```

```
    @Autowired  
    @Qualifier("myBar")  
    private Bar bar;
```

```
    // ...
```

```
}
```

```
<bean id="foo" class="x.y.Foo"/>
```

```
<bean id="bar1" class="x.y.Bar">  
</bean>
```

```
<bean id="bar2" class="x.y.Bar">  
    <qualifier value="myBar"/>  
</bean>
```

Eğer tanımlanmaz ise **default qualifier** değeri olarak bean ismi kabul edilir

# @Autowired

```
public class Foo {

    private Bar bar;

    @Autowired(required=false)
    public void setBar(Bar bar) {
        this.bar = bar;
    }

    // ...
}
```

Default **required** attribute değeri **true**'dür  
required=true durumunda Spring Container  
belirtilen tipte bean bulamadığında  
hata verir. Property'nin NULL kalması için  
**required=false** olarak belirtilmelidir.

# ApplicationContext'deki Belirli Tipteki Bütün Bean'ları Enjekte Etmek

```
public class Foo {
```

```
    @Autowired  
    private Bar[] bars;
```

```
    // ...
```

```
}
```

→ Bu sayede container'da tanımlı **belirli bir tipteki bütün beanları** bir array'e autowire etmek de mümkündür

Eğer ApplicationContext'de Bar tipinde hiç bean yoksa hata verir


# ApplicationContext'deki Belirli Tipteki Bütün Bean'ları Enjekte Etmek

```
public class Foo {  
  
    private Set<Bar> bars;  
  
    @Autowired  
    public void setBars(Set<Bar> bars) {  
        this.bars = bars;  
    }  
  
    // ...  
}
```

Enjekte edilecek bean'lerin tipini java generics'den tespit edebilir

# ApplicationContext'i Enjekte Etmek

```
public class Foo {  
  
    @Autowired  
    private ApplicationContext context;  
  
    // ...  
}
```



ApplicationContext kendisini bu bean'a enjekte eder. **ApplicationContextAware** arayüzünü implement etme gereksinimini ortadan kaldırır.

# @Autowired ve XML

- Bağımlılık tanımları **hem XML property elemanı ile hem de @Autowired ile belirtilmiş olabilir**
- Ancak **annotation injection XML injection'dan önce yapılır**
- Bu sayede **XML injection** annotation injection ile yapılanları **override edebilir**

# Sınıf Düzeyinde Anotasyon Kullanarak Bean Tanımlama

- Bean tanımları **sınıf düzeyinde anotasyon kullanarak** da yapılabilir
- Spring Container classpath'i tarayarak belirli anotasyonlarla işaretlenmiş **Java sınıflarını tespit eder** ve bu sınıflardan **birer bean yaratır**
- Tarama ve bean oluşturma işlemi için **<context:component-scan>** elemanı kullanılır



# Component Scan İşlemi

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans ...>
```

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

```
</beans>
```



**<context:annotation-config/>** elemanını da otomatik olarak aktive eder. Dolayısı ile bu elemanı tanımlaya artık gerek yoktur.

# Component Scan İşlemi

- Default olarak aşağıdaki **built-in anotasyonlar** scan edilir
  - @Component
  - @Repository
  - @Service
  - @Controller ve @RestController
  - @ControllerAdvice
  - @Configuration

# Annotasyon Tabanlı Bean Tanımlamaya Örnek

```
@Service("securityService")
public class SecurityServiceImpl implements
SecurityService {

    private SecurityDao securityDao;

    @Autowired
    public SecurityServiceImpl(SecurityDao securityDao) {
        this.securityDao = securityDao;
    }
}

@Repository("securityDao")
public class SecurityDaoImpl implements SecurityDao {
    // ...
}
```

# Component Scan İşlemi ve Bean Name Generation

- Bean isimlendirmesi için **@Component** değerine bakılır
- Default olarak bean ismi **sınıf isminin küçük harfle başlayan** halidir
- **BeanNameGenerator** ile bu davranış değiştirilebilir

```
<context:component-scan  
base-package="com.javaegitimleri"  
name-generator="x.y.CustomNameGenerator" />
```

# @Component ve Diğer Stereo Tipler

- **@Service**
  - Spring için özel bir anlamı yoktur
  - Servis katmanındaki bean'lar için eklenmiştir
- **@Repository**
  - DAO beanlarını tanımlar
  - Exception'ların otomatik çevrimini de tetikler
- **@Controller**
  - MVC controller bean'ları tanımlanır

# @Component ve Diğer Stereo Tipler

- **@ControllerAdvice**
  - MVC controller'lar için global error handling metotlarının yazıldığı bean'ları tanımlar
- **@Configuration**
  - Java tabanlı konfigürasyon bean tanımı yapar
  - Bu sınıfların içerisinde diğer bean'ları yaratan factory metotlar yer alır
- Hepsi **@Component**'ten türer

# Component Scan

## İşlemi: Include/Exclude

- İstenirse farklı anotasyonların kullanıldığı veya hiç **anotasyona sahip olmayan sınıflardan** da bean oluşturulması sağlanabilir
- Ya da default olarak taranan built-in anotasyonlar veya bazı spesifik sınıflar **göz ardı** ettirilebilir

# Component Scan İşlemi: Include/Exclude

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

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

```
<context:include-filter type="annotation"  
    expression="org.aspectj.lang.annotation.Aspect"/>
```

```
<context:include-filter type="assignable"  
    expression="com.javaegitimleri.petclinic.dao.BaseDao"/>
```

```
<context:exclude-filter type="annotation"  
    expression="org.springframework.stereotype.Controller"/>
```

```
</context:component-scan>
```

```
</beans>
```



# Component İçinde Bean Tanımları

- Component'ler business metotları dışında **bean factory metotları** da barındırabilir

**@Component**

```
public class FooFactory {
```

Metot düzeyinde tanımlama **@Bean** annotasyonu ile gerçekleştirilir

```
@Bean @Qualifier("myFoo")
```

```
public Foo foo() {  
    return new Foo();  
}
```

Metot ismi bean ismi olur,  
Ayrıca Qualifier'da tanımlanabilir

```
public void doWork() {  
    // ...  
}
```

Component normal bir bean instance'ıdır ve bean yaratma dışında normal işlemlere de sahip olabilir

```
}
```

# @Value

- Built-in Java tipli **property değerlerini** enjekte etmek için @Value anotasyonu kullanılır
- İçerisinde property **placeholder** da kullanılabilir

```
@Component
public class Foo {

    @Value("bar-value")
    private String bar;

    @Value("${foo.baz}")
    private String baz;

    // ...
}
```

# @Scope

@Scope anotasyonu bean tanımında metot veya sınıf düzeyinde kullanılabilir

```
@Scope("prototype")
@Component
public class CommandImpl implements Command {
    // ...
}
```

```
@Component
public class CommandFactory {

    @Bean @Scope("prototype")
    public Command createCommand() {
        return new CommandImpl();
    }
}
```

# @Lazy

```
@Component
@Lazy
public class FooFactory {

    @Bean @Lazy
    private Foo foo() {
        return new Foo();
    }
}
```

Sınıf veya metot düzeyinde **@Lazy** anotasyonu kullanılarak bean'lerin sadece gerektiği anda yaratılmaları sağlanabilir

# İletişim

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

