

# Spring ve REST



# REST Nedir?

- HTTP protokolü üzerinden **HTTP metotları ile gerçekleştirilen** web servis yaklaşımıdır
- **WSDL** tabanlı web servisleri HTTP'yi asıl tasarlandığı **uygulama düzeyi protokol** olarak değil, **transport protokol** olarak kullanmaktadır
- REST ile HTTP sadece transport protokol olarak değil, **uygulama düzeyi protokol** olarak da kullanılmaktadır

# Uygulama Protokolü Olarak HTTP

- Transport protokol verinin istemci ve sunucu arasında **nasıl taşınacağını** belirler
- Veri bölümlenir, encode/decode edilir, istemci – sunucu arasında transfer edilir vs.
- Ancak gerçekleştirilecek olan **işlem veya servisle ilgili** herhangi bir **anlam içermez**
- Uygulama düzeyi protokolde ise bir takım komutlar veya ifadeler ile **sunucu tarafında yapılması istenen işlem** de belirtilmektedir

# REST ve HTTP 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 Çağrıları 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

# REST Mimarisi

- **Stateless bir mimari** söz konusudur, çünkü HTTP'de stateless bir protokoldür
- State yönetimi gerekiyorsa **istemciler tarafından** yapılmalıdır
- Resource'ların **değişik formatlarda** gösterimi olabilir
  - XML, JSON, HTML vb.
- **HTTP header ve statü kodları** ile operasyonlarla ilgili bilgi istemci ve sunucu arasında paylaşılır

# Spring ve REST

- RESTful uygulamalar geliştirmek için Java EE6 standardı **JAX-RS**'dir
- JAX-RS'in Jersey, CXF, Restlet gibi gerçekleştirimleri mevcuttur
- Spring MVC 3.1 ile birlikte **RESTful** Web servisleri ve uygulamalar geliştirmek mümkün hale gelmiştir
- Fakat Spring MVC bir **JAX-RS** **gerçekleştirimi değildir**



# Spring ve REST

- REST servisleri **@Controller** bean'leri üzerinden hayata geçirilir
- Spring MVC ile REST servisleri oluşturmak için **aşağıdaki anotasyonlar** kullanılır
  - @RequestMapping
  - @PathVariable
  - @RequestBody
  - @ResponseBody
  - @ResponseStatus

# @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 {
    ...
}
```

# 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="/owners",  
                method=RequestMethod.POST)
```

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

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

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

```
@RequestMapping(value="/owners/{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

# Mevcut HttpResponseMessage Converter Nesneleri

StringHttpMessageConverter	String dönüşümü yapar
ByteArrayHttpMessageConverter	Byte[] dönüşümü yapar
ResourceHttpMessageConverter	Octet stream türündeki içeriği org.springframework.core.io.Resource'a dönüştürür
SourceHttpMessageConverter	javax.xml.transform.Source dönüşümü yapar
FormHttpMessageConverter	MultiValueMap<String,String> dönüşümü yapar
Jaxb2RootElementHttpMessageConverter	Classpath'de JAXB2 mevcut ise eklenir ve Java nesne – XML dönüşümü yapar
MappingJackson2HttpMessageConverter	Classpath'de Jackson2 mevcut ise eklenir ve Java nesne – JSON dönüşümü yapar
AtomFeedHttpMessageConverter	Classpath'de Rome mevcut ise eklenir ve Atom feed'lerin dönüşümünü yapar
RssChannelHttpMessageConverter	Classpath'de Rome mevcut ise eklenir ve RSS feed'lerin dönüşümünü yapar

# @RequestBody ile HTTP Mesajının Nesneye Dönüşümü

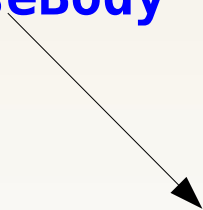
```
@RequestMapping(value="/owners", method=RequestMethod.POST)
public void createOwner(@RequestBody Owner owner) {
    // ...
}
```



Request ile gelen String verinin nesneye çevrimi için kullanılır. Nesne dönüşümü için kullanılacak `HttpMessageConverter` gelen **request'in content type'ına bakılarak** tespit edilir

# @ResponseBody ile Return Değerinin Mesaja Dönüşümü

```
@RequestMapping( value = "/pets/{petId}",  
                  method=RequestMethod.GET)  
public @ResponseBody Pet findPet(@PathVariable Long  
petId) {  
    // ...  
}
```



Nesne'nin response ile gönderilecek veriye dönüştürülmesini sağlar

Dönüşüm için uygun `HttpMessageConverter` gelen **request URI**'na, request **accept header**'ına veya spesifik bir **request parametresine** bakılarak otomatik olarak tespit edilir

# REST ve Statü Kodları

- RESTful servisler, istemcilerle haberleşmek için **http statü kodlarından** da yararlanırlar
- **@ResponseStatus** annotasyonu ile **HttpServletResponse**'a müdahale etmeden response'un statü kodu set edilebilir



# HTTP Statü Kodları

Statü Kod Grubu	Mesaj Kategorisi
1xx	Info
2xx	Success
3xx	Redirect
4xx	Client Error
5xx	Server Error

# @ResponseStatus

```
@RequestMapping(value="/{ownerId}/addPet",method=RequestMethod.POST)
```

```
@ResponseStatus(HttpStatus.CREATED)
```

```
public void addPet(@RequestBody Pet pet, @PathVariable Long ownerId) {  
    // ...  
}
```

Metot başarılı sonlandığı vakit Spring MVC DispatcherServlet response statü kodu olarak 201 CREATED değerini set edecektir

# @ResponseStatus

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class OwnerNotFoundException extends
RuntimeException {
    // ...
}
```

Bu exception herhangi bir controller metodundan fırlatıldığı zaman response statü kodu 404 NOT\_FOUND olacaktır

# İletişim

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

