

# Spring Security XML Konfigürasyonu



# Spring Security Filter Zinciri

- Spring Security **Servlet Filter** tabanlı bir framework'tür
- Her bir güvenlik gereksinimi **ayrı bir Filter** tarafından ele alınır
- Spring Container ile **entegre** çalışırlar
- Bu Filter'lar web isteği üzerinde **birbirleri ardı sıra** işlem yaparlar
- Dolayısı ile aralarındaki sıralama önemlidir

# Spring Security

## Konfigürasyonu: web.xml

```
<web-app>
  <filter>
    <filter-name>
      springSecurityFilterChain
    </filter-name>
    <filter-class>
      org.springframework.web.filter.DelegatingFilterProxy
    </filter-class>
  </filter>

  <filter-mapping>
    <filter-name>
      springSecurityFilterChain
    </filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

web.xml içerisinde  
**DelegatingFilterProxy** ile  
bir Servlet Filter tanımı  
yapılır

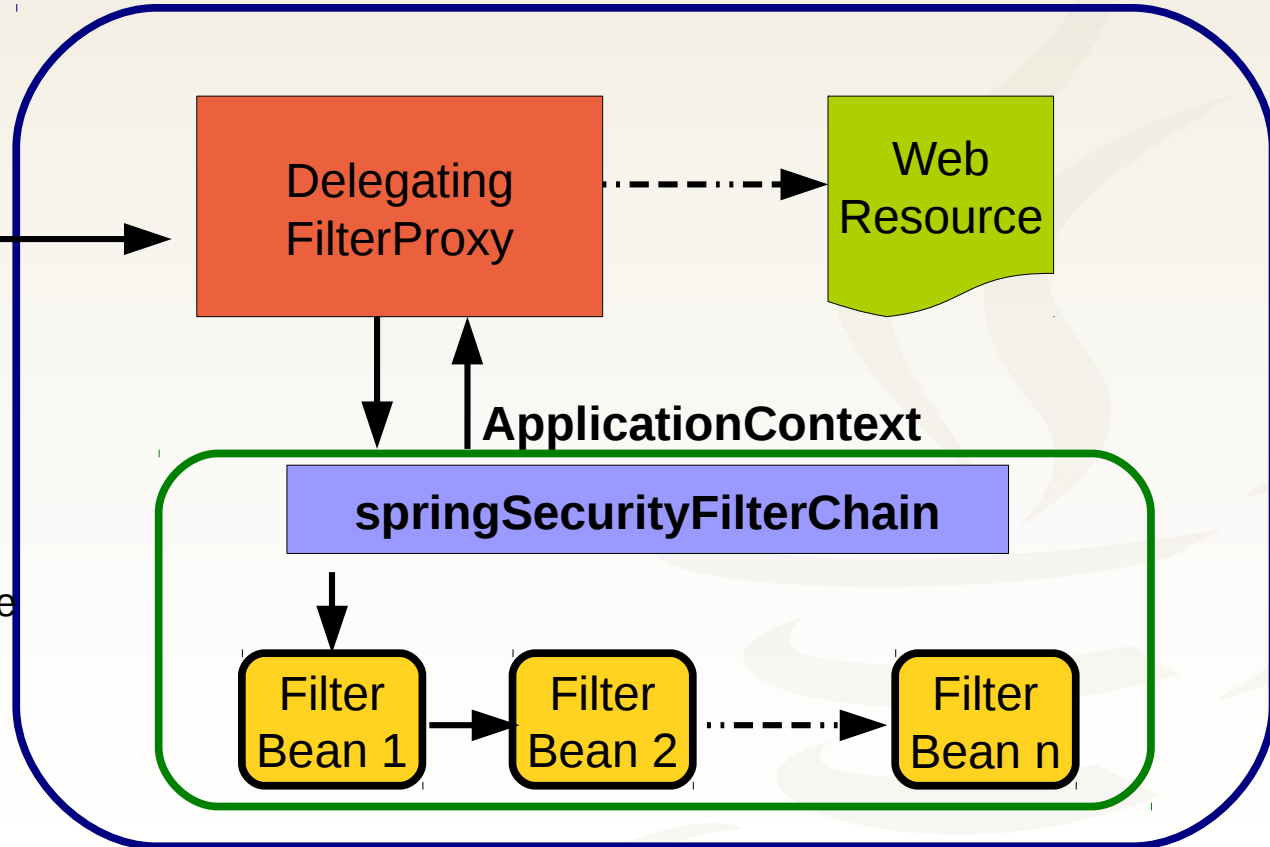
# Spring Security Filter Mimarisi

**DelegatingFilterProxy** sınıfı, web.xml içerisindeki tanım ile ApplicationContext içinde tanımlı Filter bean'ları arasında **köprü vazifesi** görür



**Web client**

## Servlet Container



**DelegatingFilterProxy** işi Spring Container'da tanımlı **FilterChainProxy** tipindeki bean'e (**springSecurityFilterChain**) delege eder. Böylece **tek bir filter** ile bütün security filter zincirinin tanımlanması sağlanır.

# Spring Security

## Konfigürasyonu: Filter Zinciri

Otomatik olarak bir login form render eder, form authentication'ın yanında basic authentication'ı da devreye alır, default logout url'i ve logout servislerini register eder

```
<security:http auto-config="true">
```

```
<security:intercept-url pattern="/**" access="hasRole('ROLE_USER')" />
```

```
</security:http>
```

**springSecurityFilterChain** id'li  
FilterChainProxy tipinde bean tanımını  
Spring Container'a otomatik olarak ekler

Bu bean içerisinde de her bir http elemanına  
karşılık gelen bir SecurityFilterChain bean'i  
eklenir

Uygulamadaki web kaynaklarına  
Erişimi yetkilendirir

## Konfigürasyonu: AuthenticationManager

Kimliklendirme işlemini gerçekleştirmesi için **authenticationManager** ve **authenticationProvider** konfigürasyonu yapılır

```
<security:authentication-manager>  
  <security:authentication-provider  
    user-service-ref="userService">  
  
  </security:authentication-provider>  
</security:authentication-manager>
```

AuthenticationProvider, kullanıcı bilgilerine erişebilmek için **UserDetailsService** bean'ine ihtiyaç duyar

## Konfigürasyonu: UserDetailsService

```
<security:user-service id="userService"  
    properties="classpath:/users.properties">  
</security:user-service>
```

Herhangi bir user realm'i kullanarak **userDetailsService** tanımlanır. Burada test amaçlı olarak **in-memory realm** kullanılmıştır

Spring UserDetailsService'in JDBC ve LDAP gerçekleştirmeleri için de built-in namespace elemanları sağlar

# İletişim

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

