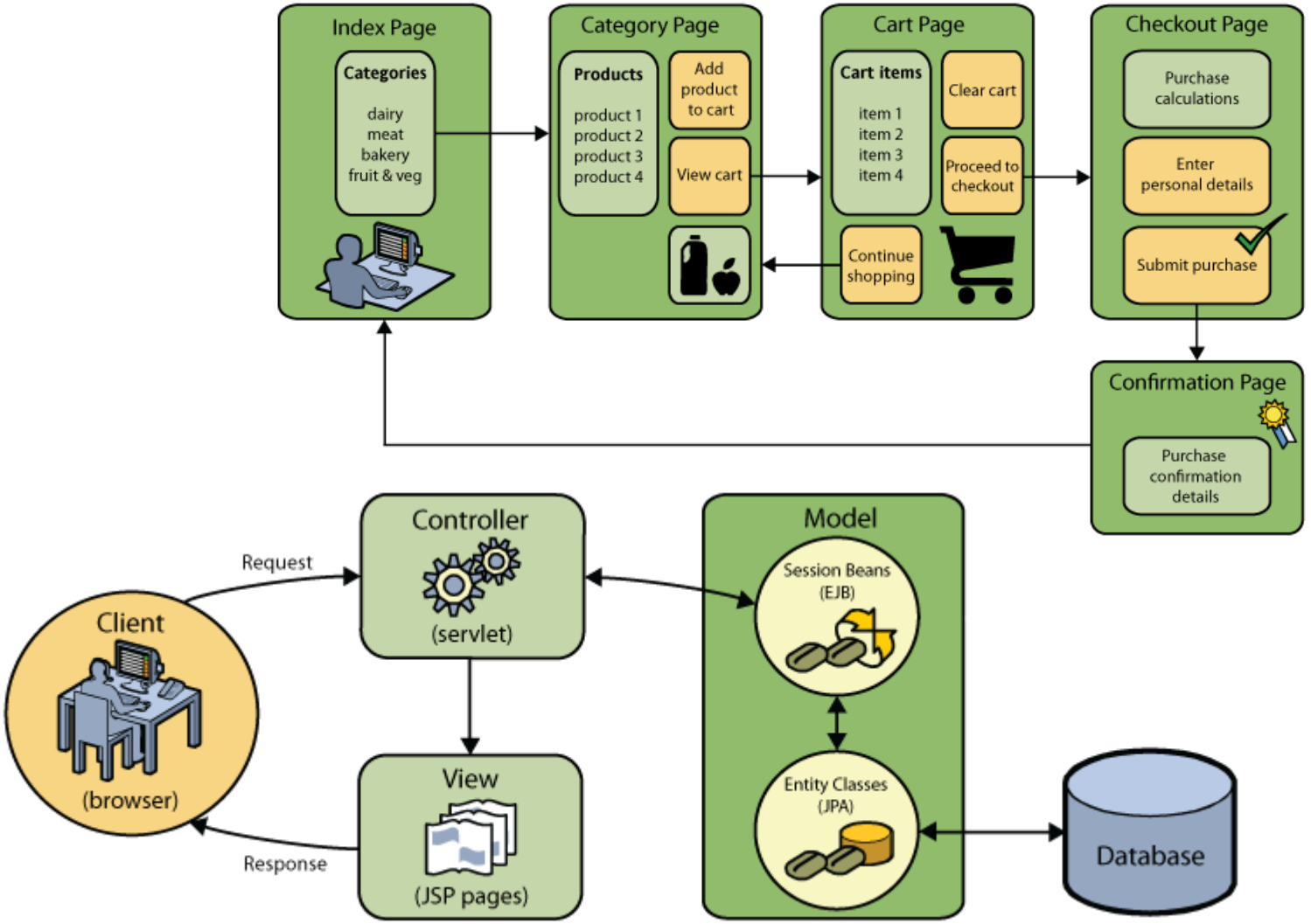
Java, http, web and services

Jfernan@ua.pt

Your application



https://netbeans.org/kb/docs/javaee/ecommerce/design.html

HTTP is one of the main glues

- Http is Text based and used on Web
  - No need for adaptations
  - Still text based, not most efficient and secure
  - No intention of explaining http in detail
- Web servers receive http requests
- Text based means
  - XML, JSON, SOAP,....

HTML on a Web server

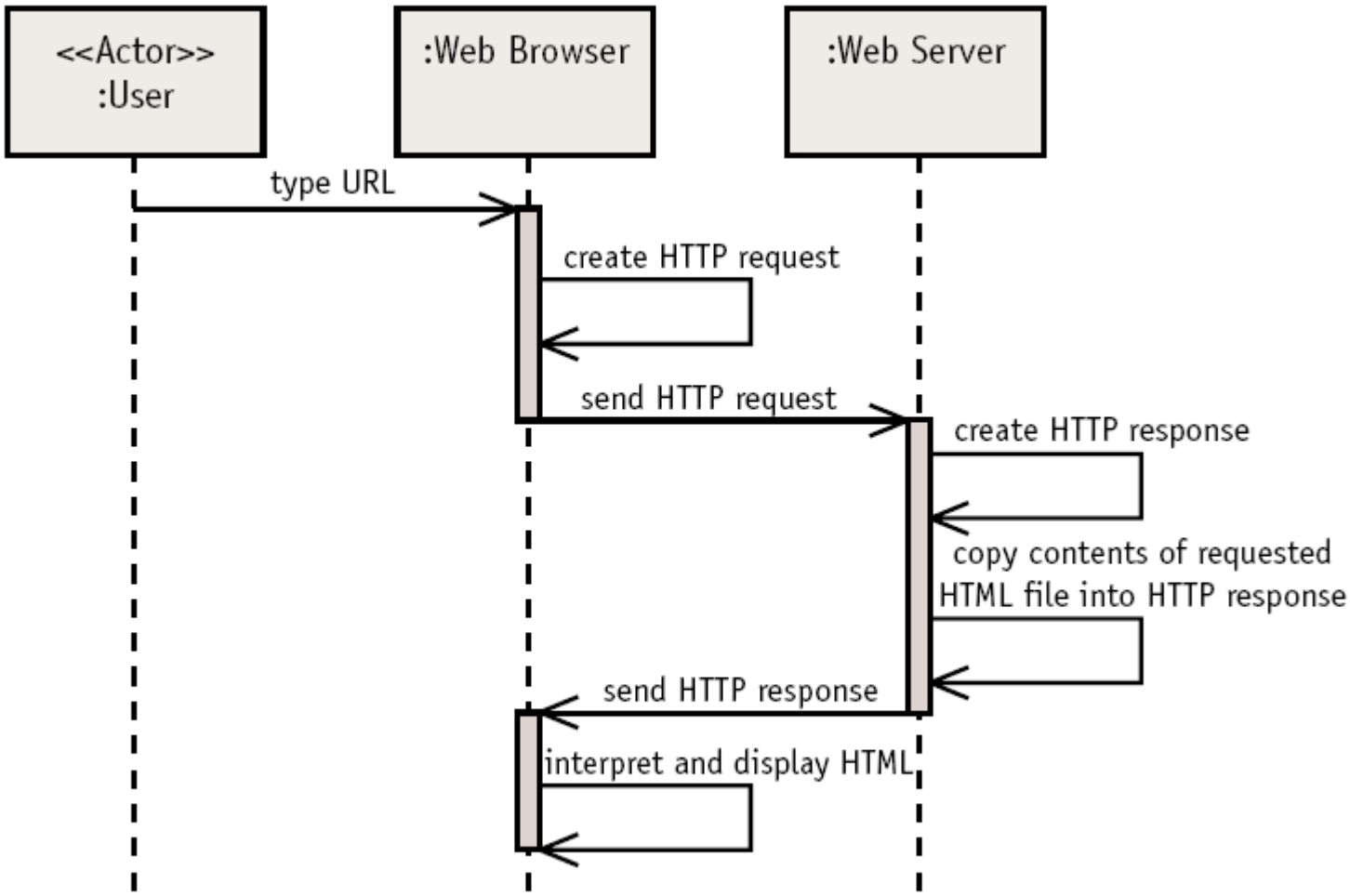


Figure 27-2 UML Sequence diagram, HTTP protocol

Examples from "UML Weekend Crash Course", Wiley, 2002

http and servlets

- Servlets handle Http requests in java
  - Look elsewhere for detailed description of HTTP
- Requests can be
  - GET – “small” request
  - POST – request with “attachments”
  - PUT - ... usually not used...
  - DELETE - ... usually not used ...

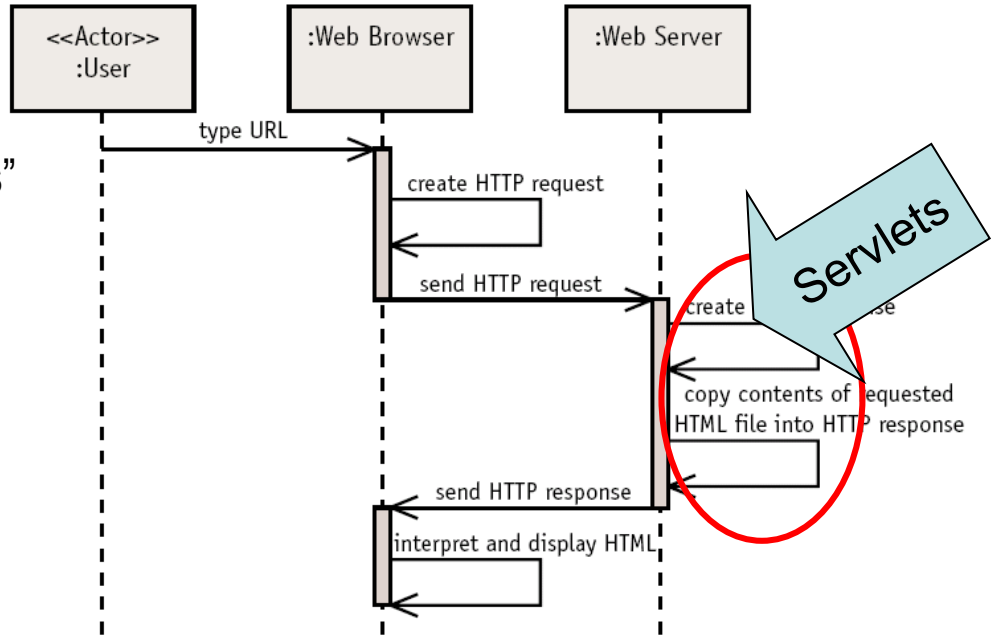
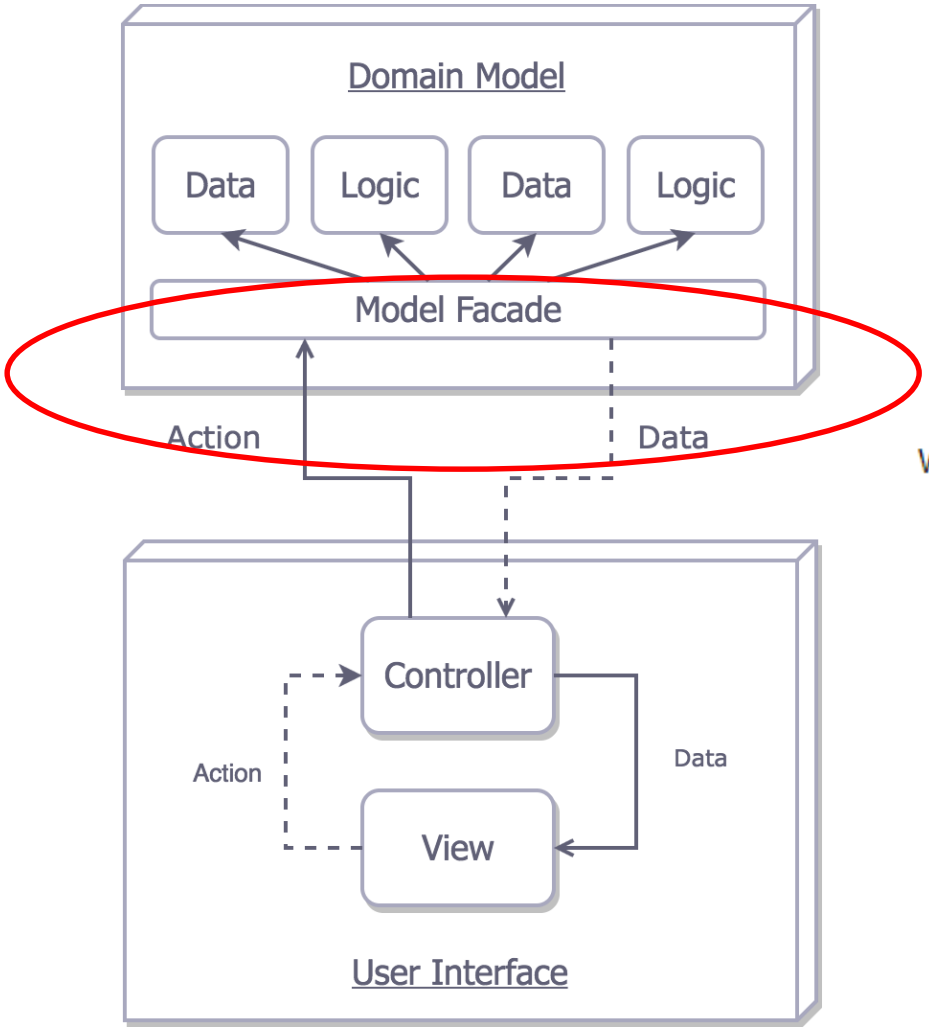
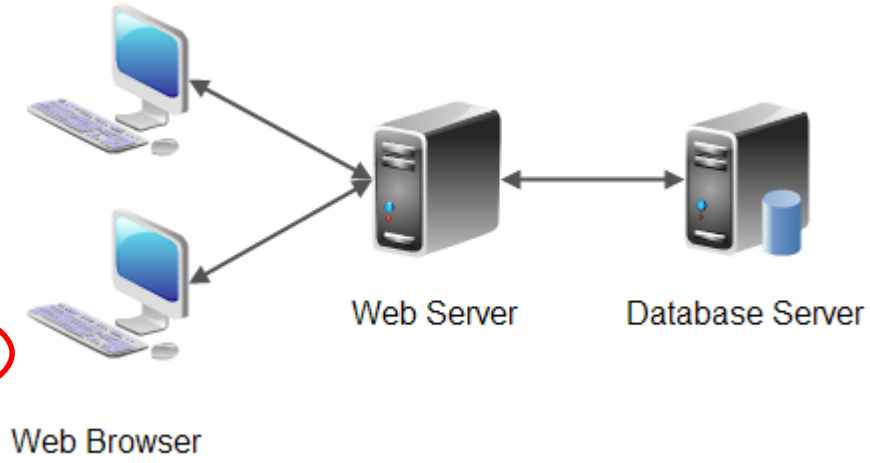
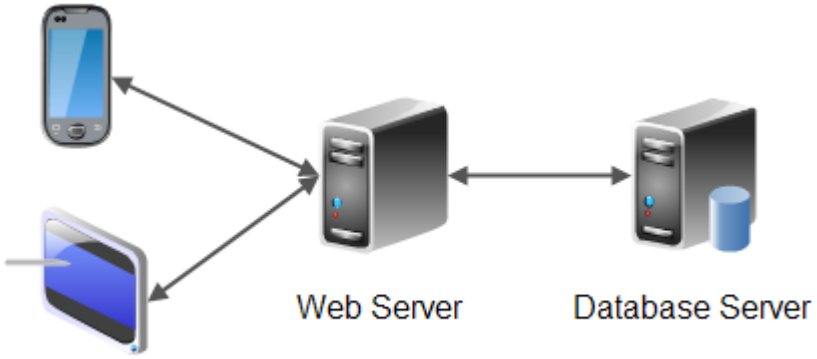


Figure 27-2 *UML Sequence diagram, HTTP protocol*

Http: the web integration







http://tutorials.jenkov.com/software-architecture/n-tier-architecture.html

Servlets

- Basic support for HTTP handling in Java

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Creating Http Servlet by Extending HttpServlet class
public class ExampleHttpServlet extends HttpServlet
{
    private String mymsg;
    public void init() throws ServletException
    {
       mymsg = "Http Servlet Demo";
    }
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException
    {
        // Setting up the content type of web page
        response.setContentType("text/html");
        // Writing the message on the web page
        PrintWriter out = response.getWriter();
        out.println("<h1>" + mymsg + "</h1>");
        out.println("<p>" + "Hello Friends!" + "</p>");
    }
    public void destroy()
    {
       // Leaving empty. Use this if you want to perform
       //something at the end of Servlet life cycle.
    }
}
```

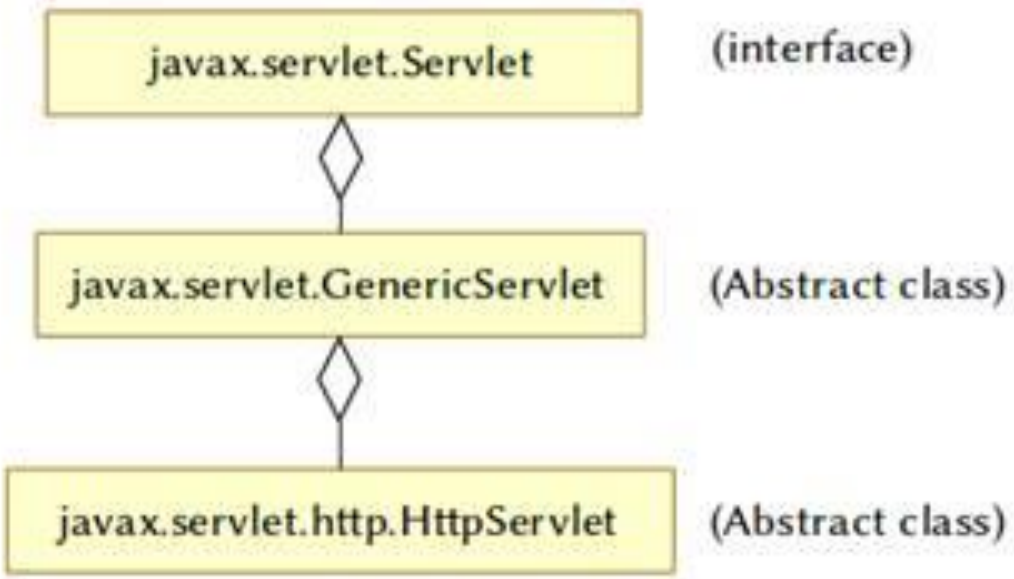
```xml
<web-app>
<display-name>BeginnersBookServlet</display-name>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
<welcome-file>index.jsp</welcome-file>
<welcome-file>default.html</welcome-file>
<welcome-file>default.htm</welcome-file>
<welcome-file>default.jsp</welcome-file>
</welcome-file-list>

<servlet>
<servlet-name>MyHttpServlet</servlet-name>
<servlet-class>ExampleHttpServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>MyHttpServlet</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

Servlets

- A servlet is a **Java programming language class** used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications **hosted by web servers**. For such applications, Java Servlet technology defines **HTTP-specific servlet classes**.

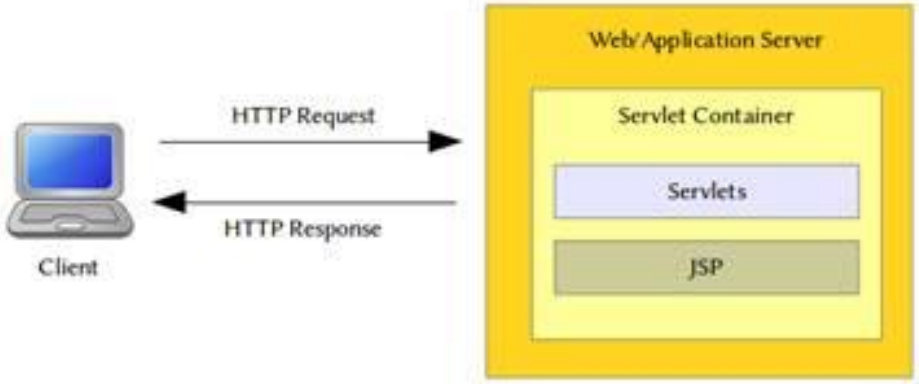
What Is a Servlet? in The Java EE 6 Tutorial
https://docs.oracle.com/javaee/6/tutorial/doc/bnafe.html

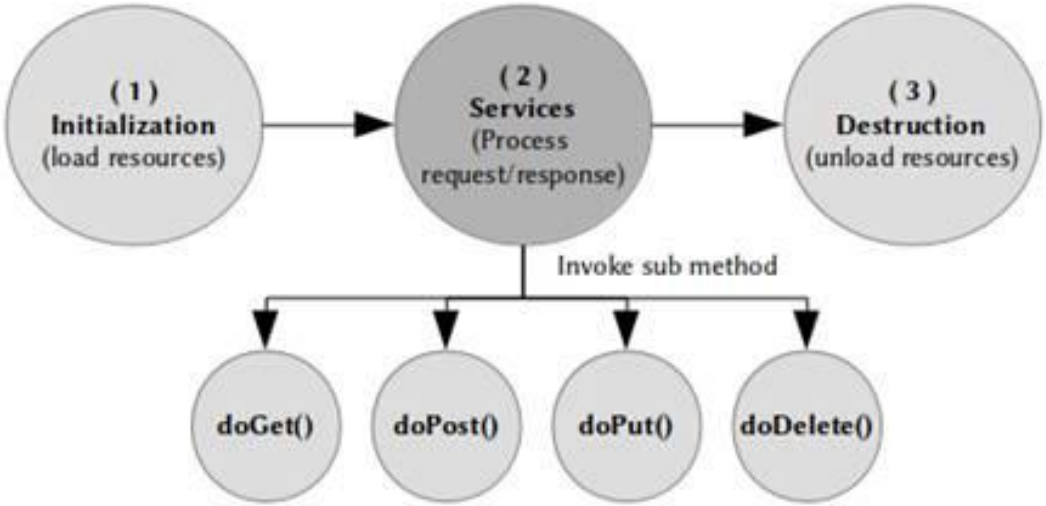
Servlet is java



Understanding the Java Servlet Life Cycle
https://www.developer.com/java/web/understanding-the-java-servlet-life-cycle.html

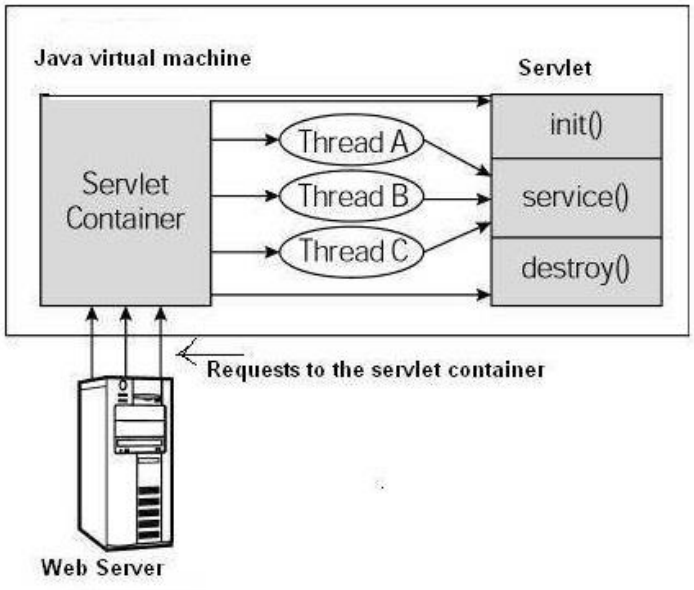
Servlets need a container

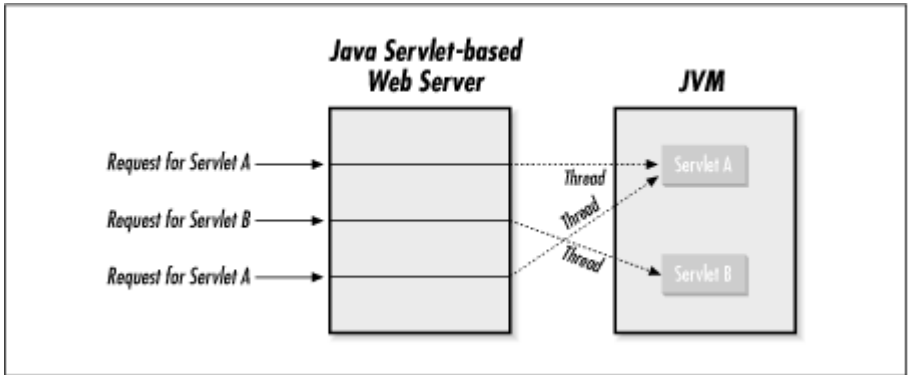


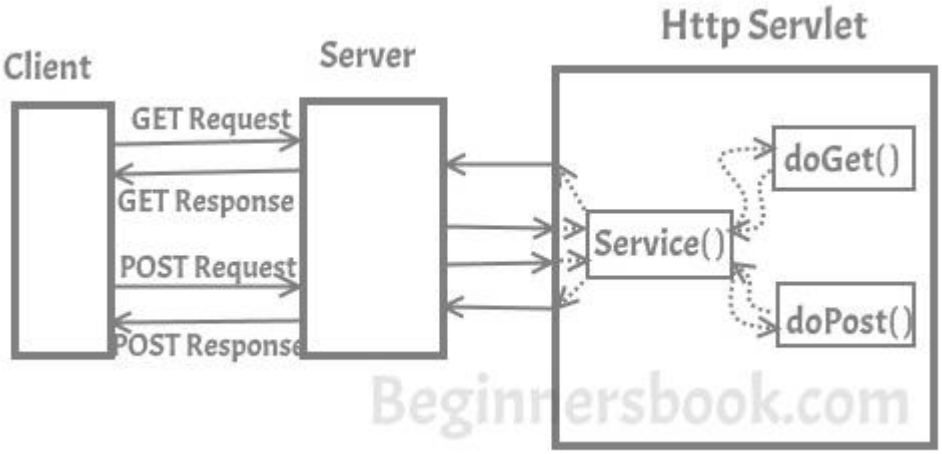


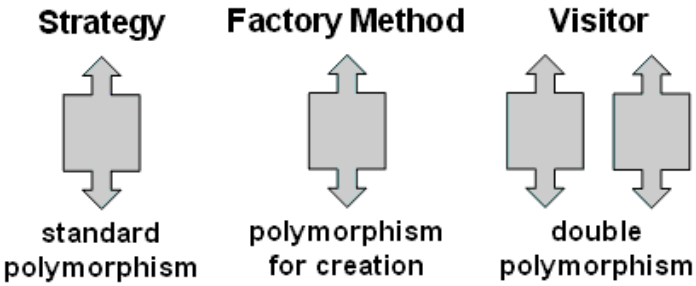
Understanding the Java Servlet Life Cycle
https://www.developer.com/java/web/understanding-the-java-servlet-life-cycle.html

Servlets need a runtime with container





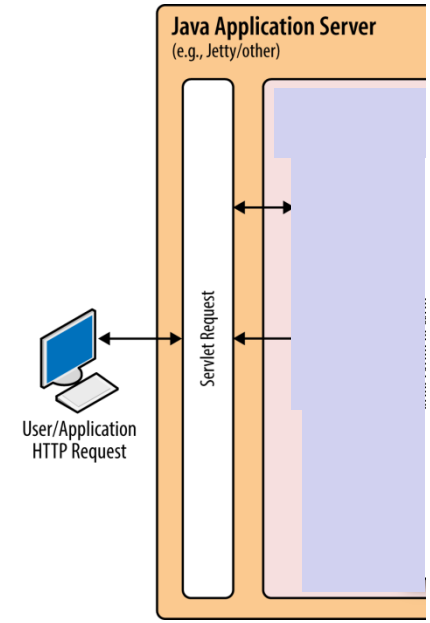




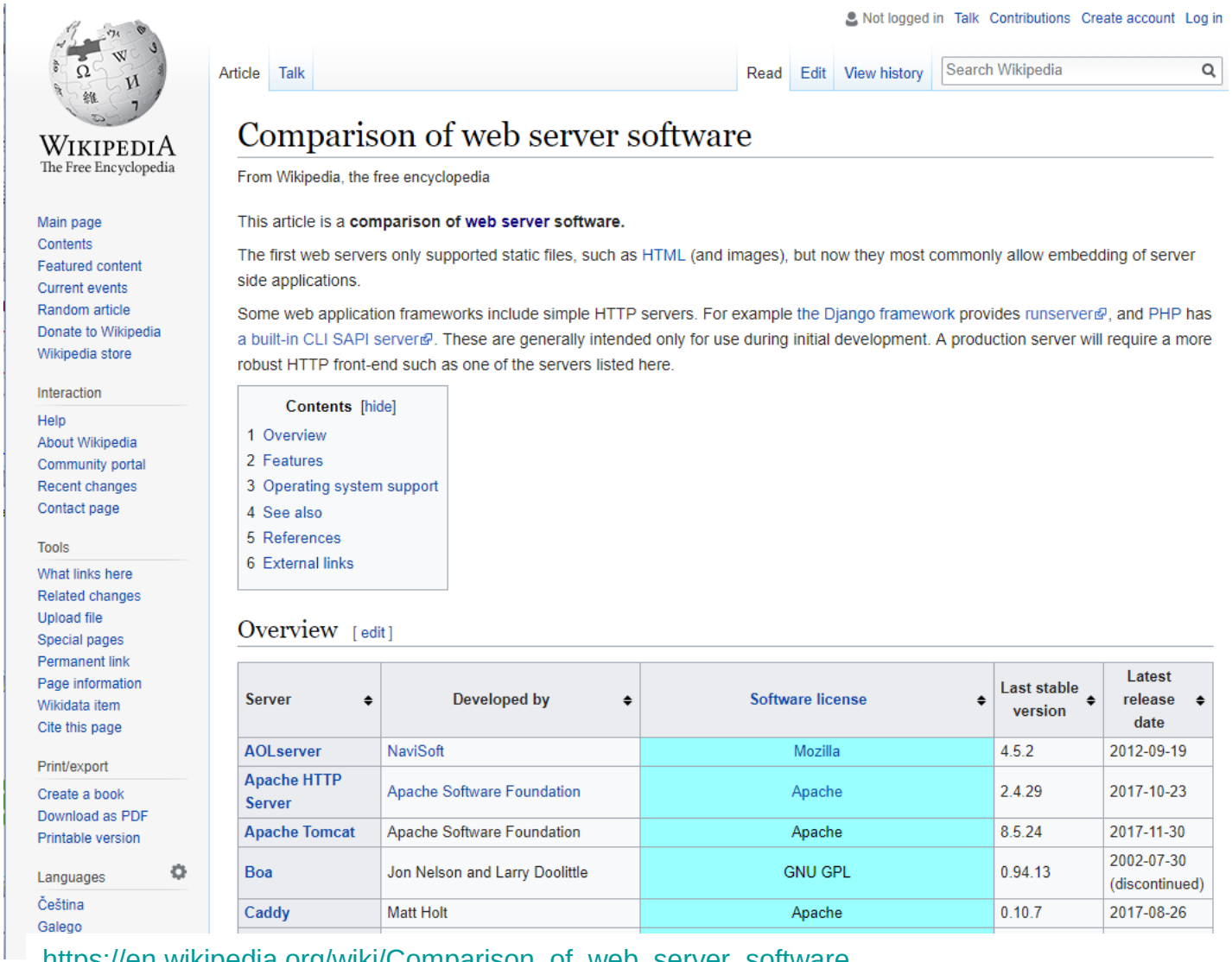
https://docstore.mik.ua/orelly/java-ent/jenut/ch05_01.htm
https://www.tutorialspoint.com/servlets/servlets-life-cycle.htm
https://beginnersbook.com/2013/05/working-of-servlets/

servers

- web servers
  - **Full** and/or **embedded**
  - Support Servlets
  - Popular
    - Jetty
    - Tomcat
    - …
- Java EE Application servers
  - Include web server



Runtimes (or container providers)



Not logged in Talk Contributions Create account Log in

Article Talk | Read Edit View history | Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Čeština
Galego

Comparison of web server software

From Wikipedia, the free encyclopedia

This article is a **comparison of web server software.**

The first web servers only supported static files, such as HTML (and images), but now they most commonly allow embedding of server side applications.

Some web application frameworks include simple HTTP servers. For example the Django framework provides runserver, and PHP has a built-in CLI SAPI server. These are generally intended only for use during initial development. A production server will require a more robust HTTP front-end such as one of the servers listed here.

Contents [hide]
1 Overview
2 Features
3 Operating system support
4 See also
5 References
6 External links

Overview [edit]

Server	Developed by	Software license	Last stable version	Latest release date
AOLserver	NaviSoft	Mozilla	4.5.2	2012-09-19
Apache HTTP Server	Apache Software Foundation	Apache	2.4.29	2017-10-23
Apache Tomcat	Apache Software Foundation	Apache	8.5.24	2017-11-30
Boa	Jon Nelson and Larry Doolittle	GNU GPL	0.94.13	2002-07-30 (discontinued)
Caddy	Matt Holt	Apache	0.10.7	2017-08-26

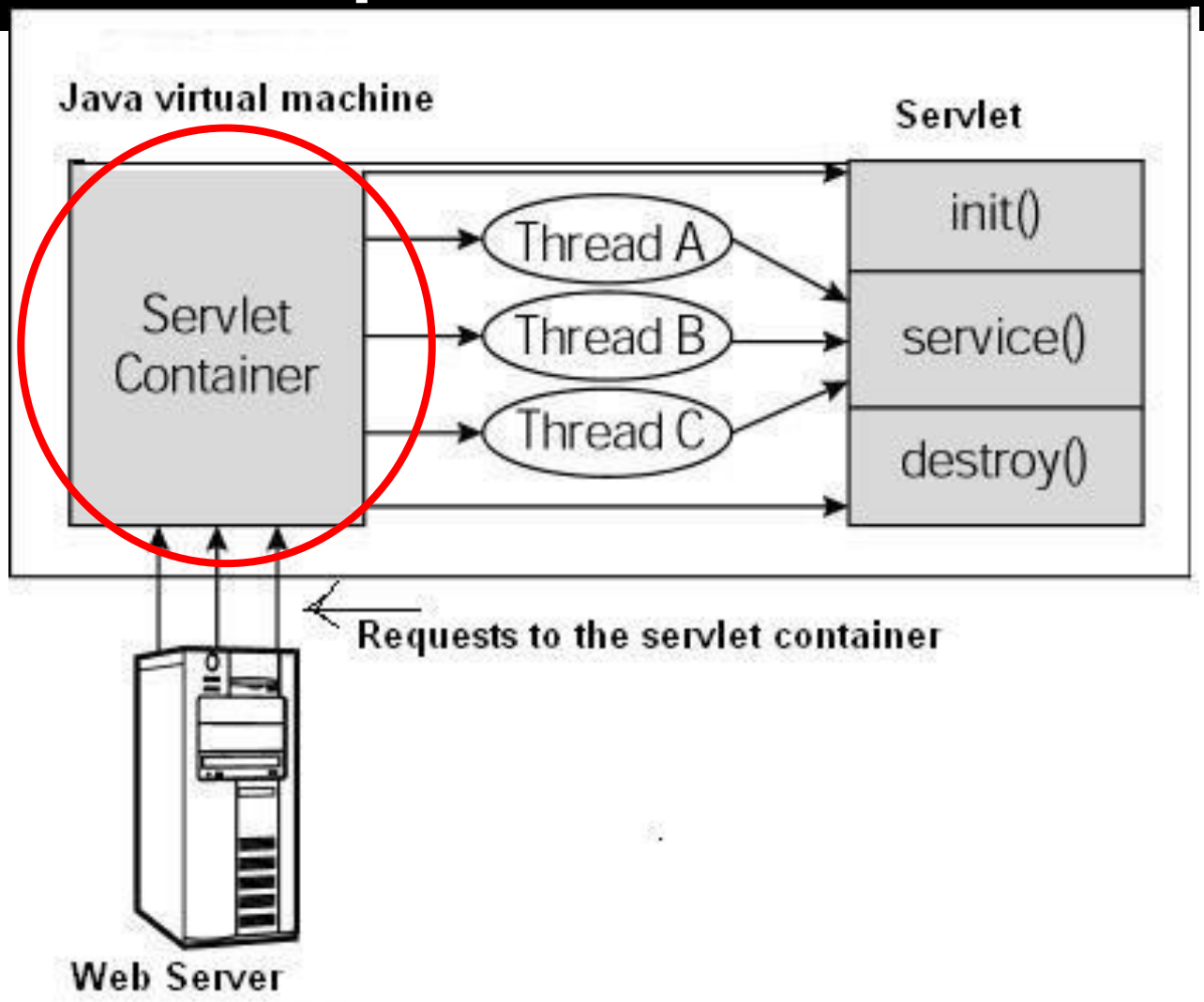
https://en.wikipedia.org/wiki/Comparison_of_web_server_software

Runtimes (or container providers)

Product	Vendor	Edition	Last release	Java EE compatibility [4]	Servlet	JSP	HTTP/2	License
ColdFusion	Adobe Systems	2016.0.1	2016-05-01	7 partial platform	3.1	2.3	No	Proprietary, commercial
Enhydra	Lutris	5.1.9	2005-03-23	No			No	Free, GPL
Enterprise Server	Borland	6.7	2007-01	1.4	2.4	2.0	No	Proprietary, commercial
Geronimo	ASF	3.0.1	2013-05-28	6 full platform	3.0	2.2	No	Free, Apache
GlassFish	GlassFish Community	5.0.0	2017-09-21	8 full platform	3.1	2.3	No	Free, CDDL, GPL + classpath exception
iPlanet Web Server	Oracle Corporation	7.0.21	2015-04	Yes[5]	2.5	2.1	No	Proprietary, commercial
JBoss Enterprise Application Platform	Red Hat	7.0.0	2016-05	7 full platform	3.1	2.3	Yes	Free, LGPL
Jetty	Eclipse Foundation	9.3.3	2015-08-27	7 partial platform[6]	3.1	2.3	Yes	Free, Apache 2.0, EPL
JEUS	TmaxSoft	8	2013-08	7 full platform	3.0	2.2	No	Proprietary, commercial
JOnAS	OW2 Consortium (formerly ObjectWeb)	5.3	2013-10-04	6 Web Profile	3.0	2.2	No	Free, LGPL
JRun	Adobe Systems	4 updater 7	2007-11-06	1.3	3.1	2.3	No	Proprietary, commercial
Lucee (Formerly Railo)	Lucee Association Switzerland	5.2.5.25	2017-12-22	7 partial platform	3.1	2.3	No	Free, CDDL, GPL + classpath exception
NetWeaver Application Server	SAP AG	7.4	2013-01-11	5	2.5	2.1	No	Proprietary, commercial
Nuno Rafael Server	Nuno Rafael Kst	45.2	2018-01-01	7 full platform	2.3		Yes	Free, CRF NR
Oracle Containers for J2EE	Oracle Corporation	10.1.3.5.0	2009-08	1.4	2.4	2.0	No	Proprietary, commercial
Orion Application Server	IronFlare	2.0.7	2006-03-09	1.3	2.3	1.2	No	Proprietary, commercial
Payara	Payara	4.1.2.172	2017-05-22	7 full platform	3.1	2.3	No	Free, CDDL, GPL + classpath exception
Resin Servlet Container (open source)	Caucho Technology	4.0.36	2013-04-25	6 Web Profile[7]	3.0	2.2	No	Free, GPL
Resin Professional Application Server	Caucho Technology	4.0.36	2013-04-25	6 Web Profile	3.0	2.2	No	Proprietary, commercial
Rupy	Rupy	1.2	2015-01-01	No			No	Free, LGPL
Tomcat	ASF	8.5.9	2016-12-08	7 partial platform	3.1	2.3	Yes	Free, Apache v2
TomEE	ASF	1.7.4	2016-03	6 Web Profile	3.0	2.2	No	Free, Apache
WebLogic Server	Oracle Corporation (formerly BEA Systems)	12.2.1.1	2016-06-21 [8]	7 full platform	3.1	2.3	No	Proprietary, commercial
WebObjects	Apple Inc.	5.4.3	2008-09-15	Partial[9]			No	Proprietary, commercial
IBM WebSphere Application Server	IBM	9.0	2016-06-24	6 & 7 full platform	3.1	2.3	No	Proprietary, commercial
WebSphere AS Community Edition	IBM	3.0.0.4	2013-06-21	6 full platform	3.0	2.2	No	Proprietary, commercial
WildFly (formerly JBoss AS)	Red Hat (formerly JBoss)	11.0	2017-11-23	7 full platform	3.1	2.3	Yes	Free, LGPL

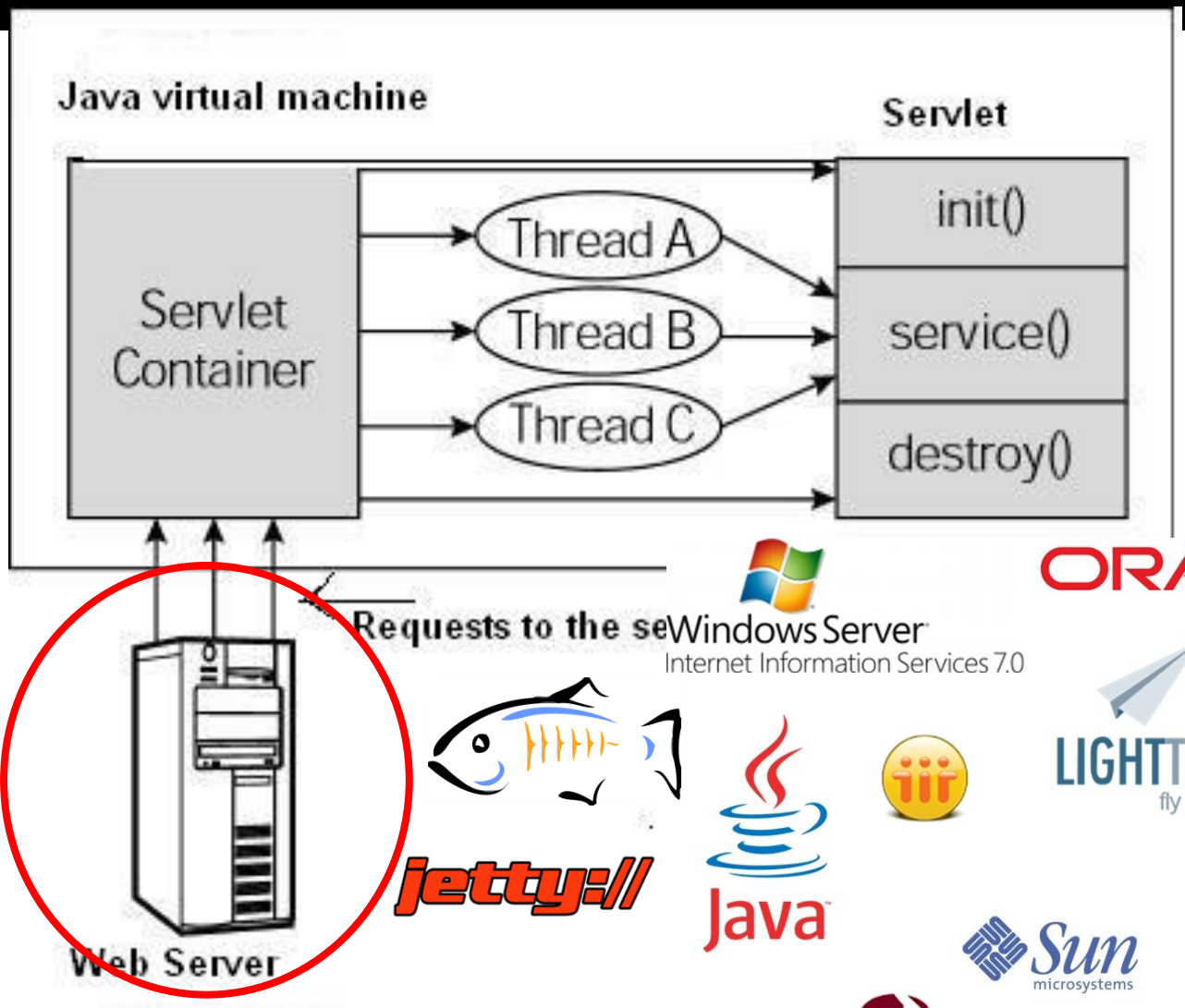
https://en.wikipedia.org/wiki/List_of_application_servers

“web server” provide container



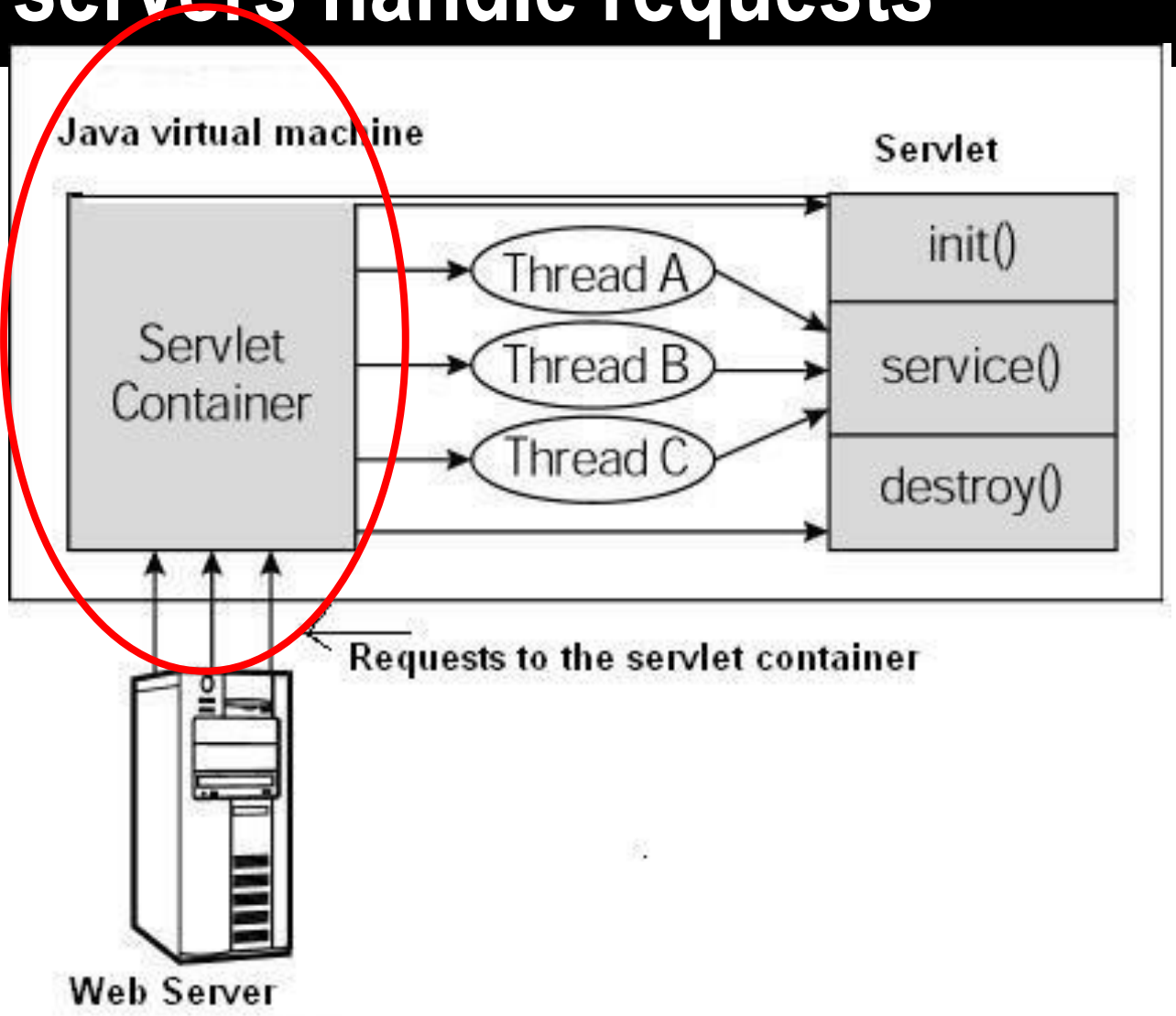
http://www.tutorialspoint.com/servlets/servlets-life-cycle.htm

Http server or “web server”



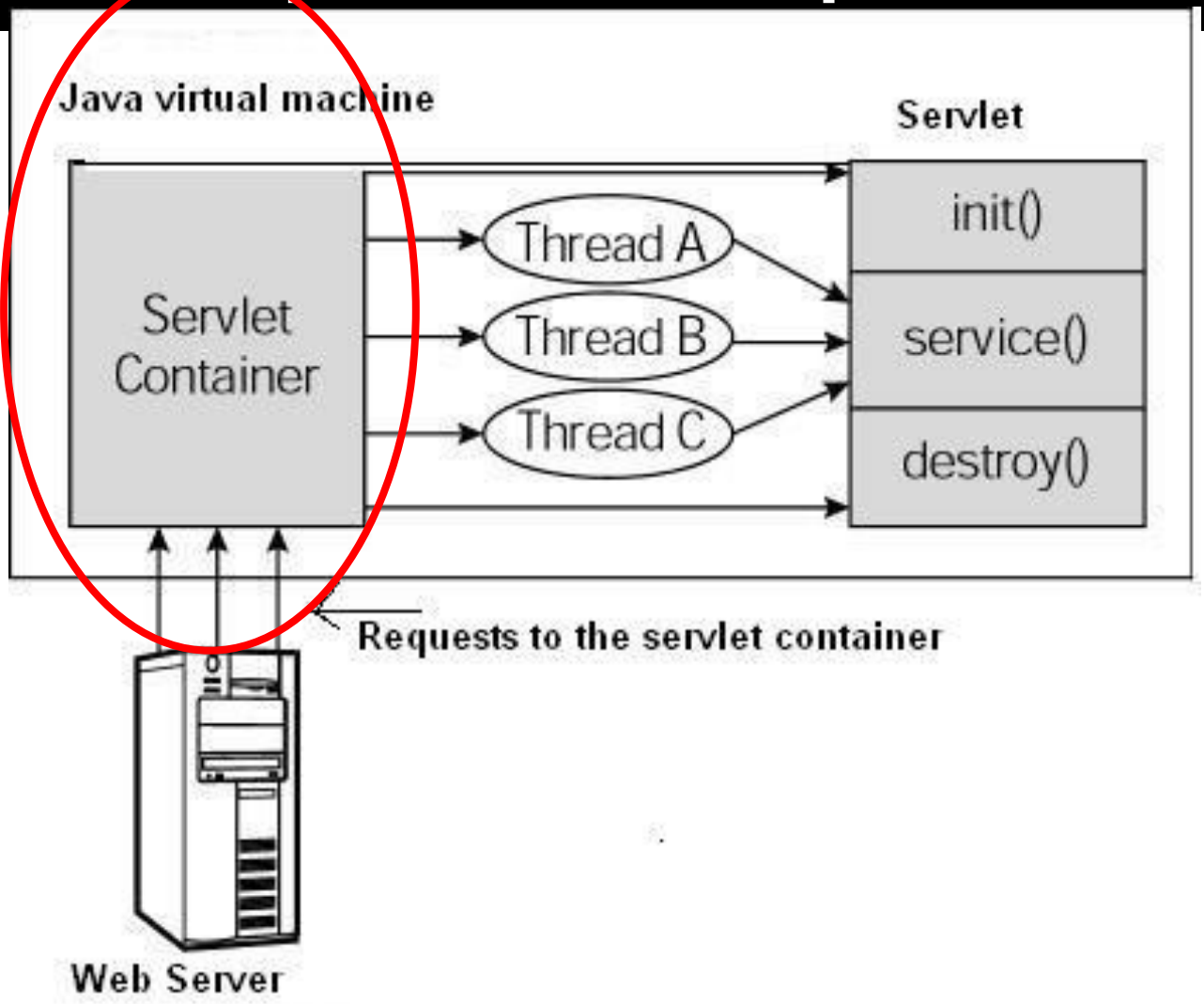
http://www.tutorialspoint.com/servlets/servlets-life-cycle.htm

Web servers handle requests



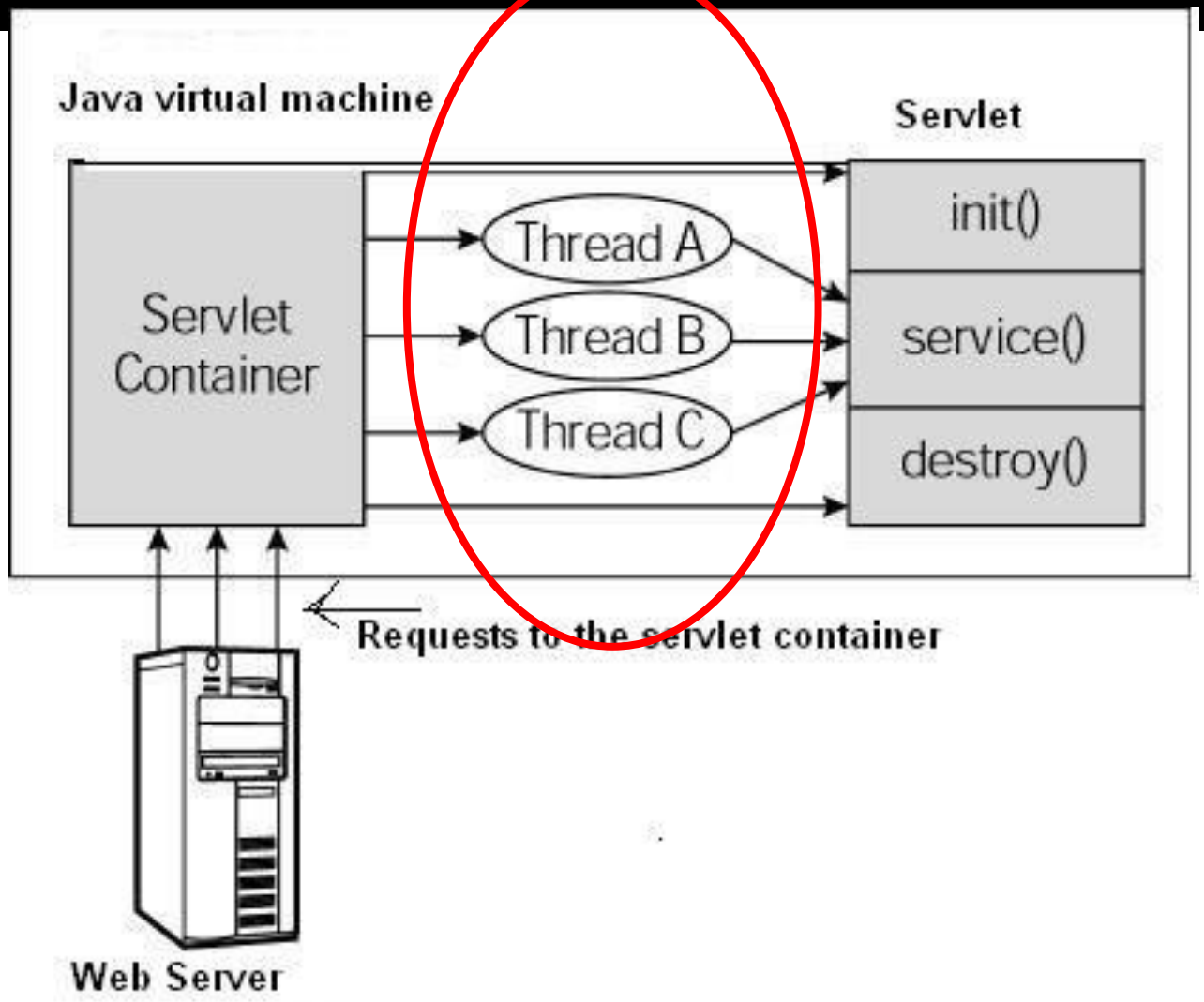
http://www.tutorialspoint.com/servlets/servlets-life-cycle.htm

Web server processes requests



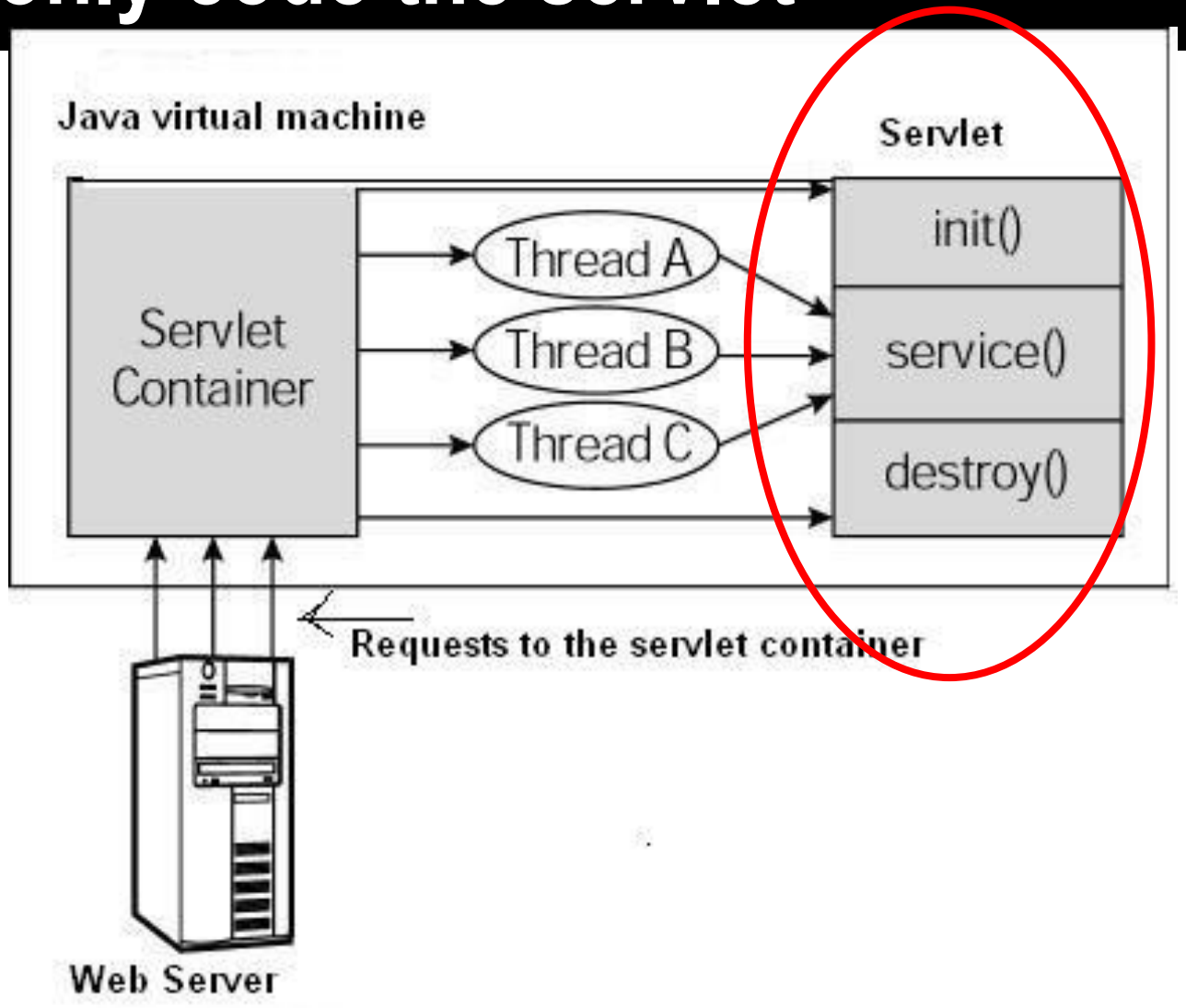
http://www.tutorialspoint.com/servlets/servlets-life-cycle.htm

You do not control the threads



http://www.tutorialspoint.com/servlets/servlets-life-cycle.htm

You only code the servlet



http://www.tutorialspoint.com/servlets/servlets-life-cycle.htm

Embedded server

- “Embedded systems have traditionally been isolated, self-contained systems”
  - Usually = software/ in code
- Web server
  - Handling HTTP
- Many flavours
  - Jetty, tomcat, micro payara, ….
- Used
  - **Spring boot, Microprofiles**

Embedded server: raw by default

```
public class EmbeddedJettyExample {

        public static void main(String[] args) throws Exception {
              Server server = new Server(8680);
              try {
        }
}
```

```
public class EmbeddedJettyExample {

        public static void main(String[] args) throws Exception {
              Server server = new Server(8680);
              try {
                 server.start();
                 server.dumpStdErr();
                 server.join();
              } catch (Exception e) {
                        e.printSta
              }
        }

        public static class HelloH

                public HelloHandle
                        this("Hell
                }

                public HelloHandle
                        this(arg,
                }

                public HelloHandle
                        this.greet
                        this.bodym
```

```
public class EmbeddedJettyExample {

        public static void main(String[] args) throws Exception {

                Server server = new Server(8680);

                ServletHandler servletHandler = new ServletHandler();
                server.setHandler(servletHandler);

                servletHandler.addServletWithMapping(HelloServlet.class, "/");

                server.start();
                server.join();

        }

        public static class HelloServlet extends HttpServlet
        {
                protected void doGet(HttpServletRequest request, HttpServletRe
                {
                        response.setContentType("text/html");
                        response.setStatus(HttpServletResponse.SC_OK);
                        response.getWriter().println("<h1>New Hello Simple Ser
```

https://exam

Why embedded??

Jetty (web server)

From Wikipedia, the free encyclopedia

Eclipse Jetty is a Java HTTP (Web) server and Java Servlet container. While Web Servers are usually associated with serving documents to people, Jetty is now often used for machine to machine communications, usually within larger software frameworks. Jetty is developed as a free and open source project as part of the Eclipse Foundation. The web server is used in products such as Apache ActiveMQ,[2] Alfresco,[3] Scalatra, Apache Geronimo,[4] Apache Maven, Apache Spark, Google App Engine,[5] Eclipse,[6] FUSE,[7] iDempiere,[8] Twitter's Streaming API[9] and Zimbra.[10] Jetty is also the server in open source projects such as Lift, Eucalyptus, Red5, Hadoop and I2P.[11] Jetty supports the latest Java Servlet API (with JSP support) as well as protocols HTTP/2 and WebSocket.

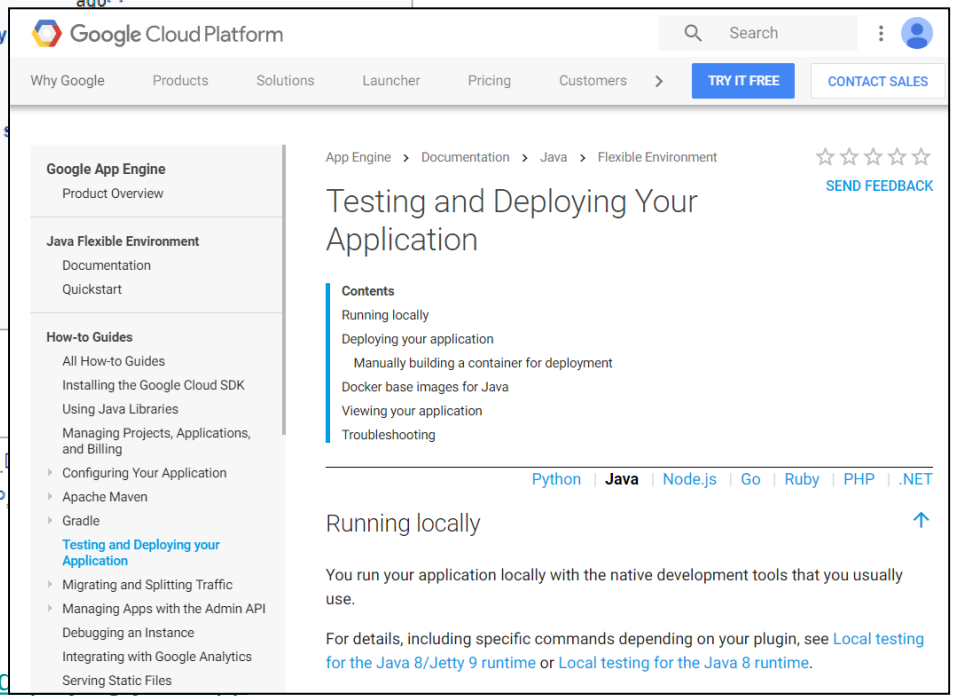
Contents [hide]
1 Overview
2 History
3 See also
4 References
5 External links

Overview [edit]

Jetty started as an independent open source project in 1995. In 2009 Jetty moved to Eclipse.[
embedded Java application and it is already a component of the Eclipse IDE. It supports AJP
other Java technologies.[5]

Jetty

Developer(s)	Eclipse Foundation
Stable release	9.4.7 / 14 September 2017; 5 months ago[1]
Repository	
Written in	
Operating s	
Type	
License	
Website	



Testing and Deploying Your Application
https://cloud.google.com/appengine/docs/flexible/java/testing-and-d
Tomcat vs. Jetty vs. Undertow: Comparison of Spring Boot Embedded Servlet Containers
https://examples.javacodegeeks.com/enterprise-java/spring/tomcat-vs-jetty-vs-undertow-comparison-of-spring-boot-embedded-servlet-containers/

HOME

Embedded containers and standalone Java applications

25 AUGUST 2015

When you create a Java application, you either choose to deploy within an external servlet container/application container or embed a container into your jar. There are developers who still refuse to use embedded containers, some for fear: feeling that somehow their application is going to crash just because it runs directly from a java -jar command (haha), others, well, I don't know why someone could not prefer to run their applications using a simple terminal command, but, there are other opinions and other views, this post is not to criticise any of it. :)

I'm going to list some containers and tools that help to develop standalone applications. I'm not going to teach you how to use each one of them, but I will provide some links to help you.

Containers

JETTY

Probably you already have heard of it. Jetty is one of the most used servlet

Embedded containers and standalone Java applications
http://www.thedevpiece.com/embedded-containers-and-standalone-java-applications/

slogan that says: "Don't deploy your application in Jetty, deploy Jetty in

Configuration: web.xml & annotations

Java web application

- Application dependent on web server
  - HTTP based provider
- Natural placeholders
  - servlets
  - Webservices and REST
  - JSF
- Java Web Application Technologies
  - http://tutorials.jenkov.com/java-web-apps/index.html

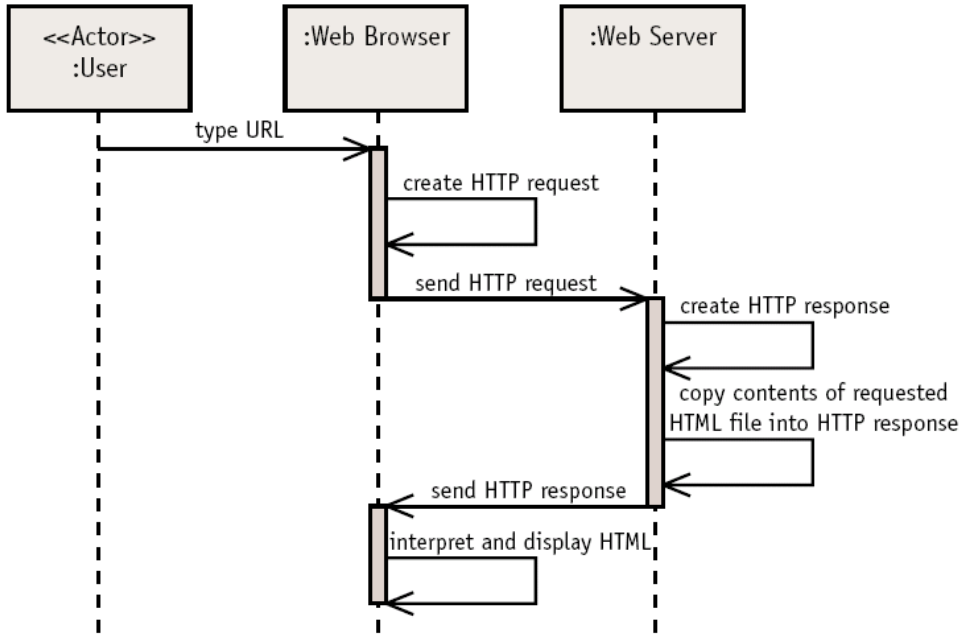
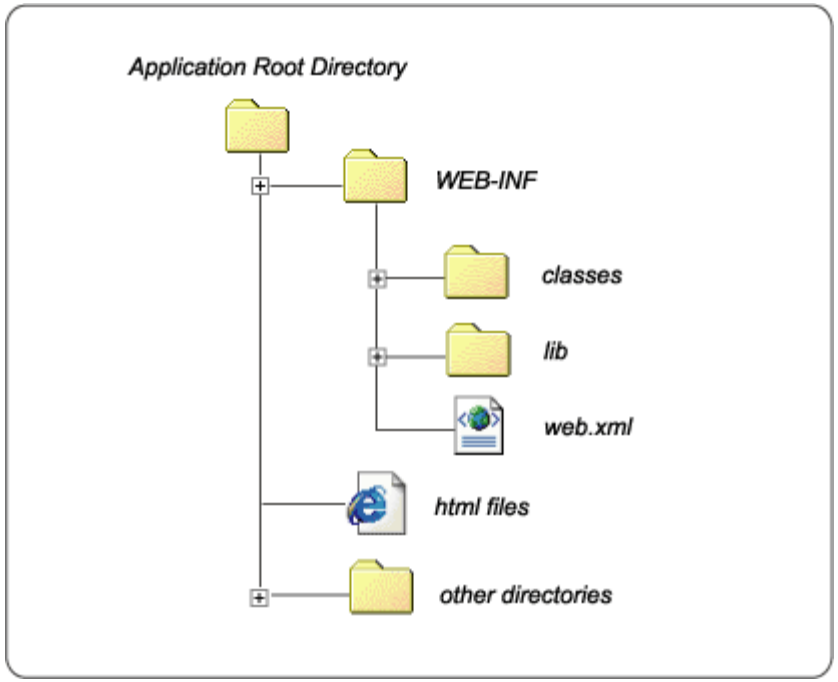


Figure 27-2 *UML Sequence diagram, HTTP protocol*

Java web application

- Uniform Structure
  - Maven template, other ….
- Specific files
  - Web.xml
  - Configurations
  - Some specific to solutions
- Some links
  - Getting Started with Web Applications – Java EE tutorial
    - http://goo.gl/jPhmdT
  - Java Web Application Technologies
    - http://goo.gl/DLXl5u
  - Deployment Descriptor: web.xml
    - http://goo.gl/ofL7I5



The web.xml

- Why?
  - Web components, such as servlets, filters, listeners, and tag handlers.
- With Java EE metadata annotations,
  - web.xml deployment descriptor is now optional
- But it can be a safe way to ..
  - Ensure compatibility among different servers

Generic Servlet

servlet

```
import java.io.*;
import javax.servlet.*;

public class ExampleGeneric extends GenericServlet{
 public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException{
   res.setContentType("text/html");
   PrintWriter pwriter=res.getWriter();
   pwriter.print("<html>");
   pwriter.print("<body>");
   pwriter.print("<h2>Generic Servlet Example</h2>");
   pwriter.print("<p>Hello BeginnersBook Readers!</p>");
   pwriter.print("</body>");
   pwriter.print("</html>");
 }
}
```

Web.xml

```
<web-app>
<display-name>BeginnersBookServlet</display-name>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
<welcome-file>index.jsp</welcome-file>
<welcome-file>default.html</welcome-file>
<welcome-file>default.htm</welcome-file>
<welcome-file>default.jsp</welcome-file>
</welcome-file-list>

<servlet>
<servlet-name>MyGenericServlet</servlet-name>
<servlet-class>ExampleGeneric</servlet-class>
</servlet>


<servlet-mapping>
<servlet-name>MyGenericServlet</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

Generic Servlet

servlet

```
import java.io.*;
import javax.servlet.*;

public class ExampleGeneric extends GenericServlet{
 public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException{
   res.setContentType("text/html");
   PrintWriter pwriter=res.getWriter();
   pwriter.print("<html>");
   pwriter.print("<body>");
   pwriter.print("<h2>Generic Servlet Example</h2>");
   pwriter.print
   pwriter.print
   pwriter.print("</html>");
 }
}
```

The implementation class

Web.xml

```
<web-app>
<display-name>BeginnersBookServlet</display-name>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
<welcome-file>index.jsp</welcome-file>
<welcome-file>default.html</wel
<welcome-file>default.htm</welc
<welcome-file>default.jsp</welcome-file>
</welcome-file-list>

<servlet>
<servlet-name>MyGenericServlet</servlet-name>
<servlet-class>ExampleGeneric</servlet-class>
</servlet>


<servlet-mapping>
<servlet-name>MyGenericServlet</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

The alias for lookup

The URL to handle

HttpServlet

Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Creating Http Servlet by Extending HttpServlet class
public class ExampleHttpServlet extends HttpServlet
{
    private String mymsg;
    public void init() throws ServletException
    {
       mymsg = "Http Servlet Demo";
    }
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException
    {
        // Setting up the content type of web page
        response.setContentType("text/html");
        // Writing the message on the web page
        PrintWriter out = response.getWriter();
        out.println("<h1>" + mymsg + "</h1>");
        out.println("<p>" + "Hello Friends!" + "</p>");
    }
    public void destroy()
    {
       // Leaving empty. Use this if you want to perform
       //something at the end of Servlet life cycle.
    }
}
```

sbook.c

Web.xml

```
<web-app>
<display-name>BeginnersBookServlet</display-name>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
<welcome-file>index.jsp</welcome-file>
<welcome-file>default.html</welcome-file>
<welcome-file>default.htm</welcome-file>
<welcome-file>default.jsp</welcome-file>
</welcome-file-list>

<servlet>
<servlet-name>MyHttpServlet</servlet-name>
<servlet-class>ExampleHttpServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>MyHttpServlet</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

Java: Declare Servlets

The code

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Creating Http Servlet by Extending HttpServlet class
public class ExampleHttpServlet extends HttpServlet
{
    private String mymsg;
    public void init() throws ServletException
    {
       mymsg = "Http Servlet Demo";
    }
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException
    {
        // Setting up the content type of web page
        response.setContentType("text/html");
        // Writing the message on the web page
        PrintWriter out = response.getWriter();
        out.println("<h1>" + mymsg + "</h1>");
        out.println("<p>" + "Hello Friends!" + "</p>");
    }
    public void destroy()
    {
       // Leaving empty. Use this if you want to perform
       //something at the end of Servlet life cycle.
    }
}
```

Web.xml

```xml
<web-app>
<display-name>BeginnersBookServlet</display-name>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
<welcome-file>index.jsp</welcome-file>
<welcome-file>default.html</welcome-file>
<welcome-file>default.htm</welcome-file>
<welcome-file>default.jsp</welcome-file>
</welcome-file-list>

<servlet>
<servlet-name>MyHttpServlet</servlet-name>
<servlet-class>ExampleHttpServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>MyHttpServlet</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

Annotations: skip the web.xml

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

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
        version="3.1">

        <display-name>Servlet with Annotations Application</display-name>

        <servlet>
                <servlet-name>simpleServlet</servlet-name>
                <servlet-class>net.javatutorial.tutorials.Servl
                <load-on-startup>1</load-on-startup>
        </servlet>

        <servlet-mapping>
                <servlet-name>simpleServlet</servlet-name>
                <url-pattern>/hello</url-pattern>
        </servlet-mapping>

</web-app>
```

```java
package net.javatutorial.tutorials;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "simpleServlet", urlPatterns = { "/hello" }, loadOnStartup = 1
public class ServletWithAnnotations extends HttpServlet {

        private static final long serialVersionUID = -3462096228274971485L;

        @Override
        protected void doGet(HttpServletRequest reqest, HttpServletResponse respo
                        throws ServletException, IOException {
                response.getWriter().println("Hello World!");
        }

}
```

Servlet Annotation Example
https://javatutorial.net/servlet-annotation-example

Runtime needs to support Servlet 3.0 API

Annotations vs web.xml

- Using annotations you can avoid using web.xml
- God practice to still use web.xml
  - Used for other servlets configurations on your web application.
    - servlets handling REST requests with Jersey,
    - Servlets handling Java Server Faces requests.
    - …
- **by default the mappings in web.xml override the mappings defined via the @WebServlet**
  - **if using both make sure they agree**

Annotations vs web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   version="2.5">
   <servlet>
      <servlet-name>Faces Servlet</servlet-name>
      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>/faces/*</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
      <welcome-file>faces/index.xhtml</welcome-file>
   </welcome-file-list>
   <context-param>
      <param-name>javax.faces.PROJECT_STAGE</param-name>
      <param-value>Development</param-value>
   </context-param>
</web-app>
```

Annotations vs web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>com.vogella.jersey.first</display-name>
 <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
     <!-- Register resources and providers under com.vogella.jersey.first package. -->
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>com.vogella.jersey.first</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

LEARN JAVA SERVLETS
web application framework

Servlets Video Tutorials

Servlets Tutorial

- Servlets - Home
- Servlets - Overview
- Servlets - Environment Setup
- Servlets - Life Cycle
- Servlets - Examples
- Servlets - Form Data
- Servlets - Client Request
- Servlets - Server Response
- Servlets - Http Codes
- Servlets - Writing Filters
- Servlets - Exceptions
- Servlets - Cookies Handling
- Ser
- Servlets - Database Access

Servlets - Annotations

Previous Page Next Page

So far, you have learnt how Servlet uses the deployment descriptor (web.xml file) for deploying your application into a web server. Servlet API 3.0 has introduced a new package called javax.servlet.annotation. It provides annotation types which can be used for annotating a servlet class. If you use annotation, then the deployment descriptor (web.xml) is not required. But you should use tomcat7 or any later version of tomcat.

Annotations can replace equivalent XML configuration in the web deployment descriptor file (web.xml) such as servlet declaration and servlet mapping. Servlet containers will process the annotated classes at deployment time.

The annotation types introduced in Servlet 3.0 are –

Sr.No.	Annotation & Description
1	**@WebServlet** To declare a servlet.
2	**@WebInitParam** To specify an initialization parameter.
3	**@WebFilter**
4	**@WebListener**

Servlets 3.0 (with CDI)

- Basic support for HTTP handling in Java

```java
@WebServlet(
        name = "EmployeeServlet",
        urlPatterns = {"/employee"}
)
public class EmployeeServlet extends HttpServlet {
```

```java
public class Main {

    public static final Optional<String> port = Optional.ofNullable(System.getenv("PORT"));

    public static void main(String[] args) throws Exception {
        String contextPath = "/";
        String appBase = ".";
        Tomcat tomcat = new Tomcat();
        tomcat.setPort(Integer.valueOf(port.orElse("8080") ));
        tomcat.getHost().setAppBase(appBase);
        tomcat.addWebapp(contextPath, appBase);
        tomcat.start();
        tomcat.getServer().await();
    }
}
```

https://dzone.com/articles/an-overview-servlet-30

Java SE 8: Creating a Web App with Bootstrap and Tomcat Embedded

http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/basic_app_embedded_tomcat/basic_app-tomcat-embedded.html

Servlet 3.x and Annotations

How do I define a servlet with @WebServlet annotation?

By Wayan in Servlet

Last modified: July 24, 2017 0 Comment

Annotations is one new feature introduces in the Servlet 3.0 Specification. Previously to declare servlets, listeners or filters we must do it in the `web.xml` file. Now, with the new annotations feature we can just annotate servlet classes using the `@WebServlet` annotation.

```
package org.kodejava.example.servlet;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(
```

How do I define a servlet with @WebServlet annotation?
https://kodejava.org/how-do-i-define-a-servlet-with-webservlet-annotation/

Servlet 3.x and Annotations

How do I define a servlet with @WebServlet annotation?

By Wayan in Servlet

Last modified: July 24, 2017 0 Comment

Annotations is one new feature introduces in the Servlet 3.0 Specification. Previously to declare servlets, listeners or filters we must
we can just annotate servlet classes

```
package org.kodejava.example.s

import javax.servlet.ServletEx
import javax.servlet.annotatio
import javax.servlet.annotatio
import javax.servlet.http.Http
import javax.servlet.http.Http
import javax.servlet.http.Http
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(
```

```
@WebServlet(
        name = "HelloAnnotationServlet",
        urlPatterns = {"/hello", "/helloanno"},
        asyncSupported = false,
        initParams = {
                @WebInitParam(name = "name", value = "admin"),
                @WebInitParam(name = "param1", value = "value1"),
                @WebInitParam(name = "param2", value = "value2")
        }
)
public class HelloAnnotationServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        out.write("<html><head><title>WebServlet Annotation</title></head>");
        out.write("<body>");
        out.write("<h1>Servlet Hello Annotation</h1>");
```

How do I define a servlet with @WebServlet
https://kodejava.org/how-do-i-define-a-servle

Servlet 3.x and Annotations

```java
@WebServlet(
        urlPatterns = "/myController",
        loadOnStartup = 1,
        asyncSupported = true
)
public class StartupServlet extends HttpServlet {

    public void init(ServletConfig config) {
        System.out.println("My servlet has been initialized"
    }

    // implement servlet doPost() and doGet()...
}
```

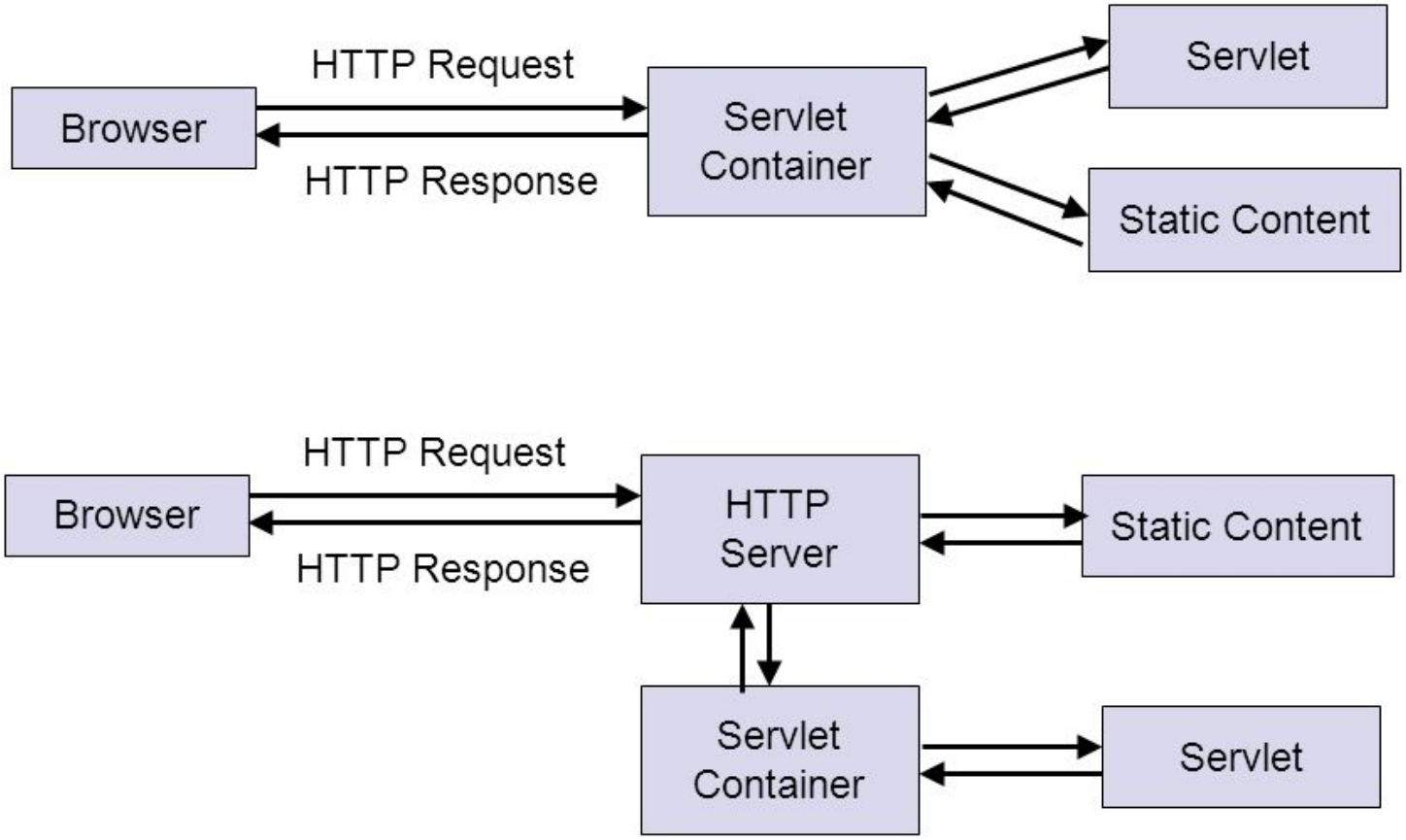
```java
@WebServlet(
        urlPatterns = "/imageUpload",
        initParams =
        {
            @WebInitParam(name = "saveDir", value = "D:/FileUpload"),
            @WebInitParam(name = "allowedTypes", value = "jpg,jpeg,gif,png")
        }
)
public class ImageUploadServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        String saveDir = getInitParameter("saveDir");
        String fileTypes = getInitParameter("allowedTypes");

        PrintWriter writer = response.getWriter();

        writer.println("saveDir = " + saveDir);
        writer.println("fileTypes = " + fileTypes);
    }
}
```

```java
@WebServlet(
    name = "AnnotatedServlet",
    description = "A sample annotated servlet",
    urlPatterns = {"/QuickServlet"}
)
public class QuickServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse
            throws IOException {

        PrintWriter writer = response.getWriter();
        writer.println("<html>Hello, I am a Java servlet!</html>");
        writer.flush();
    }

    public void doPost(HttpServletRequest request, HttpServletRespons
            throws IOException {
        String paramWidth = request.getParameter("width");
        int width = Integer.parseInt(paramWidth);

        String paramHeight = request.getParameter("height");
        int height = Integer.parseInt(paramHeight);

        long area = width * height;

        PrintWriter writer = response.getWriter();
        writer.println("<html>Area of the rectangle is: " + area + "<
        writer.flush();

    }
}
```

Quick start guide for Java servlet annotations
http://www.codejava.net/java-ee/servlet/quick-start-guide-for-java-servlet-annotations
WebServlet annotation examples
http://www.codejava.net/java-ee/servlet/webservlet-annotation-examples

Why this attention on servlets?

Java EE & Spring boot rely on servlets



Web = Servlet + ...

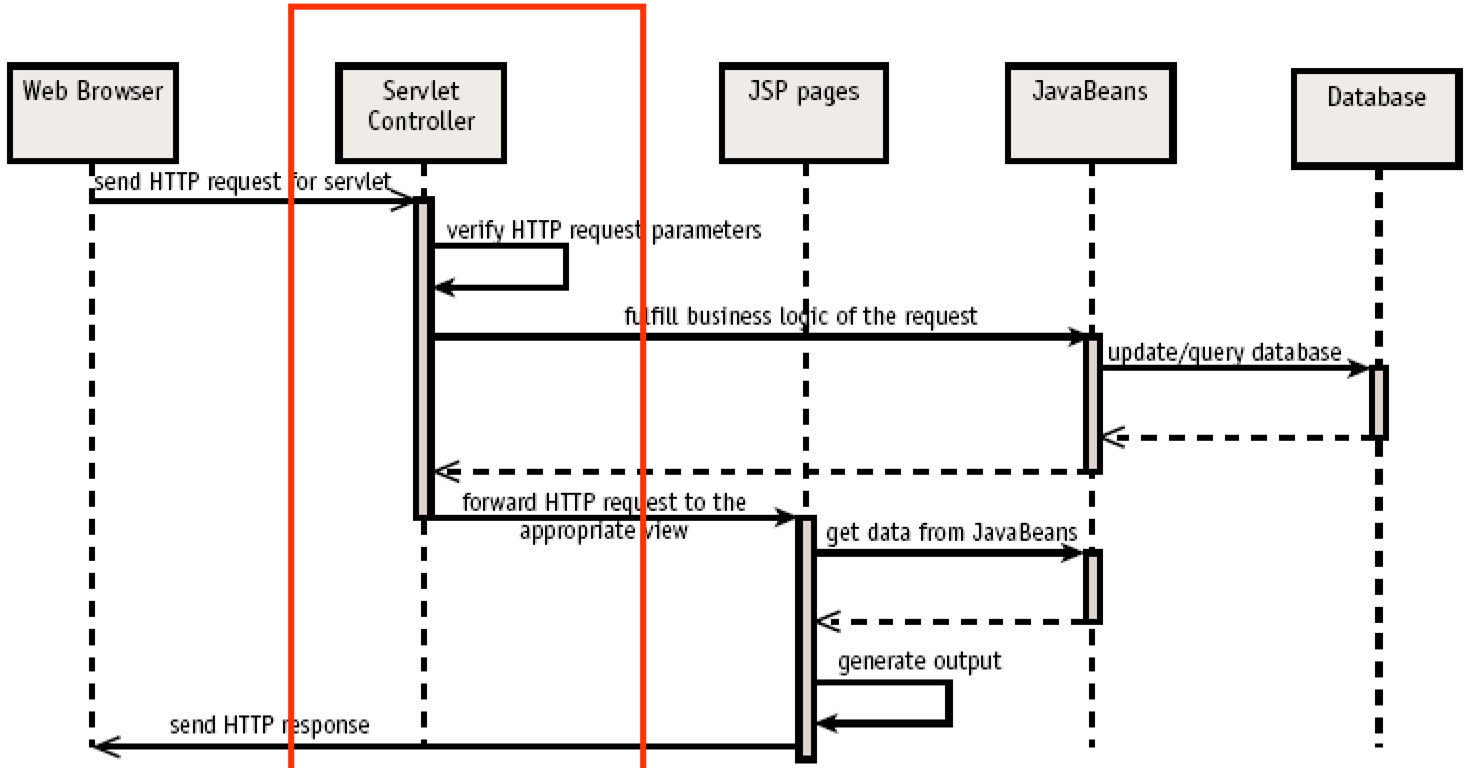


Figure 29-2 UML Sequence diagram, Model 2 Architecture

The Java (EE) instances

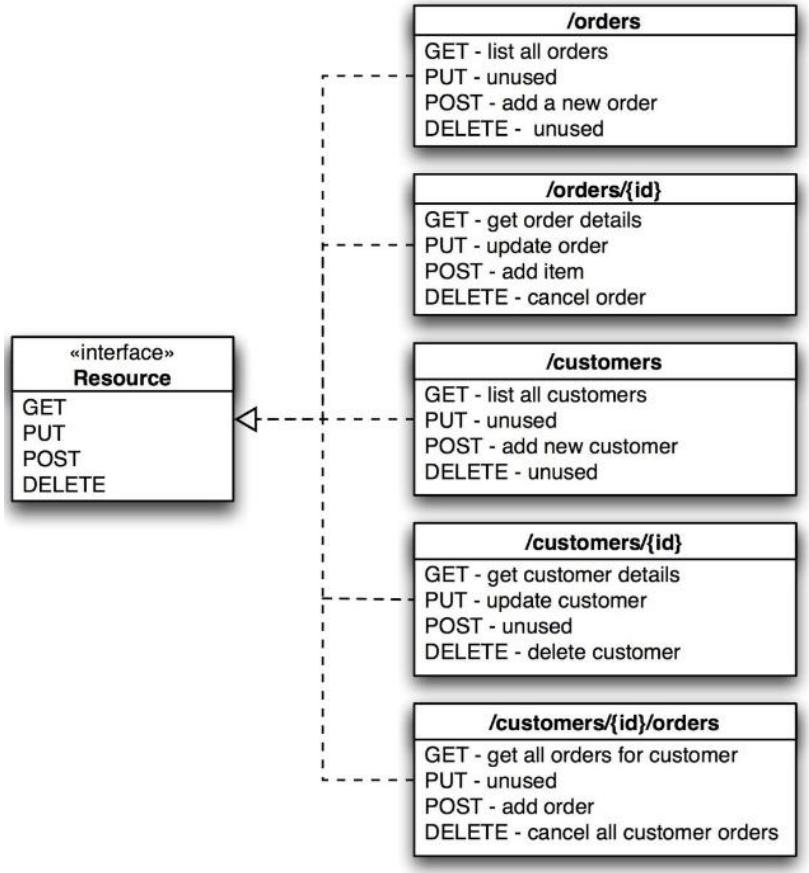
JAX-WS

- Supported on servlet / HTTP
- JavaSE
  - Endpoint, Client, provider
  - Wsimport
- JAXB mapping
  - XML<-> java
- Annotations
  - Closed Semantics
- multiplatform

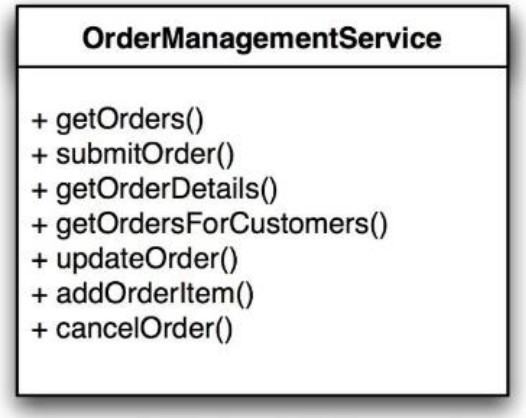
JAX-RS (RESTful)

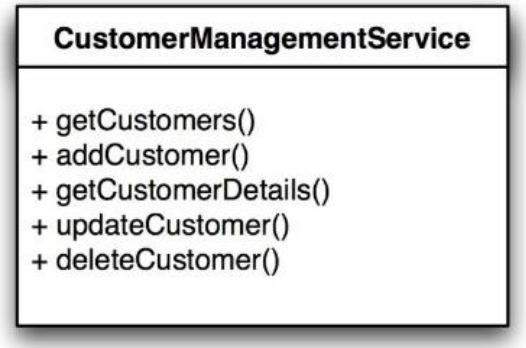
- **Supports RESTful API**
- Can implement REST
- Supported on servlet / HTTP
- JavaSE
  - Jersey
- No mapping
  - Depend on scope/programmer
- Annotations
  - Resource oriented
- multiplaform

JAX-RS & JAX-WS: JavaEE flavours



In JAX-WS
- deals with **WS protocol SOAP**
- translates (XML)
- calls your service implementation class





In JAX-RS
- Maps the **URI resource to your service**
- Translates (XML/JSON)
- Calls your service implementation class

JAX-RS & JAX-WS: rely on servlets

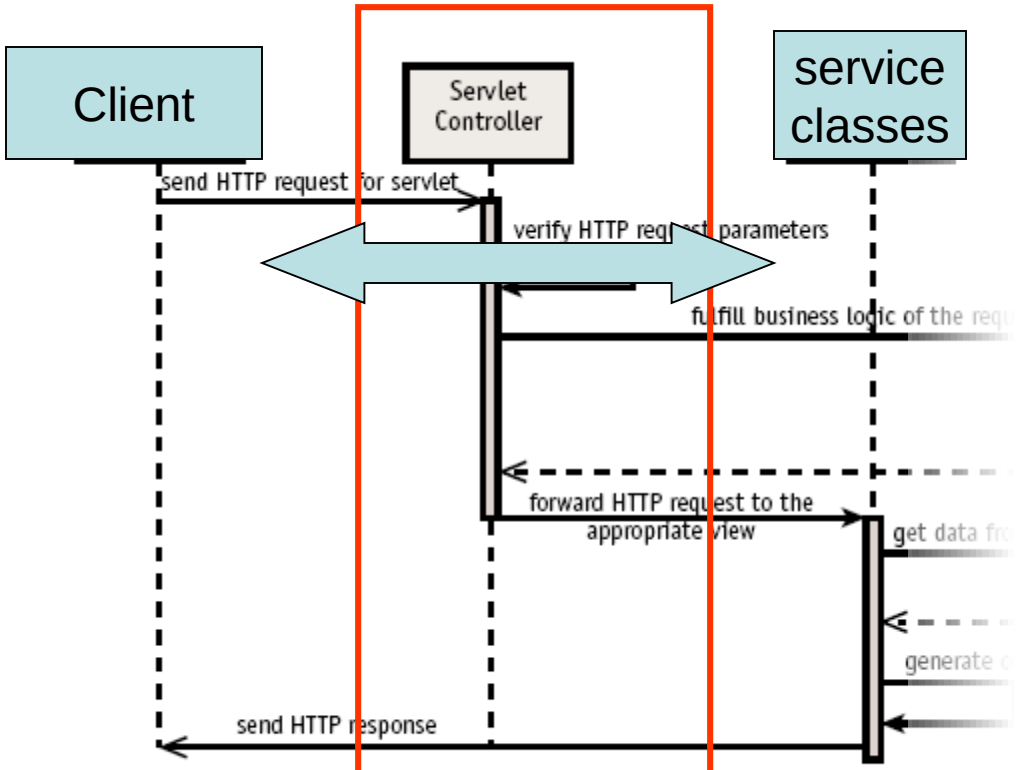
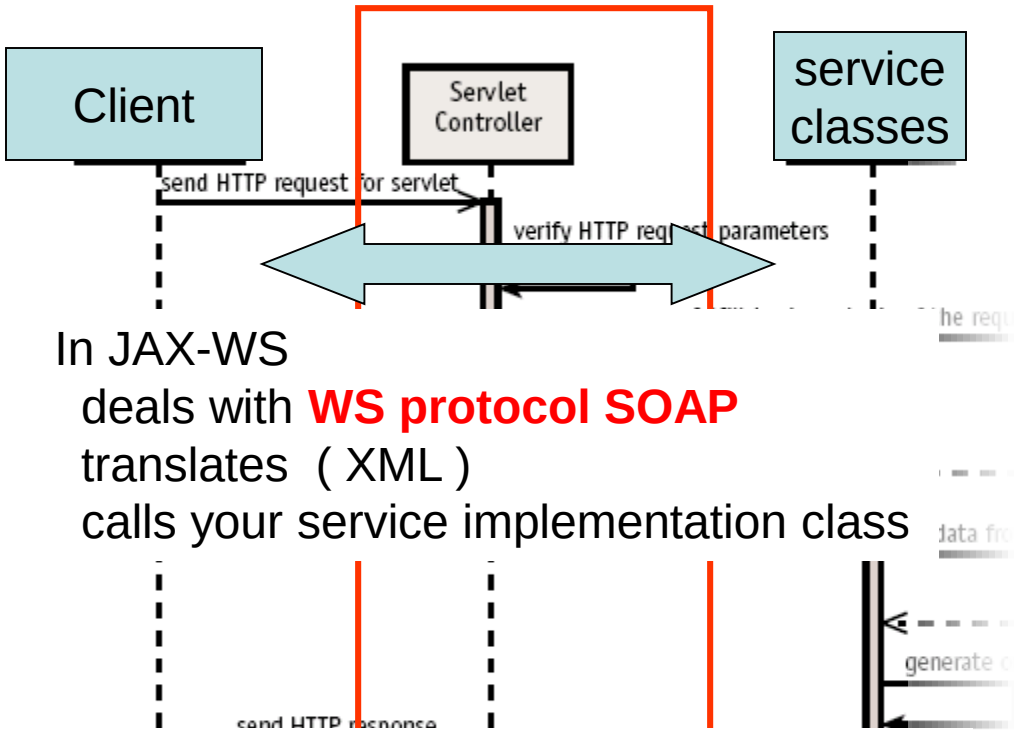


Figure 29-2 UML Sequence diagram, Model 2 Architecture

Examples from "UML Weekend Crash Course", Wiley, 2002

JAX-RS & JAX-WS: rely on servlets



In JAX-WS
- deals with **WS protocol SOAP**
- translates (XML)
- calls your service implementation class

In JAX-RS
- Maps the **URI resource to your service**
- Translates (XML/JSON)
- Calls your service implementation class

Architecture

Examples from “UML Weekend Crash Course”, Wiley, 2002

The java (EE) instances

JAX-WS

```
@WebService
public class BookCatalog {

    @WebMethod
    public List<String> getBooksCategory() {
        List<String> bookCategory = new ArrayList<>();
        bookCategory.add("Alpha");
        bookCategory.add("Bravo");
        bookCategory.add("Charlie");
        return bookCategory;
    }

}
```

JAX-RS (RESTful)

```
@Path("orders/{order_id}")
public class OrderResource {
    @GET
    @Path("customer")
    CustomerResource
    getCustomer(@PathParam("order_id") int id) {...}
}
```

```
@Path("/app")
public class SayHello {
    @GET
    @Produces(MediaType.TEXT_HTML)
    @Path("/hello")
    public String sayHello() {
        return "<h1> Hello Dude !!! </h1>";
    }
}
```

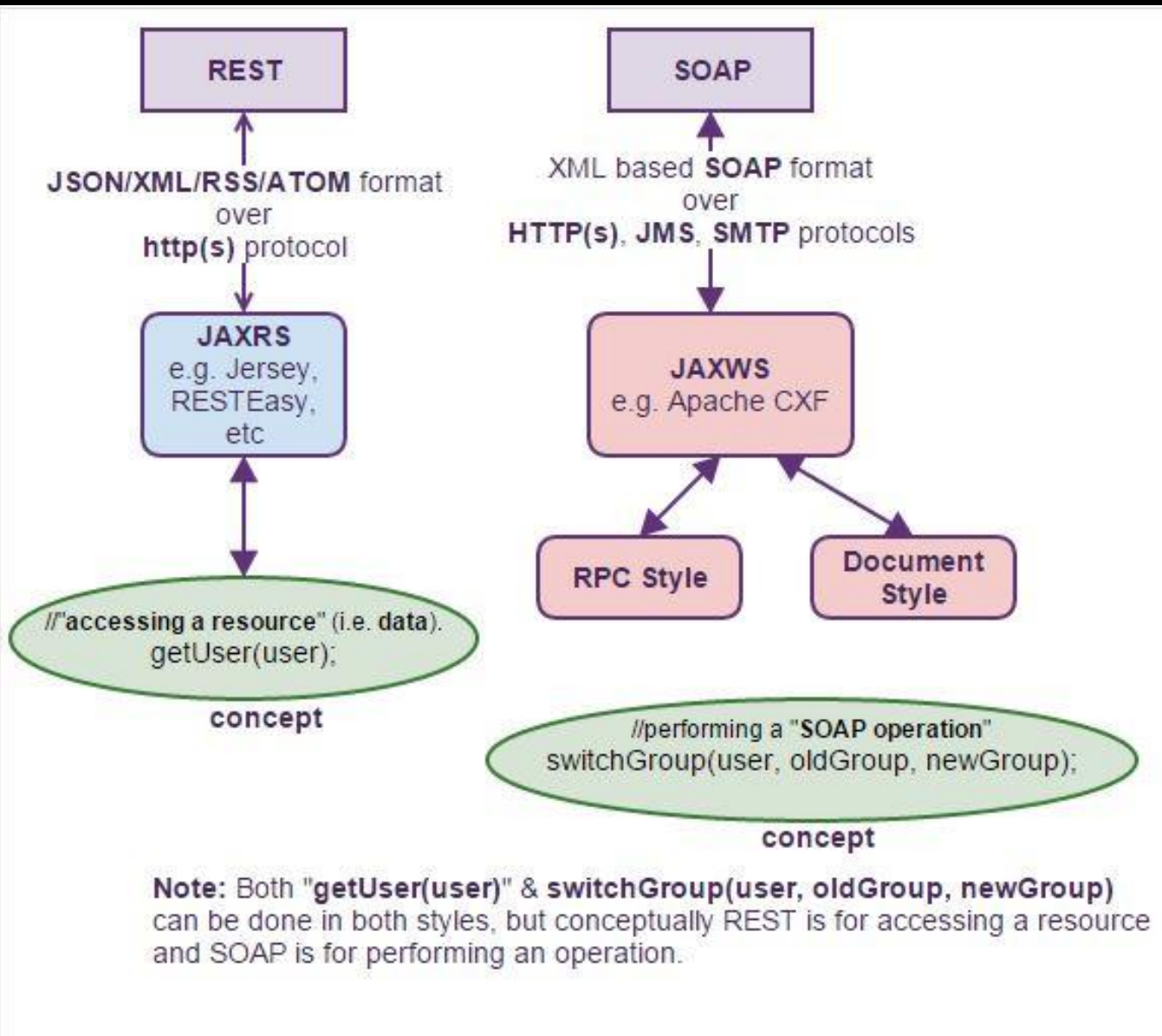
The java (EE) instances

JAX-WS

- Supported on servlet / HTTP
- JavaSE
  - Endpoint, Client, provider
  - Wsimport
- JAXB mapping
  - XML<-> java
- Annotations
  - Closed Semantics
- multiplatform

JAX-RS (RESTful)

- Supported on servlet / HTTP
- JavaSE
  - Jersey
- No mapping
  - Depend on scope/programmer
- Annotations
  - Resource oriented
- multiplaform



01: ♥♦ 40+ Java Web Services Basics Interview Q&As – Q01 – Q06
http://www.java-success.com/java-web-services-interview-questions-and-answers/

Java API for XML Web Services (JAX-WS)

- Part of Java SE
  - web service delivery
  - does not require a servlet or EJB container.
  - This makes HTTP more or less an equal peer of RMI
  - as an intrinsic protocol for distributed computing on the Java platform
- Set of APIs for creating web services in XML format (SOAP)
  - provides many annotations
  - development and deployment for both web service clients and web service providers (endpoints).

Webservices can run in…

- Not exclusive to java…
  - Webserver & application server
  - Need Servlet support
  - In java **Java-WS** implementation is **JAX-WS**
    - JavaSE, Metro, Apache CXF
- Can run in…
  - Java SE
  - Full and lightweight server JavaEE
    - Glassfish, Tomcat + plugins, jetty, Jboss, wildfly….

JAX-RS

- Relies on servlets
- **RESTful services based on REST**
- Annotations allow automation
- Conversion
  - Handling requests
  - …
- Jersey is java official implementation…

http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html

What is REST and SOAP / WS-* ?

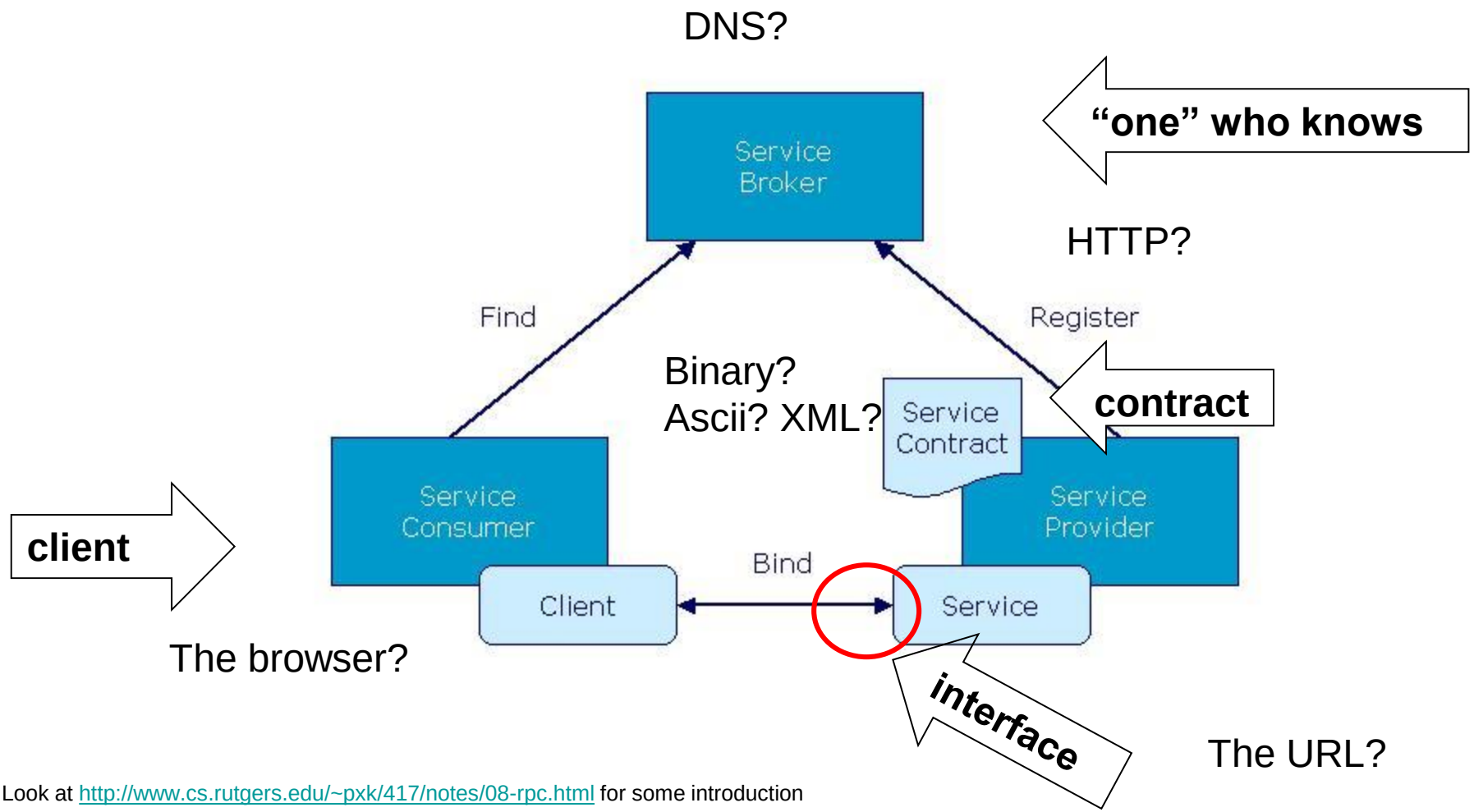
webservices

- WW3C model
- Object mapping semantics
- Transport
  - XML
- Stateless or stateful
- Functional API
  - Contract functions/Data
    - WSDL
  - Where to find them
    - UDDI
  - RPC style
    - SOAP

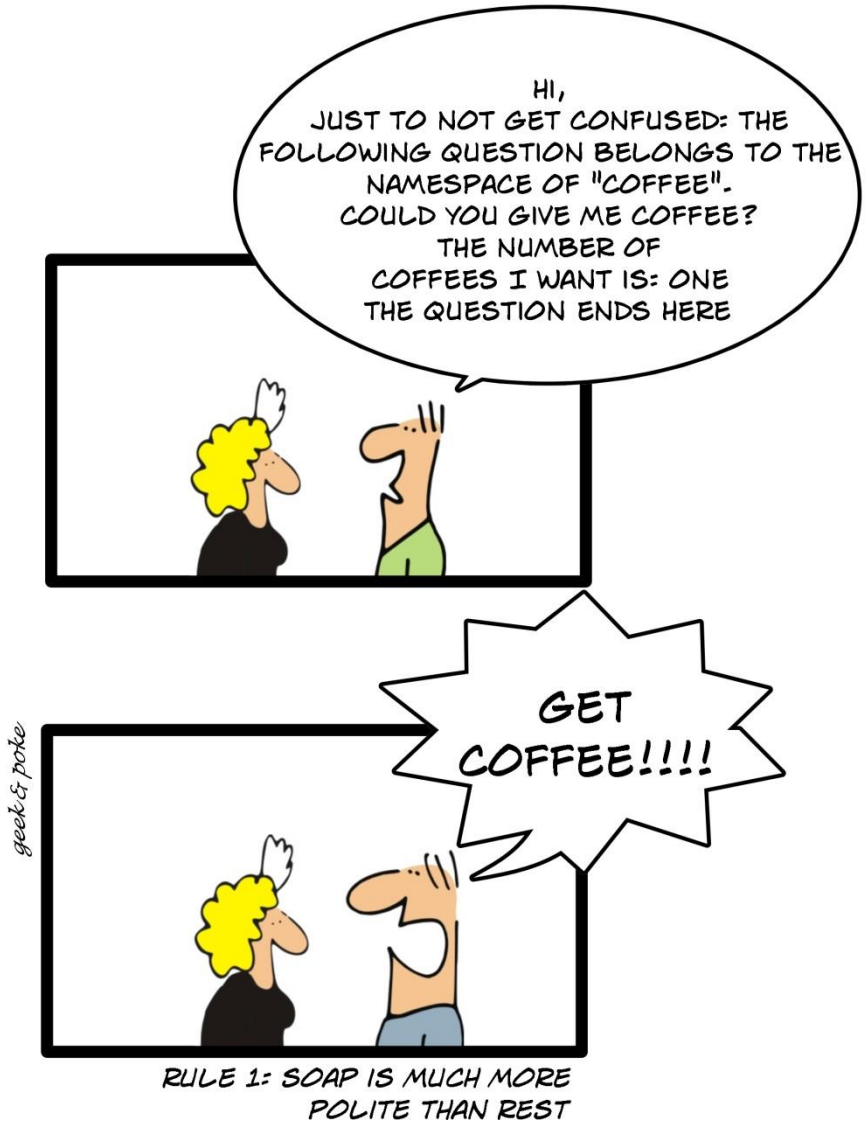
REpresentational State Transfer

- Aka REST (proposed by Felding)
- state transfer management is a mandatory
- Server = retrieve resources
  - Stateless
  - server should not remember the state of the application.
  - retrieve requested resources & related
- Client = manages the application
  - client should send all information
  - All that a client should know about a RESTful interface should be the entry point
  - Ask for record 23 not next record
- Nothing on protocol, encoding, resource address, …

Service oriented architectures



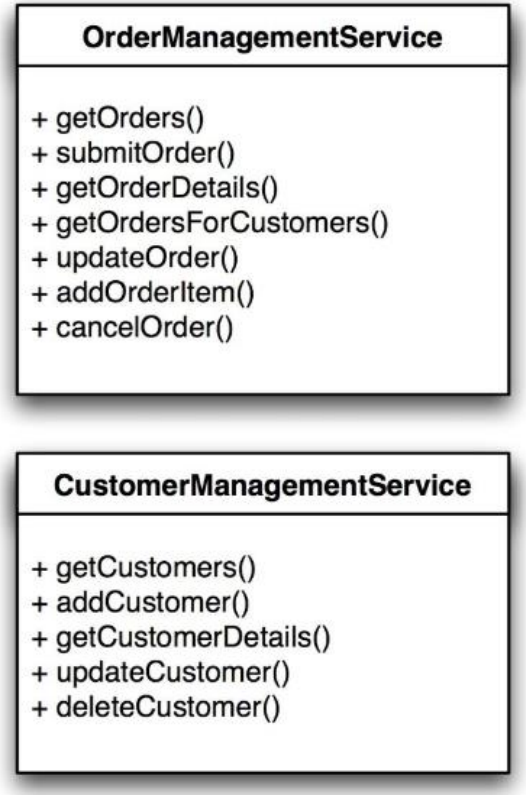
Look at http://www.cs.rutgers.edu/~pxk/417/notes/08-rpc.html for some introduction



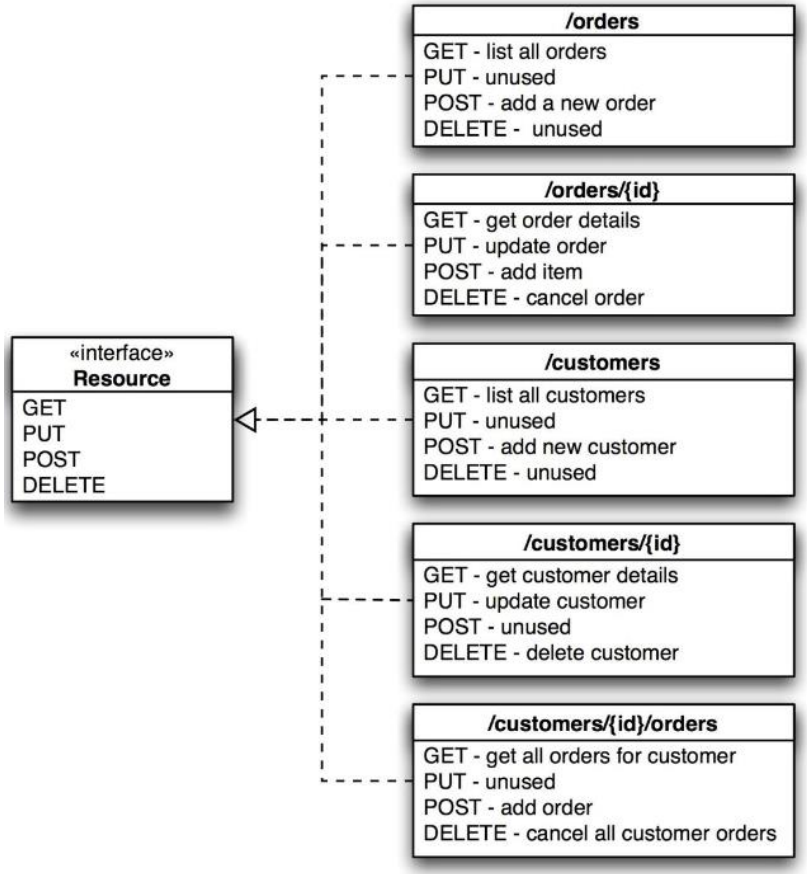
http://www.it-conservations.com/2009_11_01_archive.html

SOAP and REST

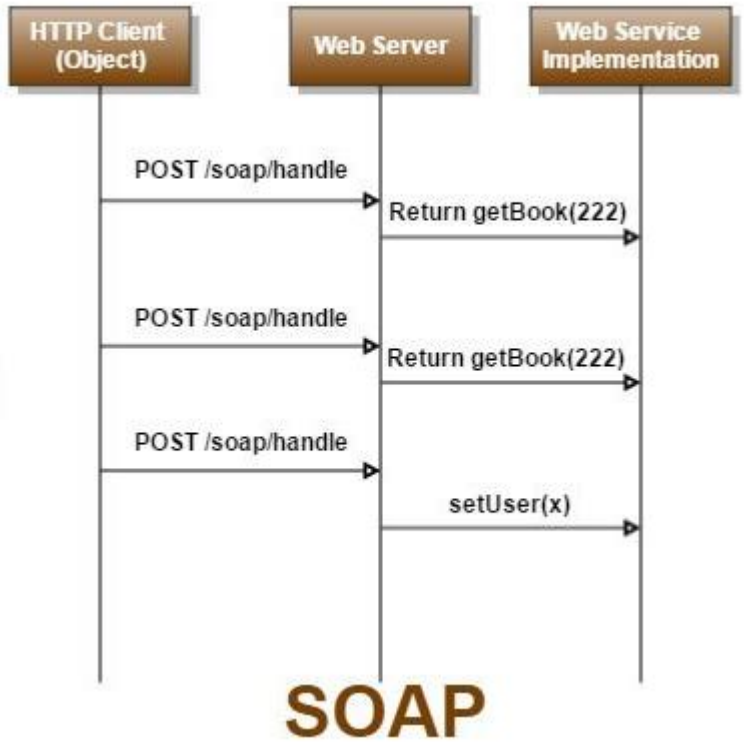
WS-Style

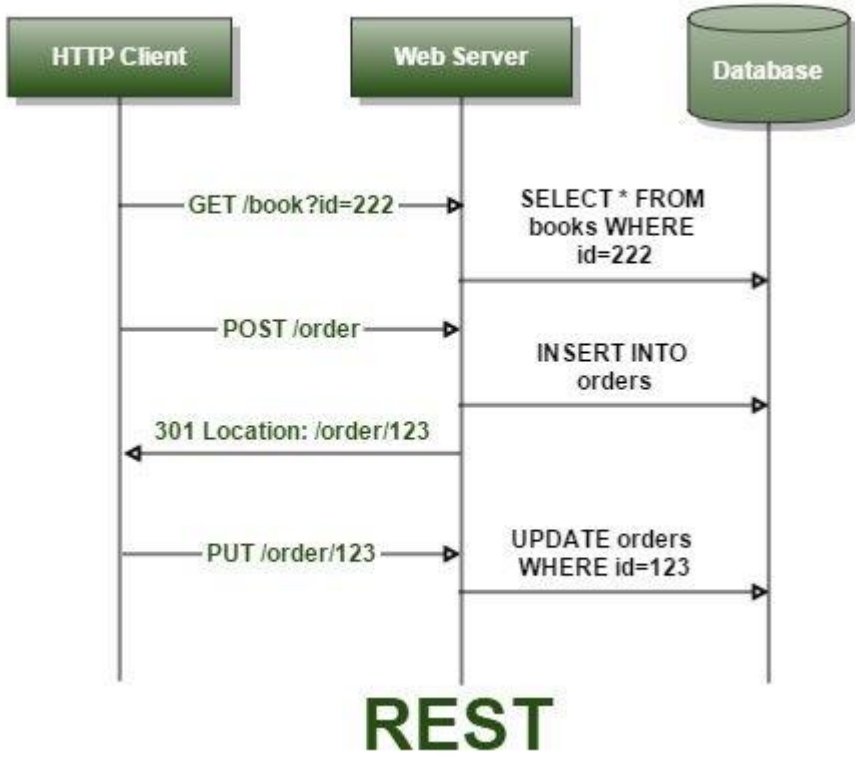


REST-Style



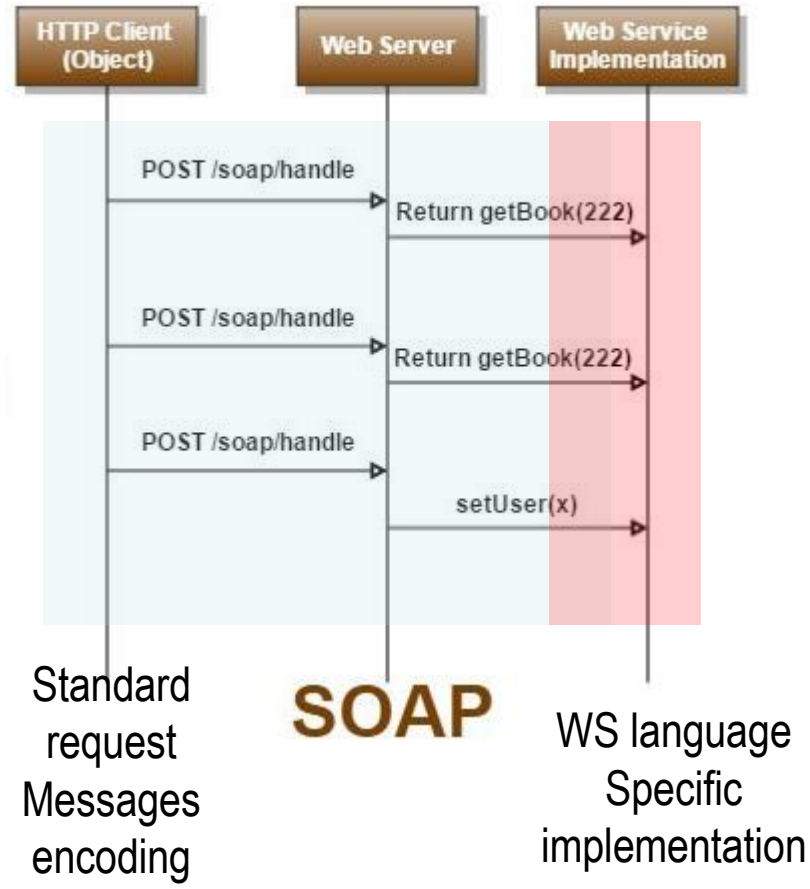
Concerns in different places

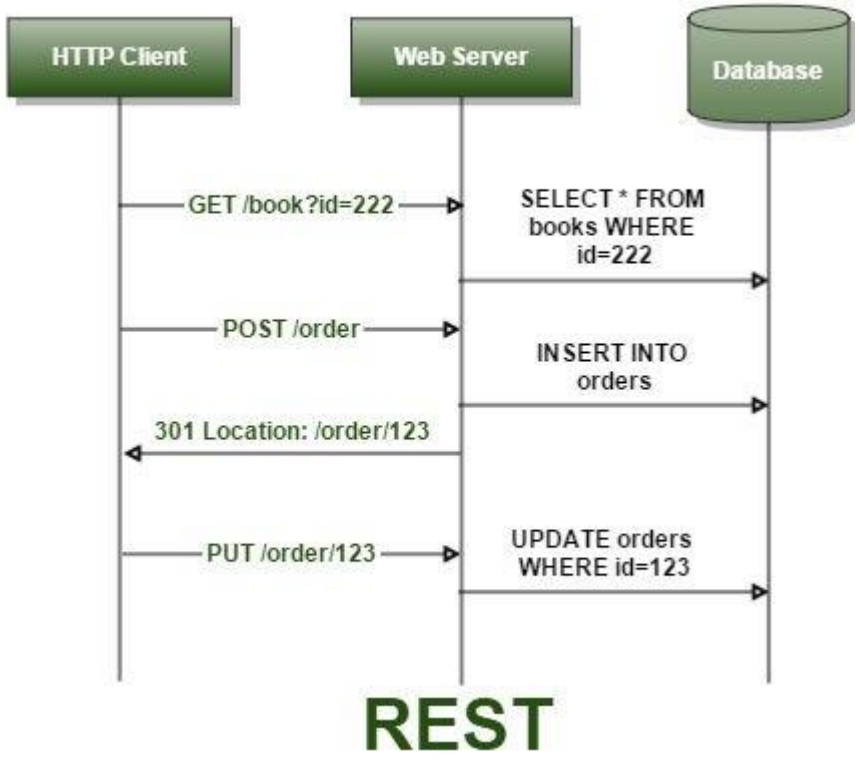




SOAP vs REST, Basic and difference
http://webtechsharing.com/soap-vs-rest/

Concerns in different places



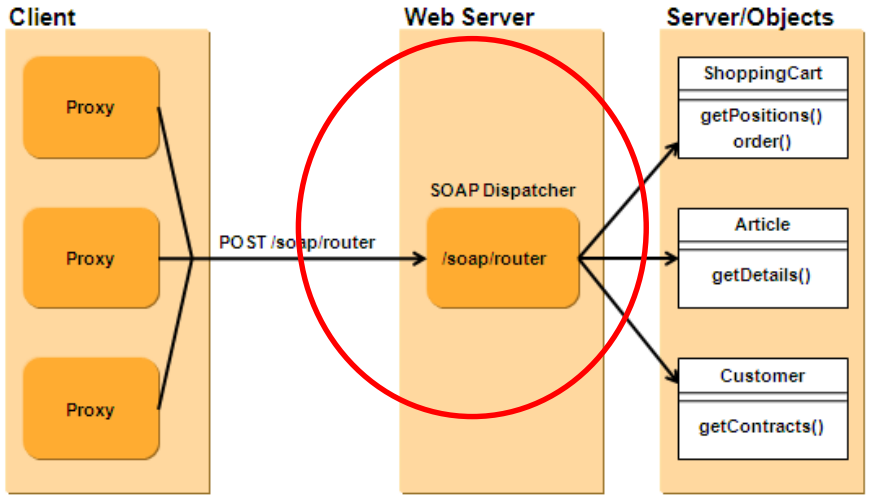


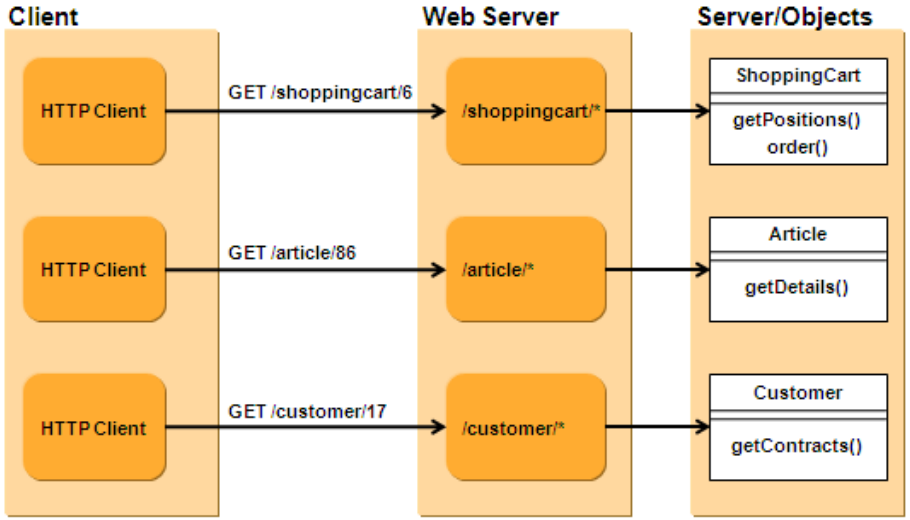
addresses

Not addressed

SOAP vs REST, Basic and difference
http://webtechsharing.com/soap-vs-rest/

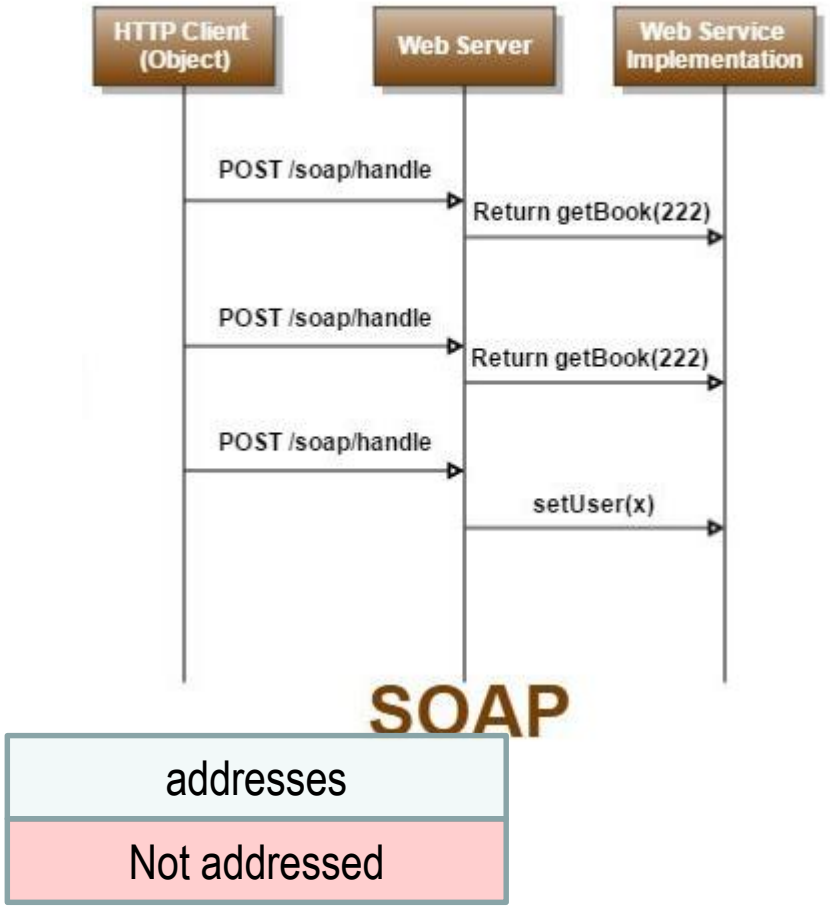
A standard naming service (server)

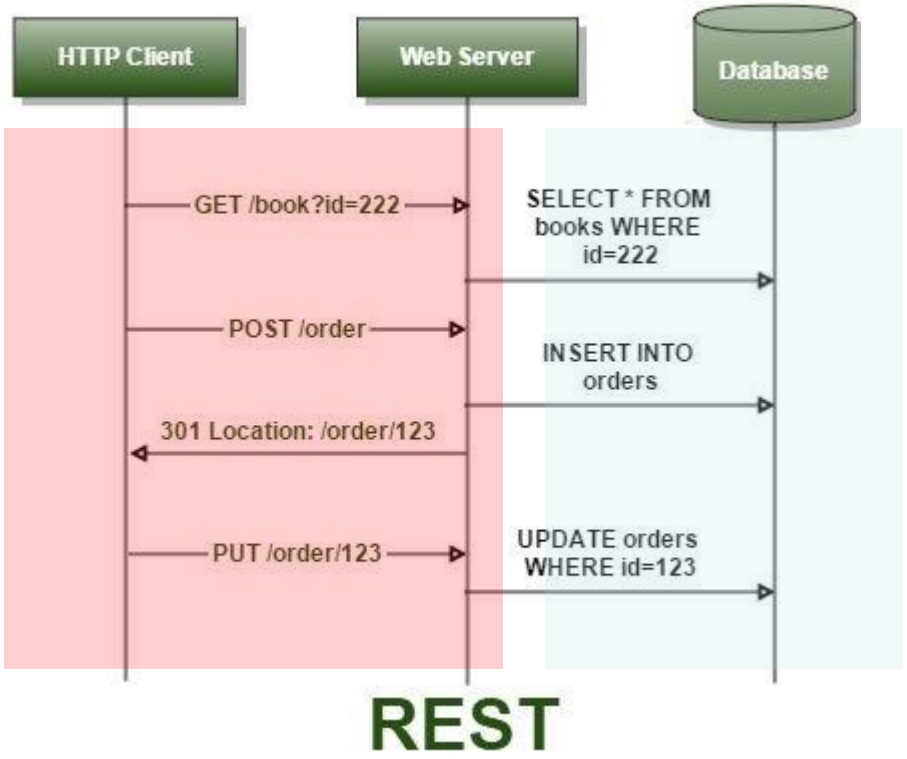




https://www.predic8.com/rest-webservices.htm

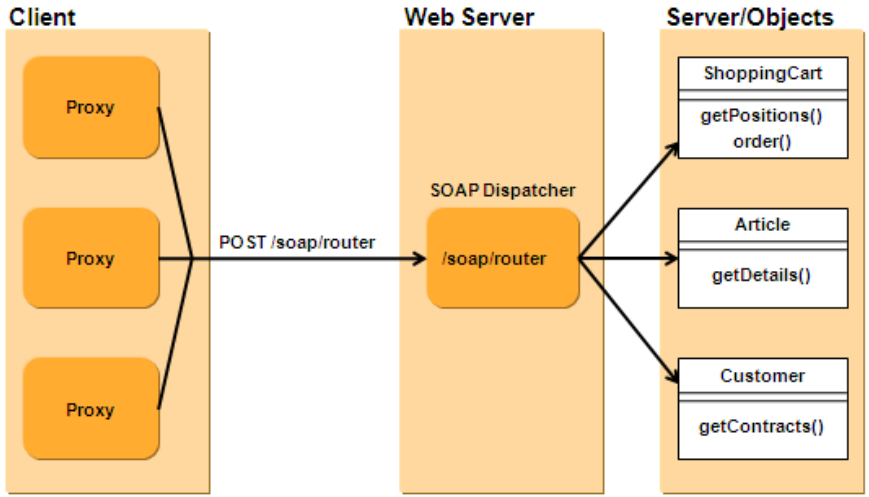
Concerns in different places

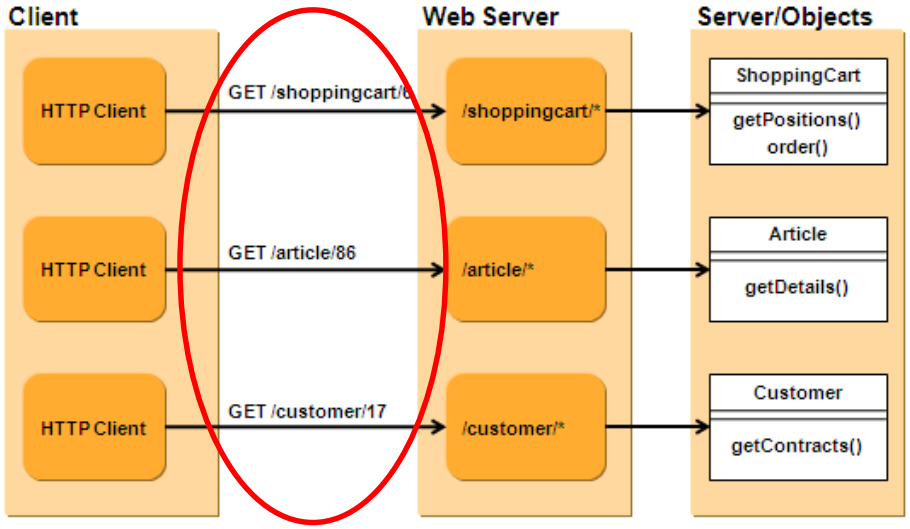




SOAP vs REST, Basic and difference
http://webtechsharing.com/soap-vs-rest/

A standard resource naming (URI)





https://www.predic8.com/rest-webservices.htm

Standards and descriptors

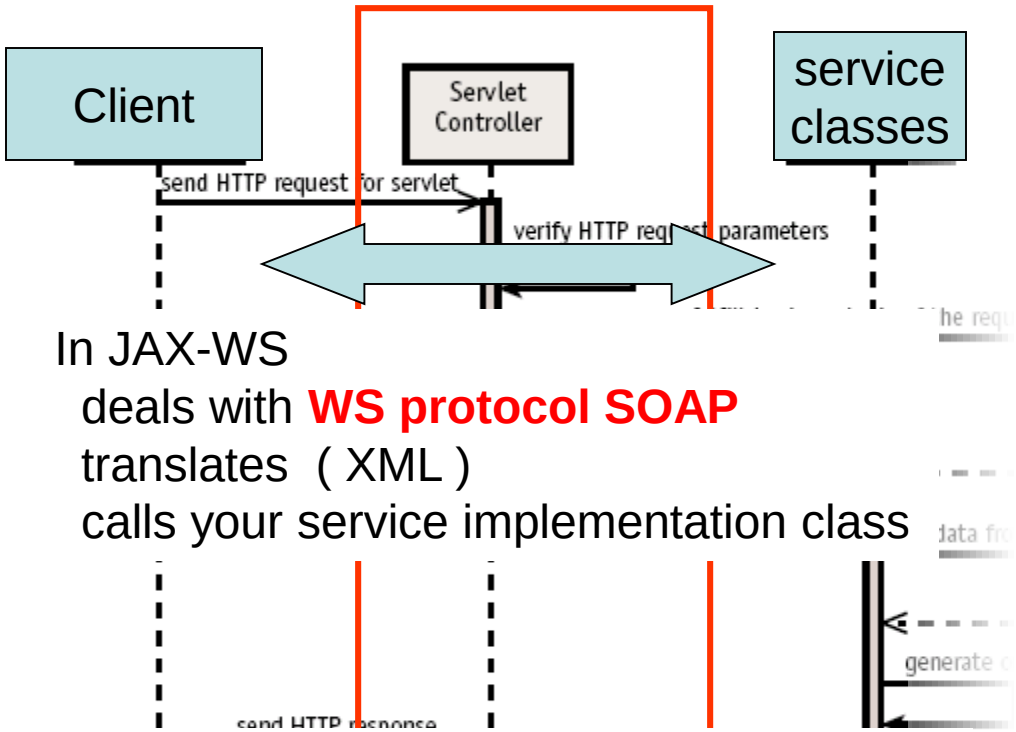
webservices

- WW3C model
- Object mapping semantics
- Transport
  - XML
- Stateless or stateful
- Functional API
  - Contract functions/Data
    - WSDL
  - Where to find them
    - UDDI
  - RPC style
    - SOAP

Rest

- REST (proposed by Felding)
- No object/data mapping semantics
- Transport
  - Open, XML, JSON, …
- Stateless
- Resource based API
  - Resources
    - Unique ID
  - Where to find them
    - URI
  - Use basic HTTP functions
    - GET,PUT, POST, DELETE, …

JAX-RS & JAX-WS: rely on servlets

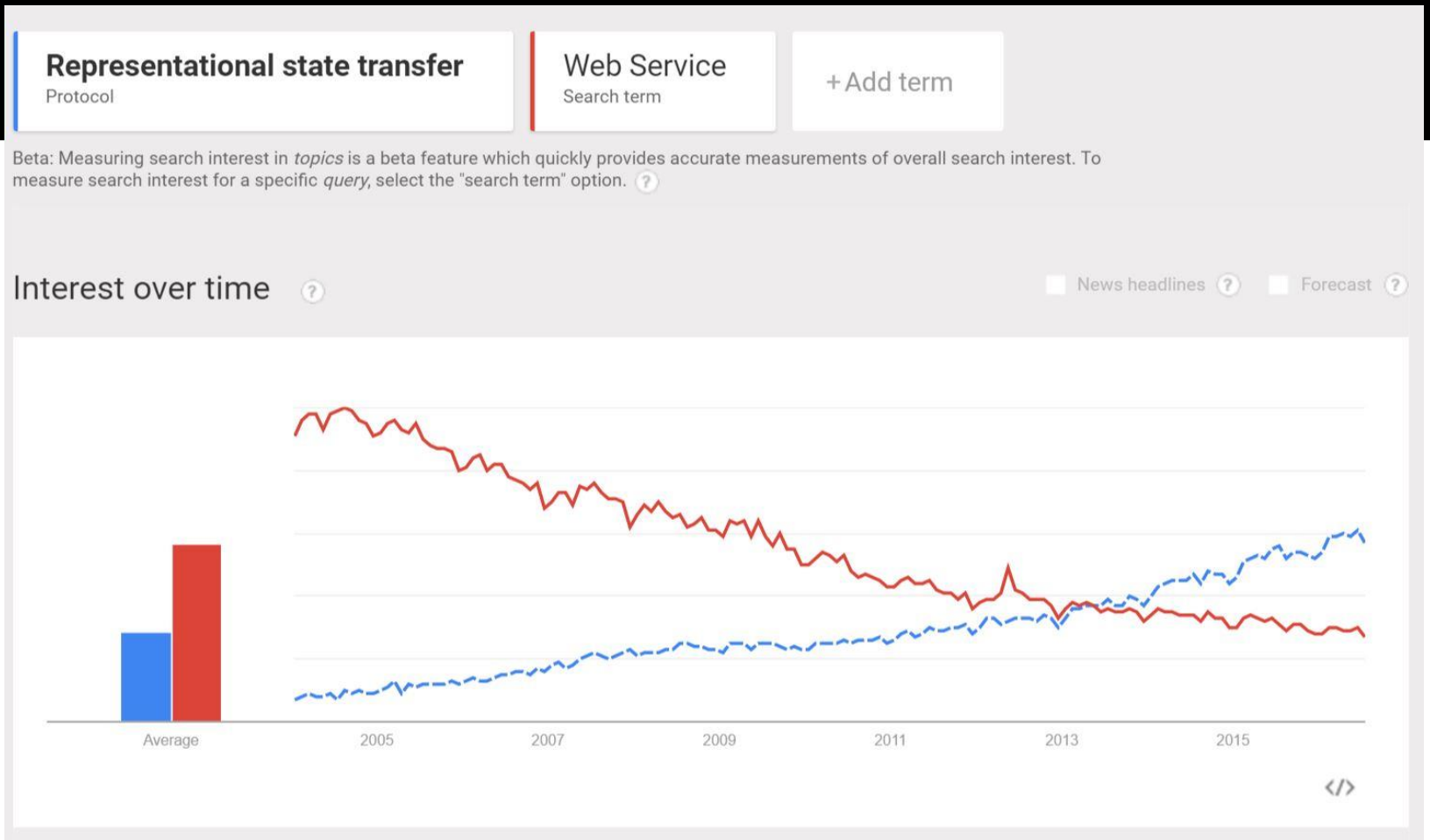


In JAX-WS
- deals with **WS protocol SOAP**
- translates (XML)
- calls your service implementation class

In JAX-RS
- Maps the **URI resource to your service**
- Translates (XML/JSON)
- Calls your service implementation class

Architecture

Examples from “UML Weekend Crash Course”, Wiley, 2002



REST API vs. SOAP Web Services Management
https://dzone.com/articles/rest-api-vs-soap-web-services-management

Representational state transfer
Protocol

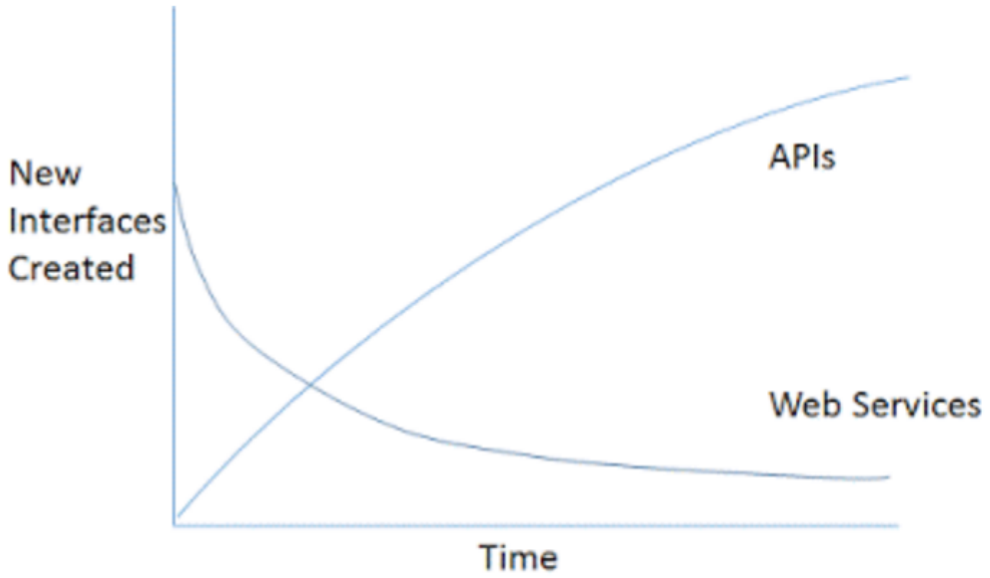
Web Service
Search term

+ Add term

Beta: Measuring search interest in *topics* is a beta feature which quickly provides accurate measurements of overall search interest. To measure search interest for a specific *query*, select the "search term" option.

Interest over ti

Average



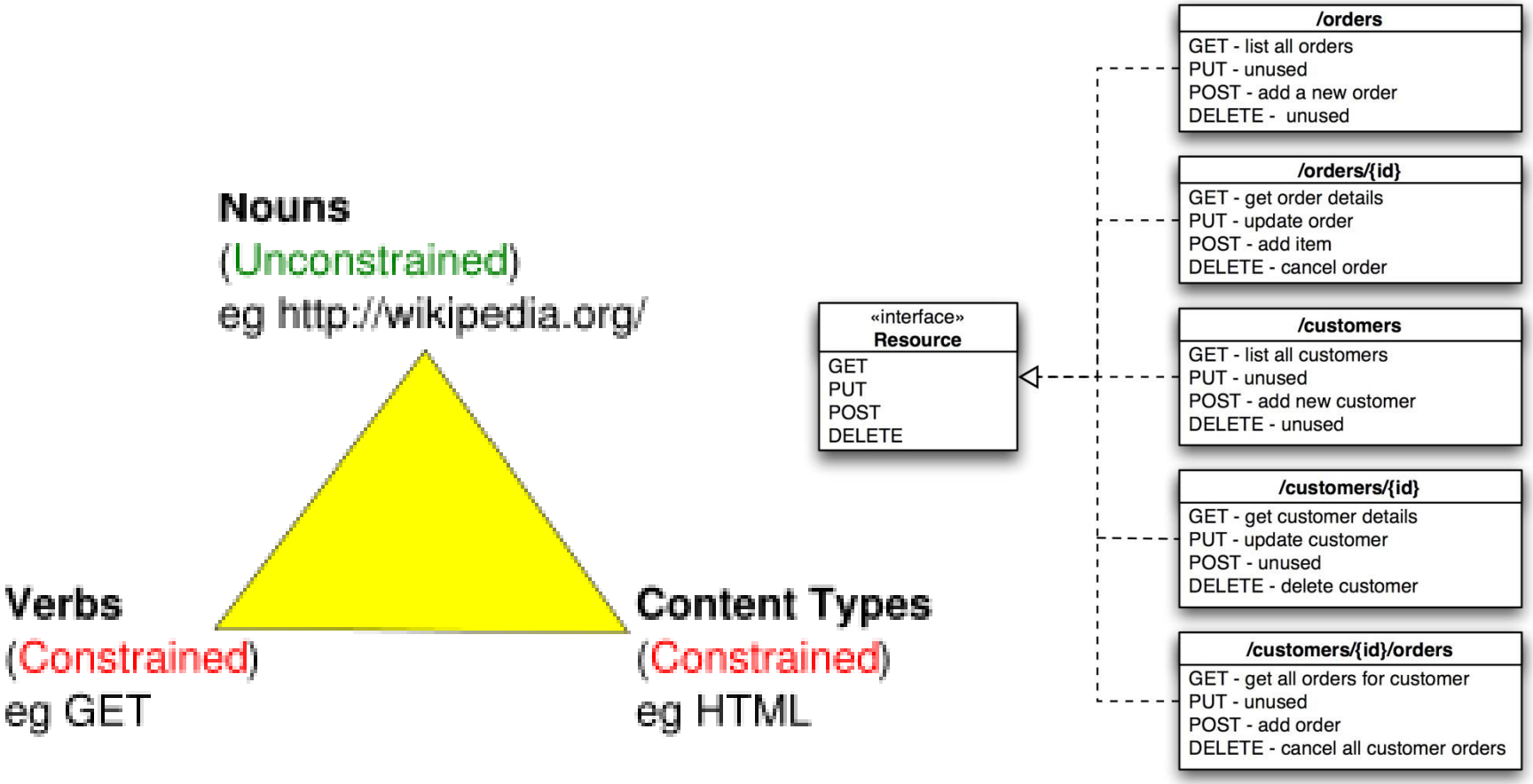
I'm convinced that we will see fewer and fewer web services used internally and externally, but they aren't dead just yet. I expect standards to play a bigger part in API management but its emphasis on simplicity and complexity hiding will stop standards being API's death knell.

BIAS?

REST API vs. SOAP Web Services Management
https://dzone.com/articles/rest-api-vs-soap-web-services-management

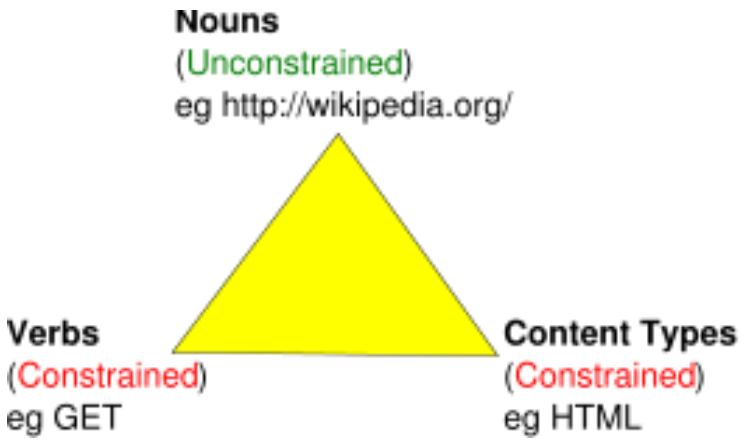
REST and SOAP

REST: resource oriented



REST- REpresentational State Transfer

- Resources
  - an ID, URI – an URL like references
- Verbs
  - The Http methods
- Types
  - with multiple representations
    - XML, JSON, ...

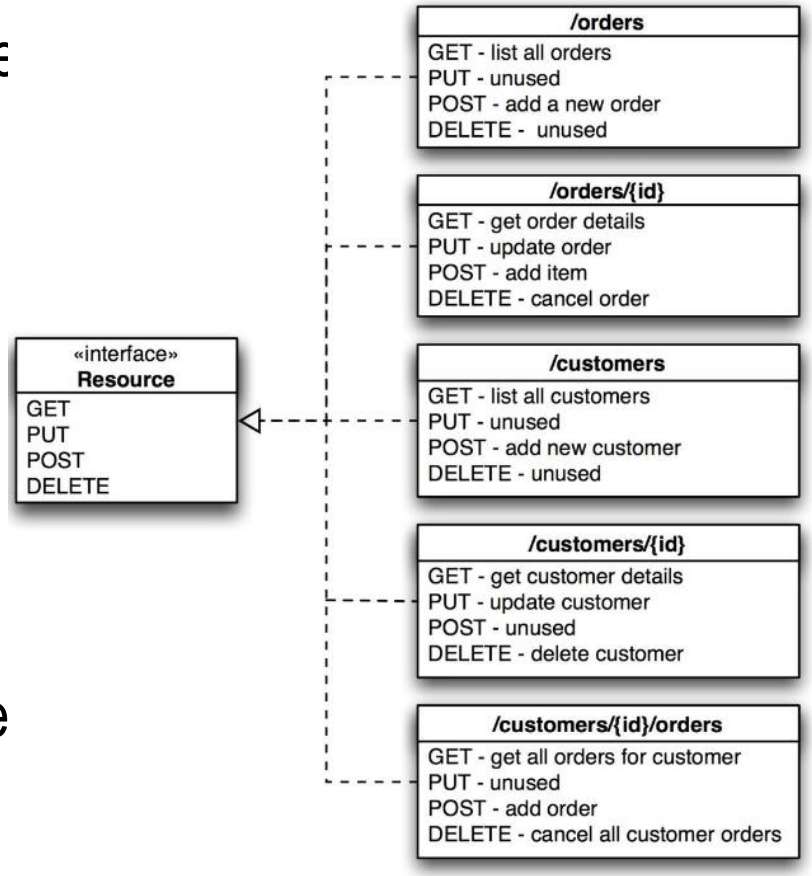


REST (REpresentational State Transfer)
https://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer

http://www.infoq.com/articles/rest-introduction

REST: a different approach

- Noun - Resources
  - Published under a unified refe
  - Referentiable
- Verb - Operation
  - Standard HTTP:
    - GET,PUT,POST, DELETE
  - Uniform semantic
- Contents – format
  - Semantics on the programme



Uniform Resource Identifier (URI)

Resource
Collection URI, such as `http://example.com/resources/`

GET	PUT	POST	DELETE
List the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URL is assigned automatically and is usually returned by the operation.	**Delete** the entire collection.

Element URI, such as `http://example.com/resources/142`

GET	PUT	POST	DELETE
Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Update** the addressed member of the collection, or if it doesn't exist, **create** it.	Treat the addressed member as a collection in its own right and **create** a new entry in it.	**Delete** the addressed member of the collection.

From http://en.wikipedia.org/wiki/Representational_State_Transfer

The Business of IT Blog

REST vs CRUD: What's The Difference?

REST: In a Nutshell

REST refers to a set of defining principles for developing API. It uses HTTP protocols like GET, PUT, POST to link resources to actions within a client-server relationship. In addition to the client-server mandate, it has several other defining constraints. The principles of RESTful architecture serve to create a stable and reliable application, that offers simplicity and end-user satisfaction.

In REST:

- Representations must be uniform with regard to resources
- Hypermedia represents relationships between resources
- Only one entry into an API to create one self-contained interface, then hyperlink to create relationships

January 18, 2018 · by Stephen Watts

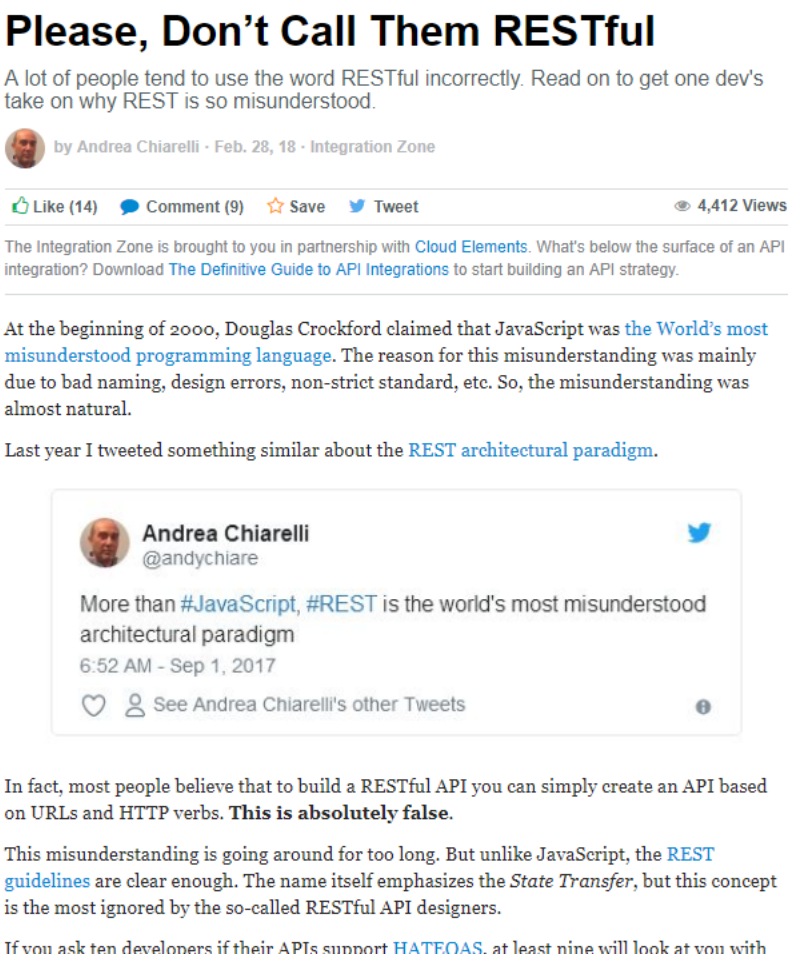
REST vs CRUD: What's The Difference?
https://www.bmc.com/blogs/rest-vs-crud-whats-the-difference/

REST API

- provides the client with a new state and ways to switch to subsequent states.
- provides a representation of a resource (not necessarily in JSON) and enriched links (*hypermedia*) to other related resources that may **move the application to another state**
- the resource describes itself and provides information about related resources.

Please, Don't Call Them RESTful
https://dzone.com/articles/please-dont-call-them-restful
https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

RESTful is not HTTP



Please, Don't Call Them RESTful

A lot of people tend to use the word RESTful incorrectly. Read on to get one dev's take on why REST is so misunderstood.

by Andrea Chiarelli · Feb. 28, 18 · Integration Zone

Like (14) Comment (9) Save Tweet 4,412 Views

The Integration Zone is brought to you in partnership with Cloud Elements. What's below the surface of an API integration? Download The Definitive Guide to API Integrations to start building an API strategy.

At the beginning of 2000, Douglas Crockford claimed that JavaScript was the World's most misunderstood programming language. The reason for this misunderstanding was mainly due to bad naming, design errors, non-strict standard, etc. So, the misunderstanding was almost natural.

Last year I tweeted something similar about the REST architectural paradigm.

Andrea Chiarelli
@andychiare

More than #JavaScript, #REST is the world's most misunderstood architectural paradigm

6:52 AM - Sep 1, 2017

See Andrea Chiarelli's other Tweets

In fact, most people believe that to build a RESTful API you can simply create an API based on URLs and HTTP verbs. **This is absolutely false.**

This misunderstanding is going around for too long. But unlike JavaScript, the REST guidelines are clear enough. The name itself emphasizes the *State Transfer*, but this concept is the most ignored by the so-called RESTful API designers.

If you ask ten developers if their APIs support HATEOAS, at least nine will look at you with

Please, Don't Call Them RESTful
https://dzone.com/articles/please-dont-call-them-restful

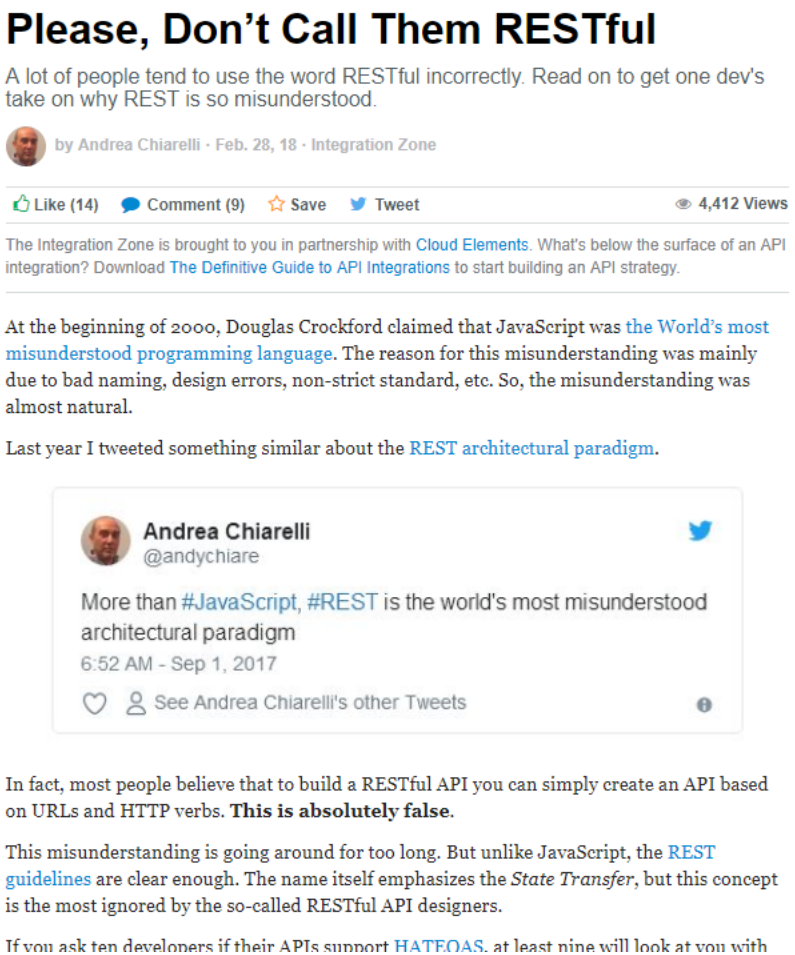
- No HTTP or other protocol
  - says nothing about the protocols to be used
  - way to identify a resource: url
  - The verbs : http verbs
  - Formats : json, xml,…
- Although it is the popular solution

Web APIs / HTTP APIs not RESTful

- most people believe that to build a RESTful API you can simply create an API based on URLs and HTTP verbs.

- APIs that simply map CRUD actions to HTTP verbs have nothing to do with *Application State Transfer*. You can call them Web APIs or HTTP APIs, **but please don't call them RESTful.**

Please, Don't Call Them RESTful
https://dzone.com/articles/please-dont-call-them-restful



Please, Don't Call Them RESTful

A lot of people tend to use the word RESTful incorrectly. Read on to get one dev's take on why REST is so misunderstood.

by Andrea Chiarelli · Feb. 28, 18 · Integration Zone

Like (14) Comment (9) Save Tweet 4,412 Views

The Integration Zone is brought to you in partnership with Cloud Elements. What's below the surface of an API integration? Download The Definitive Guide to API Integrations to start building an API strategy.

At the beginning of 2000, Douglas Crockford claimed that JavaScript was the World's most misunderstood programming language. The reason for this misunderstanding was mainly due to bad naming, design errors, non-strict standard, etc. So, the misunderstanding was almost natural.

Last year I tweeted something similar about the REST architectural paradigm.

Andrea Chiarelli
@andychiare

More than #JavaScript, #REST is the world's most misunderstood architectural paradigm
6:52 AM - Sep 1, 2017

See Andrea Chiarelli's other Tweets

In fact, most people believe that to build a RESTful API you can simply create an API based on URLs and HTTP verbs. **This is absolutely false.**

This misunderstanding is going around for too long. But unlike JavaScript, the REST guidelines are clear enough. The name itself emphasizes the *State Transfer*, but this concept is the most ignored by the so-called RESTful API designers.

If you ask ten developers if their APIs support HATEOAS, at least nine will look at you with

Please, Don't Call Them RESTful
https://dzone.com/articles/please-dont-call-them-restful

Ruben Verborgh

blog | publications | articles | teaching | contact

REST, where's my state?

HTTP interactions should be stateless, but two kinds of state are involved.

HTTP, the Hypertext Transfer Protocol, has been designed under the constraints of the REST architectural style. One of the well-known constraints of this REpresentational State Transfer style is that communication must be *stateless*. Why was this particular constraint introduced? And who is in charge then of maintaining state, since it is clearly necessary for many Web applications? This post explains how statelessness works on today's Web, explaining the difference between *application state* and *resource state*.

24 August 2012

Roy T. Fielding, the main author of the HTTP 1.1 specification, has devoted a whole

basis

he

REST, where's my state?
https://ruben.verborgh.org/blog/2012/08/24/rest-wheres-my-state/

Ruben Verborgh

blog publications articles teaching contact

REST, where's my state?

HTTP intera
of state are

**HTTP, THE HYPERT
constraints of the
this REPRESENTATI
stateless. Why wa
then of maintaini
This post explain
difference between *application state* and *resource state*.**

24 August 2012

Roy T. Fielding, the main author of the HTTP 1.1 specification, has devoted a whole

basis

he

Statelessness eliminates the need to remember

The notion of statelessness is defined from the perspective of the server. The constraint says that the **server should not remember** the state of the application. As a consequence, the **client should send all information** necessary for execution along with each request, because the server cannot reuse information from previous requests as it didn't memorize them.

Concretely, this means if you're browsing an image gallery and the server has just send you image number 23, your client **cannot simply say *next*** to the server. Instead, it asks for *image 24* to advance in the gallery. Indeed, your client has to supply all information necessary to execute the request, since the server does not remember that you were viewing image 23.

REST, where's my state?
https://ruben.verborgh.org/blog/2012/08/24/rest-wheres-my-state/

RESTful may not be REST

REST

- **Transferring**, accessing and manipulating textual data **representations**, in a **stateless** manner. When deployed correctly, it provides a uniform, interoperability, between different applications on the internet. The term stateless is a crucial piece to this as it allows applications to **communicate agnostically.**

RESTful a.k.a. web API

- A RESTful API service is exposed through a **Uniform Resource Locator (URL).** This logical name separates the identity of the resource from what is accepted or returned. The URL scheme is defined in RFC 1738, which can be found here.

RESTFUL ARCHITECTURE 101
https://blog.cloud-elements.com/restful-architecture-101

REST, RESTful, WebAPI

- Big overload of concepts
- REST
  - language-independent architectural style.
- RESTful
  - Can implement REST
  - usually means web API / CRUD

The Business of IT Blog

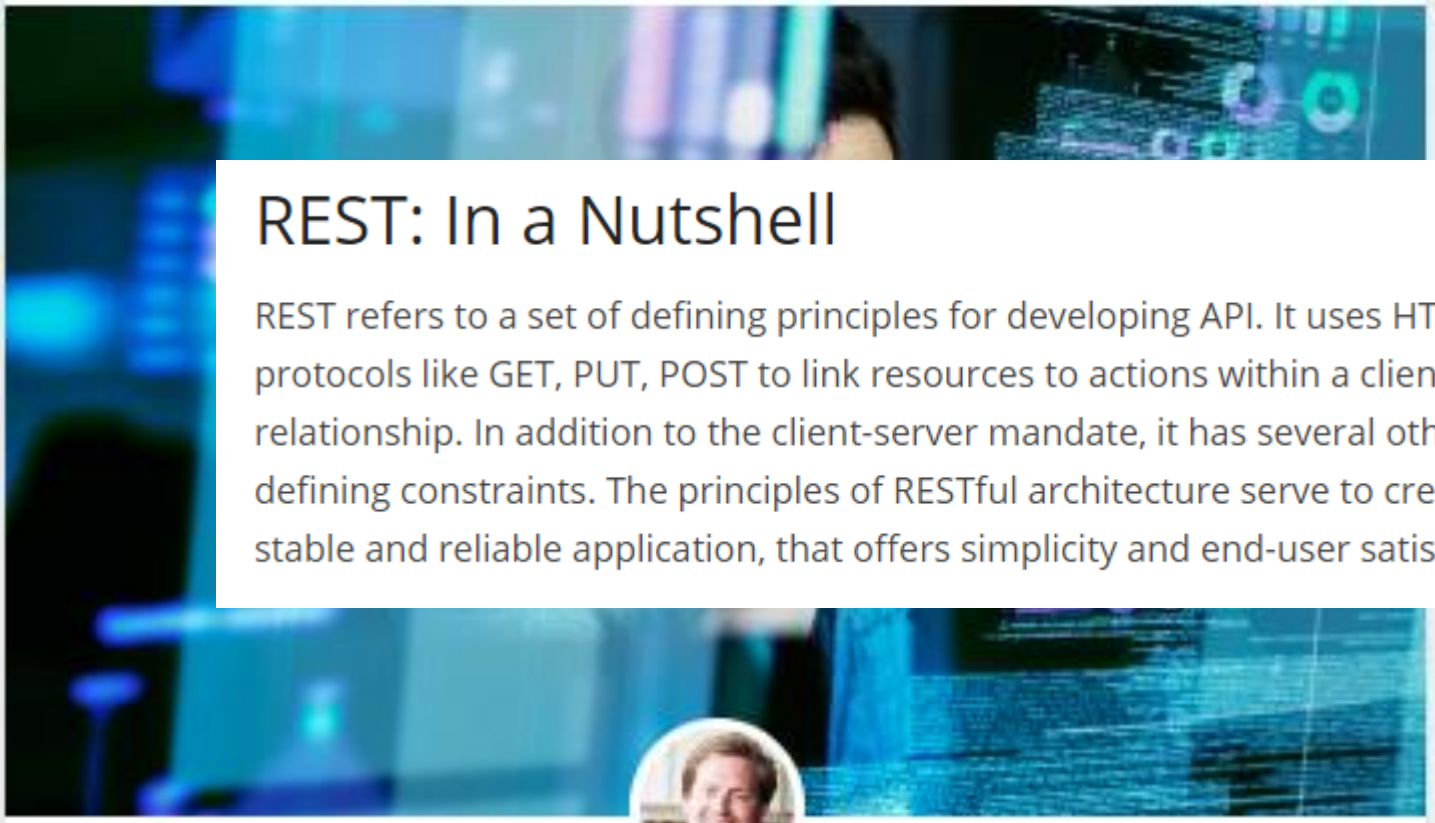
REST vs CRUD: What's The Difference?

January 18, 2018 · by Stephen Watts

REST vs CRUD: What's The Difference?
https://www.bmc.com/blogs/rest-vs-crud-whats-the-difference/

The Business of IT Blog

REST vs CRUD: What's The Difference?



REST: In a Nutshell

REST refers to a set of defining principles for developing API. It uses HTTP protocols like GET, PUT, POST to link resources to actions within a client-server relationship. In addition to the client-server mandate, it has several other defining constraints. The principles of RESTful architecture serve to create a stable and reliable application, that offers simplicity and end-user satisfaction.

January 18, 2018 · by Stephen Watts

REST vs CRUD: What's The Difference?
https://www.bmc.com/blogs/rest-vs-crud-whats-the-difference/

The Business of IT Blog

REST vs CRUD: What's The Difference?

- REST is an architectural system centered around resources and hypermedia, via HTTP protocols
- CRUD is a cycle meant for maintaining permanent records in a database setting
- CRUD principles are mapped to REST commands to comply with the goals of RESTful architecture

In REST:

- Representations must be uniform with regard to resources
- Hypermedia represents relationships between resources
- Only one entry into an API to create one self-contained interface, then hyperlink to create relationships

January 18, 2018 · by Stephen Watts

REST vs CRUD: What's The Difference?
https://www.bmc.com/blogs/rest-vs-crud-whats-the-difference/

Some references

- Part VI Web Services
  - http://docs.oracle.com/javaee/7/tutorial/partwebsvcs.htm#BNAYK
  - Chapter 28, "Building Web Services with JAX-WS"
    - http://docs.oracle.com/javaee/7/tutorial/jaxws.htm#BNAYL
  - Chapter 29, "Building RESTful Web Services with JAX-RS"
    - http://docs.oracle.com/javaee/7/tutorial/jaxrs.htm#GIEPU
- Java API for RESTful Services (JAX-RS)
  - http://jax-rs-spec.java.net/

- Links on JAX-WS and webservices
  - https://goo.gl/CphyvY
- Links on JAX-RS and RESTful
  - https://goo.gl/FWexPP

The END