Programmazione web

0. Introduction

Preliminaries

Difference between Web and Internet

What is the web

History of the web

Important references

Basic concepts

Protocol

Defines
1. **Format**
2. **Order**
- Of messages exchanged between two or more communicating entities
3. **Actions**
- Taken on the transmission and/or receipt of a message or other event

Port

> **Port**: *endpoint of communication in an operating system*

> **Binding**: *process associates its I/O channels via an Internet socket*

- With a transport protocol, a port number, and an IP address

Sockets have
- Protocol
- Local address
- Local port
- Remote address
- Remote port

RFC (Request for comment)

URI, URL, URN

> **Web resource**: *any identifiable thing, whether digital, physical, or abstract*

Uniform Resource
- **Identifier**: compact sequence of characters that identifies an abstract or physical resource
- **Locator**: subset of URI that identify resources via a representation of their primary access mechanism

    - e.g.: their network location
- **Name**: subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable
    - Serves as persistent, location-independent, resource identifier

Both URL and URN are **URI**

- **URN** identifies a **resource**
- **URL** provides a method for **finding** it
- **URN** can be associated to **many URLs**

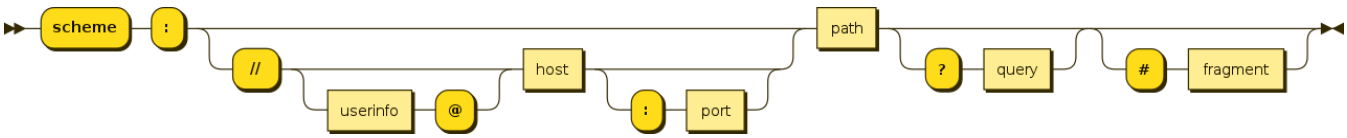
URI Schemes

- `http:`
- `https:`
- `ftp:`
- `mailto:<address>[?<header1>=<value1>[&<header2>=<value2>]]`
- `geo:<lat>,<lon>`
- `fax:<phone number>`
- `file:[//host]/path`
- `bitcoin:<address>[?[amount=<size>]…`
- `skype:<username|phonenumber>…`

URI structure

```
URI = scheme:[//authority]path[?query][#fragment]
authority = [userinfo@]host[:port]
```

URL structure



MIME types

Multipurpose Internet Mail Extensions
MEDIA_TYPE/SUBTYPE

- text -> text/plain, text/html, text/richtext …
- image -> image/jpeg, image/png, image/svg+xml…
- audio -> audio/basic, audio/ogg, audio/x-wav…
- video -> video/mp4, video/ogg…
- application -> application/x-apple-diskimage…

Web architecture

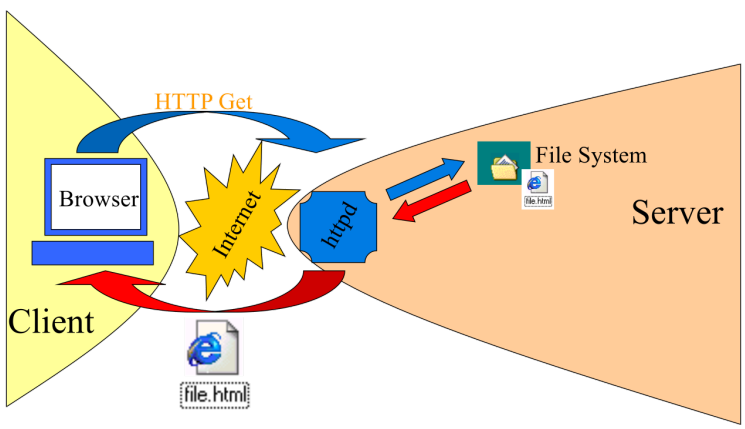
Client and Server

> **Server**: *a machine that opens a SocketServer connection, and waits for incoming calls (in order to provide a service)*

> **Client**: *a machine the starts a connection (opening a Socket to the server) and requests a service*

Server and client are **software** roles, not hardware concept

Static pages



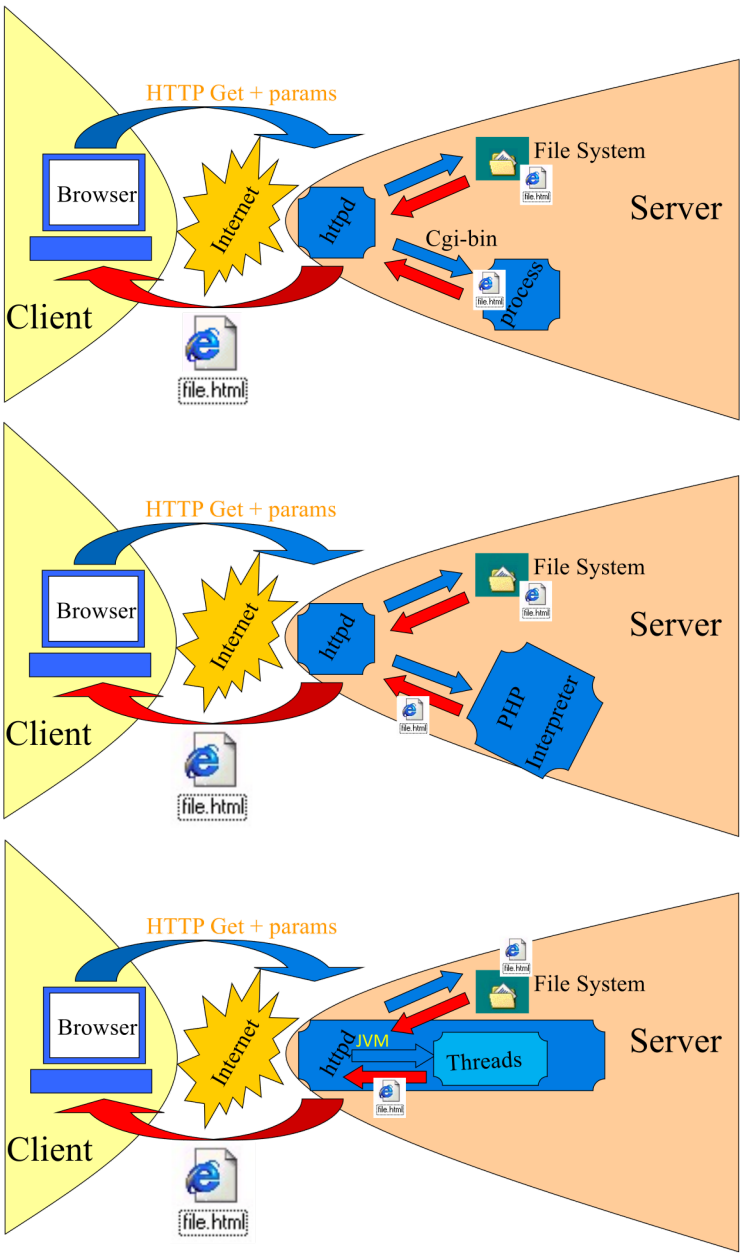
Initial idea

- Get (static) interlinked documents
- The web programmer writes collections of HTML pages

Model

- **Mapping 1:1** between URLs and (static) resources
- The Web server is nothing but a **retriever of the content** associated to an URL

Dynamic pages



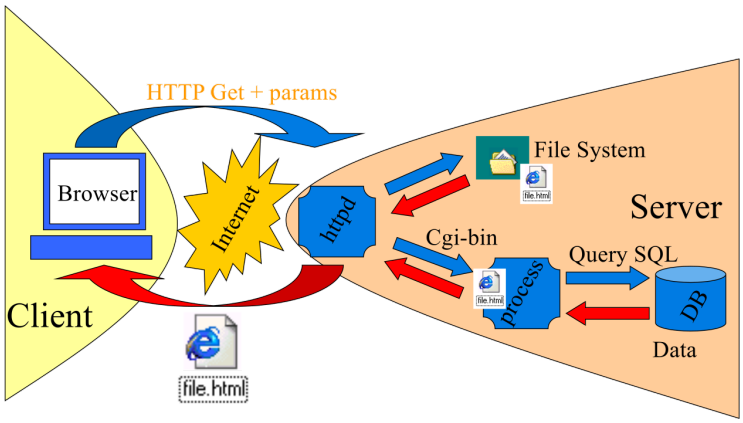
Evolution

- **Dynamically create (interlinked) documents**
- The web programmer also writes **programs**, using the programming languages

Model

- Certain **URLs** are associated with **actions** and carry **parameters** for the action
- **Web server**
  - Understands that certain **URLs** are **dynamic**
  - Parses the **parameters**
  - Starts a **process** (or thread) corresponding to the desired action
  - Obtains from the process some **data** (**for human consumption**)
  - Passed the data to the **client**

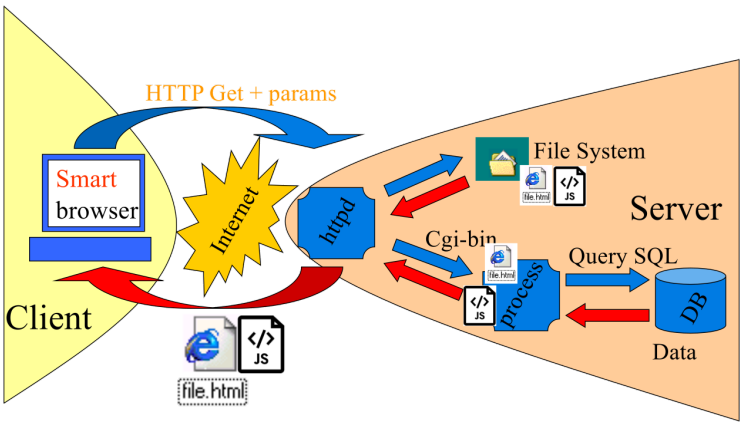
Dynamic pages with DB



Evolution

- Dynamically create (interlinked) **documents** interacting with **persistent data storage**
- The programs also need to interact with a (legacy) database

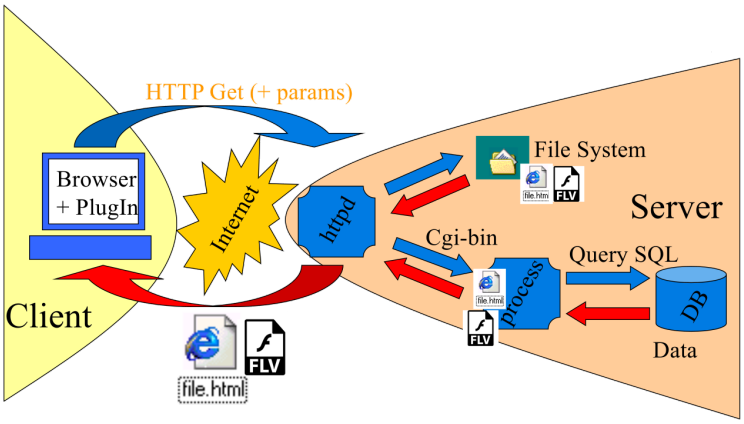
Smart browsers



Evolution

- **Execute code also on client**
- The web programmer also writes **programs** which **run on the browser**

Plugins

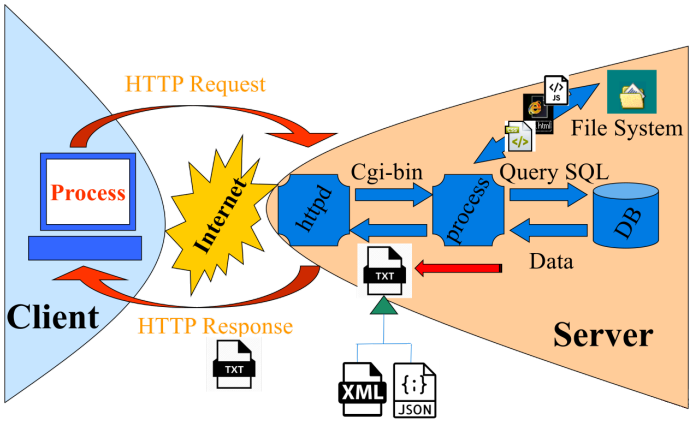


Evolution

- Augment browser with an **ad-hoc engine** to be able to execute a (proprietary) language

Web services

Web service: *software system designed to support interoperable machine-to-machine interaction over a network*



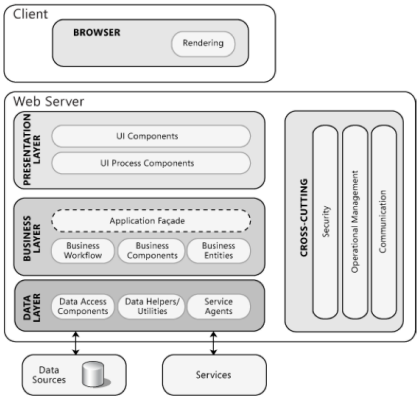
Model

- Same as dynamic, but **for machine consumption**

Ajax processing and single page applications

Same as web services, but with a **smart browser** with **data injection**, instead of client-side process

Logical Web Architecture



1. XML

Markup

Markup languages

- System for annotating a document (metadata)
- When the document is processed for display, the markup language is not shown, and is only used to format the text

Example of Markup Language

Types of Markup languages

- Presentational markup
- Procedural markup
- Descriptive markup

The roots: SGML

ISO standard, metalanguage that is used to define other languages.

SGML: the three parts

An SGML document is the combination of three parts

- **Content** of the document
  - The part that the author wants to expose to the client
- **Grammar** (DTD – Data Type Definition)

    - Defines the accepted syntax
- **Stylesheet**
    - Establishes how the content that conforms to the grammar is to be rendered on the output device

HTML vs. SGML

- **HTML** implements some of the concepts derived from **SGML**
- Some parts are **hard-coded** into the **browser** software
    - **DTD**
    - Basic **Style Sheet**
        - Can be redefined via **CSS – cascading style sheet**

Basic XML elements

Definition

> **eXtensible Markup Language**: *WWW consortium standard that permits to create custom tags*

- **Group of related technologies** that continually adds new members
- Used for **data formatting**

Applications

- **Semantic Web**
    - RDF (Resource Description Framework), OWL, Topic Maps
- **Web Services**
    - SOAP, UDDI, WSDL, XML-RPC
- **Configuration files**

Element, tag, content, attribute

```
<tag attr="value">content</tag>
```

> **Tag**: *case sensitive sequence of characters that begins with* `<` *and ends with* `>`

- The **start tag** may contain optional **attributes**
- Every tag must be **closed** with an **end tag**, which begins with `</`

> **Element**: *sequence of characters that begins with a start tag and ends with an end tag and includes everything in between*

> **Content**: *characters in between the tags*

Well-formed documents

- **XML documents**
    - Must be **well-formed**
    - Are not required to be valid
- **HTML documents**
    - Are not required to be well-formed

Requirements for well-formed XML

- XML is **case sensitive**
- **Start and end tags** are required
    - Elements that can contain **character data** must have both **start and end tags**
    - **Empty** elements have a different requirement
- Elements must **nested properly**
    - Overlapping is not admitted
    - Must **close** the **most inner** tag first
- **Empty elements**
    - Two syntaxes
        - `<book></book>`
        - `<book/>` (preferable)
    - Can contain **attributes**
- **No markup characters** are allowed

    - Text data must not have characters (entities) in the text data: `< > " ' &`
    - Must be **escaped**
- All **attribute values** must be **in quotes**
    - Single quotes can contains a double quotes
    - Double quotes can contain apostrophes

Tree structure

XML document

- Must have a **root tag**
- Is an information unit that can be seen in two ways
    - **Linear sequence of characters** that contain **characters data** and **markup**
    - **Abstract data structure** that is a **tree** of nodes

Tag definition

- XML provide a **grammar** to
    - Define **tags**
    - Define **rules** for the tags
        - Allowed **attributes**
        - **Containment** rules
- The grammar is defined in a file
    - **DTD** file
    - **XML-Schema** file
    - Or is **not defined** at all

Additional elements

- XML document can contain
    - **Processing Instructions** (PI): `<? ... ?>`
    - **Comments** `<!-- ... -->`
- When the document is analyzed
    - **Character data**
        - Within comments or PIs are **ignored**
    - **Content** of
        - **Comments** is **ignored**
        - **PIs** is passed on to **applications**

CDATA sections

CDATA section: *section used to escape character strings*

- Used with strings that may contain elements not intended to be examined by the XML engine
- e.g.: special chars or tags

PCDATA: *element content that are going to be parsed*

When a XML document is analyzed, character data within a `CDATA` section are **not parsed**, by they remain as part of the **element content**

```
<java>
<![CDATA[
    if (arr[indexArr[4] ]>3) System.out.println(“<HTML>”);
]]> <!-- Avoid having `]]>` in CDATA section -->
</java>
```

Logical structure of an XML document

- Optional
    - **XML declaration** (**prolog**)
        - If present must be the first element
        - ```
          <?xml version='1.0' encoding='utf-8'>
          ```
    - **DTD** declaration
    - **Comments** and **Processing Instructions**
- Mandatory
    - **Root element’s start tag**
    - All other **elements**, **comments** and **PIs**
    - **Root element’s closing tag**
          ```

Namespaces

Avoid tag conflicts by declaring a namespace as an attribute of the root element

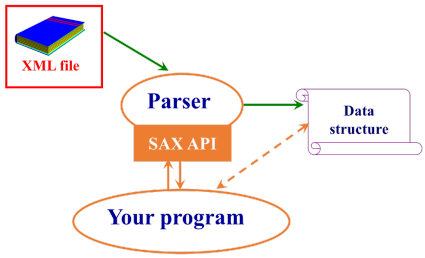
```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Parser

> **Parser**: *software tool that preprocesses an XML document in some fashion, handing the results over to an application program*

Primary purpose: do most of the hard work up front and to provide the application program with the XML information in a form that is easier to work with.

- Input: **XML file**
- Output:
  - **Data structure**
  - **Error** (if not well-formed)



Tree-based vs Event-based API

- **Tree-based API**
  - **Compiles** an XML document into an **internal tree structure**
  - **Application** programs can **navigate** the tree to achieve its objective
  - The **Document Object Model (DOM)** working group at the W3C developed a **standard tree-based API** for XML
- **Event-based API**
  - **Reports parsing events** to the application using **callbacks**
    - e.g.: start and end of elements
  - **Applications** implement and register **event handlers** for the different events
  - Code in event handlers is designed to achieve the **objective** of application

DTD

Definition

> **DTD**: *one or more files (used together) which contain a formal definition of a particular type of document*

- This sets out what **names** can be used for **elements**, **where** they may occur, and **how** they all fit together
- It's a **formal language** which lets processors automatically **parse** a document and **identify** where every **element** comes and how they relate to each other
  - So that stylesheets, navigators, browsers, search engines, databases, printing routines, and other applications can be used
- DTD contains **metadata** relative to a **collection of XML docs**

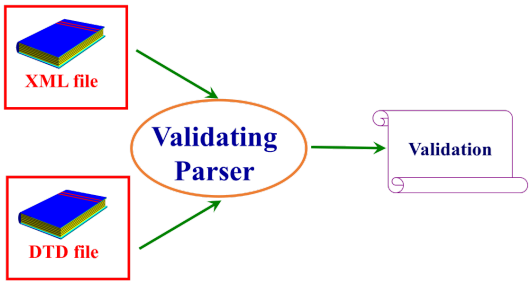
Valid documents

- An XML document is **valid** if it conforms to an existing grammar in every respect
  - e.g.: unless the DTD allows an element with the name "color", an XML document containing an element with that name is not valid according to that DTD (but it might be valid according to some other DTD)
- An **invalid** XML document can be a perfectly good and useful XML document
- A **non well-formed** document cannot be valid, and is not an XML document

Validation against a DTD can often be very **useful**, but is **not required**

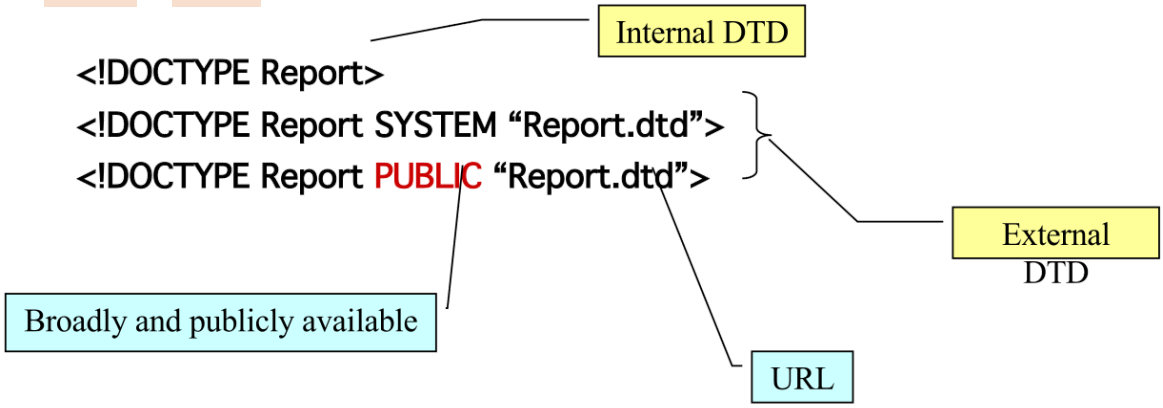
- Because XML does **not require a DTD**, in general, an XML processor cannot require validation of the document

Constraing & validating XML: DTD file



DTDs locations

DTD can be **external** or **internal** to a document



DTD Markup

ELEMENT

```
<!ELEMENT name contentModel>
```

```
<!ELEMENT book (preface?,chapter+,index)>
<!ELEMENT preface(paragraph+)>
<!ELEMENT paragraph (#PCDATA)>

<!ELEMENT chapter (title,paragraph+,reference*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT reference (#PCDATA|URL)>
<!ELEMENT URL (#PCDATA)>

<!ELEMENT index(number,title,page_number)>
<!ELEMENT number(#PCDATA)>
<!ELEMENT page_number(#PCDATA)>
```

- `?` Zero or one
- `+` One or more
- `*` Zero or more
- `,` Sequence
- `|` OR

ATTLIST

```
<!ATTLIST elementName attributeName type default>
```

```
<!ELEMENT Product (#PCDATA)>
<!ATTLIST Product
    Name CDATA #IMPLIED
    Rev CDATA #FIXED "1.0"
    Code CDATA #REQUIRED
    Pid ID #REQUIRED
    Series IDREF
    Status (InProduction|Obsolete) "InProduction"
>
```

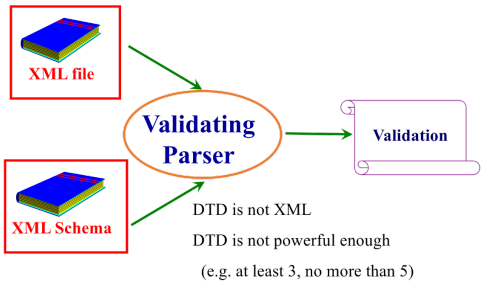
- TYPES
  - `CDATA` Character data
  - `ID` Unique key
  - `IDREF` Foreign Key
  - `(...|...)` Enumeration
- DEFAULT
  - `#IMPLIED` Optional, no default

- `#FIXED` Optional, default supplied
  - If present must match default
- `#REQUIRED` Must be provided

Problem

- Problem: DTDs are **not written in XML**
- Solution: another XML-based standard (**XML Schema**)

Constraing & validating XML: XML Schema



XML Schema

```
<?xml version="1.0"?>
<schema>
  <element name="complete_name" type="complete_name_type"/>
  <complexType name="complete_name_type">
    <sequence>
      <element name="nome" type="string"/>
      <element name="cognome" type="string"/>
    </sequence>
  </complexType>
</schema>
```

Defines **validity rules** for tags

```
<complete_name>
  <nome>Marta</nome>
  <cognome>Bassino</cognome>
</complete_name>
```

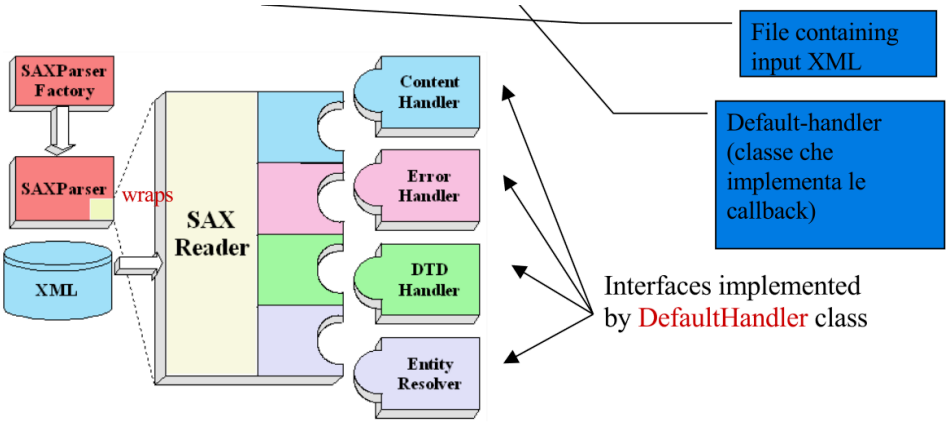
Applications

In practice

- **Navigate** its data structure
  - DOM, JDOM
  - JAXP
  - XPath
  - SAX
- **Query** XML data
  - XQuery
- **Transform** XML data
  - XSLT
- Use XML for **Single Page Web Applications**
  - AJAX
- Use XML in **configuration files**

SAX: architecture

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true); //optional - default is non-validating
SAXParser saxParser = factory.newSAXParser();
saxParser.parse(File f, DefaultHandler-subclass h)
```

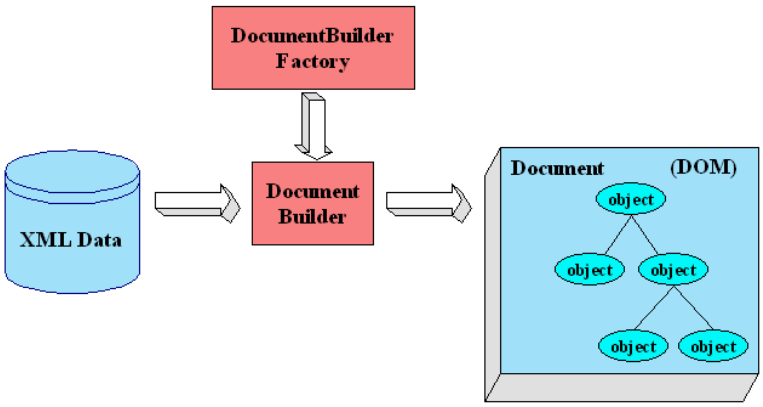


SAX: callbacks

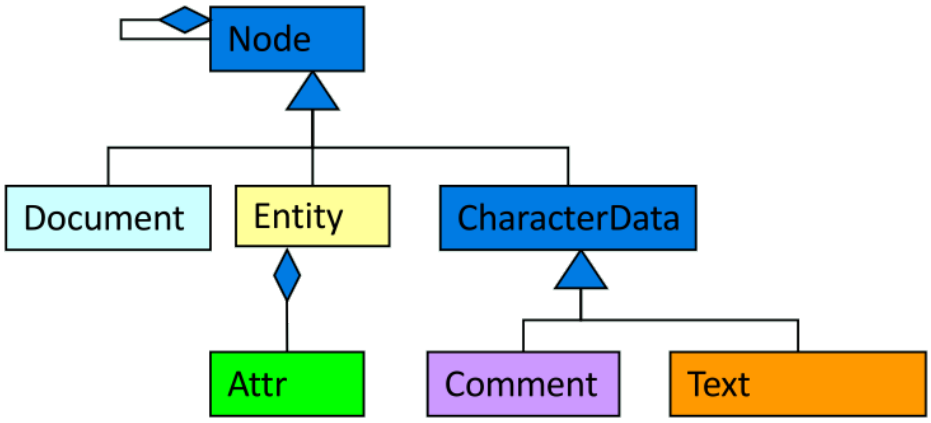
```
// ContentHandler methods
void characters(char[] ch, int start, int length)
void startDocument()
void startElement(String name, AttributeList attrs)
void endElement(String name)
void endDocument()
void processingInstruction(String target, String data)
```

JAXP: DOM

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setValidating(true); // optional - default is non-validating
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(file);
```

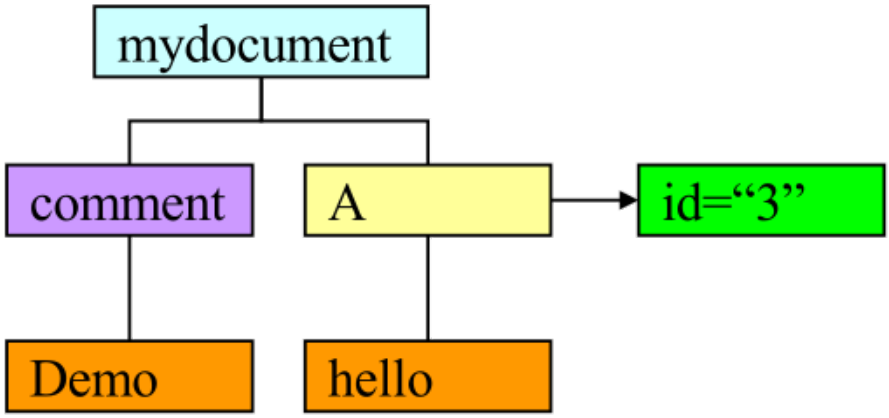


Node hierarchy



Example

```
<!-- Demo -->
<A id="3">hello</A>
```



JAXP: example

```
public int getElementCount(Node node) {
    if (null == node) return 0;
    int sum = 0;
    boolean isElement = (node.getNodeType() == Node.ELEMENT_NODE);
    if (isElement) sum = 1;
    NodeList children = node.getChildNodes();
    if (null == children) return sum;
    for (int i = 0; i < children.getLength(); i++) {
        sum += getElementCount(children.item(i));  // recursive
    }
    return sum;
}
```

Use **DOM methods** to count elements

- For each subtree
- If the root is an Element, set sum to 1, else to 0
- Add element count of all children of the root to sum

XPath

Syntax for defining parts of an XML document

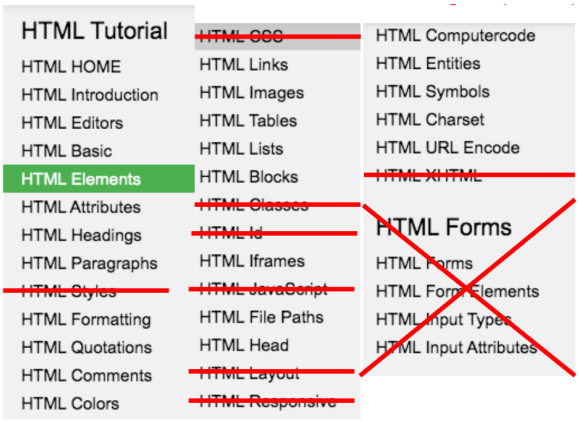
- Uses **path expressions to navigate** in XML documents
- Contains a library of **standard functions**
- Major element in XSLT
- W3C Standard

```
// prepare the XPath expression
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
XPathExpression expr = xpath.compile("//book[author='Dante Alighieri']/title/text()");
// evaluate the expression on a Node
Object result = expr.evaluate(doc, XPathConstants.NODESET);
// examine the results
NodeList nodes = (NodeList) result;
for (int i = 0; i < nodes.getLength(); i++) {
    System.out.println(nodes.item(i).getNodeValue());
}
```

2. HTML

Basics

What to know



Introduction

Readings

What is HTML

Hypertext

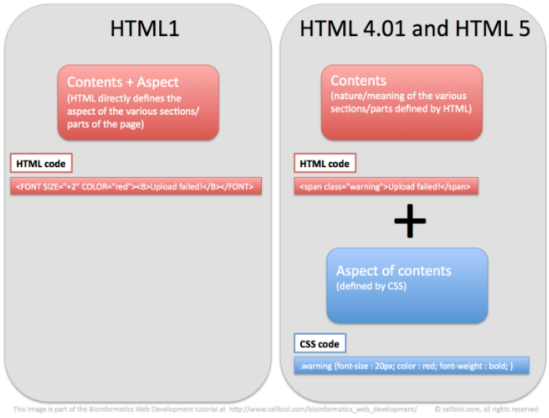
Media

Versions

The web

Structure

Content/Aspect separation



Structure of an HTML doc

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My test page</title>
  </head>
  <body>
    <p>This is my page</p>
  </body>
</html>
```

- **head**: contains information about the document (metadata)
  - **Title of the page** (which appears at the top of the browser window)
  - **Meta tags**: used to describe the content (used by Search engines)
  - **Statements or references** to JS and CSS
- **body**: contains the actual content of the document
  - This is the part that will be displayed in the browser window

DocType declaration

```
<!-- Complete -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!-- HTML 5.0 -->
<!DOCTYPE html>
```

HTML Tags

- All HTML tags are made up of a **tag name**
  - Sometimes followed by an optional **list of attributes**
- Nothing within the brackets will be displayed by the browser
  - Unless HTML is incorrect and browser interprets tags as part of the content

Tag: *keyword enclosed by angle brackets*

- Opening and closing tags use the **same command**
- Except the **closing tag** contains an additional forward slash `/`

Void elements

Ending slash

- **HTML5**: the slash is **optional**
- **HTML4**: the slash is technically **invalid**
  - It's accepted by W3C's HTML validator
- **XHTML**: The slash is **required**

Nesting

Attributes

Attributes: *name-value pair which refine or extend tag's functions*

- You can add multiple attributes within a single tag
- Some attributes only have name (no associated value)
- Values are limited to 1024 characters in length
- Attributes values are within single or double quotes

Formatting

HTML content model

Two basic **content models**

- `%inline;` character level elements and text strings
- `%block;` block-like elements

Basic formatting

Tags

- Headings: `<h1>` … `<h6>`
- Emphasis: `<em>` `<strong>` `<b>` `<i>`
- Subscript, superscript: `<sub>` `<sup>`
- Lists: `<ol>` `<ul>` `<li>`

Blocks

- Line breaks: `<br/>`
- Paragraphs and blocks: `<p>` `<div>` `<span>`
  - `<div>`: container of data (with graphic separation)
    - Line-break before and after it
  - `<span>`: logical, in-line grouping without graphic evidence
    - Mostly used for a small chunk of HTML inside a line
  - `<p>`: paragraph of content (with graphic separation)

- It cannot contain block-level elements but can contain `<span>`

> *Don't mark up your document based on how it should look,*
> *mark it up based on what it is*

Advanced formatting

- Quotation: `<blockquotes>` `<q>`
- Description lists: `<dl>` `<dt>`
- Details / Summary: `<detail>` `<summary>`
- Abbreviation: `<abbr>`
- Address: `<address>`
- Computer code: `<code>` `<pre>` `<var>` `<kbd>` `<samp>`
- Time: `<time>`

Tables

- `<table>` defines the tabella
- `<th>`, `<tr>` define the rows
- `<td>` defines every single cell
- `<caption>` defines the caption

The following attributes define the cell span over columns and rows

- `colspan="n"`
- `rowspan="n"`

Images

```
<img src="url" alt="text">
```

```
<figure>
    <img src="ninja_guy.jpg" alt="A Standing Ninja Guy.">
    <figcaption>Cool Ninja Guy</figcaption>
</figure>
```

File paths

`<img src="http://a.b/img/pic.jpg">`	Absolute Path
`<img src="pic.jpg">`	`pic.jpg` is located in the same server, same folder as the current page
`<img src="img/pic.jpg">`	`pic.jpg` is located in the images folder in the current folder
`<img src="../picture.jpg">`	`pic.jpg` is located in the folder one level up from the current folder

Hyperlinks

```
<a href="url">link text</a>
```

URLs with other protocols

```
<a href="mailto:marco.ronchetti@unitn.it">write to me!</a>
```

```
<base href="https://www.mydomain.com/" target="_blank">
```

Inner target

```
<a href="#pippo" >GO TO PIPPO!</a>
...
<a name="pippo"/>
```

Iframes - comments

```
<iframe src="demo_iframe.htm"></iframe>

<iframe height="300px" width="100%"
        src="demo_iframe.htm" name="iframe_a">
</iframe>

<p><a href="https://www.w3schools.com" target="iframe_a">
W3Schools.com
</a></p>
```

Encoding and metadata

Character Encoding

Entities

Result	Description	Entity Name	Entity Number
	non-breaking space		
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;
"	double quotation mark	&quot;	&#34;
'	single quotation mark (apostrophe)	&apos;	&#39;
¢	cent	&cent;	&#162;
£	pound	&pound;	&#163;
¥	yen	&yen;	&#165;
€	euro	&euro;	&#8364;
©	copyright	&copy;	&#169;
®	registered trademark	&reg;	&#174;

URL Encoding

Character	From Windows-1252	From UTF-8
space	%20	%20
!	%21	%21
"	%22	%22
#	%23	%23
$	%24	%24
%	%25	%25
&	%26	%26
'	%27	%27
(	%28	%28
)	%29	%29

Metadata (head)

```
<meta charset="UTF-8">
<meta name="description" content="Free Web tutorials">
<meta name="keywords" content="HTML,CSS,XML,JavaScript">
<meta name="author" content="John Doe">
<meta http-equiv="refresh" content="30">
<meta charset="utf-8">
<meta name="viewport" content="width=device-width,
                               initial-scale=1.0">
```

3. HTTP

Basics and connections

Overview

> **HTTP**: hypertext transfer protocol

- Web's application **layer protocol**: **client/server model**
  - **Client**: browser that requests, receives and displays Web objects
  - **Server**: sends (using HTTP protocol) objects in response to requests
- Uses **TCP**
  1. **Client initiates TCP** connection (creates socket) to server, port 80
  2. **Server accepts TCP** connection from client
  3. **HTTP messages** (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
  4. **TCP connection closed**

States

- HTTP is **stateless**
  - Server maintains no information about past client requests
  - Protocols that maintain state are complex
    - Past history (state) must be maintained
    - If server/client crashes, their views of state may be inconsistent, must be reconciled
- FTP is **stateful**
  - Server maintains information about past client requests
    - You can issue a `cd` command to move into a (remote) directory
    - The next commands will be executed with reference to that directory

Making a simple HTTP request using Telnet

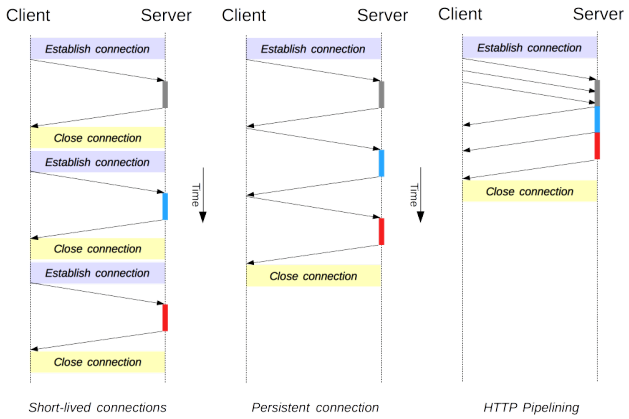
- Open a **TCP connection** to a host
  - Can borrow telnet protocol, by pointing it at the default HTTP port (80)
  - `$> telnet www.google.com 80`
- Ask for a resource using a minimal **request** syntax
  - `$> GET / HTTP/1.1`
  - `Host: www.google.com`
  -
- A Host header is required for HTTP 1.1 connections

Web and HTTP

- Web page consists of objects
  - Text file, JPEG image, Flash objects, audio file, …
- A web page contains base HTML-file which includes several **referenced objects**
- Each object is addressable by a URL
  - **www.somecompany.com/someDept/pic.gif**
  - **www.somecompany.com** is the host name
  - [someDept/pic.gif][] is the path to the object

HTTP connections

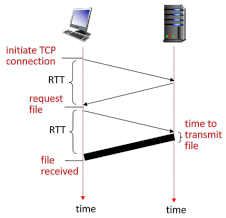
- **Non-persistent HTTP**
  - At most one object sent over TCP connection
  - Connection then closed
  - Downloading multiple objects required multiple connections
- **Persistent HTTP**
  - Multiple objects can be sent over single client-server TCP connection



Non-persistent HTTP

> **Round Trip Time (RTT):** *time for a small packet to travel from client to server and back*

- HTTP **response time**
  - **One RTT to initiate TCP** connection
  - **One RTT for HTTP request** and first few bytes of HTTP response to return
  - **File transmission** time
- **= 2 RTT + file transmission time**



Connections comparison

- **Non-persistent** HTTP issues
  - Requires **2 RTTs per object**
  - **OS overhead** for each TCP connection
  - Browsers often open **parallel TCP connections** to fetch referenced objects
- **Persistent** HTTP advantages
  - Server leaves **connection open** after sending response
  - **Subsequent HTTP messages** between same client/server sent over open connection
  - Client sends **requests as soon as** it encounters a referenced **object**
  - As little as **one RTT** for all the referenced objects

HTTP versions

HTTP 1.1

HTTP/2

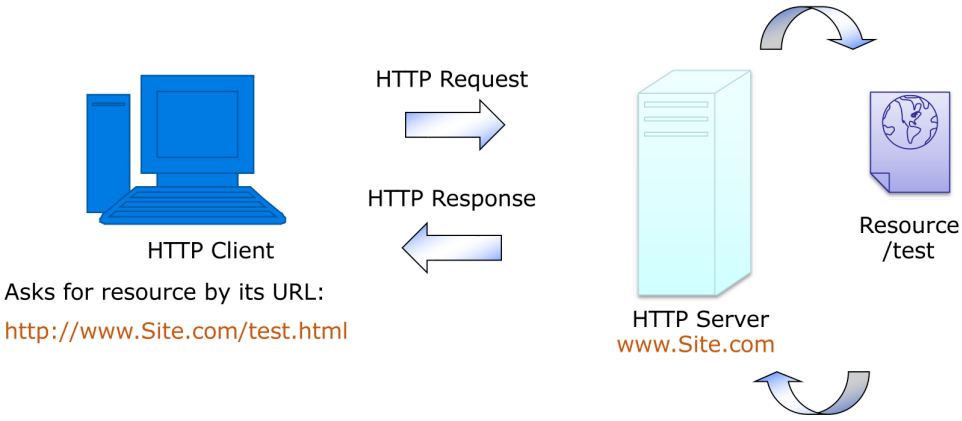
HTTP/3

Request

Request-response cycle

HTTP servers turn **URLs into resources** through a request-response cycle

- Two types of **HTTP messages**
  - **Request**
  - **Response**

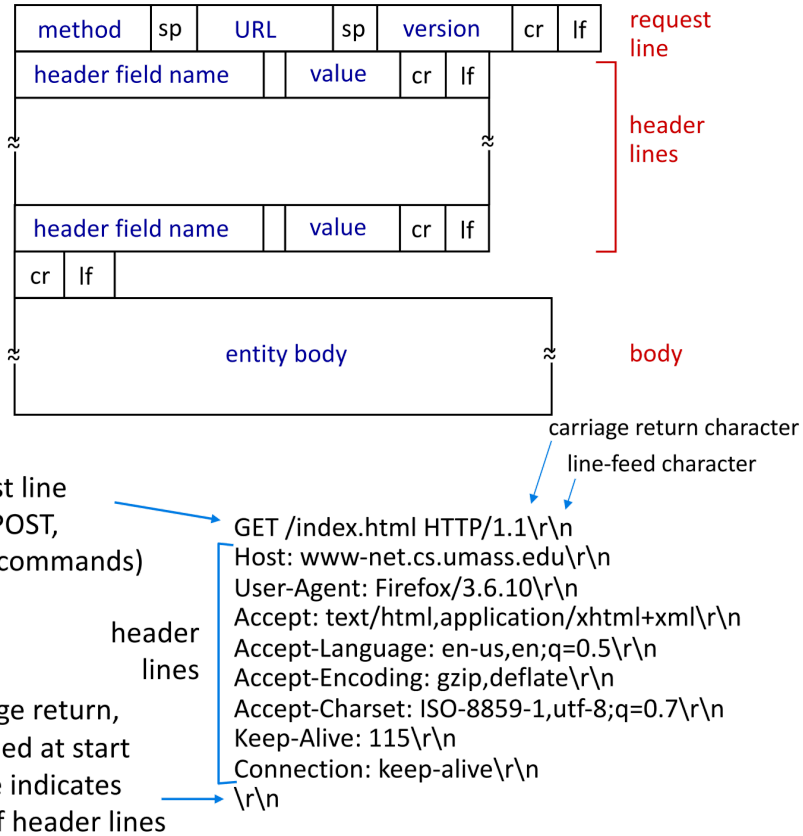


Request/response messages: general format

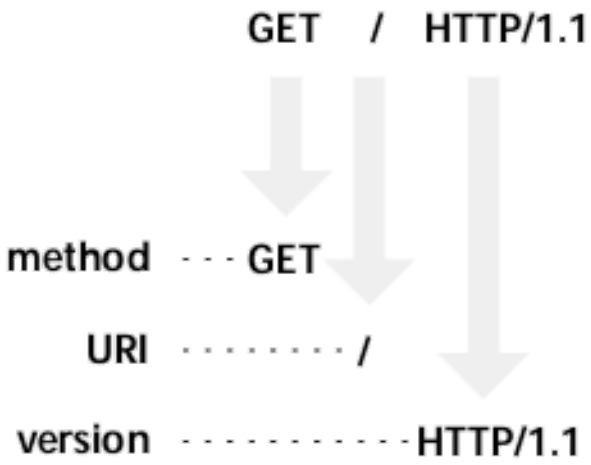
- **Start Line**
  - **Request** line for requests
  - **Status** line for responses
  - Followed by a
- **Message headers** (zero or more)
  - `field-name: [field-value]`
- **Empty line**
  - Two marks the end of the headers
- **Message body** (optional)
  - If there is a **payload**
  - All or part of the **entity body** or **entity**

Request messages: general format

ASCII, human-readable format




```
GET / HTTP/1.1
Host: www.iugaza.edu.ps
Connection: close
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Accept-Encoding: gzip
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7
Cache-Control: no-cache
Accept-Language: de,en;q=0.7,en-us;q=0.3
Referer: http://web-sniffer.net/
<CRLF>
```



Method types

- HTTP/1.0
  - **`GET`**, **`POST`**
    - Asks server to **obtain an object**
  - **`HEAD`**
    - Asks server to **leave requested object out of response**
- HTTP/1.1
  - **`GET`**, **`POST`**, **`HEAD`**
  - **`PUT`**
    - **Uploads file** in entity body to path specified in URL field
  - **`DELETE`**
    - **Deletes file** specified in the URL field

Uploading form input

- **`POST` method**
  - Web page often includes **form input**
  - Input is **uploaded** to server **in entity body**
- **URL method**
  - Uses **`GET`** method
  - Input is **uploaded in URL field** of request line:
    - www.somesite.com/animalsearch?monkeys&banana

Request Methods

- **`GET`**
  - By far **most common** method
  - **Retrieves a resource** from the server
  - Supports passing of **query string arguments**
- **`HEAD`**
  - **Retrieves only the headers** associated with a resource
    - Not the entity itself
  - Highly useful for **protocol analysis**, **diagnostics**
- **`POST`**
  - Allows **passing of data in entity** rather than URL
  - Can transmit of far **larger arguments** that GET
  - Arguments **not displayed on the URL**
- **`OPTIONS`**
  - Shows **methods available** for use
    - On the **resource** (if given a path)
    - On the **host** (if given a *)
- **`TRACE`**
  - **Diagnostic method** for assessing the impact of **proxies** along the request-response chain
- **`CONNECT`**
  - A common extension method for **Tunneling other protocols** through HTTP
- **`PUT`**, **`DELETE`**

- Used in **HTTP publishing**
  - e.g.: WebDav

Idempotent methods

> **Idempotent methods**: *an HTTP method is idempotent if an identical request can be made once or several times in a row with the same effect while leaving the server in the same state*

- Should **not** have any **side-effects**
  - Except for keeping statistics
- Implemented correctly, the **`GET`**, **`HEAD`**, **`PUT`** and **`DELETE`** method are idempotent, but not the `POST` method

Safe methods

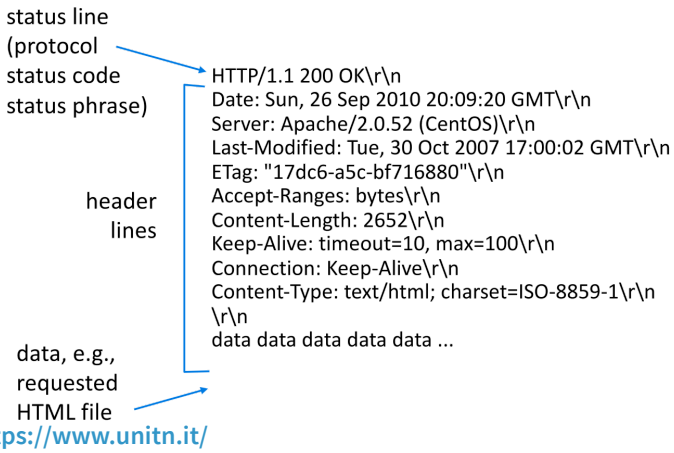
> **Safe method**: *an HTTP method is safe if it doesn't alter the state of the server*

- i.e.: it leads to a **read-only** operation
- **`GET`**, **`HEAD`**, **`OPTIONS`**
- All safe methods are also **idempotent**
  - Not all idempotent methods are safe

Safe, unsafe and idempotent

Response

Response messages: general format



https://websniffer.cc/?url=https://www.unitn.it/

Status Line

Three major parts

- **HTTP version** (~ third part of Request Line)
- **Status code**
  - 5 groups of 3 digit indicating the result of the attempt to satisfy the request
    1. Informational
    2. Success
    3. Redirection
    4. Client error
    5. Server error
- **Reason phrase**, followed by
  - Short textual description of the status code

Headers

HTTP headers

Headers come in four major types, for **requests**, for **responses** or for **both**

- **General headers**
  - Provide **info about messages** of **both kinds**

- **Request headers**
  - Provide **request-specific** info
- **Response headers**
  - Provide **response-specific** info
- **Entity headers**
  - Provide info about **request and response entities**
- **Extension** headers are also possible

General headers

- **`Connection`**: lets clients and servers manage connection state
  - `Connection: Keep-Alive`
  - `Connection: close`
- **`Date`**: when the message was created
  - Date: Sat, 31-May-03 15:00:00 GMT
- **`Via`**: shows proxies that handled message
  - `Via: 1.1 www.myproxy.com (Squid/1.4)`
- **`Cache-Control`**: enables caching directives
  - `Cache-Control: no-cache`

Request headers

- **`Host`**: hostname (and optionally port) of server to which request is being sent
- **`Referer`**: URL of the resource from which the current request URI came
  - `Referer: http://www.host.com/login.asp`
- **`User-Agent`** – Name of the requesting application, used in browser sensing
  - `User-Agent: Mozilla/4.0 (Compatible; MSIE 6.0)`
- **`Accept`** and its variants: inform servers of client's capabilities and preferences
  - Enables content negotiation
  - `Accept: image/gif, image/jpeg;q=0.5`
  - `Accept- variants for Language, Encoding, Charset`
- **`Cookie`**: how clients pass cookies back to the servers that set them
  - `Cookie: id=23432;level=3`

Response headers

- **`Server`**: server's name and version
  - `Server: Microsoft-IIS/5.0`
  - Can be problematic for security reasons
- **`Set-Cookie`**: how a server sets a cookie on a client
  - `Set-Cookie: id=234; path=/shop; expires=Sat, 31-May-03 15:00:00 GMT; secure`

Entity headers

- **`Allow`**: lists the request methods that can be used on the entity
  - `Allow: GET, HEAD, POST`
- **`Location`**: gives the alternate or new location of the entity
  - Used with `3xx` response codes (redirects)
  - `Location: http://www.iugaza.edu.ps/ar/`
- **`Content-Encoding`**: specifies encoding performed on body of response
  - Used with HTTP compression
  - `Content-Encoding: gzip`
  - (~ `Accept-Encoding` request header)
- **`Content-Length`**: size of the entity body in bytes
- **`Content-Location`**: the ctual if different than its request URL
- **`Content-Type`**: MIME type of the entity body

HTTPS

HTTP + SSL

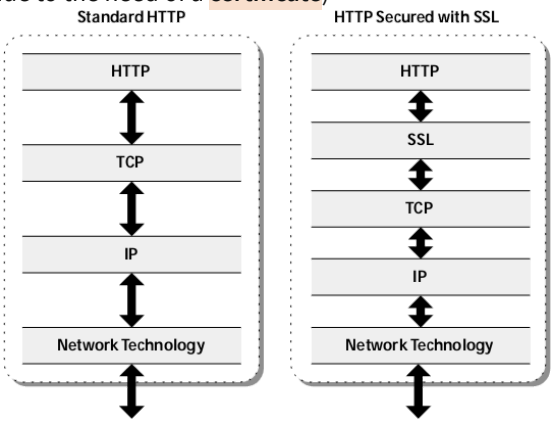
> **HTTPS**: Hypertext Transfer Protocol over Secure Socket Layer (SSL)

SSL protocol inserts itself **between HTTP** application **and TCP** transport layer

- TCP sees SSL as just another application
- HTTP communicates with SSL much the same as it does with TCP

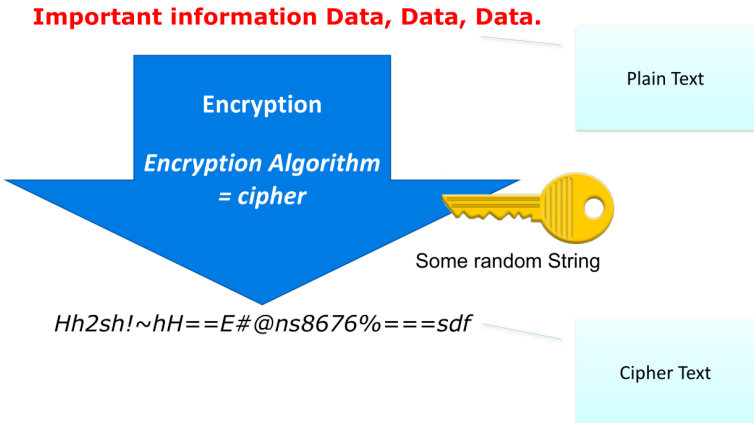
HTTP is

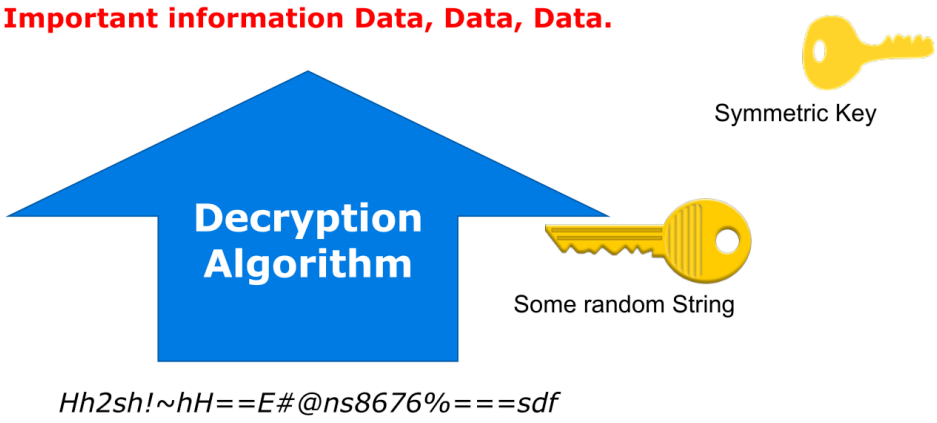
- Only **slightly slower** than HTTP
- A bit more complex to set up (due to the need of a **certificate**)



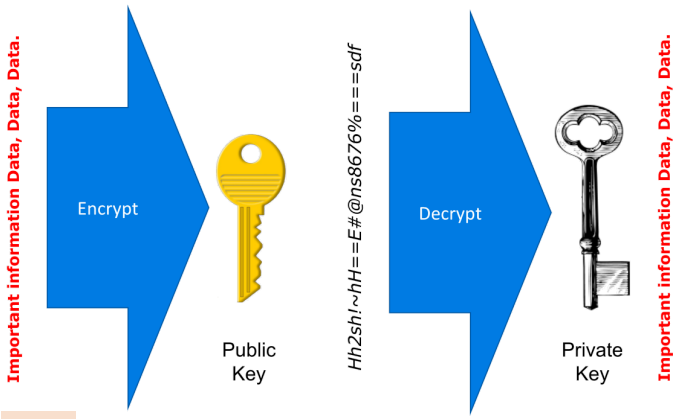
Cryptography

Symmetric





Asymmetric

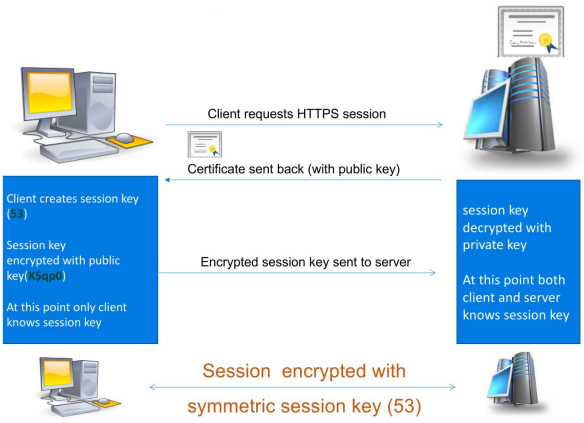


Asymmetric key decryption is **slower** than symmetric key decryption

SSL session

- **Asymmetric encryption** to privately share the **session key**
  - Lot of **overhead**
- **Symmetric encryption** to encrypt **data**
  - **Quicker** and uses less resource

Handshake process



Security issues

- The certificate is guaranteed authentic by a "**chain of trust**"

4. Traffic optimization

HTTP proxy

Web caches (proxy server)

Goal: satisfy client request without involving origin server

- **User** sets browser: web accesses via cache
- **Browser** sends all HTTP requests to cache
  - **Object in cache**: cache returns object
  - Otherwise cache requests object from **origin server**, then returns it to client

Web caching

- Cache acts as both **client and server**
  - Server for original requesting client
  - Client to origin server
- Typically cache is **installed by ISP** (university, company, residential ISP)

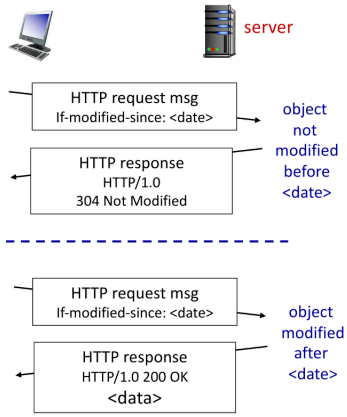
Advantages

- Reduce **response time** for client request
- Reduce **traffic** on an institution's access link
- Internet **dense** with caches
  - Enables **poor content providers** to effectively deliver content

- So too does **P2P** file sharing

Conditional GET

- Goal: don't send object if cache has up-to-date cached version
  - No object transmission delay
  - Lower link utilization
- **Cache**: specify date of cached copy in HTTP request
  - `If-modified-since: <date>`
- **Server**: response contains no object if cached copy is up-to-date
  - `HTTP/1.0 304 Not Modified`



Video streaming and CDNs

Context

Video traffic: major consumer of Internet bandwidth
- Challenge: scale
  - Reach large amount of users
- Challenge: heterogeneity
  - Different users have different capabilities
- Solution: **distributed, application-level infrastructure**

Streaming multimedia: DASH

DASH: Dynamic Adaptive Streaming over HTTP

- **Server**
  - Divides video file into multiple chunks
  - Each chunk stored, encoded at different rates
  - Manifest file: provides URLs for different chunks
- **Client**
  - Periodically measures server-to-client bandwidth
  - Consulting manifest, requests one chunk at a time
    - Chooses maximum coding rate sustainable given current bandwidth
    - Can choose different coding rates at different points in time
      - Depending on available bandwidth at time
- **Intelligence at client**
  - When to request chunk
    - So that buffer starvation, or overflow does not occur
  - What encoding rate to request
    - Higher quality when more bandwidth available
  - Where to request chunk
    - Can request from URL server that
      - Is close to client
      - Has high available bandwidth

Challenge

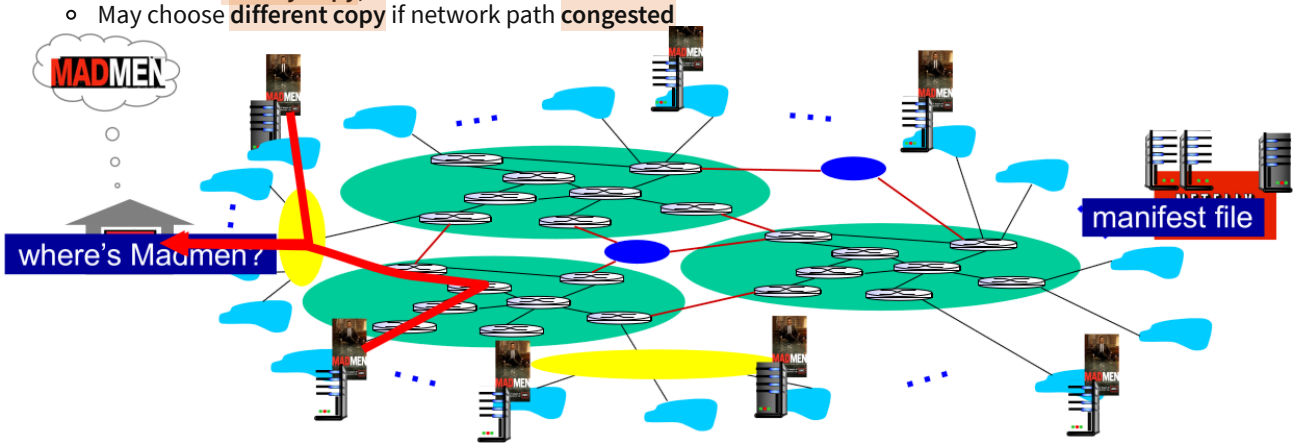
CDN: Content Distribution Networks

Challenge: stream content to several simultaneous users

- Option 1: **single**, large mega-**server**
  - Single point of failure
  - Point of network congestion
  - Long path to distant clients
  - Multiple copies of video sent over outgoing link
  - Problem: this solution doesn't scale
- Option 2: store/serve **multiple copies of videos** at **multiple CDNs** (geographically distributed sites)
  - **Enter deep**: push CDN servers deep into many access networks
    - Close to users
  - **Bring home**: smaller number (10's) of larger clusters in POPs near (but not within) access networks

Content Distribution Networks (CDNs)

- CDN stores **copies** of content at CDN **nodes**
- **Subscriber requests** content from CDN
  - Directed to **nearby copy**, retrieves content
  - May choose **different copy** if network path **congested**



5. Web server

Dynamic pages

Main idea

Obtain **non static** information from the server

- This implies **executing some code** on it, and **send the results** to the user

Common Gatway Interface

> **CGI**: *a way to tell the server to spawn a process, get its results and send them as HTTP response*

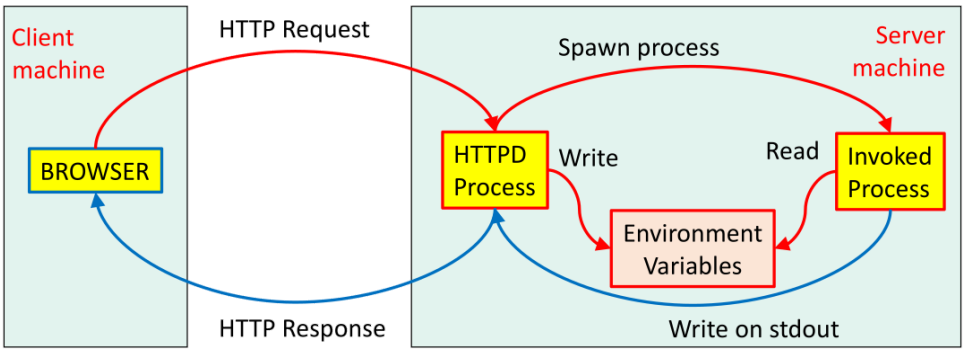
Creating dynamic pages

```
$> cd /Applications/XAMPP/xamppfiles/cgi-bin
$> touch getTime.sh
$> echo '
#!/bin/sh
echo "Content-type: text/plain; charset=iso-8859-1"
echo
echo `date`
' > getTime.sh
$> chmod 755 getTime.sh
$> ./getTime.sh

# Content-type: text/plain; charset=iso-8859-1
#
# sab 22 mag 2021, 20:40:46, CEST
```

```
localhost:8080/cgi-bin/getTime.sh
# sab 22 mag 2021, 20:40:46, CEST
```

Getting info about the request



Access to environment vars is supported in all programming languages!

6. Web server language

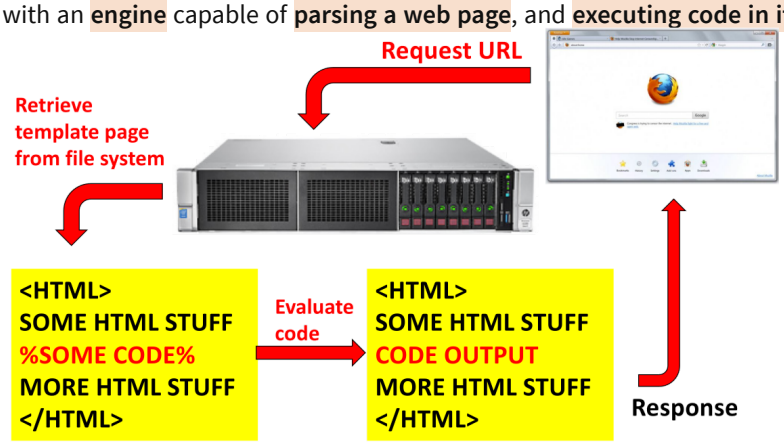
HTML and code

Embedding HTML into code

```
# readPost.sh - Read the POST DATA (full page) and print out HTML
#!/bin/sh
read MYDATA
echo "Content-type: text/plain; charset=iso-8859-1"
echo
echo "<HTML>"
echo "<HEAD><TITLE>Showing Post Data </TITLE>”
echo "<BODY>"
echo "here are the post data:<br>"
echo $MYDATA
echo "</BODY></HTML>"
```

Embedding code into HTML

Augment the **web server** with an **engine** capable of **parsing a web page**, and **executing code in it**



PHP Hypertext Processor

```
<!-- whatsTheTime.php -->
<!DOCTYPE html>
<html>
<body>
  <i>Sir, the current time is
  <?php echo date("h:i:sa"); ?>
  </i><br/>
  (as far as I know!)
</body>
</html>
```

PHP language

Language

- Interpreted
- Non-typed
- Case insensitive

https://www.jdoodle.com/php-online-editor/
https://www.w3schools.com/php/default.asp

Elements

- **Variables**: `$<name>`
- **Data types**
  - Implicitly assigned by interpreter
  - Can be enforced

    ```
    <?php declare(strict_types=1); ... ?>
    ```
- **Comments**: `//`, `#`
- **Output**: `print ...`, `echo ...`, `var_dump()`
  - `echo` and `print` are similar
    - Output data to the screen
    - `echo` has no return value
    - `print` has a return value of `1`
    - `echo` can take multiple parameters
  - `var_dump` prints type and value of the parameter

Language constructs

Arrays

Functions

Operators

Predefined functions

OOP

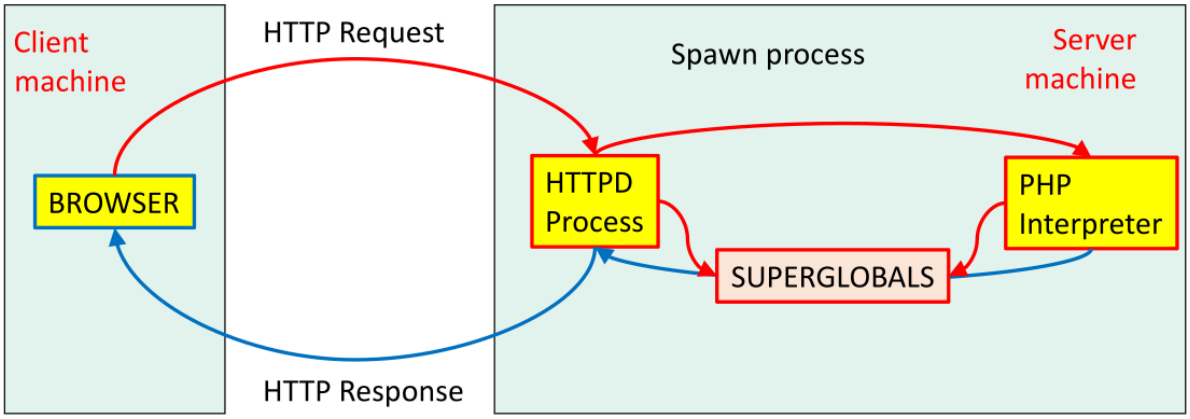
Variable scope

A variable **declared**
- **Outside a function**
  - Has a **global scope**
  - Can only be **accessed outside** a function
- **Within a function**
  - Has a **local scope**
  - Can only be **accessed within** that function

Superglobal variables

Variables visible everywhere
- `$GLOBALS['var']`: marks `var` as **global**
- `$_SERVER`: info mostly extracted from http request headers
- `$_POST`
- `$_GET`: associative arrays, with `key=name`

Getting info about the request



HTML forms

Forms

Give to the user the possibility to **send information to the Web server**
The `<form>` tag defines a form

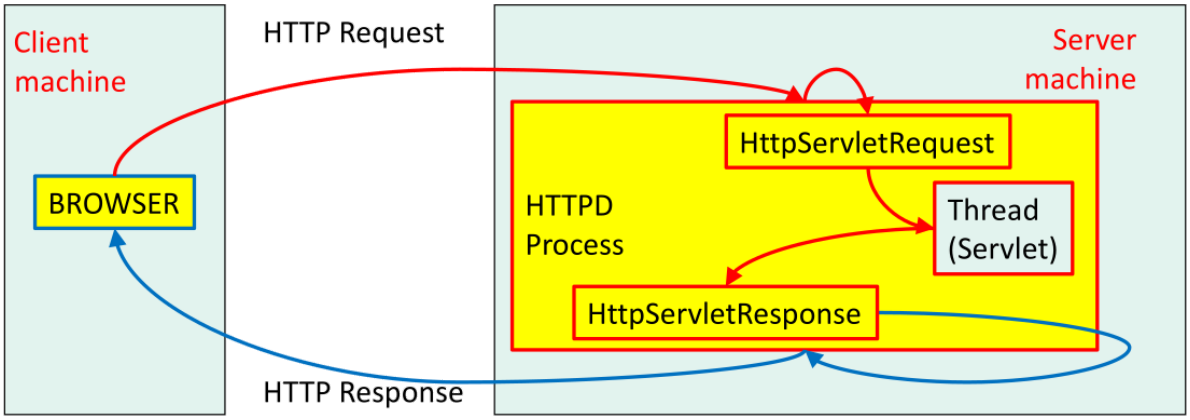
- Attributes
  - `<action>` identifies the **processing engine**
  - `<encrypte>` specificies the **MIME type** used to pass data to the server
- Sub-tags (inner tags)
  - Tags for collecting data
  - An `<input>` tag must be of type `<submit>` for **sending** the data
  - An `<input>` tag can be of type `<reset>` to **cancel** all the gathered data

Examples

7. Java servlets

Programming the web servers

Getting info about the request



Apache Tomcat

- **Java servlet container**
- Run on **JVM**
- Utilizes **Java servlet specification** to **execute servlets generated by requests**
  - Often with the help of **JSP pages**
  - Allows dynamic content to be generated **more efficiently** than CGI scripts

Example

```
import ...
```

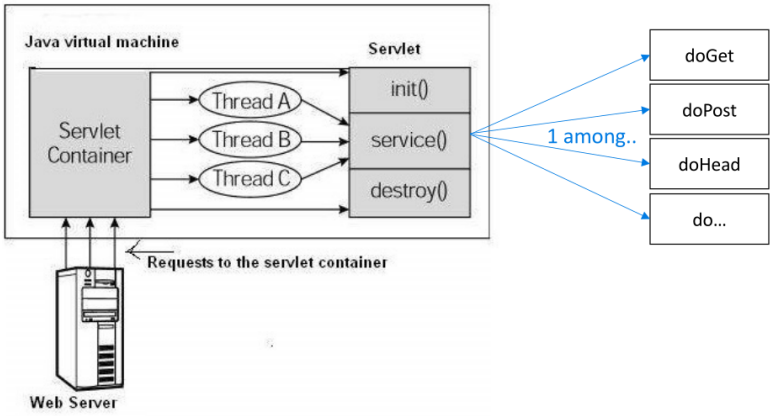
```java
@WebServlet(name = "ReadPost", urlPatterns = {"/ReadPost"}) /* ← */
public class ReadPost extends HttpServlet {                 /* ← */

    @Override                                               /* ← */
    protected void doGet(                                   /* ← */
            HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override                                               /* ← */
    protected void doPost(                                  /* ← */
            HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Processes requests for both HTTP methods
     *   - <code>GET</code>
     *   - <code>POST</code>
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(
            HttpServletRequest request,                     /* ← */
            HttpServletResponse response)                   /* ← */
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8"); /* ← */
        try (PrintWriter out = response.getWriter()) {      /* ← */
            /* TODO output your page here */
            out.println("<!DOCTYPE html>");
            out.println("<html><head>");
            out.println("<title>Servlet ReadPost</title>");
            out.println("</head><body>");
            out.println("<h1>Servlet ReadPost at "
                        + request.getContextPath()          /* ← */
                        + "</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
}
```

Servlet life-cycle



Servlet Deployment

- By default, a servlet application is located at the path `<Tomcat-installationdirectory>/webapps/ROOT`
- The class file would reside in `<Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes`
- If you have a fully qualified class name of `com.myorg.MyServlet`, then this servlet class must be located in `WEB-INF/classes/com/myorg/MyServlet.class`

Parameters passing in request

HTML form

```
<form action="http://localhost:8084/WebApplication01/ReadPost"
      method="post">
```

```
protected void processRequest(
        HttpServletRequest request,                          /* ← */
        HttpServletResponse response)                        /* ← */
        throws ServletException, IOException {
    String name = request.getParameter("fname");             /* ← */
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here */
        out.println("<!DOCTYPE html>");
        out.println("<html><head>");
        out.println("<title>Servlet ReadPost</title>");
        out.println("</head><body>");
        out.println("<h1> fname=" + name + "</h1>");          /* ← */
        out.println("</body>");
        out.println("</html>");
    }
}
```

`httpServletRequest & Response`

Advanced uses

Factoring HTML

Netbeans Services

Hit counter

Implementation

Persist the counter

In a file `counterData`

1. In `init` **check** if file exists
    - If yes, **resume** the counter
    - If no, **create** a new one
2. In `destroy`, let us **save** counter in `conterData`

Java serialization

8. Cookies

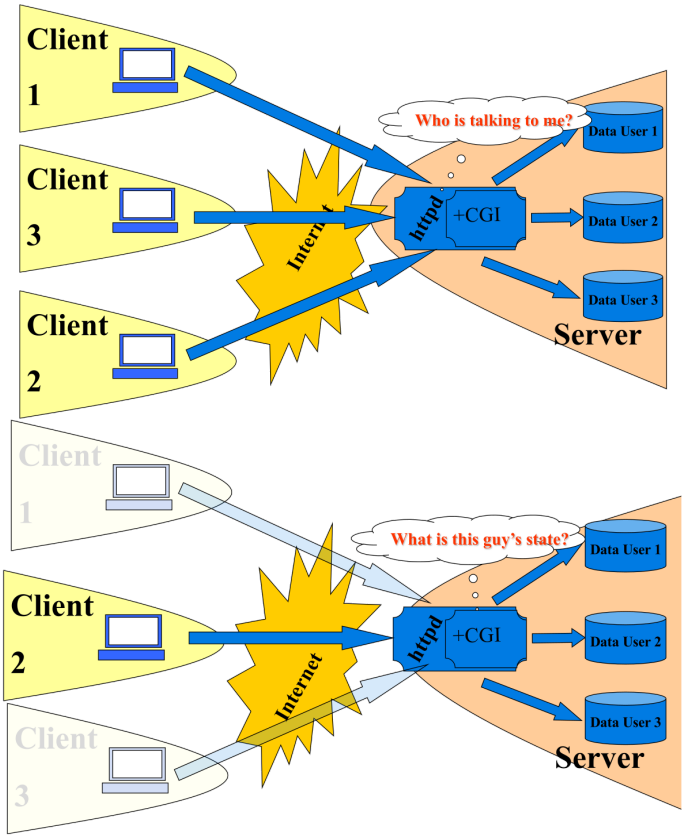
Basics

Keeping state

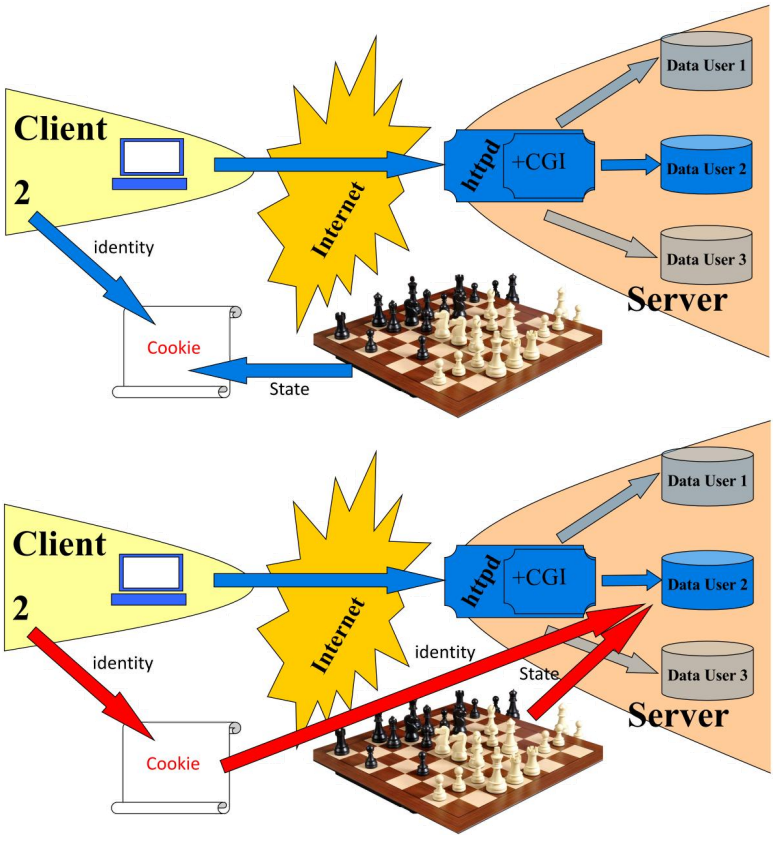
HTTP is **stateless**

- There is no way of keeping track **server-side** of
    - Who is who
    - Which state of the process it is at

Problems



Solutions



Definition

> **Cookie**: *small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server*

- A cookie's **value** can **uniquely identify a client**
  - Are commonly used for **session management**
- A cookie has a **name**, a single **value**, and optional **attributes**
  - e.g.: comment, path and domain qualifiers, maximum age, version number

Features

Cookies **maintain**

- **Status** across a **user session**
- **Info** across multiple **interactions**
  - Customer **identification**
  - Targeted **advertisement**
  - Elimination of **username e password**

Communication

- **Servlet sends** cookies to browser by using the method

```
HttpServletResponse.addCookie(javax.servlet.http.Cookie)
```

  - **Adds fields** to HTTP response **headers** to send cookies to browser
    - **One at a time**
  - The **browser** is expected to **support**
    - 20 cookies for each Web server
    - 300 cookies total
    - May limit cookie size to 4KB each
- **Browser returns** cookies to servlet by **adding fields to** HTTP request **headers**
  - Cookies can be **retrieved** from a request by using the method

```
HttpServletRequest.getCookies()
```

Caching

- Cookies affect the **caching** of the **web pages** that use them
- **Java class `Cookie`** supports both Version 0 and Version 1 cookie **specifications**
  - By default, cookies are created using **Version 0**
    - Ensure the best **interoperability**
  - **HTTP 1.0** does **not** cache pages that use cookies created with this class

Attribute summary

```
String getComment()
void   setComment(String s)
```

- Gets/Sets a **comment associated** with this cookie

```
String getDomain()
void   setDomain(String s)
```

- Gets/Sets the **domain** to which cookie **applies**
  - Normally, cookies are returned only to the **exact hostname** that sent them
  - You can use this method to instruct the browser to **return** them **to other hosts** within the **same domain**
    - The domain should contain different numbers of **dots** depending on itself

```
int  getMaxAge()
void setMaxAge(int i)
```

- Gets/Sets how much **time** (in seconds) should elapse before the cookie **expires**
  - Default: it will last only for current session and will not be stored on disk
    - i.e. until the user quits the browser
  - `LongLivedCookie` class defines a subclass of `Cookie` with a maximum age automatically set one year in the future

```
String getName()
void   setName(String s)
```

- Gets/Sets the **name** of the cookie
  - **Name** and **value** are the two most **important** parts of a cookie
  - Since the `getCookies` method of `HttpServletRequest` returns an **array of Cookie** objects, it is common to search a **particular name**, then check the value with **`getValue`** (~ `getCookieValue`)

```
String getValue()
void   setValue(String s)
```

- Gets/Sets the **value associated** with the cookie
  - In a few cases a **name** is used as a **boolean flag**, and its value is ignored
    - i.e. the existence of the name means true

```
String getPath()
void   setPath(String s)
```

- Gets/Sets the **path** to which this cookie **applies**

- Default: the cookie is returned for **all URLs** in the same directory as the current page, as well as all sub-directories
- This method can be used to **specify something** more general
  - e.g.: `cookie.setPath("/")` specifies that all pages on the server should receive the cookie
- The path specified must include the **current directory**

```
boolean getSecure()
void    setSecure(boolean b)
```

- Gets/Sets the boolean value indicating whether the cookie should only be sent over **encrypted connections** (i.e. SSL)

```
int  getVersion()
void setVersion(int i)
```

- Gets/Sets the cookie **protocol version** this cookie complies with

Cookies in response headers

The cookie is added to the **`Set-Cookie` response header**

- By means of the **`addCookie`** method of `HttpServletResponse`

```
Cookie userCookie = new Cookie("user", "uid1234");
userCookie.setMaxAge(60*60*24*365);
response.addCookie(userCookie);
// before opening the body of response!
// i.e. before any out.print
```

Reading from client

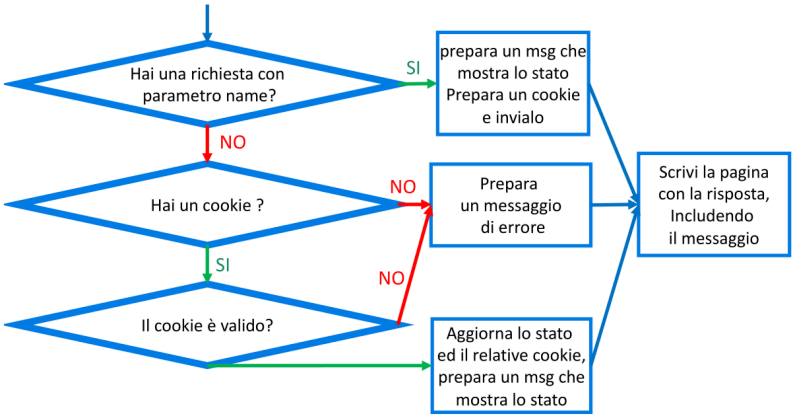
1. To **read** the cookies that come back from the client, call **`getCookies`** on the `HttpServletRequest`
   - Return: array of `Cookie` objects
     - Corresponding to values that came in on Cookie HTTP request header
2. Once this array is available, loop down it calling **`getName`** on each cookie to **search a specific name**
3. Then call **`getValue`** on the matching cookie
   - Do some processing specific to the resultant value

Cookies in action

`SetCookies`

`ShowCookies`

Cookies in action

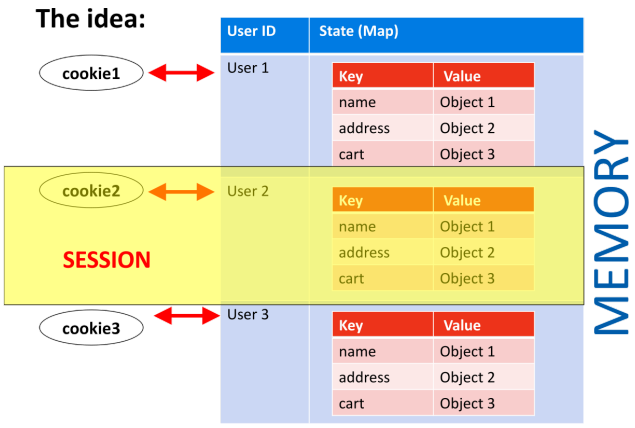


Legal aspects

Italian regulations

Session

Session tracking using cookies



Implementation

```
Hashtable globalTable = getGlobalTable();
String sessionID = makeUniqueString();
Hashtable stateTable = new Hashtable();
globalTable.put(sessionID,stateTable);
Cookie sessionCookie = new Cookie("SessionID", sessionID);
response.addCookie(sessionCookie);
stateTable.put("key", infoObject);
```

Concept

- To **support applications** that need to maintain state, **Java Servlet** technology
  - Provides an **API** for **managing sessions**
  - Allows several mechanisms for **implementing sessions**
- Sessions are represented by an **`HttpSession` object**

Session tracking

- To **associate a session with a user**, a web container can use several methods
  - All of which involve **passing an identifier** between the client and the server
    - Maintained on the **client as a cookie** or
    - The **web component** can include the **identifier in every URL** that is returned to the client

Accessing session

Access `HttpSession` object by calling **`getSession`** method of request object

- Return
  - **Current session** associated with this request or
  - **Creates a session** if the request does not have it

Associating objects with session

Associate **object-valued attributes** with an `HttpSession` by **name**

- Such attributes are accessible by **any web component** that
  - Belongs to the **same web context** and
  - Is handling a request that is part of the **same session**

`HttpSession` methods

Attributes

```
public Enumeration getAttributeNames()
```

- Return: **`Enumeration` of `String`** objects containing the **names** of all the **objects bound** to this session

```
public Object getAttribute(String name)
```

- Return: the **object bound** with the specified name in this session, or `null` if no object is bound under the name

```
public void setAttribute(String name, Object value)
```

- **Binds an object** to this session, using the name specified
- If an object of the same name is already bound, the object is **replaced**

```
public void removeAttribute(String name)
```

- **Removes the object bound** with the specified name from this session
- **Nothing** if the session does not have an object bound with the specified name

Others

```
public java.lang.String getId()
```

- Return: string containing the **unique identifier** assigned to this session
  - **Assigned by the servlet container**
  - Is **implementation dependent**

```
public boolean isNew()
```

- Return: `true` if the client
  - Does **not yet know** about the session or
  - Chooses **not to join** the session
    - e.g.: if client had **disabled** the use of **cookies**

```
public void invalidate()
```

- **Invalidates** this session then **unbinds** any objects bound to it

Session life-cycle

A session consumes **resources** (**memory**), hence it has to be managed
- Problem: http is **stateless**, so there is **no** notion of **log out**
- Solution: set an expiry time for sessions (**timeout**)

Setting session global timeout

Set the timeout period in the deployment descriptor using **NetBeans**

1. **Expand the node** of your project in the 'Projects' tab
2. **Expand the Web Pages and `WEB-INF` nodes** that are under the project node
3. If **`WebInf` is empty**, select it and right-click
4. `new → other → Standard Deployment Descriptor`
5. Double-click **`web.xml`**
6. If not present, add

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

   - 30 is the number of **minuts** after which the session will expire

`web.xml`

Deployment descriptor file used by Java web applications to determine
- How URLs map to servlets
- Which URLs require authentication
- etc…

`web.xml` resides in the app's `WAR` under the `WEB-INF` directory.

`web.xml` and annotations

Some info expected in `web.xml` can be provided via **annotation in Java code**
- Whatever is defined in `web.xml` **configuration overwrites annotations**

Example (slide 37)

Session tracking

- If the **application** uses **session objects**, ensure that **session tracking is enabled** by having the application **rewrite URLs** whenever the client turns off cookies
- Can be done by **calling the response's `encodeURL(URL)`** method on all URLs returned by a servlet
  - Includes the **session ID in the URL** only if cookies are disabled

- Otherwise, returns the **URL unchanged**

Session in action

Associating events with session objects

The **application** can **notify**

- Web context and session listener objects of **servlet lifecycle events**
- Objects of events related to their **association with a session**
  - When the object is **added/removed** to/from a session
    - The object must implement the **interface** `javax.servlet.http.HttpSessionBindingListener`
  - When session to which the object is attached will be **passivated/activated**
    - It is **moved** between virtual machines or
    - **Saved** to and **restored** from persistent storage
    - The object must implement the **interface** `javax.servlet.http.HttpSessionActivationListener`

9. JavaScript

HTML: dynamic behaviour

Web architecture with smart browser

Evolution: **execute** code also on **client**

- The web programmer writes **programs in Javascript** which run on the **browser**
- The **dynamic** behaviour is on the **client side**
  - The file can be loaded **locally**

`onmouseover`, `onmouseout`

```
<div onmouseover="this.style.color = 'red'"
     onmouseout="this.style.color = 'blue'">
I can change my colour!
</div>
```

JS event-based: `UiEvents`

Event objects that iherits the properties of the **`UiEvent`**

- `FocusEvent`
- `InputEvent`
- `KeyboardEvent`
- `MouseEvent`
- `TouchEvent`
- `WheelEvent`

https://www.jdoodle.com/html-css-javascript-online-editor/

Language

Features

ECMAScript Engine

> **ECMAScript engine**: *program that executes source code written in a version of the ECMAScript language standard*

JS vs Java

	Java	JavaScript
Browser control	NO	YES
Networking	YES	Partial
Graphics	YES	Partial

JS and HTML

- ```
  <script> JS code </script>
  <script src="url"> JS code </script>
  ```
- ```
  <server> JS code </server>
  ```
- ```
  <!-- In event handlers -->
  <input type="button" value="Click me" onClick="JS code">
  <div onmouseover="this.style.color = 'red'"
       onmouseout="this.style.color = 'blue'">
  ```

Data types

- Primitive data types
  - `number`, `string`, `boolean`, `undefined`
- Complex data types
  - `object`, `function`
- **Loosely**, **dynamic** typed variables

```
t0 == typeof(x);      // t0: undefined
var x = 3;
var t1 = typeof(x);   // t1: number
x = "pippo";
var t2 = typeof(x);   // t2: string
```

- `===` equal **value** and equal **type**
- Type **operators**
  - `typeof()` returns the type of a variable
  - `instanceof` returns `true` if an object is an instance of an object type

Strings

Strings are both

- **Primitive** data type
  - Values created from **literals**

```
var strPrimitive = "str";
```

- **Object**
  - Defined with the keyword **new**

```
var strObject = new String("str");
```

Objects and DOM

Objects

- Have **variables** and **methods**
- Can be **printed**
  - They use their **customized `toString()`** method or
  - They give a **generic indication**
- Some of them represent **fragments of an HTML** document
  - The collection of these objects represent the **whole page** (**DOM**)

> **Document Object Model (DOM):** *interface that treats HTML document as a tree structure wherein each node is an object representing a part of the page*

- Represents a document with a **logical tree**

User I/O

Core

```
n = window.prompt("Give me the value of n", 3);
for (i=1; i<n; i++) {
    document.write(i);
    document.write("<br>");
}
```

Using `document.write()` after document is loaded, will delete all existing HTML
```

JS output

Writing into
- `document.write()` → **HTML output**
- `window.alert()` → **alert box**
- `console.log()` → **browser console**

Writing into **HTML element** (`[object HTML*Element].xxx`)
- `innerHTML`: full HTML content
- `innerText`: text only, CSS aware
- `textContent`: text only, CSS unaware

Functions

Function hoisting

> **Hoisting**: *JS mechanism where variables and function declarations are moved to the top of their scope before code execution*

- Only **moves declaration**
- **Assignments** are left **in place**

Function statements

The function statement **declares a function**

```
hoisted();            // OUTPUT: "This function has been hoisted"
function hoisted() {  // Declaration hoisted
    console.log('This function has been hoisted');
};
```

- Function **declarations** are **hoisted**

Function expressions

Functions can also be **defined** using an **expression**

1. Function expression is **stored in a variable**

   ```
   var f = function(a, b) {return a * b;};
   ```

   - Function do not need name, as it are always invoked (called) using the variable name
2. The **variable** can be used **as function**

   ```
   var product = f(2, 3);
   ```

Functions stored in variables do **not need function names**
- They are always invoked using the **variable name**

Expressions and hoisting

Function expressions **load** only when the **interpreter** reaches that **line of code**

```
fundef();  // OUTPUT: "TypeError: expression is not a function"var fundef = function() {
console.log('This will not work');};
```

- Function **expressions** are **not hoisted**

Arrow functions

Function

```
function multiplyByTwo(num) {return num * 2;};
```

Redefinitions

```
const multiplyByTwo = function (num)    {return num * 2;};const multiplyByTwo =          (num) =>
{return num * 2;};const multiplyByTwo =           num  => {return num * 2;};const multiplyByTwo =
          num  =>          num * 2;
```

Usage: `multiplyByTwo(n);`

Mapping and filtering

`map`

```
var A = [1,2,3,4];document.write(A.map(num => num * 2));  // 2,4,6,8
```

`filter`

```
var A = [1,2,3,4];document.write(A.filter(num => num % 2 == 0));  // 2,4
```

Variables

Undeclared variables

An **undeclared variable** is
- Assigned the **value** **`undefined`** at execution
- Of **type** **`undefined`**

`ReferenceError` is thrown when trying to access previously undeclared variable

Variable scope

Type of declaration	Scope	Notes
`x=10;`	Always global	
`var x=10;`	Function scope	Global if external to any function
`let x=10;`	Block scope	ES 6

```
{ var x = 2; }// x CAN be used here{ let y = 2; }// y can NOT be used here
```

Variable hoisting

Variable **declarations** are processed **before any code** is executed

```
···          hoist      var x;···          ==>      ···var x=10;          x=10;
```

Variable redefinition

`let` **variables** can **not** be redefined with a **larger scope** in an inner block

```
var x = 1;  /* x=1 */{ x = 2;  /* x=2 */ }/* x=2 */
```

```
var x = 1;  /* x=1 */{ var x = 2;  /* x=2 */ }/* x=2 */
```

```
var x = 1;  /* x=1 */{ let x = 2;  /* x=2 */ }/* x=1 */
```

Objects

JS objects

JS is all **dynamic**, all **runtime**, and it has **no classes** at all
- It's all just **instances** (**objects**)
- Simulated "classes" are **function objects**

JS vs Java

	JS	Java
Structure	Can change	Cannot change
Class	Objects without class	Objects must belong to a class

Objects as data structures

JS objects
- Are **collections of named values** (**properties**)

```
var person = {    first: "Dorothea",    last: "Wierer",    birth: 1990};
```

- Can contain also **methods**

```
var person = {    first: "Dorothea",    last: "Wierer",    year: 1990,    full: function() {
       return this.first + " " + this.last;    }};
```

- Accessing **properties**

```
person.firstName == person['firstName']
```

Dynamic management

- **Add new properties** to an existing object by giving it a value

```
person.place = "Brunico";
```

- **Delete** existing **properties**

```
delete person.first;
```

Creating objects

- Objects can also be created **empty and populated** later

```
var person = {};person.first = "Dorothea";person.last = "Wierer";person.year =
1990;person.full = function() {return this.first + " " + this.last;};
```

- Using **`Object` constructor**

```
var object1 = new Object();Object.defineProperty(    object1, 'name', {        value: "AA",
     writable: false    });object1.name = 77;document.write(object1.name);// OUTPUT: AA (no
error)// if `writable: true` → OUTPUT: 77
```

Objects constructors

Object constructor: *template for creating objects of a certain type* (~ class)

- Constructor **function** is JavaScript's version of a **class**

```
function Rectangle(w, h) {    this.width = w; this.height = h;  // Instance variables    this.area
= function() {        return this.width * this.height;  // Method    }}r = new Rectangle(3,4);//
r.area() => 12// r.width => 3
```

Prototype

`prototype` feature

Allows to **add** new properties and methods **to object constructors**

```
Rectangle.prototype.area = function() {    return this.width * this.height;}
```

Prototype name space

Prototype is a **separate naming space**

- **Shared** by all instances of the **same constructor**
- Prototype properties can be **redefined** for **single instances**

Prototype inheritance

Prototype is used for providing a sort of **inheritance**

`for ... in` cycle

Iterates over **all object properties**

- Including the ones in the **prototype**
- `enumerable:false` in `Object.defineProperty()` **skip** the property

```
···let person1 = new Person('Dorothea', 'Wierer');···for (x in person1) {    document.write(x + "
- ");  // first - last - year - full}
```

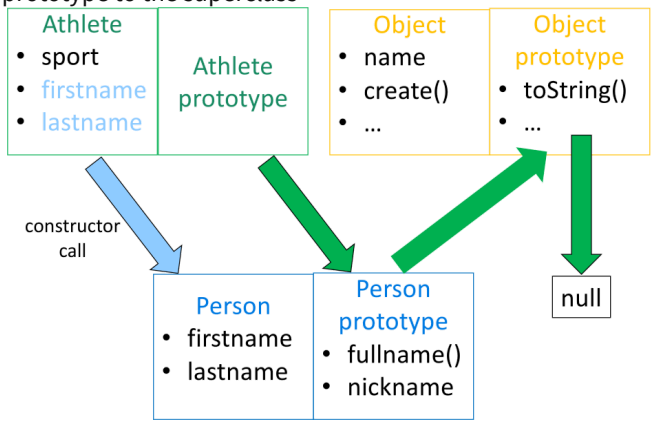
Prototype-based inheritance

Definitions

- **Instance variables** in constructor
- **Methods** and **shared variables** in prototype

Inheritance

- **Instance variables**, by invoking the superclass constructor
- **Prototype**, by associating prototype to the superclass



Predefined objects

```
var x1 = new Object();var x2 = new Date();// DO NOT USE ↓var x3 = new String();var x4 = new
Number();var x5 = new Boolean();var x6 = new Array();     // Use []var x7 = new RegExpr();   //
Use /( )/var x8 = new Function();
```

- **Native prototypes** should never be extended
  - Unless it is for the sake of **compatibility** with newer JavaScript features
- **`Math`** is an object to **never instantiate**
  - It's a **container** for math **functions**

Classes

JS classes

Syntactic sugar over JS's existing prototype-based inheritance
- Class syntax does **not** introduce a new **object-oriented inheritance** model to JS
- Unlike function declarations, class **declarations** are **not hoisted**

```
class B [extends A] {    constructor(a, b, c) {        super(a, b);        this c = c;    }
}B.prototype.d = 'd';
```

Variables and methods

- **Instance variables** can only be **defined within methods**
- **Class variables/methods** can be **defined**, but they can **not be called** on instances (~ Java)

Arrays

Features

Arrays are
- **Sparse**
- **Inhomogeneous**
- **Associative**

```
a = [];a = new Array(); //discourageda[0] = 3; a[1] = "hello"; a[10] = new
Rectangle(2,2);a.length() == 11;a["name"] == a.name;
```

Functions

- **Add/Remove** an element
  - At the **end**
  - At the **front**
  - By **index**
- **Remove** a **number of elements** starting from an index
- **Find the index** of an element
- **Make a copy** of an array

Plus operator

Rules

1. **Conversion**: any `Object` operand is converted to a **primitive value**
    1. If an operand is `String`, the other is converted to **`String`** (if not `String`)
    2. Any remaining operand is converted to **`Number`**
        - `Boolean → 0|1`
        - `null → 0`
        - `Undefined → NaN`
2. **Execution**: if both operands are
    - `String →` **concatenation**
    - `Number →` **sum**

Unary operator

Used to **convert `String` to `Number`**

```
var s = "1";  // stringvar n = + s;  // number (1)var s = "s";  // stringvar n = + s;  // number (NaN)
```

Objects

Rules to execute the conversion

1. **Check** if the object verifies all the following
    - Is **not** a **`Date`** and
    - Has a **`valueOf()`** method and
    - Its `valueOf()` method returns a **primitive value**
2. If yes, use the **`valueOf()`** method
3. If no, use the **`toString()`** method

Notes

- **Array** `[1,"a",2]` would be converted to `"1,a,2"`
- **Empty object** `{}` is converted to `"[object Object]"`

10. DOM

Introduction

JS and DOM

When a web **page is loaded**, the browser creates a **Document Object Model** of it

- **Tree of objects**
- **Every element** in a document is part of its DOM
    - They can be accessed/manipulated using **DOM** and a **scripting language**

With DOM, **JS** can

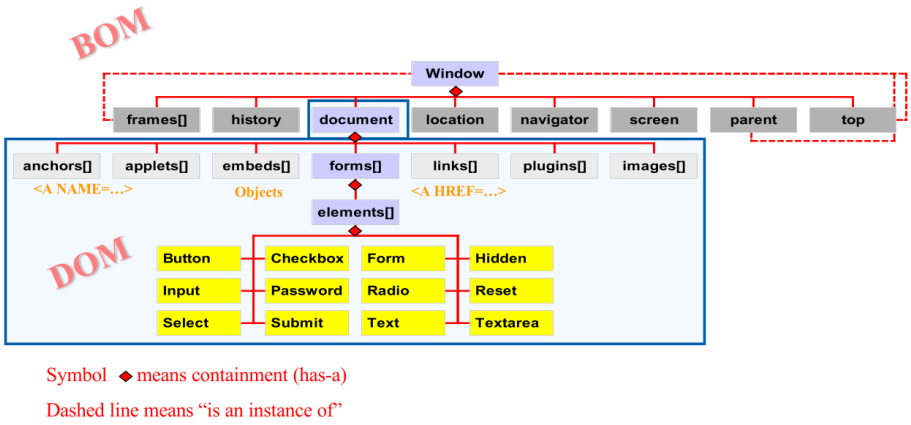
- **Change all the HTML** elements, attributes, styles in the page
- **Add/Remove** existing HTML elements and attributes
- React to and create new **HTML events** in the page

Languages

Implementations of DOM can be built for **any language**

- **JS** is the only one that can work **client-side**

Object hierarchy

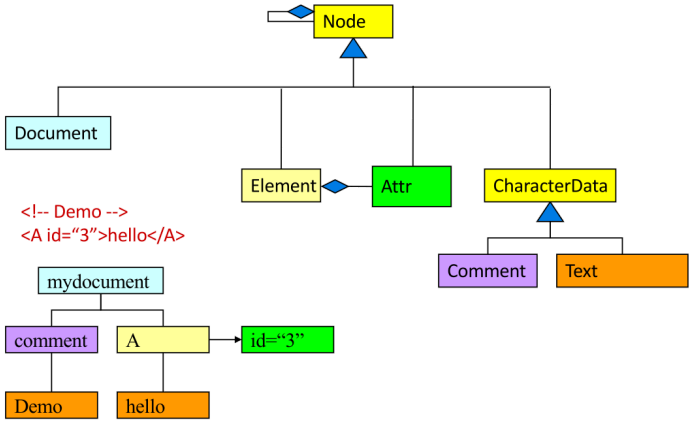


Datatypes

Fundamental datatypes

- **`Document`** (is-a `Node`)
  - The **root** document object itself
- **`Node`**
  - **Every object** located within a document is a node
  - In an HTML document, an **object** can be
    - Element node
    - Text node
    - Attribute node
- **`Element`** (is-a `Node`)
  - The most **general base class** from which all element objects (i.e. objects that represent elements) in a `Document` **inherit**
- **`Attr`** (is-a `Node`)
  - **Object reference** that exposes a special **interface** for attributes
  - Attributes are nodes in the DOM just like elements are
- **`NodeList`**
  - **Array of elements**, where items are accessed by **index**
    - `list.item(1)` or `list[1]`
- **`NamedNodeMap`**
  - **Associative array**, where items are accessed by **name**
  - They can also be accessed by index using the `item()` method
  - It's possible to add and remove items

Node: hierarchy



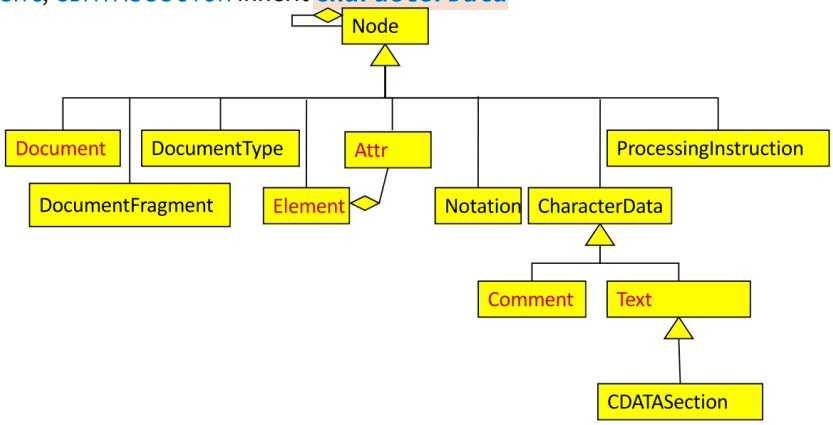
Warning

- The implied semantic of this model is **wrong**
- A comment **can't contain** any other **node**
- The **integrity** is delegated to a series of **`Node`'s attributes**
  - That the **programmer** should check

Node: inheritance

The following types **inherit the `Node` interface**'s methods and properties

- `Document`, `Element`, `Attr`, `CharacterData`, `ProcessingInstruction`, `DocumentFragment`, `DocumentType`, `Notation`
  - `Text`, `Comment`, `CDATASection` inherit **`CharacterData`**



Node: properties

`Node.*`

- `nodeType` (read only)
- `nodeName` (read only)
  - `HTMLElement` → name of the corresponding tag
  - `Text` → `"#text"`
  - `Document` → `'#document'`
- `baseURI` (read only)
- `textContent` (read/write)

Node: types

`Node.nodeType`

1. `ELEMENT_NODE`
2. `ATTRIBUTE_NODE`
3. `TEXT_NODE`
4. `CDATA_SECTION_NODE`
5. `ENTITY_REFERENCE_NODE`
6. `ENTITY_NODE`
7. `PROCESSING_INSTRUCTION_NODE`
8. `COMMENT_NODE`
9. `DOCUMENT_NODE`
10. `DOCUMENT_TYPE_NODE`
11. `DOCUMENT_FRAGMENT_NODE`
12. `NOTATION_NODE`

Node: read only navigation properties

`Node.*`

- `childNodes`
- `firstChild`
- `lastChild`
- `nextSibling`
- `previousSibling`
- `parentNode`
- `parentElement`
- `ownerDocument`

Element: inheritance

The **most general base class** inherited by all element objects in a `Document`

- It only has **methods and properties common** to all kinds of elements
- More specific **classes inherit** from `Element`
  - e.g.: `HTMLElement` and `SVGElement` interfaces are the base interface for HTML and SVG elements

Element: properties

`Element.*`

- `id`
- `attributes`
- `className`
- `innerHTML`
- `outerHTML`
- `clientHeight` (inner height)
- `clientLeft` (width of the left border)
- `clientTop` (width of the top border)
- `clientWidth` (inner width)

Element: main properties

`Element.*`

- `innerHTML`
- `style.left`
- `setAttribute()`
- `getAttribute()`
- `addEventListener()`

Element: **name** property

`name` **gets/sets the name** property of an element in the DOM

- It only applies to: `<a>` `<applet>` `<button>` `<form>` `<frame>` `<iframe>` `<img>` `<input>` `<map>` `<meta>` `<object>` `<param>` `<select>` `<textarea>`
- The `name` property **doesn't exist** for other elements
- It is **not a property of `Node`, `Element` or `HTMLElement`** interfaces
  (!~ `tagName`, `nodeName`)
- Can be used in **`document.getElementsByName()`**, a **form** and with the form elements collection
  - Return: single element or collection

JS output

Writing into an **HTML element**

- `Element.innerHTML`

```
<div onmouseover="this.innerHTML='How are you?';">Hello</div>
```

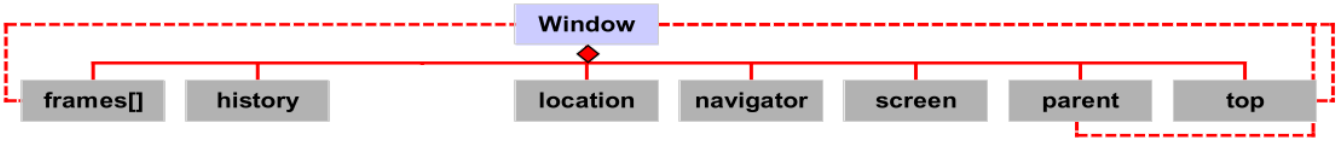
- `HTMLElement.innerText`

```
<div onmouseover="this.innerText='How are you?';">Hello</div>
```

- `Node.textContent`

```
<div onmouseover="this.textContent='How are you?';">Hello</div>
```

BOM subset



Window

Web **browser** window or **frame**

- Properties: objects
  - `history`
  - `frames[]`
  - `document`
  - `location`
  - `navigator`
  - `screen`
  - `parent`, `top`
- Properties: others
  - `status`, `defaultStatus`
  - `name`
- Methods
  - `alert()`, `prompt()`, `confirm()`
  - `focus()`, `blur()`
  - `moveBy()`, `moveTo()`
  - `resizeBy()`, `resizeTo()`
  - `scroll()`, `scrollBy()`, `scrollTo()`
  - `setInterval()`, `clearInterval()`
  - `setTimeout()`, `clearTimeout()`

Screen

Information about the **display**

- Properties
  - `availHeight`, `availWidth`
  - `height`, `width`
  - `colorDepth`, `pixelDepth`
  - `hash`

Navigator

Information about the **browser** in use

- Properties
  - `appName`
  - `appVersion`
  - `Platform`
- Methods
  - `javaEnabled()`

History

The **URL history** of the browser

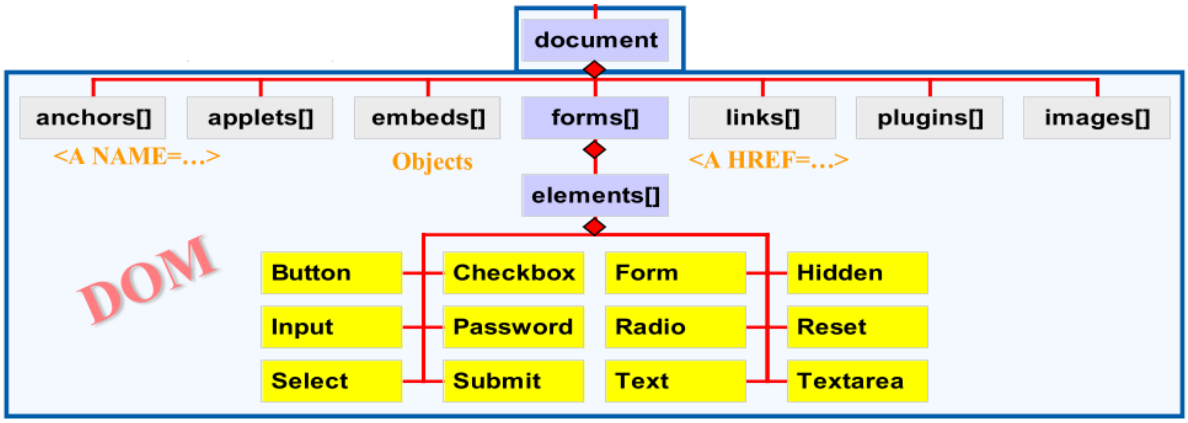
- Properties
  - `lenght`
- Methods
  - `back()`
  - `forward()`
  - `go(+/-n)`
  - `go(target_substring)`

Location

The specification of the **current URL**

- Properties
  - `href`
  - `protocol`, `hostname`, `port`
  - `search`
  - `hash`
- Methods
  - `reload()`
  - `replace()`

Document and its components



Document

HTML document

- Properties: arrays of component objects
  - `anchors[]`
  - `applets[]`
  - `embeds[]`
  - `forms[]`
  - `links[]`
  - `plugins[]`
- Properties: others
  - `bgColor`, `fgColor`, `linkColor`, `vlinkColor`
  - `lastModified`
  - `title`, `URL`, `referrer`, `cookie`
- Main methods
  - `open()`
  - `close()`
  - `clear()`
  - `write()`

Element selection

```
document.getElementById(id);document.getElementsByName(name);document.getElementsByTagName(name);document.getElementsByClassName(name);
```

DOM structure modification

```
document.createElement(element)document.removeChild(element)document.appendChild(element)document.replaceChild(newElement, oldElement)
```

Image

Image embedded in an HTML document

- Properties
  - `border`
  - `height`
  - `width`
  - `src`

Applet

Applet embedded in a Web page

- Properties
  - Same as the public **fields** of the **Java applet**
- Methods
  - Same as the public **methods** of the **Java applet**

Form

HTML **input form**

- Properties
  - `action` (destination URL)
  - `method` (get/post)
  - `name` (name of Form)
  - `name` (destination Window)
  - `Elements[]` (list of contained elements)
- Methods
  - `reset()`
  - `submit()`

Events

Event

The **`Event` interface** represents an event which takes place in the DOM

- An event can be
  - Triggered by **user action**
  - **Generated by APIs** to represent the progress of an **asynchronous task**
  - **Triggered programmatically**
    - Calling the `HTMLElement.click()` method of an element
    - Defining the event, then sending it to a specified target using `EventTarget.dispatchEvent()`
- `Event` contains the **properties and methods** which are **common** to all events
  - Some types of events use **other interfaces** based on main `Event` interface

Types

- **UI events**
  - Input
  - Keyboard
  - Focus
  - Mouse
  - Drag and drop
- **Generic events**
  - Events
  - Animation
  - Clipboard

Handlers

Events can be managed by **`EventHandlers`**

```
on<event> = "JS code"
```

Examples

Server side execution

Server-Side JS

Server-dependent technology to **process** web page **before** passing it to client

- Substitute for **CGI**

Node.js

- Open-source, cross-platform **JS run-time environment** for executing **JS code server-side**
- Has an **event-driven architecture** capable of **asynchronous I/O**
- Optimize **throughput** and **scalability**
  - In web applications with many **I/O operations**
  - For **real-time** web applications

11. CSS

Types of cascading style sheets

Main idea

- **Separation** between content and presentation
- **Customize** the behaviour of **tags**

Inline

- **Local** definition
- HTML **tags**

```
<tag style="formatting-element: value;"></tag>
```

Internal

- Definition of a **page style**
- **All instances** of a type of HTML tag

```
<head>
<style>
  tag {
    color: blue;
    text-align: center;
  }
</style>
</head>
```

External

- Definition of a **shared style**
- All instances of a type of HTML tag for **many files**

```
<head>
<style type="text/css"> @import url("myStyle.css"); </style>
<!-- OR -->
<link rel="stylesheet" type="text/css" href="myStyle.css">
</head>
```

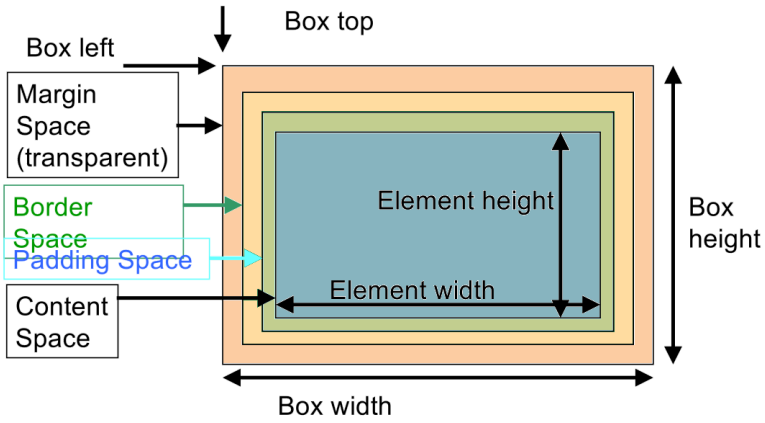
```
/* myStyle.css */
tag {
  color: blue;
  text-align: center;
}
```

Formatting elements

Length units

- **Absolute length**
  - **International System**
    - `cm` centimetrs
    - `mm` millimetrs
  - **Anglosaxon units**
    - `in` inch
    - `pt` point (`1/72 in`)
    - `pc` pica (`12 pt = 1/6 in`)
  - **System dependent**
    - `px` pixel
- **Relative length**
  - `em` height of element font
  - `ex` extended height of element font

Box model



Formatting elements

```
text {
color:              color
font-family:        name
font-size:          xx-small|x-smallsmall|medium|large|x-large|
                    xx-large|larger|smaller|absoluteSize|
                    relativeSize|percentage|lenght
font-style:         normal|italic
font-weight:        bold|bolder|lighter|normal|100|200|...|900
line-height:        normal|length|percentage
text-align:         left|center|right|justify
text-decoration:    blink|line=through|overline|underline
text-indent:        lenght|percentage
text-transform:     none|capitalize|uppercase|lowercase
}

background {
background-color: color
background-image: uri
}

box-model {
border-width:       thin|medium|thick|n
border-color:       color
border-style:       double|groove|none|inset|outset|ridge|solid
border-width:       thin|medium|thick|length
margin|padding:     thickness
}

ol { list-style:  decimal | [lower|upper]-[alpha|roman] }
ul { list-style:  circle|disc|square }
ul { list-style-image: url('img.gif') }
```

Selectors

Basic selectors

- **Simple**

```
tag { ... }
```

```
<tag>
```

- **Class**

```
.class { ... }
```

```
<tag class="class">
```

- **Universal**

```
* { ... }
```

```
<tag>
```

- **ID**

```
#id { ... }
```

```
<tag id="id">
```

Combinator selectors

- **Simple + class**

```
tag.class { ... }
```

```
<tag class="class">this</tag>
```

- **Grouping**

```
tag1, tag2 { ... }
```

```
<tag1>this</tag1> <tag2>and that</tag2>
```

- **Descendant**

```
tag1 tag2 { ... }
```

```
<tag1>
  <···>
    <tag2>this</tag2>
  </···>
</tag1>
```

- **Child**

```
tag1 > tag2 { ... }
```

```
<tag1>
  <tag2>this</tag2>
</tag1>
```

- **Adjacent sibling**

```
tag1 + tag2 { ... }
```

```
<tag1></tag1> <tag2>this</tag2>
```

- **General sibling**

```
tag1 ~ tag2 { ... }
```

```
<tag1></tag1> <···></···> <tag2>this</tag2>
```

Advanced selectors

- **Pseudo-class**: defines a special state of an element
  - Mouse over, get focus, link visited, n-th child, current language
- **Pseudo-element**: styles specified parts of an element
  - First letter, first line, before/after the content, selection

Attribute selectors

```
input[type="button"] { ... }
```

Selector	Example	Example description
`[attribute]`	`[target]`	Selects all elements with a `target` attribute
`[attribute=value]`	`[target=_blank]`	Selects all elements with `target="_blank"`
`[attribute~=value]`	`[title~=flow]`	Selects all elements with a `title` attribute containing the word `"flow"`
`[attribute|=value]`	`[lang|=en]`	Selects all elements with a `lang` attribute value starting with `"en"`
`[attribute^=value]`	`a[href^="https"]`	Selects every `<a>` element whose `href` attribute value begins with `"https"`
`[attribute$=value]`	`a[href$=".pdf"]`	Selects every `<a>` element whose `href` attribute value ends with `".pdf"`
`[attribute*=value]`	`a[href*="w3s"]`	Selects every `<a>` element whose `href` attribute value contains the substring `"w3s"`

Cascading and positioning

`!important` clause

`!important` has precedence over the other clauses

Positionable elements

Specifications

- **Type of position**
  - `position: absolute|relative`

- **Position**
  - `top: size`
  - `left: size`
- **Dimension**
  - `width: size`
  - `height: size`
- **Visibility**
  - `visibility: hidden|inherit|show`
- **Clip**
  - `Clip: rect(top|right|bottom|left)`
  - Clip an element (leaving it in place)
- **Superposition**
  - `z-index: n`
  - Decides the superposition ranking

More stuff

Fonts

- **Multiple declaration**

```
p { font-family: Gotham, 'RM Neue', sans-serif; }
```

- **Import fonts**

```
@font-face {
  font-family: 'Museo';
  src: url('/fonts/museo.otf') format ('opentype');
}
```

Misc

```
div {
  box-shadow:  10px 10px 25px #ccc;
  text-shadow: 10px 10px 25px #ccc;
  border-radius: 20px;
  border-image: url('border.png') 30 round 5 repeat;
}
```

Advanced

- Pagination
- Multiple column
- Tooltips
- Graphic transformations
- Animation

Libraries

Popper

Bootstrap

Inclusion, CDN, starter template

12. Typescript

Language

Polyfilling and transpiling

- **Polyfilling**: methodology used as a backward compatibility measurement
  - **Polyfill** / **Polyfiller**: piece of code (or plugin) that provides the technology that the developer expect the browser to provide natively
- **Transpiler**: tool that transforms code with newer syntax into older code equivalents (transpiling)

Programming language

TypeScript
- Is compiled into **JS** (**transpiled**)
- Gives the **capabilities** required to develop **large scale applications** using JS
- Is a **superset** of JS
  - It includes the entire JS programming language together with additional capabilities
- Allows to use JS as if it was a **strictly type** programming language
  - Allows to specify the **type of the variables**
  - Allows to define **classes** and **interfaces**
  - Variables can **not** be treated as **dynamic type** variable
- In general, nearly every code in **JS can be included** in code **in TS**
- Compiling TS into JS leads to a clean simple ES3 compliant code runnable in **any web browser**
- Compiler will still try to **execute** the code when **errors** occures

Variable typing

```
var id:number = 221255;
var aname:string = "Dorothea";
var tall:boolean = true;
var names:string[] = ['pippo', 'pluto', 'minnie'];
var temp:any = <number|string|bool|Obj>;  // dynamic type
```

JS `"use strict"`

- It is **forbidden** to use **undeclared variables**
  - `x=3;  // must become var x=3;`
- Any assignment to the following will throw an **error**
  - Non-writable property
  - Getter-only property
  - Non-existing property
  - Non-existing variable
  - Non-existing object
- **Deleting** a variables, objects, functions is **not allowed**
- **Function parameter names be unique**
  - In normal code the last duplicated argument hides previous identically named arguments
    - Those previous arguments remain available through `arguments[i]`

Functions

Function typing

Define the type of the variables

```
function name():type {}
```

Number of params

When calling a **function** passing over **arguments** (!~ JS)
- The **number of arguments** must match the number of the **parameters**
- Otherwise **compilation error**

Optional params

- Adding the **question mark** to the name of a parameter will turn that parameter into an optional one
- The optional parameters should be **after any required** one
  - They should be the **last ones**

```
function sum(a:number, b:number, c?:number):number {
    var total:number = 0;
    if (c !== undefined) { total += c; }
    return total + a + b;
}
```

Default params

- Function definition can specify **default values** for any of its parameters
- If an argument is **not passed** over to the parameter then the default value we specified will be set instead

```
// TS
function sum(a:number, b:number, c:number=0):number {
    return a + b + c;
}
```

```
// JS
"use strict";
function sum(a, b, c) {
    if (c === void 0) { c = 0; }
    return a + b + c;
}
```

Rest params

- Function can have an **arbitrary number of params** (~ `main` in Java)

```
// TS
function sum(...numbers:number[]):number {
    var total:number = 0;
    for (var i = 0; i < numbers.lenght; i++) {
        total += numbers[i];
    }
    return total;
}
```

```
// JS
function sum() {
    var numbers = [];
    var total = 0;
    for (var _i = 0; _i < arguments.lenght; _i++) {
        numbers[_i] = arguments[_i];
    }
    for (var i = 0; i < numbers.lenght; i++) {
        total += numbers[i];
    }
    return total;
}
```

Classes

Scheme

```
class Car {
    // field
    engine:string;
    // constructor
    constructor(engine:string) {
        this.engine = engine;
    }
    // method
    disp():void {
        console.log("Engine is: " + this.engine);
    }
}
```

Constructor

- All classes have a **default constructor**
- Defining a **new constructor**, the default one will be **deleted**
- There is **no polymorfism**
- New definition can specify each one of its **parameters** with an **access modifier**
    - This indirectly define those parameters as **instance variables**

Access modifiers

The available access modifiers are **`private`**, **`public`** and **`protected`**

- **`public`** is the default one
- **Not specified** access modifier are **`public`**

Instance variables and methods

- **Variables** are usually **declared before constructor**
  - **Definition** includes **three parts**
    - **Access modifier** (optional)
    - **Identifier**
    - **Type** annotation
- **Methods** are declared **without** using the **`function`** keyword
  - Declaration is the same as variables, but type is optional

Static variables and methods

- Static variables and methods are **defined** with the **`static`** keyword
- **Accessing** static variables and methods is done using the **name of the class**

Class inheritance

```
class Shape {
    Area:number;
    constructor(a:number) {
        this.Area = a;
    }
}
class Circle extends Shape {
    disp():void {
        console.log("Area of the circle:" + this.Area);
    }
}
var obj = new Circle(223);
obj.disp();
```

Type assertion

Type assertion

- Is like a **type cast** in other languages
  - But performs **no** special **checking**
- Is used purely by the **compiler**
  - It has **no runtime impact**

Generics

Class related issues

- TS does **not** support **multiple inheritance**
  - **`super`**
- Classes implement **interfaces**
  - `class Student implements Iprintable {...}`

Interfaces

- Interfaces can be used as **data type definition**
- They fully **disappear in JS**

```
interface IPerson {
    firstName: string,
    lastName: string,
    sayHi: () => string
}
var customer:IPerson = {
    firstName: "Tom",
    lastName: "Hanks",
    sayHi: ():string => { return "Hi there!"; }
}
```

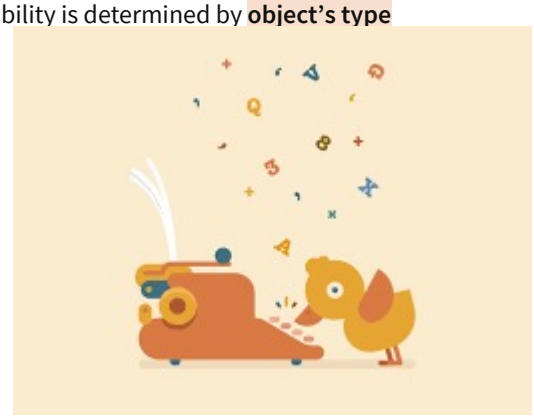
Interfaces multiple inheritance

```
interface IParent1 { v1:number; }
interface IParent2 { v2:number; }
interface Child extends IParent1, IParent2 { }
var Iobj:Child = { v1:12, v2:23 };
console.log("value 1: " + this.v1 + " value 2: " + this.v2);
```

Duck typing

Duck test:
" If it walks like a duck and it quacks like a duck, then it must be a duck "

- Duck typing is an application of the duck test to **determine if an object is suitable for a particular purpose**
  - With **normal typing**, suitability is determined by **object's type**



Duck inheritance

```
class Vehicle {
    public run(): void { console.log('Vehicle.run'); }
}
class Task {
    public run(): void { console.log('Task.run'); }
}
function runTask(t:Task) {
    t.run();
}
runTask(new Task());
runTask(new Vehicle());
```

Avoiding duck inheritance

13. JSP

Basics

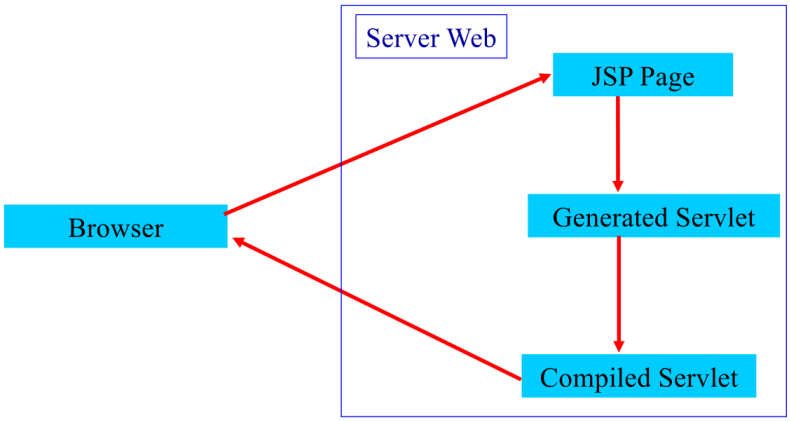
JSP Technology

- A technology similar to PHP, ASP and ASP.net
- Java-based
- Dual to **servlets**

Simple.jsp

```
<%@ page import=java.util.* %>
<html>
<body>
  <% out.println(Calendar.get(Calendar.HOUR_OF_DAY)); %>
</body>
</html>
```

Lifecycle



Nuts and bolts

- **Syntactic elements**

```
<%@ directives %>     →  Interaction with the CONTAINER
<%! declarations %>   →  In the initialization of the JSP
<% scriptlets %>      →  In the service method
<%= expressions %>    →  Eq. to <% out.println(expression %)>
<jsp:actions/>
<%-- Comment --%>
```

- **Implicit Objects**
  - `request`
  - `response`
  - `session`
  - `application`
  - `out`
  - `config`
  - `page`
  - `pageContext`

Scriptlets

Block of **Java code** executed during the **request-processing** time

- In **Tomcat** all the scriptlets gets put into the **`service()`** method of the **servlet**
  - They are therefore **processed for every request** that the servlet receives

```
<% z = z + 1; %>
<%
  // Get the Employee's Name from the request
  out.println("<b>Employee: </b>" +
  request.getParameter("employee"));
  // Get the Employee's Title from the request
  out.println("<br><b>Title: </b>" +
  request.getParameter("title"));
%>
```

Declarations

Block of **Java code** used to

- **Define** class-wide **variables** and **methods** in the generated servlet
- They are initialized when the **JSP page is initialized**

```
<%! DECLARATION %>
```

```
<%! String nome = "pippo"; %>
<%! public String getName() { return nome; } %>
```

Directives

Used as a **message mechanism** to pass information **from JSP** code **to container**

```
<%@ DIRECTIVE {attribute=value} %>
```

- Main directives
  - `page`
  - `include`
    - For including other `STATIC` resources at compilation time
- Main attributes

```
<%@ page language=java session=true %>
<%@ page import=java.awt.*,java.util.* %>
<%@ page errorPage=URL %>
<%@ page isErrorPage=true %>
```

Standard actions

Tags that affect
- **Runtime behavior** of JSP
- **Response** sent back to client

```
<jsp:include page="URL" />
<jsp:forward page="URL" />
```

- `include` for including `STATIC` or `DYNAMIC` resources at request time

Predefined objects

???	???	Note
`out`	`Writer`	
`request`	`HttpServletRequest`	
`response`	`HttpServletResponse`	
`session`	`HttpSession`	
`page`	`this`	Nel Servlet
`application`	`servlet.getServletContext`	Area condivisa tra i servlet
`config`	`ServletConfig`	
`exception`		Solo nella `errorPage`
`pageContext`		Sorgente degli oggetti

`request`

```
<%@ page errorPage="errorpage.jsp" %>
<html>
<head>
  <title>UseRequest</title>
</head>
<body>
  <%
    // Get the User's Name from the request
    out.println("<b>Hello: "
                + request.getParameter("user")
                + "</b>");
  %>
</body>
</html>
```

`include`

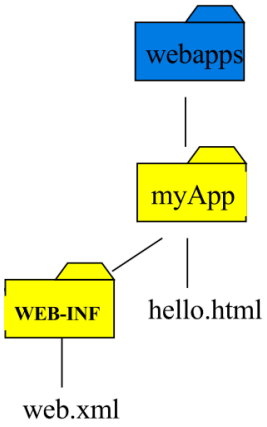
- **`<%@include@%>` directive**
  - Includes **literal text** as is in the JSP page
  - Is **not** intended for use with content that **changes at runtime**
  - Occurs only when the servlet implementing JSP page is **built** and **compiled**
  - e.g.: `<%@ include file="myfile.jsp" @%>`
- **`<jsp:include>` action**
  - Allows to include either **static or dynamic content** in the JSP page
  - **Static** pages are included as if the `<%@include@%>` **directive** had been used
  - **Dynamic** included files, though, act on the given **request** and return results that are included in the JSP page
  - Occurs **each time** the JSP page is **served**

WebApps: Tomcat configuration

Static pages

`web.xml` file must be provided

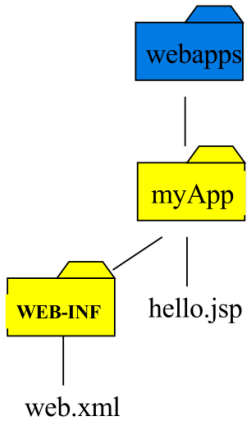
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD WebApplication 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
</web-app>
```



JSP pages

Same **procedure** used for **static pages**

- In **myApp folder** it's deposited the **JSP files**
- On **Tomcat server**, the **URL for** **hello.jsp** file becomes
  `WEB-INF http://machine/port/myApp/hello.jsp`
- The `WEB-INF` directory can be empty



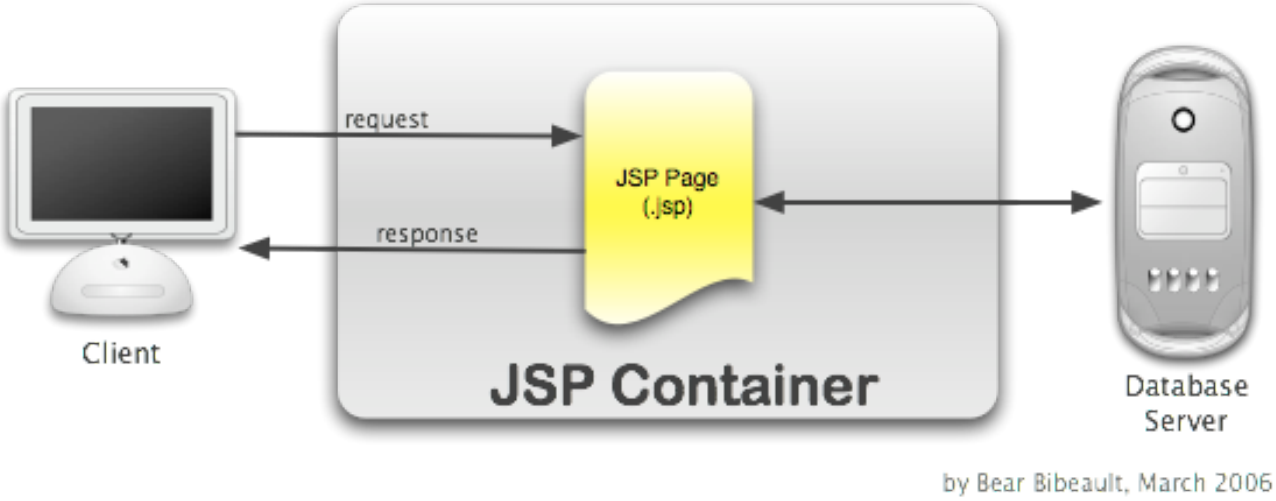
JSP in action

Generated code

Usage: MVC pattern

Wrong simple approach

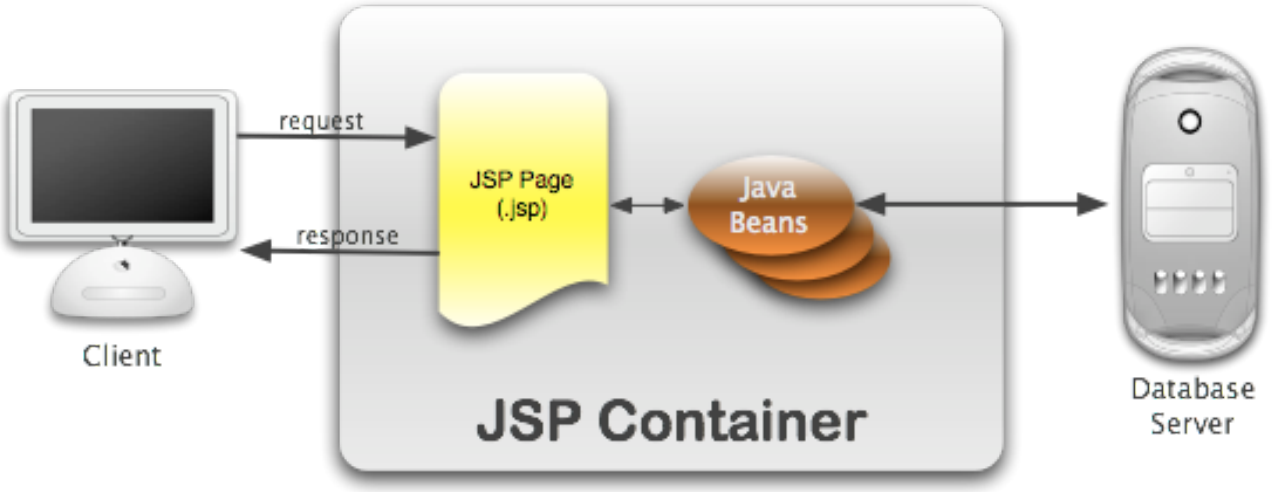
Control, data management and presentation in the **same page**



by Bear Bibeault, March 2006

Better solution

Control logic is delegated to **Java classes**



by Bear Bibeault, March 2006

Java bean

Java class that

- Provides a public no-argument **constructor**
- Implements **`java.io.Serializable`**
- Follows JavaBeans **design patterns**
  - Has **set/get** methods for **properties**
  - Has **add/remove** methods for **events**
- Is **thread safe/security conscious**
  - Can **run** in an applet, application, servlet

```
public class SimpleBean implements Serializable {
    private int counter;
    SimpleBean()           { counter = 0; }
    int getCounter()       { return counter; }
    void setCounter(int c) { counter = c; }
}
```

Standard actions involving beans

```
<jsp:useBean id="name" class="fully_qualified_pathname"
             scope="{page|request|session|application}" />
<jsp:setProperty name="nome" property="value" />
<jsp:getProperty name="nome" property="value" />
```

Example
Examples

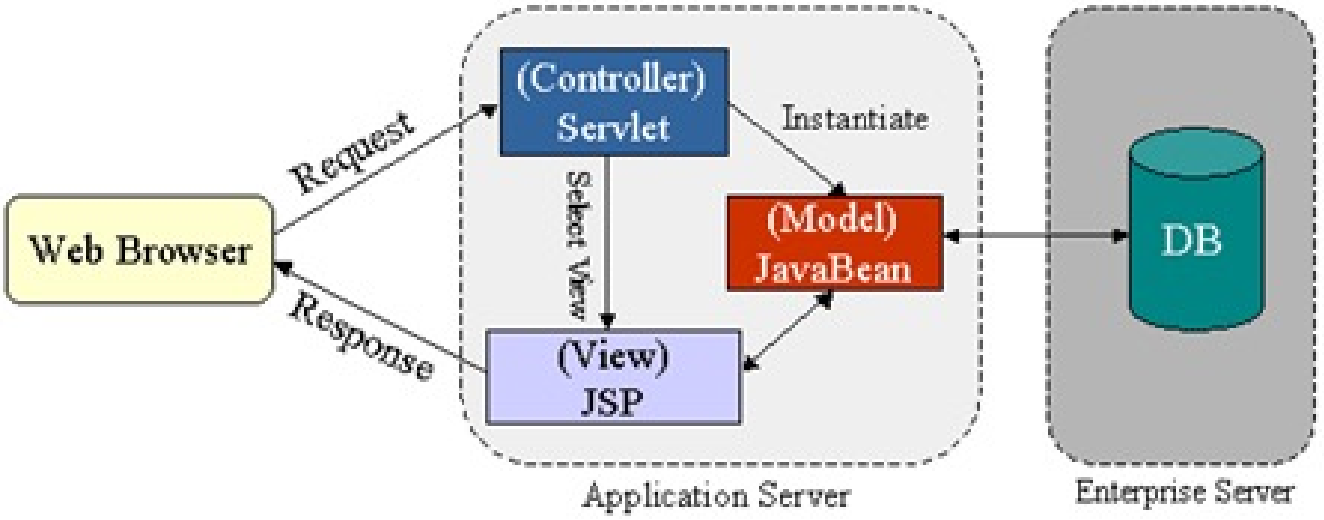
Scoping

```
<jsp:useBean id="myBean"
             class="beans.BeanA"
             scope="{session|application|request}"/>  <!--
```

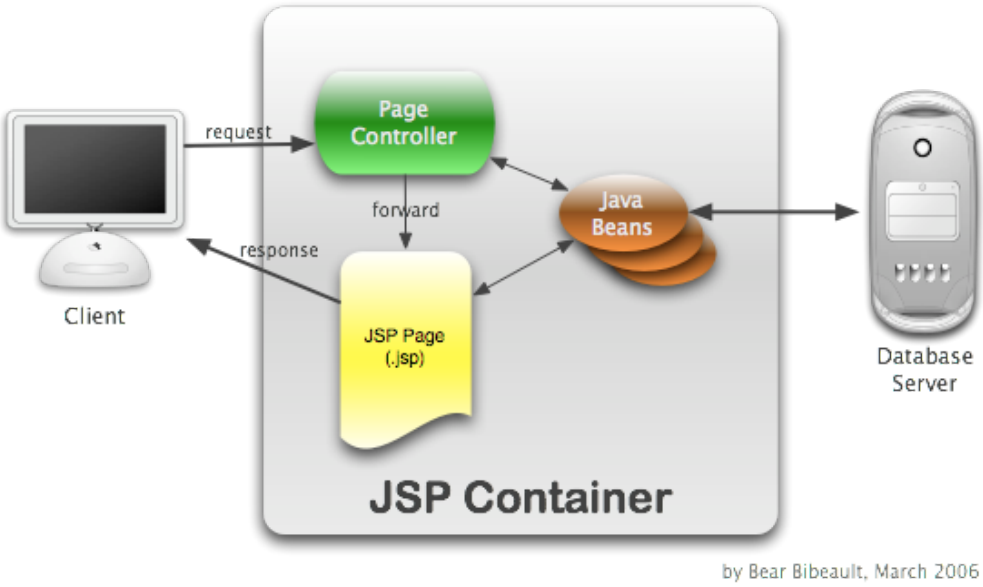
JSP	Servlet
	`Session session=request.getSession();`
`session`	
	`BeanA x=(BeanA)session.getAttribute("myBean");`
	`ServletContext c=request.getServletContext();`
`application`	
	`BeanA x=(BeanA)context.getAttribute("myBean");`
`request`	`BeanA x=(BeanA)request.getAttribute("myBean");`

JSP models

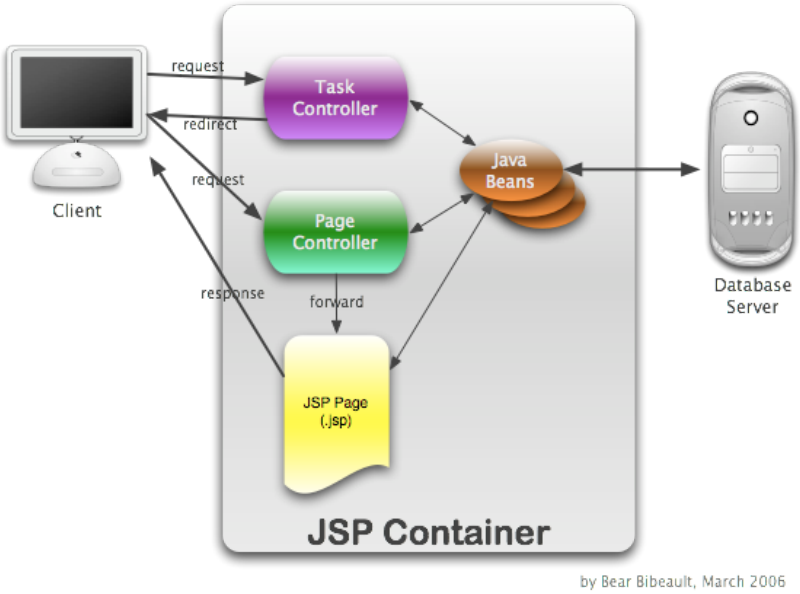
Model 2



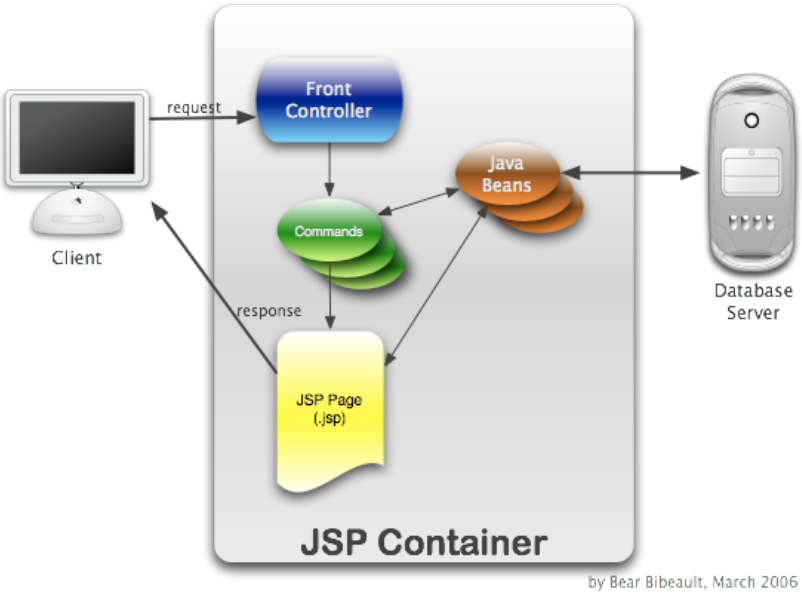
Only for presentation



Separate post and get



Front controller pattern



Best practices

- **Don't overuse Java** code in HTML pages
- Choose the right **include mechanism**
  - **Static data** such as headers, footers, and navigation bar content is best kept in **separate files** and not regenerated dynamically
  - Once such content is in separate files, they can be **included in all pages** using one of the following mechanisms
    - Include **directive**: `<%@ include file="filename" %>`
    - Include **action**: `<jsp:include page="page.jsp" />`
- **Don't mix** business **logic** with **presentation**
  - **JSP** code should be limited to **front-end presentation**
- Use **filters** if necessary
- Use a **database** for persistent information
  - Use **connection pooling**

Examples

Exercise

Filters

AOP

Aspect-Oriented Programming attempts to aid the **separation of concerns**
- Specifically **cross-cutting** concerns, as an advance in **modularization**
- **Logging** and **authorization** offer two examples of crosscutting concerns
  - Logging strategy necessarily affects every single **logged part** of the system
  - Logging thereby **crosscuts all logged classes and methods**
  - Same is true for **authorization**

Filters

> **Filter**: class that preprocess/postprocess request/response

Object that performs **filtering tasks** on either
- **Request to a resource**
  - Servlet or static content
- **Response from a resource**

Implementation: **`javax.servlet.filter`**
- Filters perform **filtering** in the **`doFilter`** method
- Every **`Filter`** has access to a **`FilterConfig`** object from which it can obtain its **initialization parameters**
  - Reference to **`ServletContext`** which it can use to **load resources** needed for filtering tasks
- They provide the ability to **encapsulate recurring tasks in reusable units**

Filters are **configured**
- In the **deployment descriptor** of a web application
- Via **annotation**

Applications

- **Authentication**: blocking requests based on user identity
- **Logging and auditing**: tracking users of a web application
- **Image conversion**: scaling maps, and so on
- **Data compression** making downloads smaller
- **Localization**: targeting the request and response to a particular locale
- **XSL/T**: transformations of XML content-Targeting web application responses to more that one type of client
- Encryption, tokenizing, triggering resource access events, mime-type chaining, caching

Filtering API

- Defined by `Filter`, `FilterChain`, `FilterConfig` **interfaces**
  - In **`javax.servlet`** **package**
- **Definition** means implementing the **`Filter`** **interface**

The most important **method** is **`doFilter`**
- Is passed **request**, **response**, and **filter chain** objects
- Actions
  1. **Examine** the request headers
  2. **Customize** the request/response objects
  3. **Invoke** the next entity in the filter chain
     - Configured in the WAR
     - By calling `doFilter` on the chain object
       - Passing in the request and response it was called with, or the wrapped versions it may have created

Filter methods

```
public void doFilter (ServletRequest, ServletResponse, FilterChain)
```

- Called by the container each time a request-response pair is passed through the chain due to a **client request for a resource** at the end of the chain

```
public void init(FilterConfig filterConfig)
```

- Called by web container to indicate to a filter that it is being **placed into service**

```
public void destroy()
```

- Called by web container to indicate to a filter that it is **taken out of service**

Example

Configuration

Filters Application Order

The **order** of filter-mapping elements **in `web.xml`** determines the order in which the **web container applies the filter to the servlet**
Example

Sessions and parameters

18. Access to DB

JDBC in servlets

Installation and usage

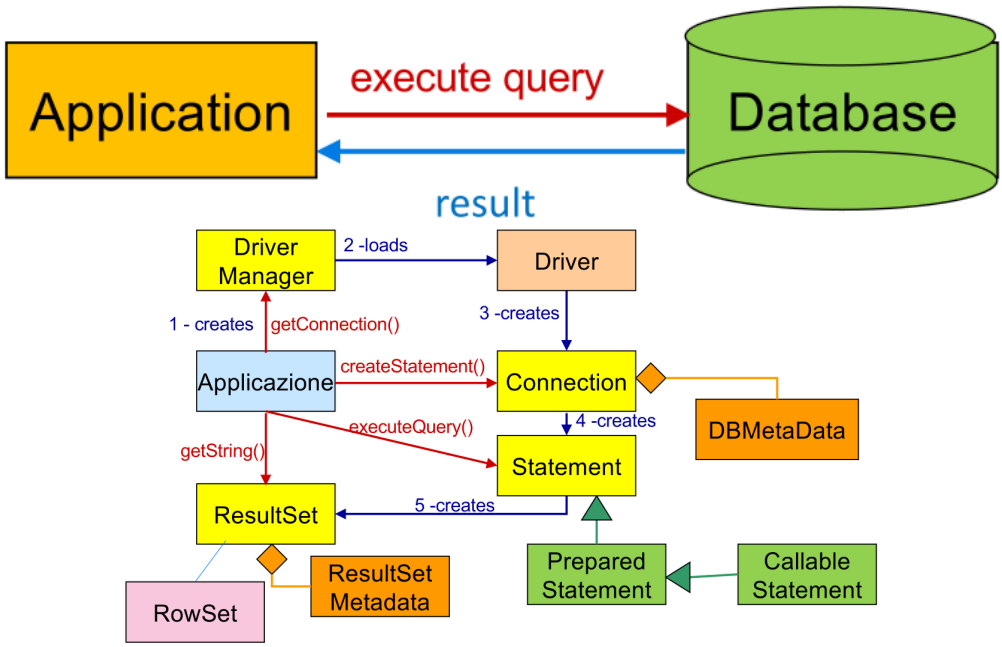
0. **Install a driver** on the machine
1. **Load the driver**
2. **Open a connection**
3. **Create Statement**
4. **Retrieve Values**

Always **catch exceptions**

- JDBC tracks warnings and exceptions generated by DBMS and Java compiler
- To show them, **print** them out from a **catch block**

Access database from Java

`java.sql` Object Model



Get the driver

```
Class.forName("org.apache.derby.jdbc.ClientDriver");
```

Reminder: **`Class.forName`**

```
static Class forName(String className);
```

Return
`Class` object associated with the class or interface with the given string name

```
Object o = Class.forName("java.lang.String").newInstance();
Object o = new String();  // equivalent
```

Load the driver

```
Connection con = DriverManager.getConnection(
    url, "myLogin", "myPassword");
```

The last part of the JDBC url supplies information for identifying the **data source**
Third party JDBC driver

Create statement

`Statement` object **sends SQL statement to DBMS**

- **`executeQuery`** for a **`SELECT`** statement
- **`executeUpdate`** for statements that create/modify **tables**

```
Statement stmt = con.createStatement();
stmt.executeUpdate(query);
```

```
CREATE TABLE COFFEES (
    COF_NAME VARCHAR(32),
    SUP_ID INTEGER,
    PRICE FLOAT,
    SALES INTEGER,
    TOTAL INTEGER
)
```

Retrieving values

JDBC returns results in a **`ResultSet` object**

```
String query = "SELECT COF_NAME, PRICE FROM COFFEES";
ResultSet rs = stmt.executeQuery(query);
```

To **access values**, go to each row and retrieve the values according to their types
The method **`next`**

- Moves a **cursor** to the **next row**
- Makes that line the one to **operate** on

Cursor

- **Initially** it is positioned just **above the first row** of a `ResultSet` object
- The **first call** to `next` moves the cursor to the **first row** and makes it the current
- **Successive** invocations move the cursor down **one row at a time** to bottom
- JDBC 2.0 API can move the cursor backwards and to absolute/relative positions

Prepared statements

`PreparedStatement` object will reduce execution time of the **multiple repetition** of the same `Statement` object
Main feature: it is given an **SQL** statement when it is **created**

- Advantage: the statement will be sent to the **DBMS when** it will be **compiled**
- Result: the object contains an SQL statement that has been **precompiled**
    - When the `PreparedStatement` is **executed**, the DBMS can just **run** the SQL statement without having to compile it first

```
PreparedStatement updateSales = con.prepareStatement(query);
updateSales.setInt(1, 75);
```

```
UPDATE COFFEES
SET SALES = ? WHERE COF_NAME LIKE ?
```

Callable statements

Stored procedure: *group of SQL statements that form a logical unit and perform a particular task*

- Used to **encapsulate a set of operations/queries** to execute on database server
- To call them use **`CallableStatement`** (is-a `PreparedStatement`)
- WARNING: stored procedures move the business logic **within the DB**

Netbeans configuration

Create DB

1. Create **WebApp**
2. In Project Properties add Library “**Java DB**”
3. In Services create **Database**
    1. Right cllck on JavaDB
    2. Choose “create”

    3. Add requested info
    4. Click on properties
    5. Make the DB Location within your project
4. Rename the created connection (if needed), right-click it and "**connect**"
5. Create **table**
6. **Populate** DB

Images

Create servlet

1. Go to the **project**
2. Create a **servlet** called "TheServlet"
3. Edit it as shown in the slides

Manage DB connections

Connection management

- **`perServlet`**
  - Methods
    - **Create** connection in **`init`**
    - **Close** it in **`destroy`**
  - Many connections **simultaneously open**
  - **Concurrency bottleneck**
    - Connection's methods are **synchronozed**
- **`perRequest`**
  - Methods
    - Create connection in **`doXXX`**
    - Or `processRequest`
  - **Lots of open/close** (slow)
- **`perSession`**
  - Every user has **one connection**, and reuses it
  - Potentially **many** connections, with **low usage** each
  - **Sessions** remain alive as long as the **connection** lives
  - `HttpSessionBinding` interface to **monitor closing of sessions** due to timeout
- **Connection pooling**
  - **Servlets share** a set of existing **connection**
  - More complex
  - Infrastructures exist to allow it
- **One connection per Web App**

Data Access Object