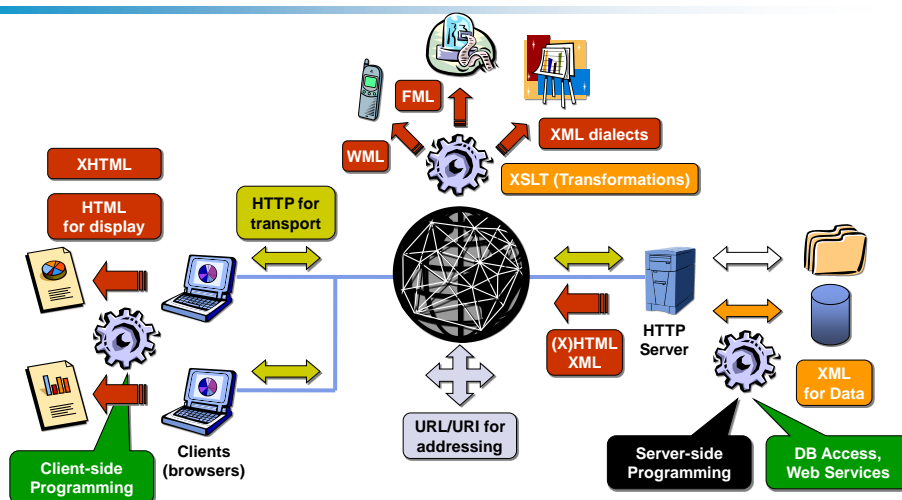




Lab session Google Application Engine - GAE

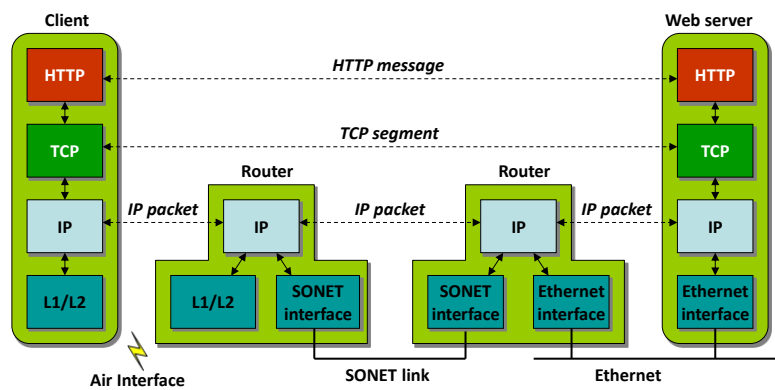
Navid Nikaein

Overall Interactions



Source: P. Michiardi, S. Crosta

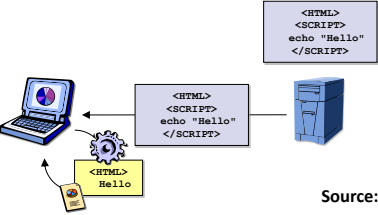
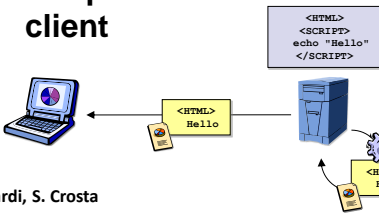
Protocol Interaction



Source: P. Michiardi

3

Client- vs Server-side Programming

Client-side	Server-side
<ul style="list-style-type: none">HTML and script sent to clientScript processed before displayScript visible to client	<ul style="list-style-type: none">Script processed before sending HTML to clientClient receives and displays processed HTMLScript hidden from client
	

Source: P. Michiardi, S. Crosta

4

IaaS vs PaaS vs SaaS



06/10/2015 -

- p 5

Platform as a service

- **Enabling communication and synchronization among devices using cloud**

- Centralized resources
- Separate clients
- Scalability
- ...



- **Different solutions exist**

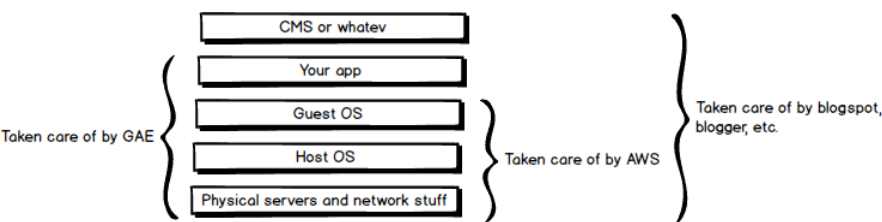


06/10/2015 -

- p 6

GCP/GAE and AWS

- IaaS vs PaaS



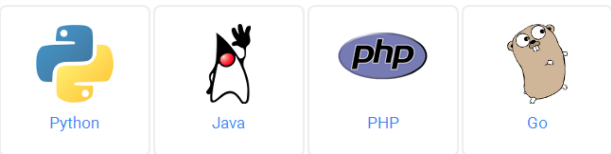
Read <http://notwastingtime.blogspot.fr/2010/05/google-app-engine-gae-versus-amazon-web.html>

06/10/2015 -

- p 7

GAE: a PaaS

- Google App Engine lets you build and run applications on Google's infrastructure.
- App Engine applications are easy to create, easy to maintain, and easy to scale as your traffic and data storage needs change.



- With App Engine, there are no servers for you to maintain. You simply upload your application and it's ready to go

06/10/2015 -

- p 8

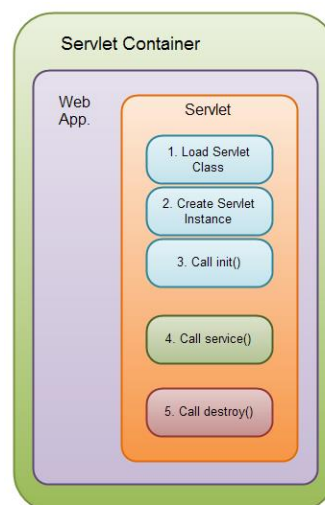
SERVLET

06/10/2015 -

- p 9

Servlet job and life cycle

- **Extend the server capabilities by means of request-response programming model**
- **Usage**
 - Read and store data and process request from the client
 - Send data / response back to the client
 - Build a dynamic content
 - Manage state information
- See http://en.wikipedia.org/wiki/Java_Servlet & http://en.wikibooks.org/wiki/J2EE_Programming/Servlet

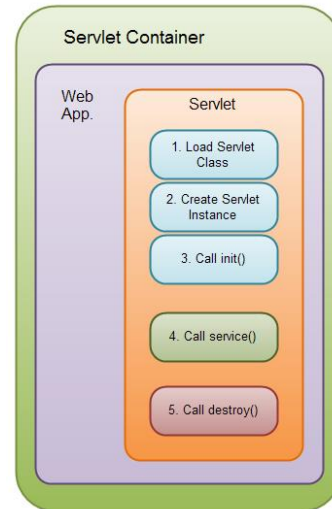


06/10/2015 -

- p 10

Servlet

- **Servlet container**
 - Manage the life cycle
 - Dispatch/mapping the URL
- **Servlet API**
 - Package : `javax.servlet.*`
- **Different data format may exist**
 - Html, xml, json



06/10/2015 -

- p 11

Servlet

- **Three methods to manages the servlet lifecycle**
 - `init()`, `service()`, `destroy()`
- **Service method of HttpServlet class dispatches requests to the methods**
 - `doGet()`, `doPost()`, `doPut()`, `doDelete()`, etc according to the HTTP request

06/10/2015 -

- p 12

Project structure and files

Project structure

- **src/**
- **war/**
 - WEB-INF/
 - appengine-generated/
 - classes/
 - lib/
 - appengine-web.xml
 - web.xml

Web.xml

```
<servlet>
<servlet-name>Helloworldgae</servlet-name>
<!-- class name -->
<servlet-
class>eurecom.fr.helloworldgae.Helloworldg
aeServlet</servlet-class>
<!-- class tree -->
</servlet>
<servlet-mapping>
<servlet-name>Helloworldgae</servlet-name>
<!-- class name -->
<url-pattern>/helloworldgae</url-pattern>
</servlet-mapping>
<!-- Class pattern in the URL-->
```

06/10/2015 -

- p 13

Decomposition of the URL

- **Prototype:**
 - [Protocol:][DNS]:[PORT]/[RootEntryDirectory]/[ServletName]
- **Example:**
 - <http://localhost:8888/hellomoon>
 - <http://localhost:8888/hellomoon?name=navid>

06/10/2015 -

- p 14

Servlet

- **Java class that runs on the server side that receives HTTP data and does a processing or more following the constraints of the HTTP protocol**

```
@WebServlet("/Hello")
public class Hello extends HttpServlet {
    // called at the reception of the http GET request
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("Hello World");
        } finally {
            out.close();
        }
    }
    // called at the reception of the http POST request
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/plain");
        String msg = "Hello, world";
        if (request.getParameter("name") != null) {
            msg += " from " + request.getParameter("name");
        }
        response.getWriter().println(msg);
    }
}
```

- p 15

Notes on HTTP GET and POST

GET	POST
<ul style="list-style-type: none">▪ GET requests can be cached▪ GET requests remain in the browser history▪ GET requests can be bookmarked▪ GET requests should never be used when dealing with sensitive data▪ GET requests have length restrictions▪ GET requests should be used only to retrieve data	<ul style="list-style-type: none">▪ POST requests are never cached▪ POST requests do not remain in the browser history▪ POST requests cannot be bookmarked▪ POST requests have no restrictions on data length
➔ Get requests are idempotent.	➔ POST requests are NOT idempotent.

06/10/2015 -

- p 16

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

06/10/2015 -

- p 17

HTTP Response

- **Generate error**

protected void **doXXX**(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

 response.setContentType("text/html;charset=UTF-8");

 PrintWriter out = response.getWriter();

 try {

 response.sendError(HttpServletResponse.SC_SERVICE_UNAVAILABLE, "the server is overloaded. Please try later.");

 } finally {

 out.close();

 }

}

- See http://en.wikibooks.org/wiki/J2EE_Programming/Servlet

06/10/2015 -

- p 18

Addressing: URI

- **URI: Uniform Resource Identifier**

- Universal naming mechanism for identifying resources on the Web
 - A resource is “anything to which we can attach identity” (Web page, image, anchor in page, database record, etc.)
- Web is an information space, URIs are handles
- **Unique** Web naming/addressing technology (HTML/HTTP: not the only data format/Web protocol)

- **URL: Uniform Resource Locator**

- Subset of URIs for some existing Internet protocols (http, ftp, mailto, etc.)
 - No longer used in specifications
-

©Navid Nikaein2014

19

Establishing Connectivity

- **HTTP traffic need a transport infrastructure (Client/server protocol)**

- clients open *connections* for sending HTTP traffic
- HTTP is a set of conventions regulating traffic over the transport

- **Any Internet host can establish Internet connections**

- IP is responsible for sending packets from and to computers
- TCP is responsible for handling connections between programs

- **Routing on the Internet is a complex process**

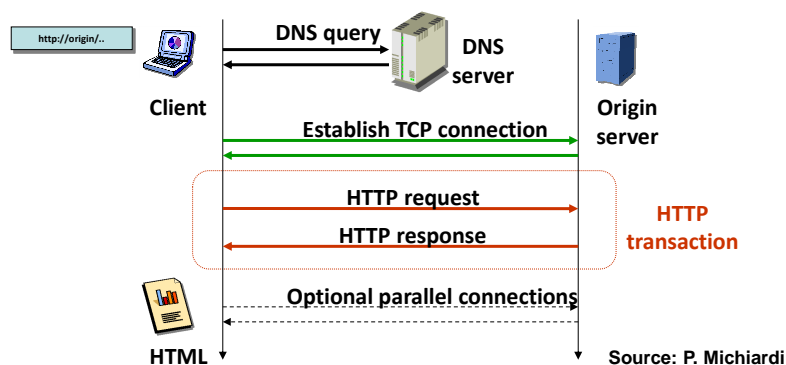
- backbone routers routinely exchange large routing tables
 - Internet traffic always is packet-based, each IP packet travels alone
 - IP is packet-based for robustness
-

©Navid Nikaein2014

20

HTTP in Context

■ Major steps in a "browser process"



©Navid Nikaein2014

21

HTTP Request

■ HTTP request is ASCII text (human-readable)

- Request line (method, URI, HTTP version)
- Header lines
- <CR>
- Optional payload

```
GET /foo.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)
Host: www.eurecom.fr
Connection: Keep-Alive
<CR>
```

GET /foo.html HTTP/1.1 — request "/foo.html" using HTTP version 1.1

Accept: — types of documents accepted by browser

Accept-Language: — preferred language is english

Accept-Encoding: — browser understands compressed documents

User-Agent: — identification of browser (real type is IE 5.01)

Host: — what the client thinks the server host is

Connection: — keep TCP connection open until explicitly disconnected

Source: P. Michiardi

©Navid Nikaein2014

22

HTTP Reply

- **Sample reply returned to client**

```
HTTP/1.1 200 OK
Date: Mon, 15 Jul 2002 08:49:00 GMT
Server: Apache/1.3.26 (Unix) PHP/4.2.1
Last-Modified: Wed, 12 Jun 2002 08:49:49 GMT
ETag: "2a-50ea-3d070b2d"
Accept-Ranges: bytes
Content-Length: 20714
Connection: close
Content-Type: text/html
<CR>
<html>...
```

HTTP/1.1 200 OK — document found (code 200); server is using HTTP 1.1
Date: — current date at the server
Server: — software run by the server
Last-Modified: — most recent modification of the document
ETag: — entity tag (unique identifier for the server resource, usable for caching)
Accept-Ranges: — server can return subsections of a document
Content-Length: — length of the body (which follows the header) in bytes
Connection: — the connection will close after the server's response
Content-Type: — what kind of document is included in the response
<html>... — document text (follows blank line)

Source: P. Michiardi

©Navid Nikaein2014

23

Client Methods

- **GET**
Retrieve a resource from the server (static file, or dynamically-generated data)
- **HEAD**
Get information about a resource (but not the actual resource)
- **POST**
Client provides some information to the server, e.g., through forms (may update the state of the server)
- **PUT**
Provide a new or replacement resource to put on the server
- **DELETE**
Remove a resource from the server
- **OPTIONS (HTTP 1.1)**
Request other options for an URI (methods understood by a server or allowed for a given URI)
- **TRACE (HTTP 1.1)**
Ask proxies to declare themselves in the headers (used for debugging)
- **CONNECT (HTTP 1.1)**
Used for HTTPS (secure HTTP) through a proxy

©Navid Nikaein2014

24

Server Response Codes

- **100 range: Informational**
 - 100 Continue — 101: Switching protocols
 - **200 range: Client request successful**
 - 200: OK — 201: Created — 204: No content ...
 - **300 range: Redirection**
 - 301: Moved permanently — 305: Use proxy ...
 - **400 range: Client request incomplete**
 - 400: Bad request — 401: Unauthorized ...
 - **500 range: Server errors**
 - 500: Internal error — 501: Not implemented ...
-

©Navid Nikaein2014

25

Lab Session Steps

- 1. Helloworldgae [10 minutes]**
 - 2. Hellomoongae [40 minutes]**
 1. Create a servlet
 2. JSP
 3. Publish and deploy the code
 - 3. AddressBook [1h30]**
 1. Database
 2. POST and GET
 3. List of contact and contact details
 4. Modify contact
 5. Output format in json
 - 4. Website [10 minutes]**
-

06/10/2015 -

- p 26

Technology

- **Google App Engine GAE**

- PaaS cloud computing platform
- Developing and hosting web applications in google data center



- **AWS**

- IaaS cloud computing infrastructure
- Developing and hosting web applications in Amazon web service



- **Apache Tomcat (formerly Jakarta Tomcat)**

- open source web server and servlet container
- Implements java servlet and javaServer pages



06/10/2015 -

- p 27

Java server pages - JSP

- **The JavaServer Pages is a technology for inserting dynamic content into a HTML or XML page using a Java servlet container**

- **Declaration:** `<%! int serverInstanceVariable = 1; %>`
- **Scriptlet :** `<% int localStackBasedVariable = 1; out.println(localStackBasedVariable); %>`
- **Expression:** `<%= "expanded inline data " + 1 %>`
- **Comment:** `<%-- This is my first JSP. --%>`
- **Directive :** `<%@ page import="java.util.*" %>`

06/10/2015 -

- p 28