



Génie logiciel

Philippe Dugerdil

14.11.2019



Principles of REST architecture

Implementing web services



Introduction

Representational State Transfer (REST)

- Fielding, R. T., Taylor, R. N. - Principled design of the modern Web architecture. IEEE ICSE 2000, Limerick, Ireland
- Fielding, R. T. - Architectural Styles and the Design of Network-based Software Architectures. PhD Thesis, UC Irvine, 2000



Today's challenge : system's integration

- Application heterogeneity is a fact
- Technology heterogeneity is a fact
- Platform heterogeneity is a fact

But...

- Enterprise (and IT) agility is must

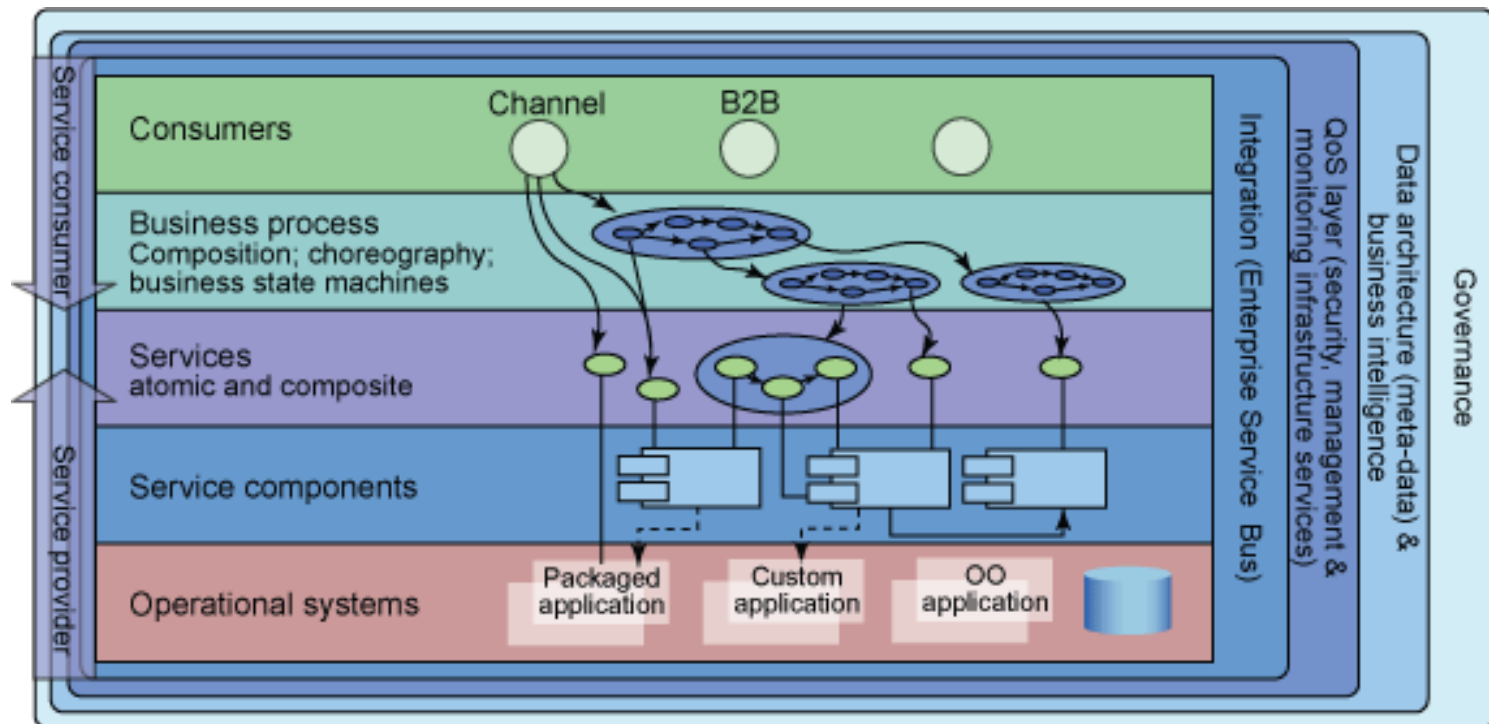


We need a standard way to integrate all this stuff



Integration at the enterprise level: SOA


IBM's view on SOA





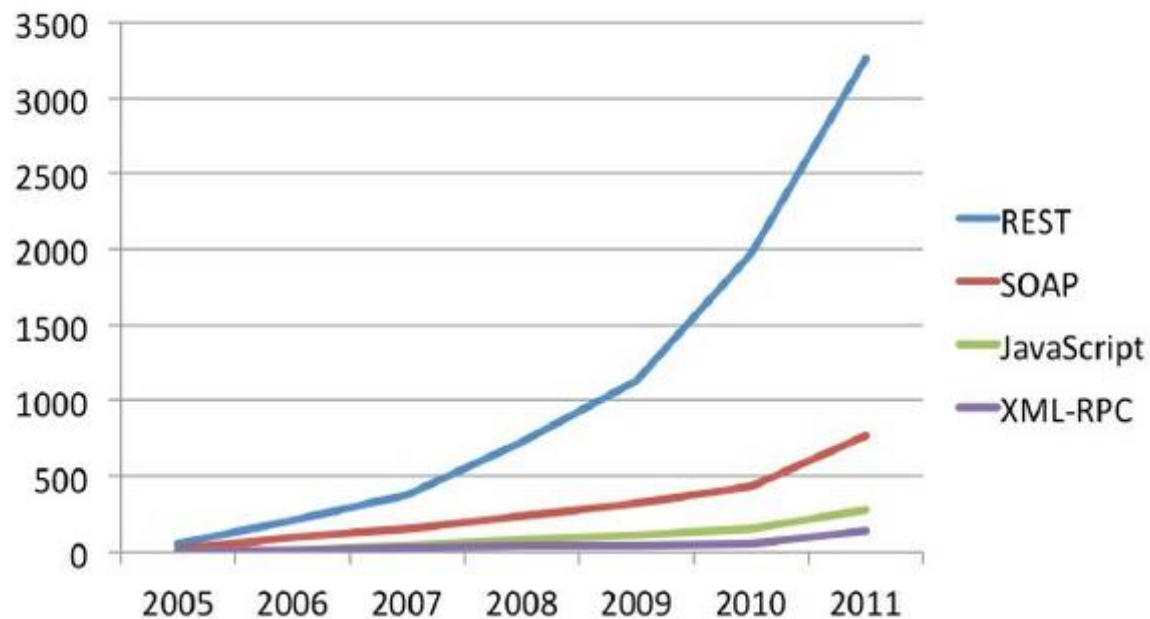
This is very complicated. Would there be a simpler way? HTTP to the rescue

- HTTP is here to stay
- Plenty of services are available through HTTP
- Could we use HTTP as the glue with which all the components are integrated ?

YES  REST



An old statistic, that shows REST dominance



API protocols and styles

Based on directory of 5,100 web APIs listed at ProgrammableWeb, February 2012

Source: computed by Dino Chiesa, <http://www.dinochiesa.net/?p=259>



HTTP

HTTP (Hypertext Transfer Protocol) sits on top of TCP

- Request/Response kind of handshake
- Format: HTTP://<host>:<port>/<path>?<query>

URL

<query> is a sequence of <key>=<value> pairs separated by “&”

HTTP defines **methods** (sometimes referred to as *verbs*) to indicate the desired action to be performed on the identified **resource**. What this resource represents depends on the implementation of the server. [Wikipedia]



Resource

The target of an HTTP request

“HTTP does not limit the nature of a resource; it merely defines an interface that might be used to interact with resources. Each resource is identified by a Uniform Resource Identifier (URI)”

[RFC 7231 – HTTP Semantics and Contents <http://tools.ietf.org/html/rfc7231>]

- In other words, a resource is anything interesting enough to be reachable through an URI (URL).



HTTP Representation

“Information that is intended to reflect a past, current, or desired state of a given resource

in a format that can be readily communicated via the protocol, and that consists of a set of representation metadata and a potentially unbounded stream of representation data.”

[RFC 7231 – HTTP Semantics and Contents <http://tools.ietf.org/html/rfc7231>]

- In other words, a representation is information about a resource communicated to and from the HTTP client



Main HTTP verbs

GET: requests a representation of the specified resource. Such request should only retrieve data and should have no other effect.

POST : requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI.

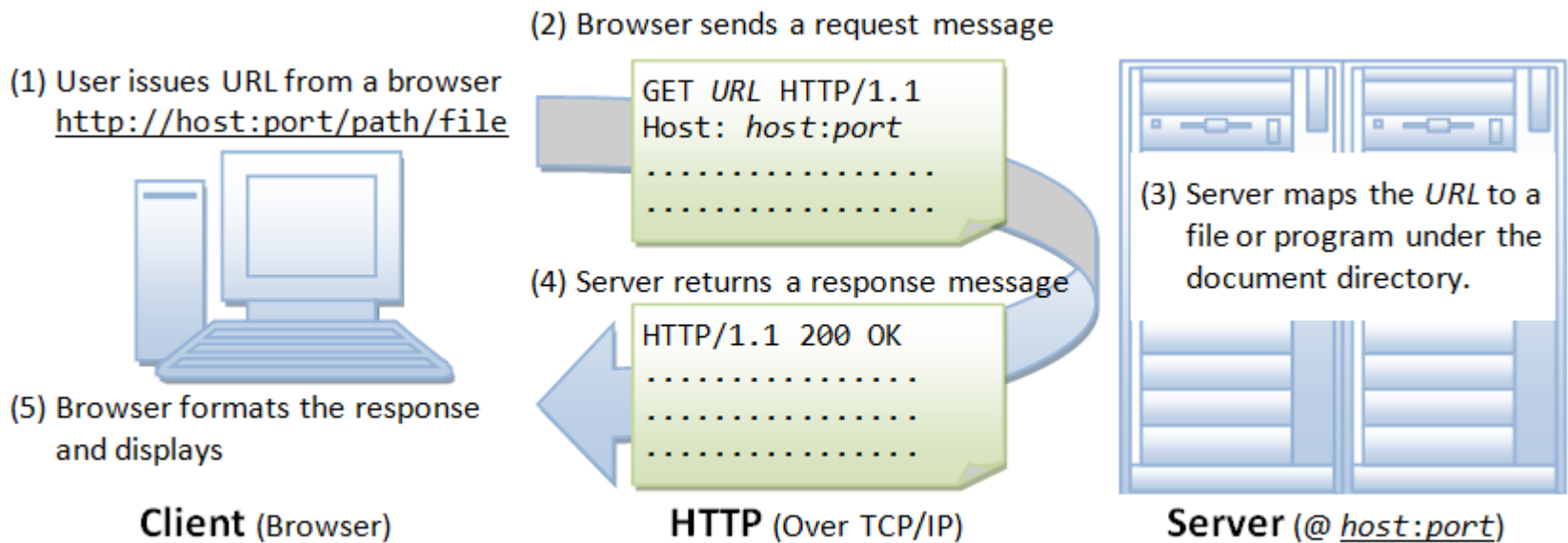
PUT: requests that the enclosed entity be stored under the supplied URI.

DELETE: deletes the specified resource.



Example of interaction

Status code: “the first line of the HTTP response is called the *status line* and includes a numeric *status code* (such as “404”) and a textual *reason phrase* (such as “Not Found”).”
[Wikipedia]



Source: https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html



HTTP Requests

Request structure:

- Request-line (i.e. the “verb” and URL)
- Zero or more header fields followed by CRLF
- An empty line (CRLF) indicating the end of the header fields
- Optional a message-body

Request-line

GET /hello.htm HTTP/1.1

User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

Host: www.tutorialspoint.com

Accept-Language: en-us

Connection: Keep-Alive

Headers

<person>

<firstname> jules </firstname>

<lastname> bonnot</lastname>

</person>

Body (Payload)



HTTP Response

Response structure:

- Status-line
- Zero or more header fields followed by CRLF
- Empty line (CRLF) indicating the end of the header fields
- Optionally a message-body

HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache/2.2.14 (Win32)

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

Content-Length: 88

Content-Type: text/html

Connection: Closed

Status line

Headers

<html>

<body>

<h1>Hello, World!</h1>

</body>

</html>

Body (Payload)



Representational State Transfer

REST

“REST is an **architectural style** consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements, **within a distributed hypermedia system.**”

“The REST architectural style is also applied to the development of web services as an alternative to other distributed-computing specifications such as SOAP.”

[Wikipedia]



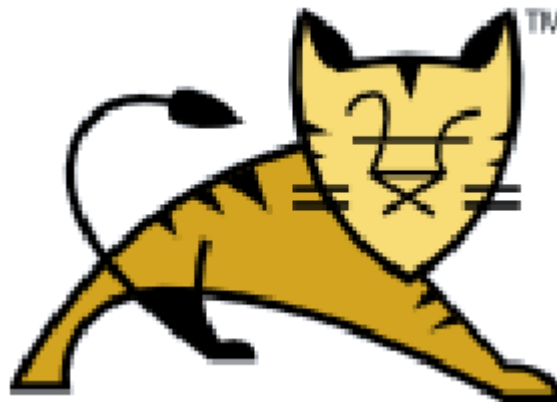
Key features

- Simple, lightweight, easy to implement
- HTTP standard compliant
- Large adoption by the market
- No client state on the server

REST services run in a container



So we need to learn a bit about **Apache Tomcat**





Apache Tomcat

Mainly: a **Servlet** machine with HTTP front end

- A Java **servlet** is a Java program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers. Such Web servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET.
- Servlets could in principle communicate over any client–server protocol, but they are most often used with the HTTP protocol.

[wikipedia]



Servlet communication architecture

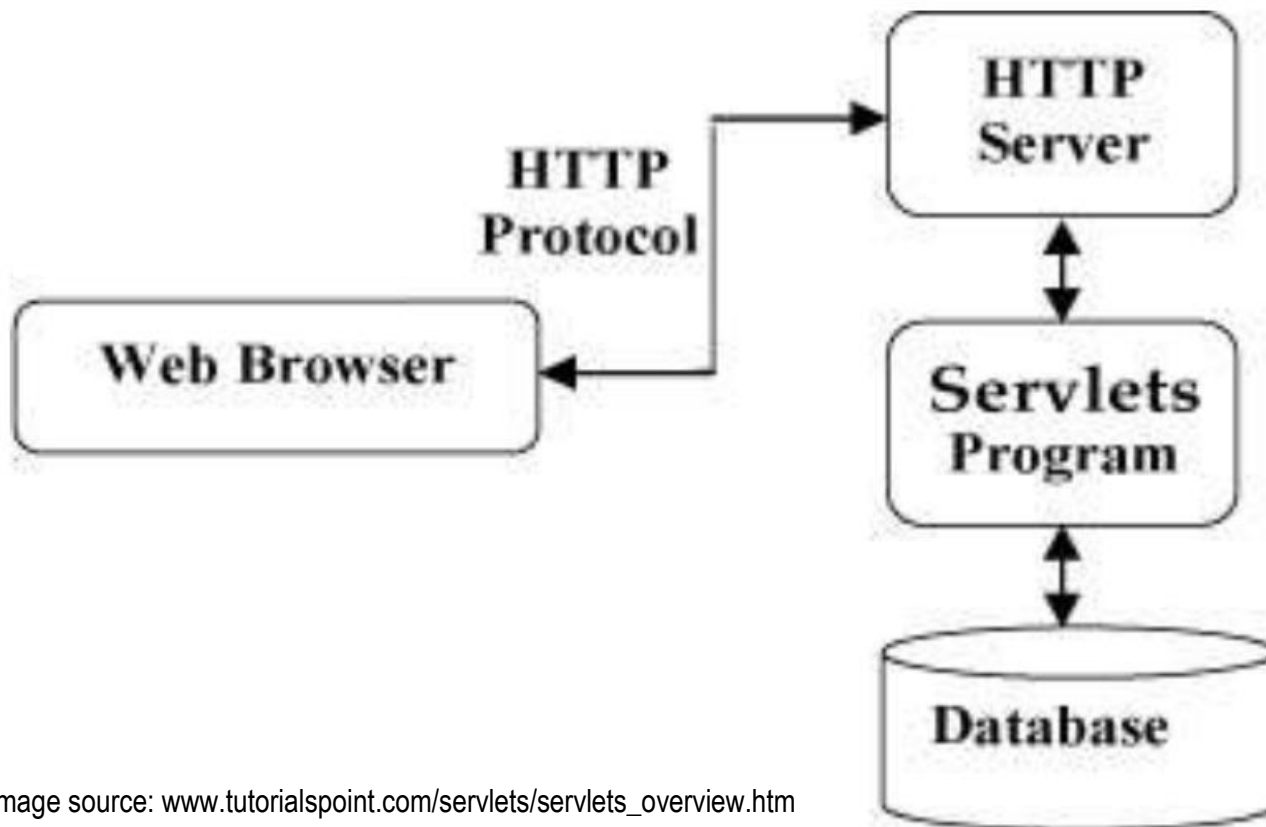
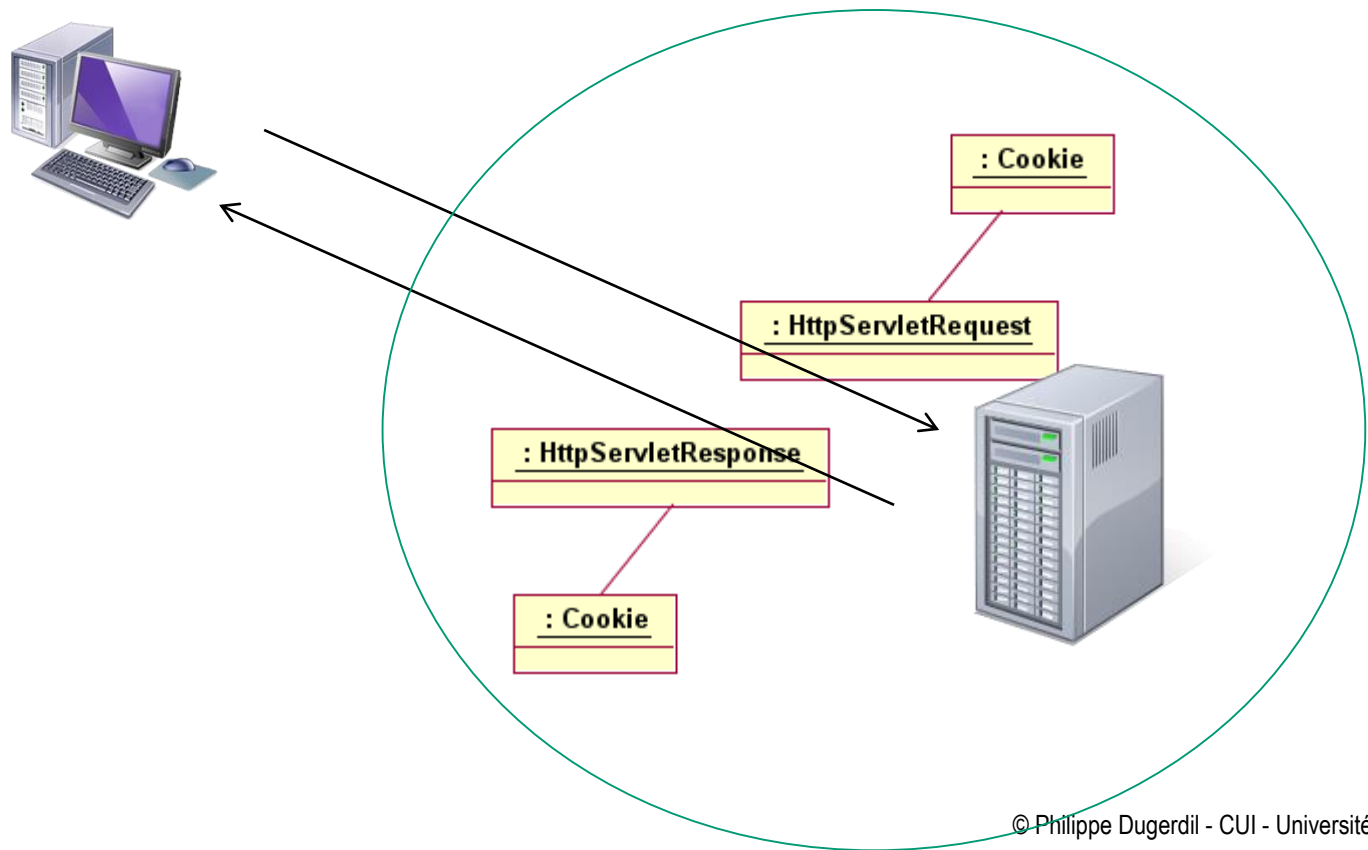


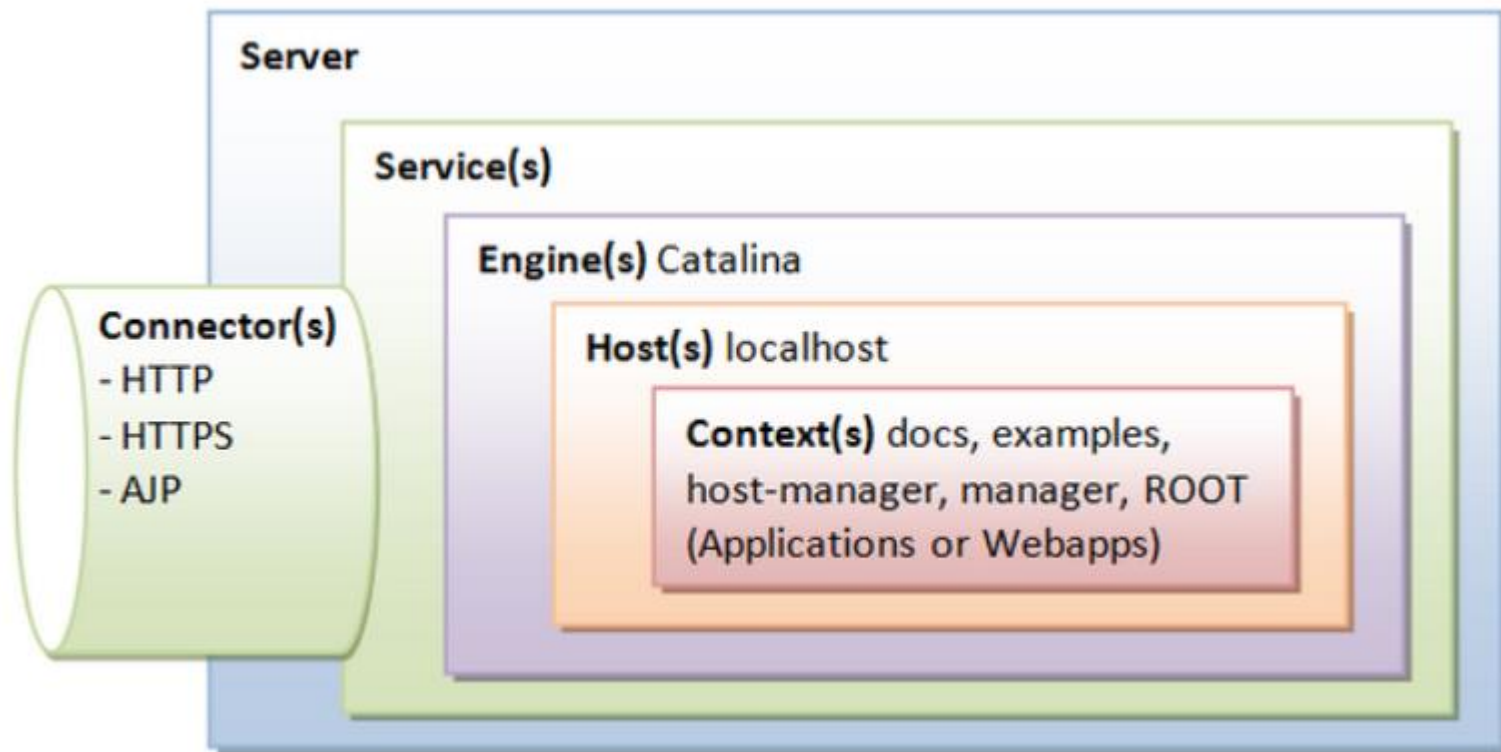
Image source: www.tutorialspoint.com/servlets/servlets_overview.htm

Examples of servlet Java classes: request/response





Tomcat Architecture



Source: <http://examples.javacodegeeks.com/enterprise-java/tomcat/tomcat-server-xml-configuration-example/>



Architecture explained

- **Server:** the whole container
- **Engine:** request processing pipeline receiving requests from one or more Connectors and handing the response back to the appropriate Connector
- **Connector:** handles the communications with the client using some protocol (such as HTTP) and port.
- **Service:** combination of one or more Connector that share a single Engine for processing incoming requests.
- **Host:** virtual host that contains the apps. A host is linked to a domain name that must be associated to the IP address of the server.
 - The host component will direct the request to the domain name to the proper app

[inspired from : tomcat.apache.org/tomcat-8.0-doc/architecture/overview.html]



Server configuration file (server.xml)

server.xml excerpt:

```
<Server port="8005" shutdown="SHUTDOWN">
  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />
    <Engine name="Catalina" defaultHost="localhost">
      <Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">
        </Host>
      </Engine>
    </Service>
  </Server>
```

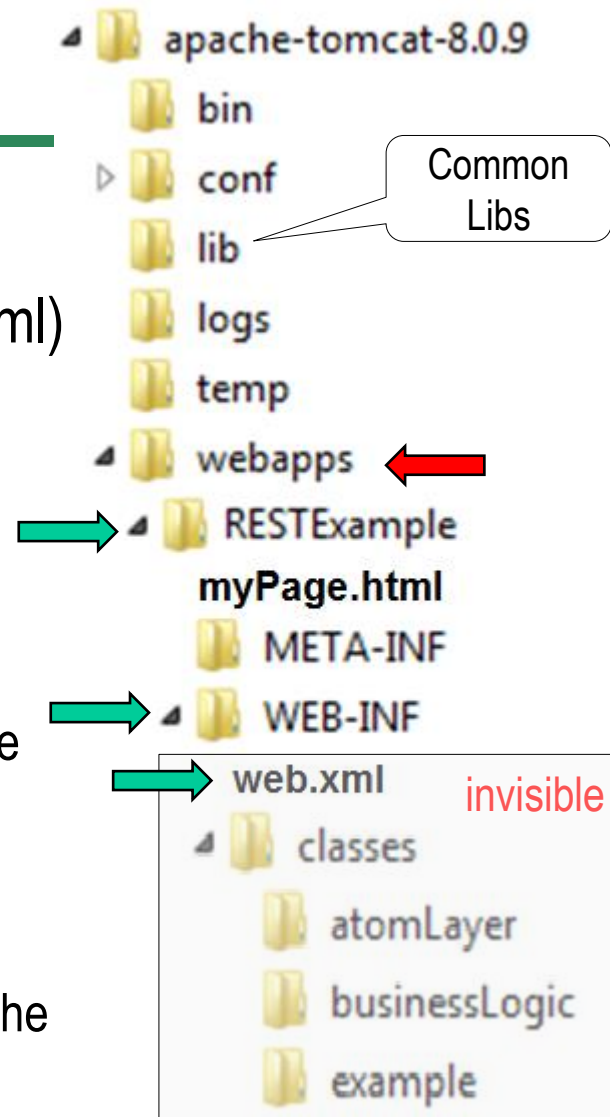
Server port is the one to listen to for the shutdown command

Location: c:\Program Files\apache-tomcat-8.0.26\conf)



Tomcat default application directory hierarchy

- Each individual application should be (server.xml) located in a direct subdirectory of **/webapps**
- **/WEB-INF** separates the files of the application that are visible and invisible from the outside:
 - Files outside are visible i.e. can be reached using the path `ServerURL/AppName/FileName`
`http://myserverURL/REStExample/myPage.html`
 - Files inside are invisible.
 - The URL path to the servlets should be declared in the deployment descriptor: **web.xml**





Tomcat webapps folders example...

[tomcat root dir]

/webapps

/[AppName]

*.html

Application directory

/WEB-INF

web.xml

Deployment descriptor

/classes

//pojo and servlet classes

/custom directory hierarchy

*.class

/lib

*.jar //specific libraries

Specific to
each app

C:\Program Files\apache-tomcat-8.0.9

/webapps

/test

index.html

/WEB-INF

web.xml

/classes

/mainervlets

MyServlet.class

/lib

mylib.jar



..and the associated **web.xml** deployment descriptor

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<web-app >
<display-name>Test Application </display-name>
<description>this is a test application</description>
<servlet>
  <servlet-name>selectionServlet</servlet-name>
  <servlet-class>mainervlets.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>selectionServlet</servlet-name>
  <url-pattern>/choice</url-pattern>
</servlet-mapping>
</web-app>
```

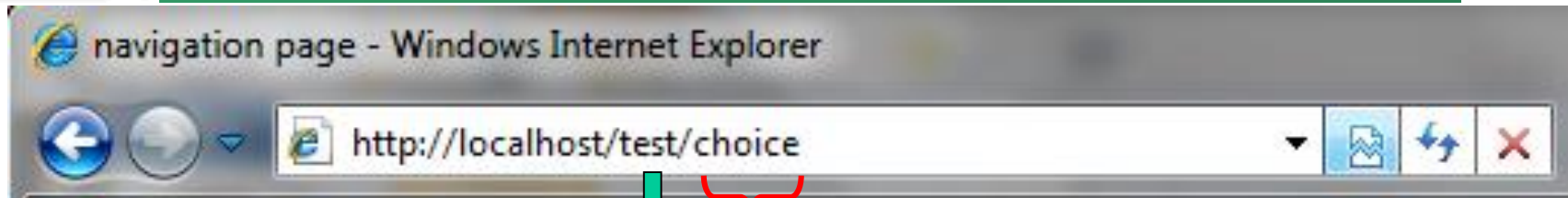
Servlet ID in the server

Qualified class name of the servlet class relative to **/classes**

Servlet ID in the server

Path to launch the servlet, relative to **/AppName**

Launching the example servlet



[tomcat]

/ webapps

/ test

/ WEB-INF

web.xml

/classes

/mainservlets

MyServlet.class

```
<servlet>
  <servlet-name>choiceServlet</servlet-name>
  <servlet-class>mainservlets.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>choiceServlet</servlet-name>
  <url-pattern>/choice</url-pattern>
</servlet-mapping>
```

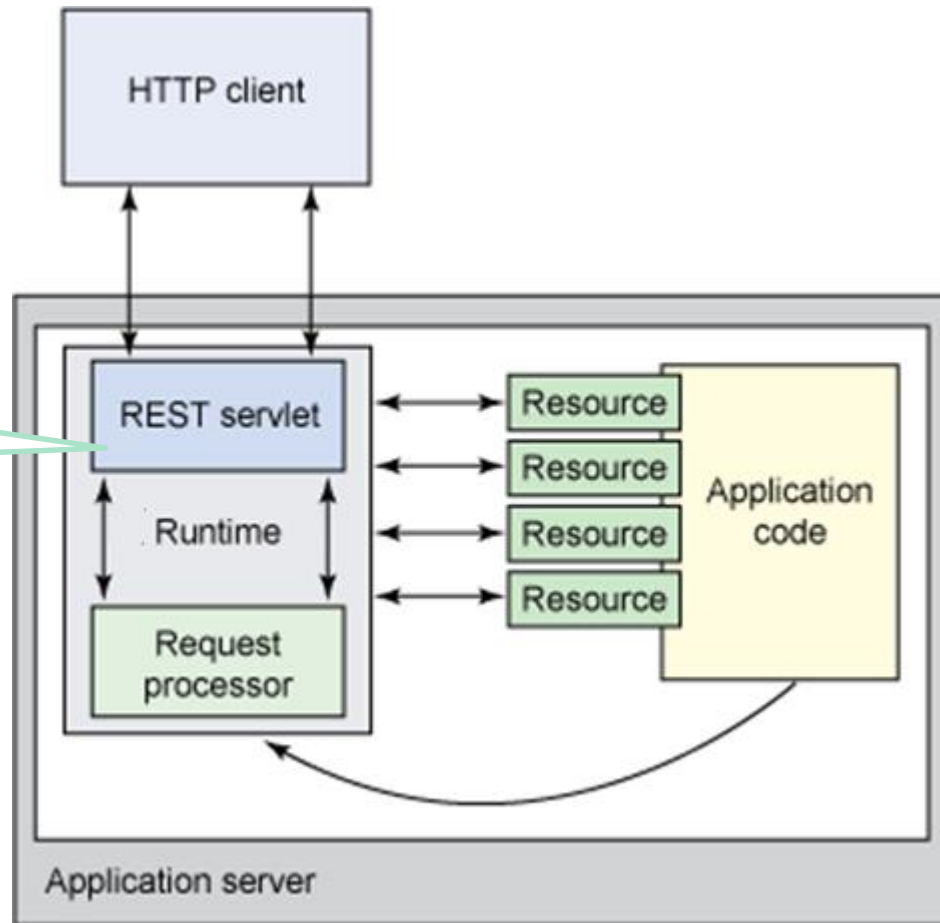


REST and Servlets : JAX RS

- All incoming requests will be processed by a **single standard servlet** called **Jersey**
 - This is the reference implementation of the REST engine
 - **jersey-bundle-1.18.jar** from <https://jersey.java.net>
- To build REST services, one uses annotations in Java classes. One does not create servlets.
 - Jersey will know how to launch the proper classes
 - Must import Java REST library : `javax.ws.rs.*`;



REST and Servlets



Unique
Standard
servlet



JAX RS & Jersey

The Jersey engine scans the directories for classes annotated with the JAX RS – defined annotations.

- Hence the Jersey library must be known by the server.
 - the **jersey-bundle-1.18.jar** must be installed in the common library.
 - Servlet: `com.sun.jersey.spi.container.servlet.ServletContainer`.
- The link between the URL and this servlet must be declared in the deployment descriptor (`web.xml`)

Standard web.xml file

Must be included in all JAX RS applications



```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
```

```
  <display-name>RESTWebApp</display-name> //displayed in the Tomcat manager's app list
```

```
  <servlet>
```

```
    <servlet-name>jersey-servlet</servlet-name>
```

```
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
```

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

```
  </servlet>
```

```
  <servlet-mapping>
```

```
    <servlet-name>jersey-servlet</servlet-name>
```

```
    <url-pattern>/*</url-pattern>
```

```
  </servlet-mapping>
```

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

The servlet will be launched as soon an URI segment is attached to the ServerDomainName /AppName/ URL prefix

The URL to reach the services will be **ServerDomainName / AppName / ResourceURL**



Path explained

- **ServerDomainName** : domain name of the Tomcat Server
- **AppName** : name of the **webapps** subdirectory.
- **ResourceURI**: resource specific URL segment following the REST URL-building principles

If one wishes the Tomcat Manager to display some specific name for the app, one must change the <display-name>

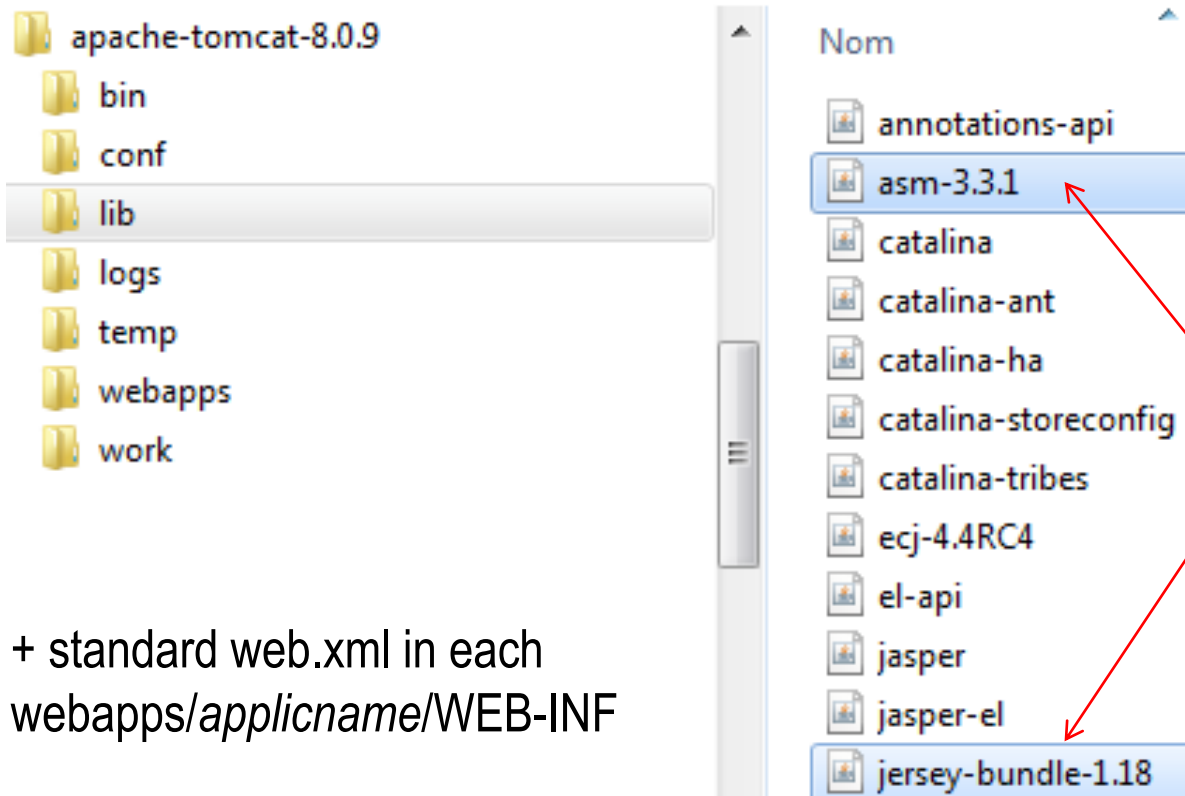


Installing JAX RS on the server

To process REST queries using JAX RS we must :

- Install 2 libraries (in apache-tomcat-8.0.9/**lib**)
 - Jersey: open source, production quality, JAX-RS (JSR 311) Reference Implementation for building RESTful Web services:
jersey-bundle-1.18.jar from <https://jersey.java.net>
 - In particular, this contains the **javax.ws.rs** library
 - Java bytecode manipulation and analysis framework :
asm-3.3.1.jar from <http://forge.ow2.org/projects/asm>

Installing required JAX RS libraries in Tomcat



Added libraries
that will be
available to all
the webapps

+ standard web.xml in each
webapps/applicname/WEB-INF



Getting access to the Tomcat console

Users configuration file (tomcat-users.xml)

To manage deployment, we must have the proper rights that are defined in tomcat-users.xml

```
<tomcat-users version="1.0" xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://tomcat.apache.org/xml">
<!-- NOTE: By default, no user is included in the "manager-gui" role required to operate the "/manager/html" web application. If you
wish to use this app, you must define such a user - the username and password are arbitrary. -->
<!-- NOTE: The sample user and role entries below are wrapped in a comment and thus are ignored when reading this file. Do not
forget to remove <!-- .. --> that surrounds them. -->
<!-- <role rolename="tomcat"/> <role rolename="role1"/> <user username="tomcat" password="tomcat" roles="tomcat"/> <user
username="both" password="tomcat" roles="tomcat,role1"/> <user username="role1" password="tomcat" roles="role1"/> -->
<user roles="manager-gui" password="manager" username="manager"/>
</tomcat-users>
```

Location: C:\Program Files\apache-tomcat-8.0.26\conf)




http://localhost:8080

Home Documentation Configuration Examples Wiki Mailing Lists Find Help

Apache Tomcat/8.0.9

The Apache Software Foundation
http://www.apache.org/

If you're seeing this, you've successfully installed Tomcat. Congratulations!


 Recommended Reading:
[Security Considerations HOW-TO](#)
[Manager Application HOW-TO](#)
[Clustering/Session Replication HOW-TO](#)

Server Status
Manager App
Host Manager

Launching the deployment manager

http://localhost:8080/manager/html/undeploy?path=/Atom

The Apache Software Foundation
http://www.apache.org/



Gestionnaire d'applications WEB Tomcat

Message:

Gestionnaire

Lister les applications	Aide HTML Gestionnaire	Aide Gestionnaire	Etat du serveur
---	--	-----------------------------------	---------------------------------

Applications

Chemin	Version	Nom d'affichage	Fonctionnelle	Sessions	Commandes
--------	---------	-----------------	---------------	----------	-----------



Building REST services



REST APIs

- Client-server interaction through HTTP requests only
- The HTTP verbs are the only actions available
 - There are NO procedure names in REST requests (unlike remote procedure calls)
- Resource and representation architecture should be carefully designed so that all operations could be performed with the HTTP verbs



REST Resource

In REST web services there are basically three kinds of resources.

1. One-off resource for important aspects of the application or unique resources such as the top level directory
2. A resource for every object exposed through the services.
3. The result of algorithms applied to the resources such as the result of a search engine



REST & HTTP

The interface is represented by the following HTTP verbs :
GET, PUT, DELETE, POST and PATCH

- GET : fetches a representation
NO change to the state of the resource
- DELETE: deletes a resource.
- PUT: changes the state of a resource
- POST : creates a resource
- PATCH: changes the state of a resource by sending partial information

JAX RS annotations



The corresponding classes must import `javax.ws.rs.*`;



Annotations

- Java classes representing REST services will use REST-specific annotations
 - These will be processed by the Jersey engine
- Once jersey is installed on the server there are no further development on the servlet side. All REST programming is made from annotations.



@Path (class level)

@Path(*URI-String*)

- **This is the KEY annotation**: it identifies a class that can service REST requests. It must be placed before the class declaration.
- *URI-String* is a String that denotes a URI relative to the **AppName** URI segment.
 - The minimal path is `/`: `@Path("/")`
 - At class level, `path @Path("/xyz")` is equivalent to `@Path("xyz")`
- Example:

```
@Path("students")  
public class StudentResource {...}
```



@Path (method level)

@Path(*URI-String*)

- Defines the URI path segment associated to a method. It is placed before the method declaration
- The final value of the URL segment to reach a method is the concatenation of the values of the @Path annotations at the method and class levels
 - There must be a "/" somewhere to separate the class' URI segment and the method's URI segment

```
@Path("students")
public class StudentResource {
.....
    @Path("/id2345")
    public String getStudent() {...}
```



Path example

- servlet url-pattern : **/***
- webapps subdirectory: **testApplication**
- **@Path("students")** (class level)
- **@Path("/id2345")** (method level)

Complete URL :

domainName/testApplication/students/id2345



Attaching method to HTTP verbs

@GET, @PUT, @POST, @DELETE

- Binds a method to the corresponding HTTP operation.
- Must be placed before the method declaration (like @Path())
- The parameters from the HTTP query can be *injected* into the java method to process the query. The contents returned by the query is the element returned by the method.

Exemple:

```
@GET  
public String helloWorld() {}
```

Type of the output contents

A green line originates from the word 'String' in the code snippet above and points to a green-bordered box containing the text 'Type of the output contents'.



Simplest @GET example

```
import javax.ws.rs.*;

@Path("/")
public class GetExample {

    @GET
    public String helloWorld() {return "hello world";}
```