

Java Servlets



Servlet Nedir?

- **Web server** tarafında çalışan java nesneleridir
- Client'dan gelen request'i ele alarak **dinamik olarak web sayfaları** oluşturmayı sağlarlar
 - HTML form ile **submit edilen veriyi** ele almakta kullanılırlar
 - Veritabanı **sorgu sonuçlarını** kullanıcılara göstermek için kullanılırlar
 - HTTP requestleri arasında **state bilgisini** korumaya yardımcı olurlar
- Çalışmaları için Servlet uyumlu bir **Web Container** (web server)'a deploy edilmeleri gerekir

Java Servlets vs CGI

- Web server tarafında program çalıştırarak, **dinamik içerik üretmek** için daha önce **CGI** ve **Perl** gibi programlar kullanılırdı
- Servlet'lerin CGI programlarına **üstünlükleri**
 - Her bir request için **ayrı bir process** üretmeye gerek yoktur
 - Request'ler arasında Servlet instance **hafızada hazır** bekler
 - Requestleri ele alan **tek bir instance** vardır, concurrent biçimde çalışabilirler

Java Servlets ve HTTP

- Günümüzde etkileşimli, dinamik web uygulamaları yapmak için **temel teknoloji** haline gelmişlerdir
- Servlet teknolojisi herhangi bir **client-server protokolüne bağımlı değildir**
- Ancak HTTP **en yaygın kullanılan protokoldür**
- Java Servlets, HTTP ile özdeşleşmiştir, çoğunlukla **HTTP Servlet** olarak da bilinirler
- Servlet'lar **Java sınıflarıdır** ve nesneleri JVM içerisinde çalıştırılır

HTTP Nedir?

- Web'in text tabanlı **network protokolüdür**
- Client ve Server arasındaki **request** ve **response**'larla iletişim söz konusu olur
- Bir web sunucusundaki **URL** ile tespit edilebilen herhangi bir “**resource**”a erişim sağlar
- Resource **herhangi bir büyüklükteki veridir**
 - Text dosya, resim, ses, video gibi **statik** dosya olabilir
 - Yada **dinamik** olarak üretilen bir veri olabilir
 - **URL ile tespit edilebilmelidir**

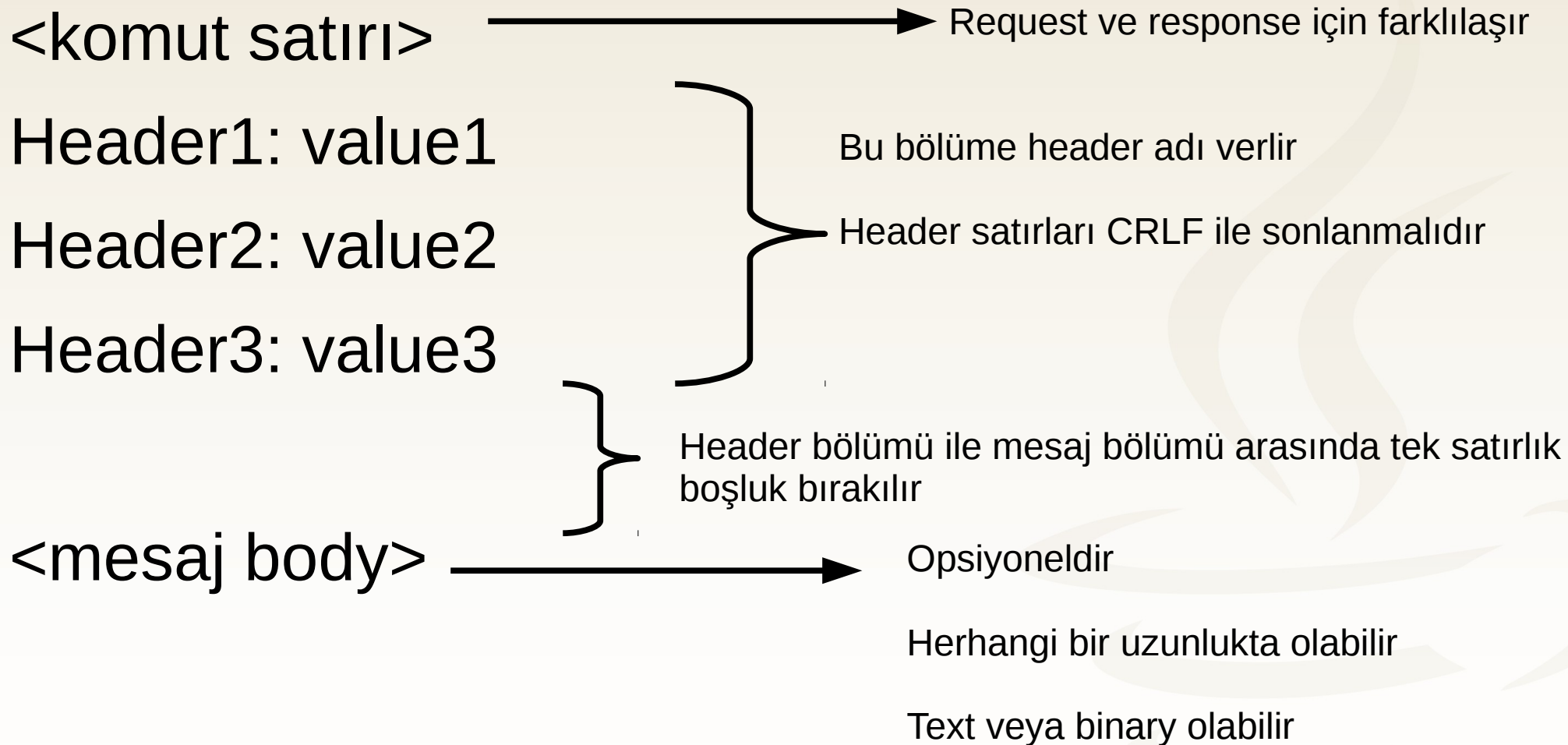
URL, URN ve URI Nedir?

- Uniform Resource Locator (**URL**) bir Uniform Resource Identifier (URI)'dir
- Bir **resource'un adresini ve ona erişim şeklini** tanımlar
- Uniform Resource Name (**URN**) de bir URI'dir
- Bir resource'un kimliğini lokasyonundan bağımsız olarak tanımlamayı sağlar
- URN **kimlik tanımı**, URL ise **lokasyon bilgisi** olarak düşünülebilir
- Kişinin kimliği ve bu kişinin yaşadığı yerin adresi gibi

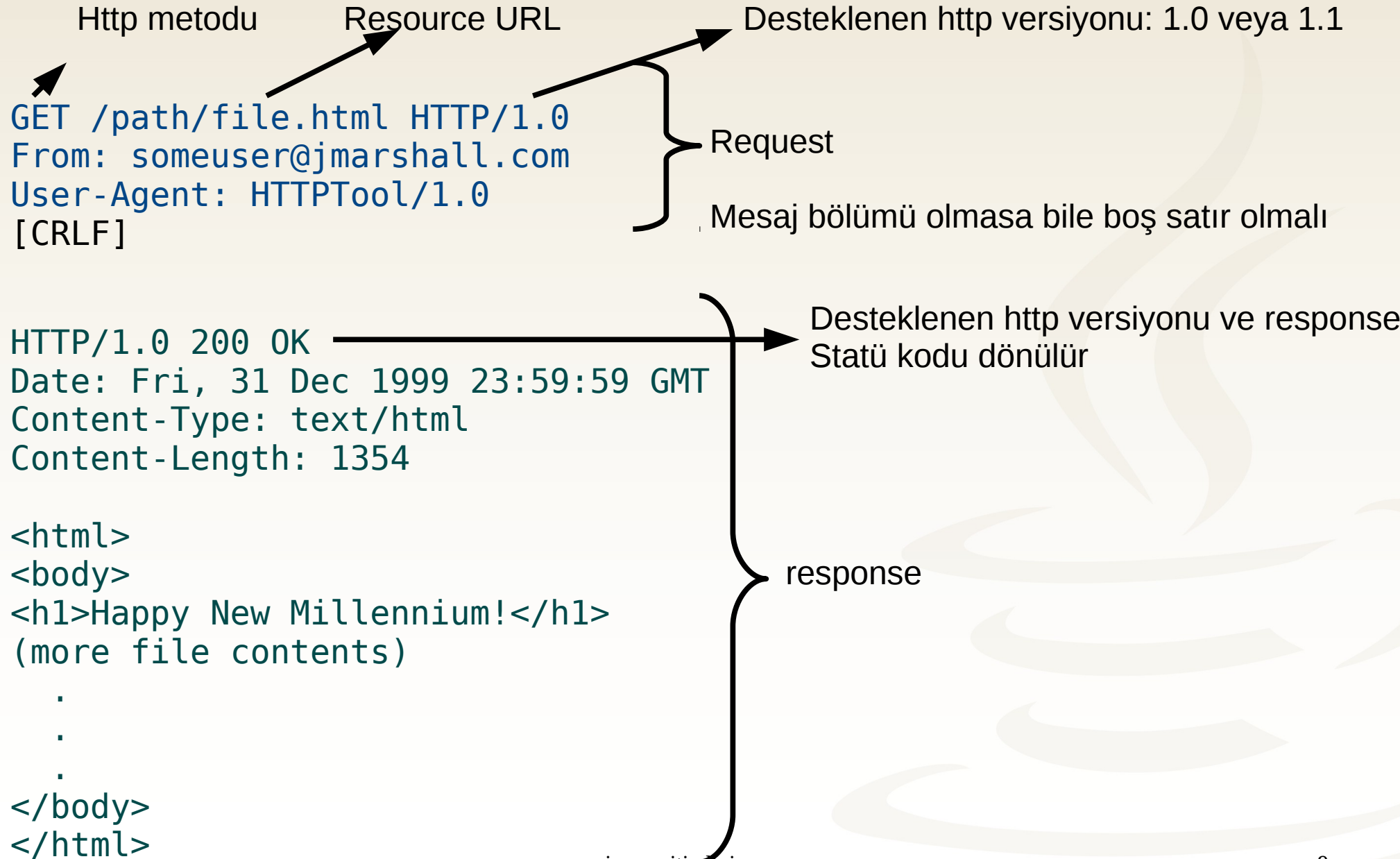
URL'in Yapısı

- *scheme://username:password@domain:port/resource_path?query_string#fragment_id*
- Scheme, **URL'in geri kalan syntax'ını** belirler
- <http://www.java-egitimleri.com:80>
- <mailto:ksevindik@harezmi.com.tr>
- <ftp://guest:secret@harezmi.com.tr>
- [http://10.10.0.1/myapp?
firstName=kenan&lastName=sevindik](http://10.10.0.1/myapp?firstName=kenan&lastName=sevindik)
- <http://java.sun.com/docs/jdk6.html#toc>

HTTP Request ve Response Yapısı



HTTP Request ve Response Örneği



HTTP GET ve POST Metotları

■ GET

- **Resource URI** ile belirtilen veriyi sunucudan alır
- Mesaj bölümü **Request URL**'inde taşınır
- Buna **query string** adı verilir
- Mesaj bölümünün **uzunluğu** bu nedenle **sınırlıdır**
- Aynı GET request'nin sunucu tarafında tekrar tekrar çalıştırılması herhangi bir **side effect yaratmamalıdır**
- Başka deyişle sunucu tarafında herhangi bir **değişikliğe neden olmamalıdır**

HTTP GET ve POST Metotları

- **POST**
 - **Resource URI** ile belirtilen veriyi sunucudan alır
 - Request ile gönderilen **mesaj bölümü stream** olarak sunucuya aktarılır
 - Bu nedenle gönderilecek **verinin uzunluğu** için herhangi bir **sınır yoktur**
 - Sunucu tarafında bir **değişikliğe neden olacak** işlemler bu metot ile gerçekleştirilir
 - Örneğin bir hesaptan başka hesaba para transferi yapılması

HTTP Response Statü Kodları

- 200: OK
- 201: Resource created
- 204: Response empty
- 30x: Redirect
- 404: Resource not found
- 405: HTTP method not supported
- 409: Resource conflict
- 500: Internal server error

Java Servlets ve Web Uygulamaları

Jetty, Tomcat yaygın kullanılan
iki web container'dır

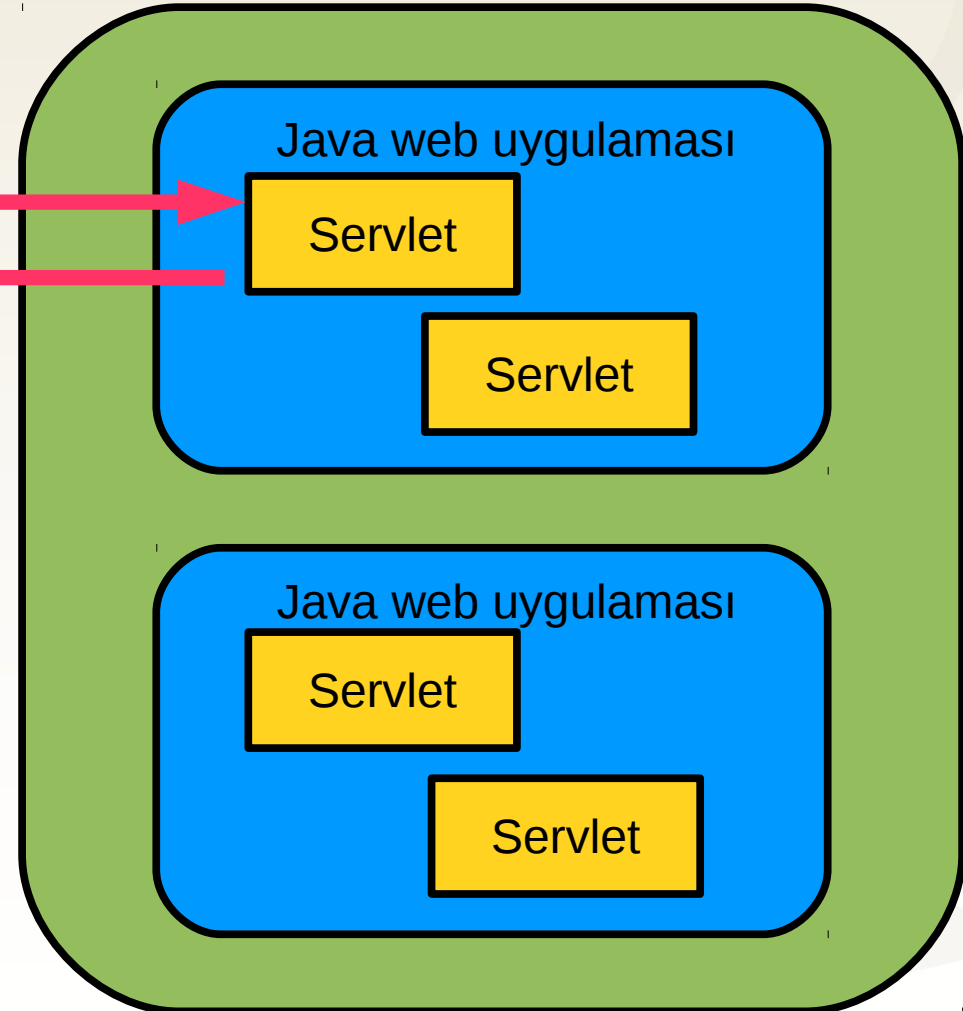
Web/Servlet container



Client Browser

HTTP request




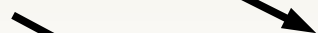
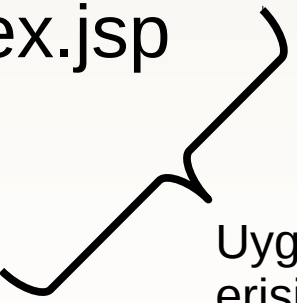

HTTP response



Bir web container içerisinde birden fazla web uygulaması aynı anda çalışıyor olabilir

Aynı web uygulaması içinde yer alan Servlet instance'ları birbirleri arasında veri alışverişinde bulunabilir

Java Web Uygulamalarının Yapısı

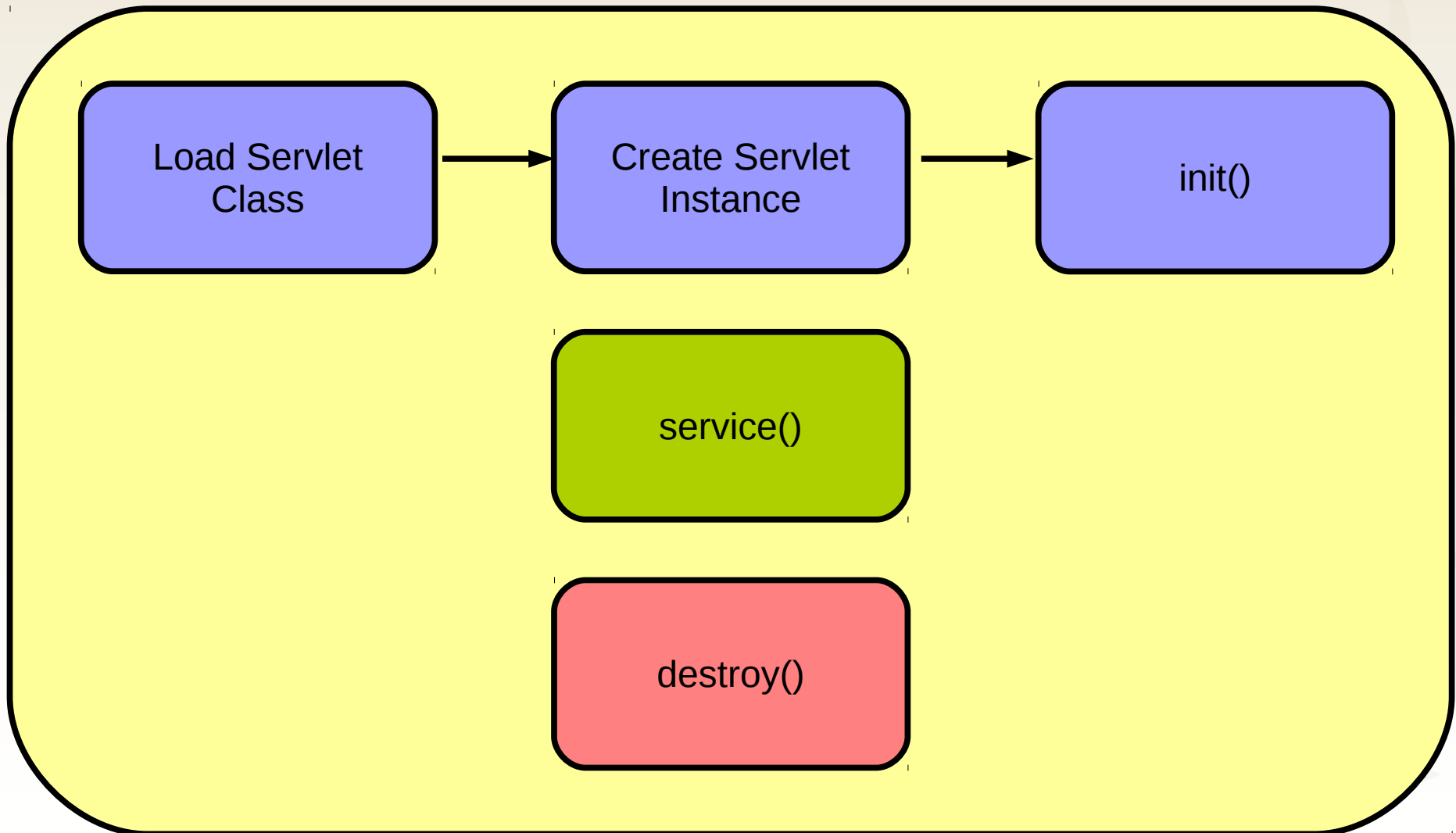
- petclinic (dizin)  Root dizindir, bu dizin altındaki dosyalar uygulama server'a deploy edildiğinde erişilebilir hale gelir
 - WEB-INF (dizin)  Özel bir dizindir, client tarafından doğrudan erişilemez
 - classes (dizin)  Uygulamanın java sınıflarını ve Classpath'de bulunması gereken diğer Resource'larını içerir
 - lib (dizin)
 - web.xml  Web uygulamasının ihtiyaç duyduğu Jar'ları içerir
 - index.jsp
 - ...
-  Uygulamaya ait erişilebilir dosya ve dizinler root dizinin altında oluşturulur
-  Web uygulamasının konfigürasyon dosyasıdır

Web Archive Dosyası (WAR)

- Java Archive (**JAR**) dosyasıdır
- JAR dosya formatı **ZIP** dosya formatı üzerine bina edilmiştir
- Web uygulama **dizin yapısının tek bir dosya halinde paketlenip deploy edilmesini sağlar**
- Dosyanın uzantısı “**.war**” şeklinde olmalıdır

Servlet Yaşam Döngüsü

Servlet Container



Servlet Instance'ın Oluşturulması

- Servlet container **servlet sınıfını yükler** ve bir **instance oluşturur**
- Ardından **init() metodu** çağrılarak servlet instance'ın kendisini initialize etmesi sağlanır
- **init()** metoduna web.xml vasıtası ile **parametreler** geçilebilir
- Bu işlem servlet container'ın **start aşamasında** veya servlet'a gelen **ilk request öncesinde** gerçekleşebilir
- Bu davranış da web.xml içerisinden kontrol edilebilir

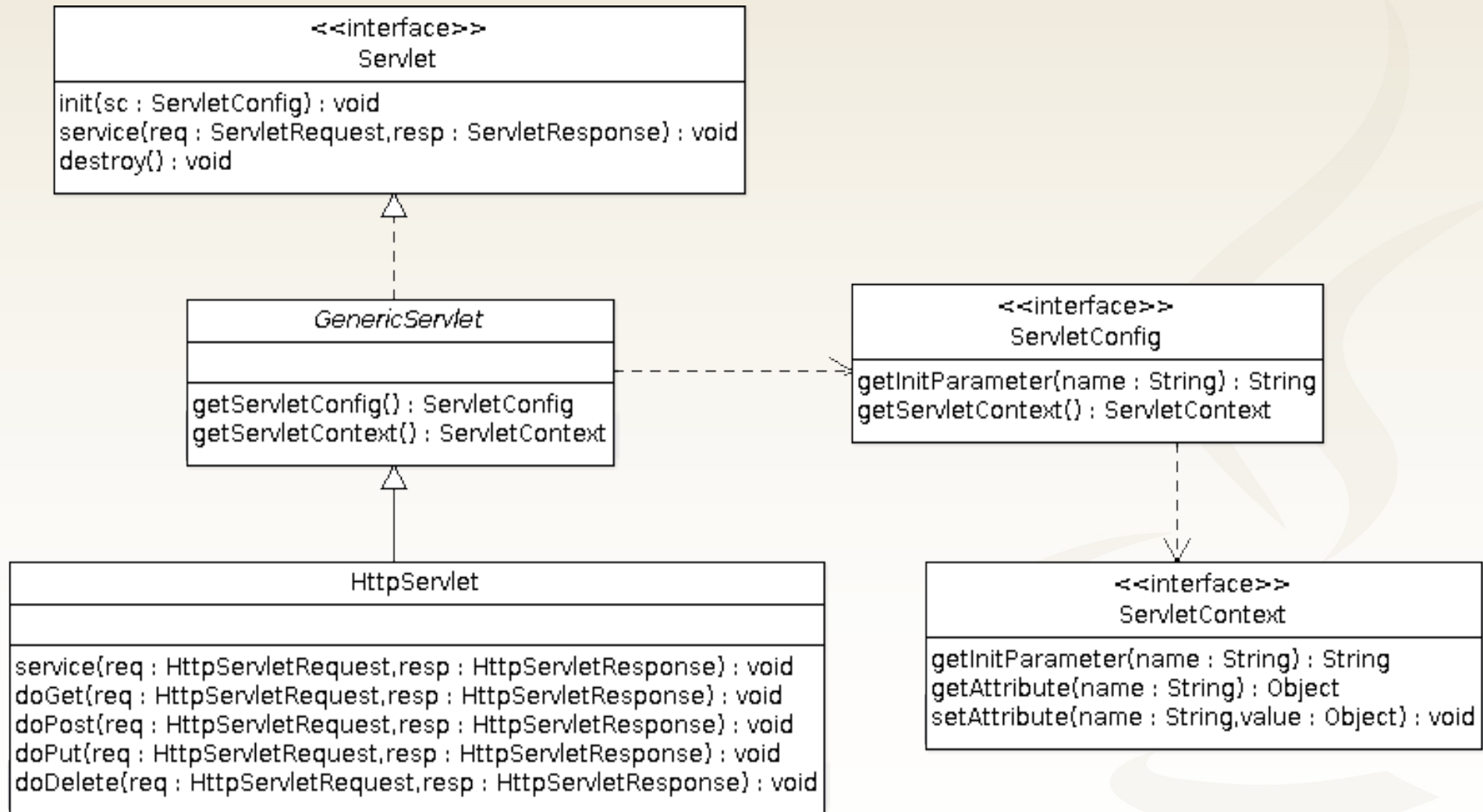
HTTP İsteklerinin Karşılanması

- HTTP istekleri geldiğinde **service()** metodu çalıştırılır
- Bu metot HttpServlet sınıfında **doGet()** veya **doPost()** metotlarından herhangi birisini tetikler
- Servlet **aktif olduğu müddetçe** HTTP isteklerine cevap verebilir
- Dolayısı ile service() metodu **pek çok defa çalıştırılabilir**

Servlet Instance'ının Destroy Edilmesi

- Servlet instance unload edilirken **destroy()** metodu çalıştırılır
- **Unload işleminin yapıldığı noktalar**
 - Servlet **container'ın kapatılması**
 - Web uygulamasının dinamik olarak container içerisinde **reload edilmesi**

Servlet Class Diagram



HttpServlet ve web.xml Konfigürasyon Örneği

```
public class HelloWorldServlet  
    extends HttpServlet {
```

```
    protected void  
doGet(HttpServletRequest request,  
HttpServletResponse response) throws  
ServletException, IOException {
```

```
        doPost(request, response);  
    }
```

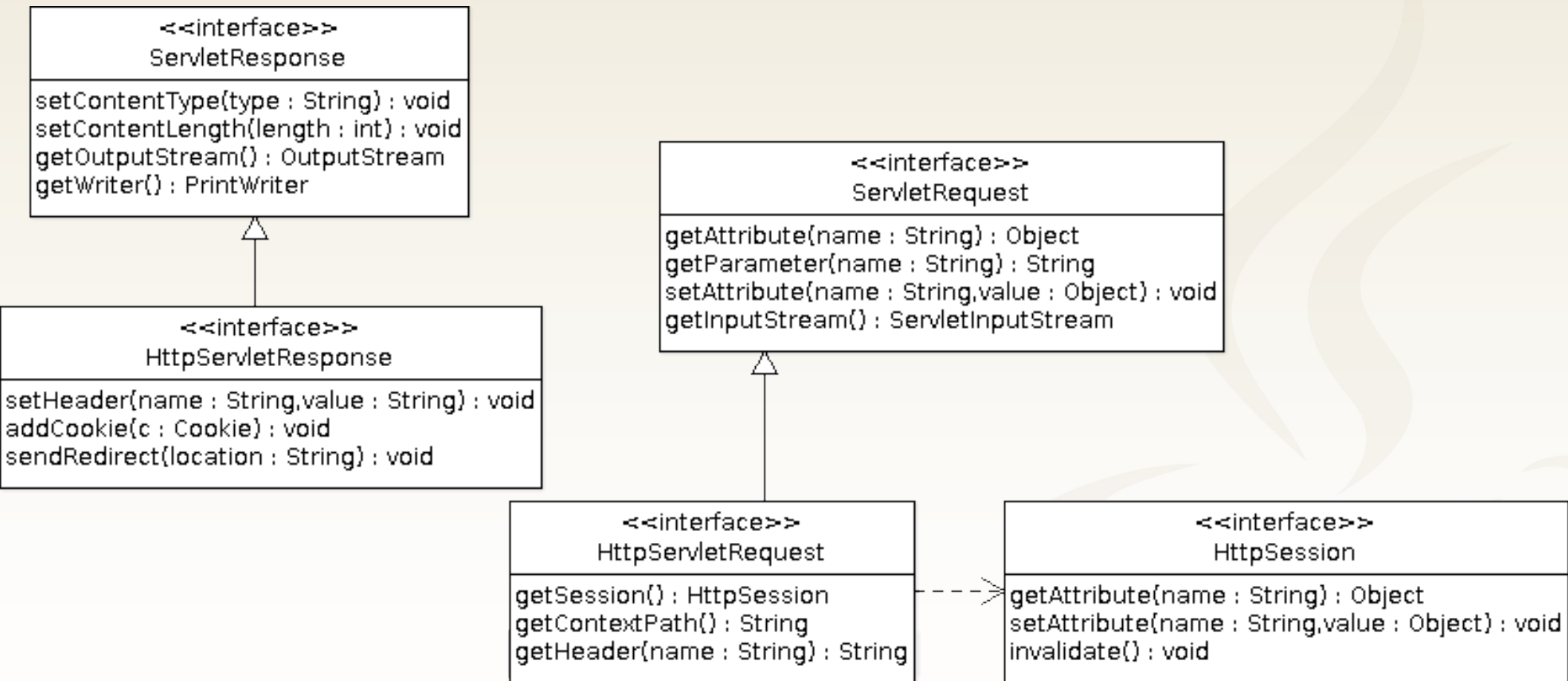
```
    protected void  
doPost(HttpServletRequest request,  
HttpServletResponse response) throws  
ServletException, IOException {
```

```
        response.getWriter().write("Hello  
world");  
    }  
}
```

```
<web-app>  
    <display-name>petclinic</display-  
name>  
    <servlet>  
        <servlet-name>  
HelloWorldServlet  
        </servlet-name>  
        <servlet-class>  
com.javaegitimleri.petclinic.web.He  
lloWorldServlet  
        </servlet-class>  
    </servlet>  
    <servlet-mapping>  
        <servlet-name>  
HelloWorldServlet  
        </servlet-name>  
        <url-pattern>  
/hello  
        </url-pattern>  
    </servlet-mapping>  
</web-app>
```

Web.xml konfigürasyonu

ServletRequest/Response Class Diagram



HttpServletRequest

- Request **parametrelerine** erişmeyi sağlar
- Request **header**'larına erişmeyi sağlar
- Request **body**'sine **InputStream** olarak erişmeyi sağlar
- **HttpSession** ve **ServletContext** nesnelerine erişim sağlar
- **Request attribute**'ları vasıtası ile request'in ömrü boyunca içerisinde **veri tutulabilir**, diğer servlet'ler ile attribute'lar vasıtası ile **veri paylaşılabilir**

HttpServletResponse

- Response **header** değerlerini **set etmeyi** sağlar
- Response'un **Content-Type** ve **Content-Length** bilgilerini set etmek mümkündür
- Response içeriğini **text veya binary** olarak oluşturmaya izin verir
- Farklı bir **URL'e redirect** etmeye imkan tanır

Text veya HTML İçerik Oluşturma

http://localhost:8080/petclinic/hello?name=kenan

```
String name = request.getParameter("name");
```

```
response.setContentType("text/html");
```

Response içeriği oluşturulmadan
Evvel Content-Type header
Uygun mime type değerine set
edilmelidir

```
PrintWriter writer = response.getWriter();
```

```
String message = "<html><body><h2>Merhaba " + name +  
"</h2></body></html>";
```

```
response.setContentLength(message.length());
```

```
writer.write(message);
```

```
writer.close();
```

Response içeriğinin uzunluğu da
Content-Length header'ına set edilmelidir

Özellikle binary içerikte önemlidir

Response içeriğini oluşturma işlemi
Bitince Writer nesnesi kapatılmalıdır

Binary İçerik Oluşturma

```
InputStream is = new
FileInputStream("/home/ksevindik/Desktop/harezmi.jpg");
ByteArrayOutputStream bout = new ByteArrayOutputStream();
while(is.available() != 0) {
    bout.write(is.read());
}
is.close();
```

Öncelikle
Binary içerik
Servlet
Instance'a
yüklenir

```
byte[] image = bout.toByteArray();
```

```
response.setContentType("image/jpeg");
```

→ Mime type set edilir

```
ServletOutputStream out = response.getOutputStream();
```

```
response.setContentLength(image.length);
```

```
out.write(image);
```

```
out.close();
```

Binary içeriğin uzunluğu
Content-Length header ile
Client browser'a belirtilir

→ İçerik yazıldıktan sonra stream kapatılır

Web.xml'de Servlet Konfigürasyonu

```
<web-app>
  <servlet>
    <servlet-name>PetClinicServlet</servlet-name>
    <servlet-class>x.y.z.PetClinicServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>PetClinicServlet</servlet-name>
    <url-pattern>/petClinicServlet</url-pattern>
  </servlet-mapping>
</web-app>
```



Url-pattern Servlet'e nasıl erişileceğini tanımlar

* ile wildcard kullanımda mümkündür: *.html, /petClinic* /*

Web.xml'de Servlet Konfigürasyonu

```
<web-app>
```

```
<context-param>
```

```
  <param-name>contextParam</param-name>
```

```
  <param-value>value</param-value>
```

```
</context-param>
```

Context param tanımlarına
ServletContext ile erişilebilir

Bütün Servlet'lar erişebilir

```
<servlet>
```

```
  <servlet-name>PetClinicServlet</servlet-name>
```

```
  <servlet-class>x.y.z.PetClinicServlet</servlet-class>
```

```
  <init-param>
```

```
    <param-name>servletParam</param-name>
```

```
    <param-value>value</param-value>
```

```
  </init-param>
```

```
  <load-on-startup>1</load-on-startup>
```

Init-param servlet tanımına
Özeldir

ServletConfig ile erişilebilir

```
</servlet>
```

```
</web-app>
```

Servlet'in startup sırasında yüklenmesini sağlar
Değer servlet tanımları arasındaki sıralamayı belirler,
Küçük değer önce yüklenir, Negatif değerde veya tanım yoksa
yükleme herhangi bir anda yapılabilir

@WebServlet ile Servlet Konfigürasyonu

Web.xml deki tanımlar yerine **WebServlet** anotasyonu ile de Servlet konfigürasyonu yapılabilir. Servlet'in hangi url'lerde devreye gireceği, init parametreleri vs buradan belirtilebilir

```
@WebServlet(name="HelloServlet",  
            urlPatterns={"/hello"},loadOnStartup=1)  
public class HelloWorldServlet extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request,  
                          HttpServletResponse response)  
        throws ServletException, IOException {  
  
        doPost(request, response);  
    }  
  
    protected void doPost(HttpServletRequest request,  
                          HttpServletResponse response)  
        throws ServletException, IOException {  
  
        response.getWriter().write("Hello world");  
    }  
}
```

RequestDispatcher Ne İşe Yarar?

- Bir **servlet** içerisinde **başka bir servlet'i** **çağırarak** için kullanılır
- RequestDispatcher **HttpServletRequest** üzerinden elde edilir
- **Forward ve include** metotları vardır
- Forward, ilk servlet'in **response'unu sonlandırdıktan sonra** ikinci servlet'i çağırır
- Include, iki servlet'in **response içeriğini merge** etmeye imkan tanır

RequestDispatcher Kullanım Örneği

```
public class HelloWorldServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
        response.getWriter().write("hello ");  
        request.getRequestDispatcher("/bye").include(request,  
            response);  
    }  
}
```

Servlet'in web.xml'de map edildiği url-pattern ile
RequestDispatcher nesnesi oluşturulur

Include ilk servlet'in response'u ile
İkincisini merge eder
Forward kullanılsaydı response
Sadece ikinci servlet tarafından
oluşturulacaktı

```
public class GoodByeServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
        response.getWriter().write("and goodbye!");  
    }  
}
```

ServletContext Nedir?

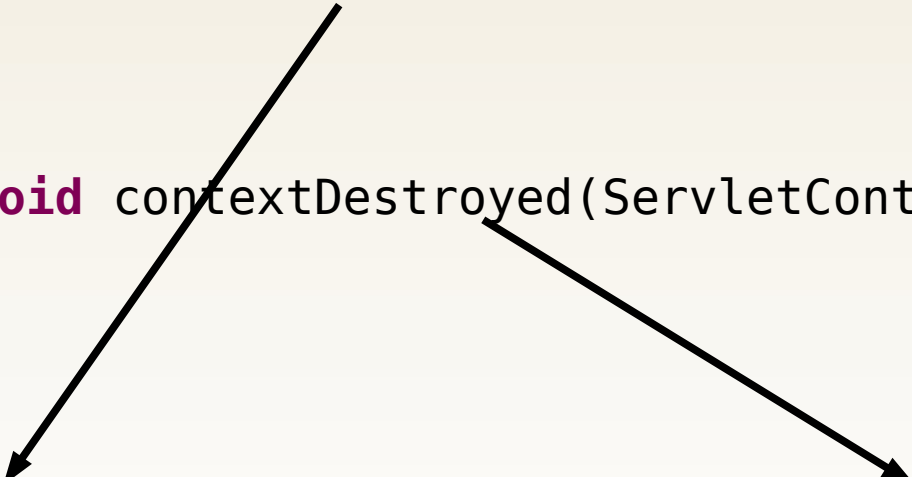
- HttpSession'da tutulan attribute'lar sadece **tek bir kullanıcı tarafından** erişilebilir
- ServletContext ise **uygulama genelinde bilgi tutmayı** sağlayan global bir **veri yapısıdır**
- ServletContext nesnesine **HttpServletRequest** aracılığı ile erişilebilir
- ServletContext'de tutulan attribute'lar ise **bütün kullanıcılar tarafından** erişilebilir

ServletContextListener Ne İşe Yarar?

- Web uygulamasının **servlet context'inde meydana gelen değişikliklerden** haberdar edilirler
- Değişikliklerden haberdar olabilmesi için **web.xml içerisinde listener** sınıfın tanımlanması gerekir

ServletContextListener Örneği

```
public class MyContextListener implements ServletContextListener {  
  
    public void contextInitialized(ServletContextEvent sce) {  
    }  
  
    public void contextDestroyed(ServletContextEvent sce) {  
    }  
  
}
```



Web uygulamasının initialization
Sürecinin başladığını haber verir

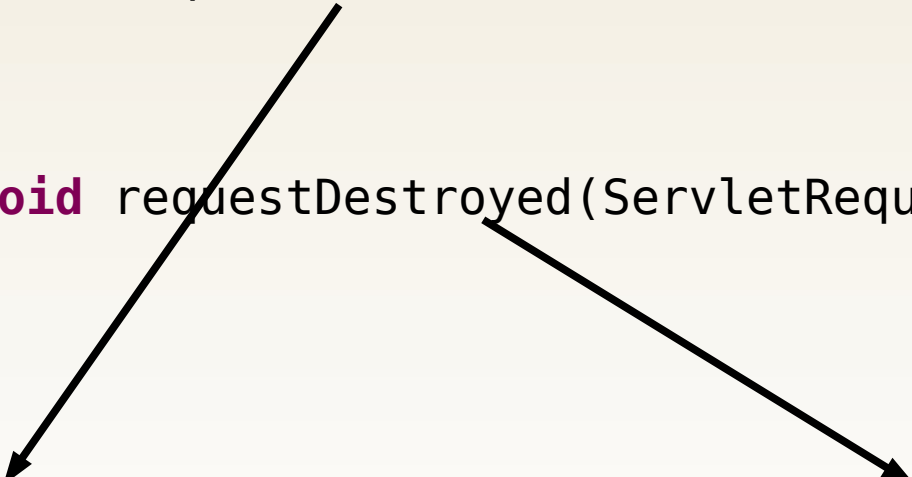
Bu metot çağrıldığında henüz herhangi bir
Servlet veya filter init edilmemiştir

Web uygulamasının ServletContext'inin
Sonlandırıldığını haber verir

Bu metot çağrılmadan evvel bütün
Servlet ve filter instance'ları destroy
edilmiştir

ServletRequestListener Örneği

```
public class MyRequestListener implements ServletRequestListener {  
    public void requestInitialized(ServletRequestEvent sce) {  
    }  
    public void requestDestroyed(ServletRequestEvent sce) {  
    }  
}
```



Her web request'i sırasında devreye girer
HttpServletRequest'in initialize edildiğini
bildirir. Event içerisinde current request'e
erişilebilir

Web request'i sonlandığı vakit çağrılır

Web.xml'de ServletContext ve RequestListener Konfigürasyonları

```
<web-app>
```

```
  <listener>
```

```
    <listener-class>x.y.z.MyContextListener</listener-class>
```


```
  </listener>
```

```
  <listener>
```

```
    <listener-class>x.y.z.MyRequestListener</listener-class>
```

```
  </listener>
```

```
</web-app>
```



ServletContextListener ve ServletRequestListener Tanımlarıda web.xml içerisinde listener elemanları ile ayrı ayrı yapılmalıdır

@WebListener ile Listener Konfigürasyonları

ServletRequestListener, ServletContextListener, HttpSessionListener tanımları web.xml yerine WebListener anotasyonu ile yapılabilir



@WebListener

```
public class MyRequestListener implements ServletRequestListener {  
  
    @Override  
    public void requestInitialized(ServletRequestEvent sre) {  
        System.out.println("request initialized");  
    }  
  
    @Override  
    public void requestDestroyed(ServletRequestEvent sre) {  
        System.out.println("request destroyed");  
    }  
}
```

İletişim



www.harezmi.com.tr

www.java-egitimleri.com



info@harezmi.com.tr

info@java-egitimleri.com



[@HarezmiBilisim](https://twitter.com/HarezmiBilisim)

[@JavaEgitimleri](https://twitter.com/JavaEgitimleri)