

Spring MVC ve REST

Web Uygulamalarında Java Tabanlı Container Konfigürasyonu

- **ContextLoaderListener** default olarak **WebApplicationContext** sınıfından bir Container oluşturur, bu da XML konfigürasyon dosyaları ile çalışır
- Java tabanlı konfigürasyon sınıflarının yüklenmesi isteniyorsa **contextClass** isimli bir context-param ile **AnnotationConfigWebApplicationContext**'i kullanması söylenmelidir
- **contextConfigLocation** context-param'a ise bundan böyle değer olarak FQN şeklinde konfigürasyon sınıfları verilebilir

Web Uygulamalarında Java Tabanlı Container Konfigürasyonu

```
<web-app>
  <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.config.AppContextConfig
    </param-value>
  </context-param>

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

Web.xml'siz ApplicationContext Konfigürasyonu

- web.xml'siz ApplicationContext konfigürasyonu **Servlet 3.0+ spesifikasyonu** ile uyumlu uygulama sunucular için geçerlidir
- **WebApplicationInitializer** arayüzünü implement ederek web uygulamasında **ApplicationContext bootstrap işlemi tamamen web.xml'siz** gerçekleştirilebilir

WebApplicationInitializer

- Web uygulamalarında **ServletContext** nesnesini programatik olarak konfigüre etmek için kullanılan **callback interface**'tir
- Classpath'deki **WebApplicationInitializer** sınıflarını tespit edip yükleyen **SpringServletContainerInitializer** sınıfıdır
- SpringServletContainerInitializer sınıfı ise **javax.servlet.ServletContainerInitializer** SPI arayüzünden türer


SpringServletContainer Initializer

- ServletContainerInitializer implemantasyonları /META-INF/services/javax.servlet.ServletContainerInitializer dosyaları içerisinde bulunup yüklenir
- SpringServletContainerInitializer'da **spring-web.jar** içerisindeki ilgili dosya içerisinde tanımlıdır

Web.xml'siz ApplicationContext Konfigürasyonu

```
public class PetClinicWebApplicationInitializer
    extends AbstractContextLoaderInitializer {

    @Override
    protected WebApplicationContext createRootApplicationContext() {
        AnnotationConfigWebApplicationContext wac =
            new AnnotationConfigWebApplicationContext();
        wac.register(DaoBeansConfig.class, ServiceBeansConfig.class);
        wac.refresh();
        return wac;
    }
}
```



WebApplicationInitializer arayüzünden türer. ServletContext'de ContextLoaderListener'ın konfigürasyonunu sağlar

Birden fazla WebApplicationInitializer olması durumunda **Ordered** arayüzü ile aralarında sıralama yapmak mümkündür

Web.xml'siz DispatcherServlet Konfigürasyonu

```
public class PetClinicSpringMvcWebApplicationInitializer
    extends AbstractDispatcherServletInitializer {

    @Override
    protected WebApplicationContext createServletApplicationContext() {
        AnnotationConfigWebApplicationContext wac =
            new AnnotationConfigWebApplicationContext();
        wac.setEnvironment(new StandardServletEnvironment());
        wac.register(ControllerBeansConfig.class);
        return wac;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[]{"/mvc/*"};
    }

    @Override
    protected WebApplicationContext createRootApplicationContext() {
        AnnotationConfigWebApplicationContext wac =
            new AnnotationConfigWebApplicationContext();
        wac.register(DaoBeansConfig.class, ServiceBeansConfig.class);
        return wac;
    }
}
```


Web.xml'siz DispatcherServlet Konfigürasyonu

- DispatcherServlet ayrı bir WAC oluşturmak yerine parent WAC'ı da kullanabilir

```
public class PetClinicSpringMvcWebApplicationInitializer
    extends AbstractDispatcherServletInitializer {

    private AnnotationConfigWebApplicationContext wac;
    @Override
    protected WebApplicationContext createServletApplicationContext() {
        return wac;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[]{"mvc/*"};
    }

    @Override
    protected WebApplicationContext createRootApplicationContext() {
        AnnotationConfigWebApplicationContext wac =
            new AnnotationConfigWebApplicationContext();
        wac.register(DaoBeansConfig.class, ServiceBeansConfig.class,
            ControllerBeansConfig.class);
        return wac;
    }
}
```

@RestController

- REST Controller bean'larını tanımlamak için kullanılır
- **@Controller** ve **@ResponseBody** anotasyonlarını bir araya getirir
- Her bir handler metoduna **@ResponseBody** eklemekten kurtarır
- Sadece kolaylık sağlar

```
@RestController
public class PetClinicRestController {
    ...
}
```

HTTP Protokolünün Metotları

- HTTP protokolünde sunucu tarafında yapılacak iş **HTTP metotları** ile belirtilir
 - **GET** : Mevcut bir resource'a erişim sağlar (**SELECT/READ**)
 - **POST** : Yeni bir resource yaratır (**INSERT/CREATE**)
 - **PUT** : Mevcut bir resource'u güncellemeyi sağlar (**UPDATE**)
 - **DELETE** : Mevcut bir resource'u siler (**DELETE**)

POST ve PUT Metot Çağrılarları Arasındaki Fark

- Hem POST hem de PUT metotları sunucu tarafında **yeni bir resource yaratmak** (INSERT/CREATE) için kullanılabilir
- Ancak **yaratılacak resource'un sunucudaki lokasyonu bilindiği zaman** PUT metodunu kullanmak anlamlıdır
- Bu da yeni bir nesne yaratılıyorsa bu **nesne'nin ID'sini istemcinin bilmesi** veya karar vermesine karşılık gelir

POST ve PUT Metot Çağrılarları Arasındaki Fark

- Eğer ID sunucu tarafından üretilen bir değer ise **POST**'u kullanmak doğru olacaktır
- Genel kural olarak yeni bir resource yaratma işlemleri için **POST** metodunu, mevcut resource'u güncellemek için ise **PUT** metodunu kullanmak doğrudur

Spring ve REST API Örneği

```
@RequestMapping(value="/owners",  
                method=RequestMethod.GET)
```

```
@ResponseStatus(HttpStatus.OK)  
@ResponseBody Owners  
petClinicService.findOwners()
```

```
@RequestMapping(value="/owners/{id}",  
                method=RequestMethod.GET)
```

```
@ResponseStatus(HttpStatus.OK)  
@ResponseBody Owner petClinicService.  
loadOwner(@PathVariable long id)
```

```
@RequestMapping(value="/owner",  
                method=RequestMethod.POST)
```

```
@ResponseStatus(HttpStatus.CREATED)  
@ResponseBody Long petClinicService  
.createOwner(@RequestBody Owner owner)
```

```
@RequestMapping(value="/owner/{id}",  
                method=RequestMethod.PUT)
```

```
@ResponseStatus(HttpStatus.OK)  
void petClinicService.updateOwner(  
@RequestBody Owner owner, @PathVariable long id)
```

```
@RequestMapping(value="/owner/{id}",  
                method=RequestMethod.DELETE)
```

```
@ResponseStatus(HttpStatus.OK)  
void petClinicService.deleteOwner(  
@PathVariable long id)
```

HttpMessageConverter ile Text – Nesne Dönüşümü

- HttpMessageConverter'lar ile REST çağrılarında transfer edilen nesnelerin **text-nesne dönüşümleri** gerçekleştirilir
- **<mvc:annotation-driven/>** elemanı tarafından değişik converter implementasyonları **built-in register** edilmektedir
- Başka converter'lar da **ilave** olarak register edilebilir

Response Content Tipinin Belirlenmesi

- Web isteği sonucunda dönülecek içeriğin formatı default durumda önce **request URI'in uzantısına**, yoksa **format request parametresine**, diğer durumda ise **Accept request header'ına** bakılarak belirlenir
- Bu ayarlar `<mvc:annotation-driven/>` elemanı built-in **ContentNegotiationManager** bean'i register etmektedir

Response Content

Tipinin Belirlenmesi: URI Path Ext.

GET /petclinic/pets/123.json

...

HTTP/1.1 200 OK

...

Content-Length:...

Content-Type: application/json

{

pet: {id:"123", ...}

}

Response Content Tipinin Belirlenmesi: Request Parameter

GET /petclinic/pets/123?format=json

...

HTTP/1.1 200 OK

...

Content-Length:...

Content-Type: application/json

{

pet: {id:"123", ...}

}

Response Content Tipinin Belirlenmesi: Accept Header

GET /petclinic/pets/123

...

Accept: application/json

...

HTTP/1.1 200 OK

...

Content-Length:...

Content-Type: application/json

{

pet: {id:"123", ...}

}

ContentNegotiationManager

Konfigürasyonu

```
<mvc:annotation-driven content-negotiation-manager="contentNegotiationManager"/>
```

```
<bean id="contentNegotiationManager"  
class="org.springframework.web.accept.ContentNegotiationManagerFactoryBean">
```

```
  <property name="favorPathExtension" value="true"/>
```

```
  <property name="favorParameter" value="false"/>
```

```
  <property name="ignoreAcceptHeader" value="false"/>
```

```
  <property name="mediaTypes">
```

```
    <map>
```

```
      <entry key="json" value="application/json"/>
```

```
      <entry key="xml" value="application/xml"/>
```

```
      <entry key="html" value="text/html"/>
```

```
    </map>
```

```
  </property>
```

```
  <property name="useJaf" value="false"/>
```

```
  <property name="defaultContentType" value="text/html"/>
```

```
</bean>
```

1. request URI uzantısı'na bakılır
2. Accept header değerine bakılır
3. "format" request param değerine bakılır

MediaTypes mevcut değil ise Java Activation Framework Kütüphanesi classpath'de mevcut ise mime type belirlemek için kullanılabilir

Response'un Farklı Tiplerde Üretilmesine Örnek

```
@RequestMapping(value="/owners/{id}", produces={"application/json", "application/xml"})  
@ResponseBody  
public Owner getOwner(@PathVariable Long id) {  
    return petClinicService.getOwner(id);  
}
```

http://localhost:8080/petclinic/mvc/owners/1.json -> application/json
http://localhost:8080/petclinic/mvc/owners/1.xml -> application/xml

http://localhost:8080/petclinic/mvc/owners/1.html -> text/html

```
@RequestMapping("/owners/{id}")  
@ResponseBody  
public String getOwnerAsHtml(@PathVariable Long id) {  
    StringBuilder builder = new StringBuilder();  
    Owner owner = petClinicService.getOwner(id);  
    builder.append("<html><body>");  
    builder.append(owner.getFirstName() + " " + owner.getLastName() + "<br>");  
    builder.append("</body></html>");  
    return builder.toString();  
}
```

RestTemplate Kullanımı

HTTP Metodu	RestTemplate Metodu
GET	<code>getForObject(URI url, Class<T> responseType):T</code>
POST	<code>postForObject(URI url, Object request, Class<T> responseType):T</code>
PUT	<code>put(URI url, Object request):void</code>
DELETE	<code>delete(URI url):void</code>

RestTemplate Kullanım

Örneği: GET

```
RestTemplate restTemplate = new RestTemplate();

ResponseEntity<String> entity =
    restTemplate.getForEntity(
        "http://localhost:8080/petclinic/mvc/vets/1/lastName",
        String.class);

String body = entity.getBody();

MediaType contentType =
    entity.getHeaders().getContentType();

HttpStatus statusCode = entity.getStatusCode();
```

RestTemplate Kullanım

Örneği: POST

```
RestTemplate restTemplate = new RestTemplate();
```

```
Vet vet = new Vet("Kenan", "Sevindik", 1999);
```

```
ResponseEntity<Long> entity =  
    restTemplate.postForEntity(  
        "http://localhost:8080/petclinic/mvc/vet", vet,  
        Long.class);
```

```
Long id = entity.getBody();
```

```
MediaType contentType =  
    entity.getHeaders().getContentType();
```

```
HttpStatus statusCode = entity.getStatusCode();
```


RestTemplate Kullanım

Örneği: Exchange

```
RestTemplate restTemplate = new RestTemplate();

Vet vet = new Vet("Kenan", "Sevindik", 1999);

RequestEntity<Vet> requestEntity =
    RequestEntity.post(
        new URI("http://localhost:8080/petclinic/mvc/vet"))
        .header("Authorization", "Basic aHR0cHdhGNo0mY=")
        .accept(MediaType.APPLICATION_JSON)
        .body(vet);

ResponseEntity<Long> responseEntity =
    restTemplate.exchange(requestEntity, Long.class);

Long id = responseEntity.getBody();
```

RestTemplate ve ClientHttpRequestInterceptor

- **ClientHttpRequestInterceptor** nesneleri vasıtası ile istemci tarafında HTTP request öncesi ve HTTP response döndükten sonra ilave işlemler yapılabilir
- Örneğin **BasicAuthorizationInterceptor** yardımı ile HTTP isteğine basic authentication header değerleri yerleştirilebilir

BasicAuthorization Interceptor Örneği

```
RestTemplate restTemplate = new RestTemplate();

BasicAuthorizationInterceptor basicAuthorizationInterceptor =
    new BasicAuthorizationInterceptor("user1", "secret");
restTemplate.setInterceptors(
    Arrays.asList(basicAuthorizationInterceptor));

Owner owner = restTemplate
    .getForObject("http://localhost/owners/1", Owner.class);
```

RestTemplate ve HTTP Client Konfigürasyonu

- RestTemplate default constructor ile yaratılırsa Java'nın **built-in http client kütüphanesi** kullanılır
- **ClientHttpRequestFactory** alan constructor'ı ile bu değiştirilebilir
- Bu sayede **Apache Commons HttpClient** kullanılabilir
- Apache Commons HttpClient **credentials desteği** sunar

RestTemplate ve HTTP Client Konfigürasyonu

```
CredentialsProvider credentialsProvider = new BasicCredentialsProvider();

credentialsProvider.setCredentials(
    AuthScope.ANY,
    new UsernamePasswordCredentials("user1", "secret"));

HttpClient httpClient = HttpClientBuilder.create()
    .setDefaultCredentialsProvider(credentialsProvider).build();

RestTemplate restTemplate = new RestTemplate(
    new HttpClientHttpRequestFactory(httpClient));

Vet vet = new Vet("Kenan", "Sevindik", 1999);

restTemplate.postForObject("http://localhost:8080/petclinic/mvc/vet",
    vet, Void.class);
```

Browser ile PUT ve DELETE Çağrıları

- HTML tarayıcılar **sadece GET ve POST** metotlarını destekler
- RestTemplate dışında **tarayıcı üzerinden** PUT ve DELETE tipindeki REST servislerine erişmek için iki yol mevcuttur
 - Javascript ile http metot çağrısı yapmak
 - veya POST metot üzerinden asıl metodu hidden field ile taşımak gerekir

Browser ile PUT ve DELETE Çağrıları

- Spring MVC, **POST** metot üzerinden asıl metodu **hidden field** olarak taşımayı seçmiştir
- **HiddenHttpMethodFilter** gelen hidden field değerini alarak asıl metodu tespit eder
- **_method** isimli parametre değerine bakılır

Browser ile PUT ve DELETE Çağrıları

```
<filter>
    <filter-name>httpMethodFilter</filter-name>
    <filter-class>
org.springframework.web.filter.HiddenHttpMethodFilter
    </filter-class>
</filter>

<filter-mapping>
    <filter-name>httpMethodFilter</filter-name>
    <servlet-name>DispatcherServlet</servlet-name>
</filter-mapping>
```


Browser ile PUT ve DELETE Çağrıları (Bug)

- **PUT** metodu ile form üzerinden gönderilen input, sunucu tarafında **@RequestBody** ile işaretlenmiş ise **MultiValueMap** nesnesine dönüştürülememektedir
- Nedeni **HiddenHttpMethodFilter**, HTTP request body'ye **_method** için eriştikten sonra **FormHttpMessageConverter**'in HTTP request body'sine tekrar erişmeye çalıştığında body'nin boş gelmesidir
 - **Workaround:** Veri **MultiValueMap** yerine **@PathVariable** ve **@RequestParam** ile metoda aktarılabilir

İletişim

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

