# Web programming

**Henning Schulzrinne**  
Dept. of Computer Science  
Columbia University

---

## Web services vs. web programming
- web services = remote procedure call  
- we saw SOAP  
- structured data (XML)  
- methods and responses  
- generally, for machine consumption  
- web programming → generate HTML pages  
  - for humans  
  - often, database-driven  
  - replacement for IBM 3270 terminals ...

---

## Client vs. server programming
- Execute code on client:  
  - download Java applet → self-contained programming environment  
  - JavaScript (aka ECMAScript):  
    - modify and get values from HTML ("document object model" – DOM)  
- Execute code on server → generate document  
  - state maintenance (HTTP stateless)  
    - login, shopping cart, preferences

---

## Taxonomy

<table>
<thead>
<tr>
<th>server</th>
<th>client</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSI, ASP, PHP, JSP, CFM</td>
<td>JavaScript</td>
</tr>
<tr>
<td>server API (NSAPI), CGI, servlets</td>
<td>Java applets, plug-in</td>
</tr>
</tbody>
</table>

---

## Example: JavaScript - cookies

```javascript
var expires = new Date();
var today = new Date();

function setCookie(name, value, hours) {
  var expire = new Date();
  expire.setTime(expires.getTime() + (1000 * 60 * 60 * hours));
  document.cookie = name + "=" + escape(value) + "; expires=" + expire.toGMTString();
}

setCookie("cookie", "my data", 1);
```

---

```javascript
var exp = new Date();
exp.setTime(today.getTime() - 86400 * 365);

document.cookie = name + "=" + " " + "; expires=" + exp.toGMTString();

setCookie("cookie", "my data", 1);
```
### JavaScript - DOM

```javascript
function tz (f,v) {
  var t = -2;
  switch (f[v].value) {
    case "US": t=328; break;
    case "CT": t=0; break;
    case "UT": t=-2; break;
    ....
  }
  if (t != -2) {
    f.form.timezone.options[t].selected = true
  }
}
```

### Web as RPC

- request = HTTP GET, PUT
- response (result): headers + body
- object identifier ~ URL
- typed data (XML) vs. HTML
- from constant → mostly constant → completely on-demand

### Server-side include

- .shtml documents (or configured by default for all .html documents)
- include in HTML/XML comments
  ```html
  <!-- element attribute=value
  attribute=value ... -->
  ``
- limited scripting: if/else, include, exec, variables
- primarily for conditional inclusion, boilerplate
- security issues: exec

### SSI example

- Columbia CS home page
  ```html
  <html>
  <head><title>Computer Science: Welcome</title>
  <script language="JavaScript">
  var section = "Home";
  var subsection = "";
  <script /
  </head>
  <body>
  <!-- #sect element value="HOME" -->
  <!-- #include file="top.html" -->
  <!-- #include file="home.txt" -->
  <table>
  <!-- #include file="bottom.txt" -->
  </html>
  ```
  ```html
  <html>
  <head>
  <h1>SSI Test</h1>
  The document was last modified on
  <!-- #lastmod file="$DOCUMENT_NAME" -->
  and has <!-- #fsize
  file="$DOCUMENT_NAME" -- > bytes.
  </head>
  <h2>Environment</h2>
  <!-- #printenv -->
  </pre>
  ```
cgi arguments
- application/x-www-form-urlencoded format
  - space characters → "+"
  - escape (%xx) reserved characters
  - name=value pairs separated by &
- GET: foo.cgi?name=John&gender=male&family=5&city=kent&city=miami&other=abc12345678&nick=xyz
- POST: include in body of message

Web state
- State:
  - stateless
  - state completely stored on client
  - state referenced by client, stored on server (most common)
- Mechanisms:
  - hidden form fields
  - URL parameters
  - cookies (HTTP headers)

cgi forms
- single form per submission
  - form action=scripturl method=GET|POST
  - form fields:
    - input type="text" name="text1" size=30
    - maxlength=15 value=Initial text
    - input type="hidden" name="state" value="secret"
    - input type="radio" name="radio" value=NYC checked
    - input type="radio" name="radio" value=KQED
    - input type="submit" value="Submit"

cgi mechanics
- either called .cgi in HTML directory or stored in cgi-bin
  - in CS, both /home/alice/html/foo.cgi or /home/alice/secure_html/foo.cgi work
  - executable (script file)
  - runs as nobody or as owning user (~user/mycgi.cgi)
  - store secret data off the document tree!

SQL interface
- Most common web model:
  - cgi script (or Java servlet) accesses database
  - database via TCP connection (ODBC, JDBC, script)
- n-tier model:
  - delegate "business logic" to RPC-based server
- XML-based model:
  - generate XML render via XSLT

Tcl cgi example
```tcl
set env(LD_LIBRARY_PATH) /home/bgs/syn5/11b
load Env(LD_LIBRARY_PATH)/libbash.so
lappend auto_path /home/bgs/html/adams3
lappend auto_path /home/bgs/11b
package require cgi

cgi_eval {
  sql connect dbhost.columbia.edu dbuser secret
cgi_body {
    ...
    sql disconnect
  }
}```
Tcl cgi

cgi_body {
    html "database view"
    set conflist [sql "SELECT conference, name, url, logo
                FROM conference WHERE conference=$c"]
    foreach conf $conflist {
        maplist $conf c name url logo
        table_row {
            td "$name"
            td "$url"
        }
    }
}

Python for cgi

- Handles processing cgi variables
- Need to generate HTML by print
- But separate object-oriented routines

```python
#!/usr/local/bin/python
# /opt/CICPython/bin/python2.2
import os, string, sys
from types import ListType

print "Content-Type: text/html" # HTML is following
print # blank line, EOF
```

cgi python

```python
print "<title>Python cgi script<title>
print "<body>
print "<h3>Python script</h3>
print "Before script"
print sys.path
try:
    import cgi
except:
    print "error", sys.exc_info()[0]
    # only for Python 2.2!
import cgitb; cgitb.enable()
```

cgi python

```python
form = cgi.FieldStorage()
if not (form.has_key("name")):
    print "<form action=cgi.cgi method=get>"
    print "<input type=text name=name size=30>"
    print "<input type=submit value=Submit>"
    print "</form>"
else:
    print "<p name="", form["name"].value
    print "</body>"
```

SQL interface

- SQL = more-or-less standard retrieval language for databases
- Examples:
  - Oracle
  - Sybase
  - IBM DB/2
  - Microsoft SQL Server
  - mySQL
  - PostgreSQL

SQL architecture

- Library interface
  - Proprietary
  - JDBC, ODBC
- Driver that connects (via TCP) to database
  - Same or different host
- Issue queries, get results
- Modify content
- Transactions
### SQL basics
- **Relational database**: tables with labeled columns, combined into database
- **Columns** are atomic types:
  - `create table person (` 
    - `person` integer `auto_increment` 
    - `primary key`, 
    - `name varchar(40)`, 
    - `state enum ('NY', 'AL', ...)`, 
    - `biography text`, 
    - `verified date`, 
    - `index(name)`) 

### SQL basics: joins
- Join two tables that have a common value ("product")
- **Example**: `SELECT lastname, city.name FROM person.cities WHERE city.zip = person.zip AND lastname = 'John'`

### SQL Python interface
```python
import MySQLdb
import MySQLdb.cursors

try:
    db = MySQLdb.connect('grandcentral',
                         user='cs3995',
                         passwd='cs3995',
                         db='grades')
except MySQLdb.Error, e:
    print 'Error %d: %s' % (e.args[0], e.args[1])
    sys.exit(1)

    c = db.cursor()
    c.execute("SELECT ... FROM ...")
    results = c.fetchall() # list of tuples
    c.close()
```

### SQL
- **Get description of table**:
  ```bash
  $ mysql -h grandcentral -u cs3995 -p
  mysql> use grades
  mysql> describe students;
  +----------+-------+------+-----+---------+-------+------------------------------------------------------------------+
  | Field    | Type  | Null | Key | Default | Extra |
  +----------+-------+------+-----+---------+-------+------------------------------------------------------------------+
  | firstname | text   | YES  | | NULL | | "first name of student"
  | lastname  | text   | YES  | | NULL | | "last name of student"
  | points    | int(11)| YES  | | NULL | | "points of student"
  +----------+-------+------+-----+---------+-------+------------------------------------------------------------------+
  3 rows in set (0.00 sec)
  ```
**SQL Python interface**

- Results are just tuples, with fields in order of table definition
- can also fetch one row at a time:
  ```python
c.execute("SELECT firstname, lastname FROM students ORDER BY lastname")
  print("%s, %s" % student)  
  while student = c.fetchone():  
    if student == None: break  
    print("%s, %s" % (student[0], student[1]))
  print "%d rows were returned" % c.rowcount
```

**Python SQL - dictionary cursor**

- Map rows to dictionary elements instead of list elements:
  ```python
c = db.cursor(MySQLdb.cursors.DictCursor)
c.execute("SELECT firstname, lastname FROM students")
results = c.fetchall()  
for row in results:  
  print "%s, %s" % (row["firstname"], row["lastname"])
print "%d rows were returned" % c.rowcount
```

**Servlet life cycle**

- server application loads `ServletClass`
- creates instance via no-args constructor
- servers call servlet’s `init()` method
- server calls `service(req, res)` method for each request (often, with class name as URL), possibly concurrently
- servers calls `destroy()` on shutdown

**HTTP requests as servlets**

- HTTP method GET, PUT, POST, ... → `doGet`, `doPut`, `doPost`
- subclass of `HttpServlet` overrides default implementation

**Servlet example**

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloClientServlet extends HttpServlet{
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException{
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.print("<HTML><HEAD><TITLE>Hello Client</TITLE></HEAD>" +
    "<BODY><H1>Hello Client</H1><BODY>" +
    "<HTML>" +
    out.close();
  }
  public String getServletInfo() {
    return "HelloClientServlet 1.0 by Stefan Zeiger";
  }
}
```

**2-tier architecture**

- "client-server", "fat client"
- e.g., ODBC on client (PC), accessing SQL database
- business logic on PC
- (-) transport data across network
- (-) need applications for each platform
- (-) need to update applications on many desktops
n-tier architecture

- client tier:
  - receives user events (keyboard, mouse)
  - presentation of data
  - user interface
  - e.g., Java applets, web browser, thin client application
- application-server tier:
  - “business logic” → actual data processing, algorithms
  - can be component-based (Java Beans)

Data-server tier
- data storage
- relational and legacy databases
- all tiers could run on same machine, but usually separated
- HTTP (or SOAP) from client to server
- Corba or SOAP or remote-SQL between server tiers

Advantages:
- independent of storage model
- simpler authentication to database