ISAPI & ASP

Antonio Lioy < lioy@polito.it >

english version created by
Marco D. Aime < m.aime@polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

ISAPI

- **Internet Server API**
- **Microsoft's proprietary mechanism to create dynamic pages with IIS:**
  - every ISAPI application is a DLL
  - … loaded into memory at the first request
  - … kept into memory to satisfy other requests
  - same memory space of IIS (bi-directional communication through specific objects shared between IIS and the ISAPI application)
  - can be removed from memory only by the system administrator
- **the ISAPI application must be thread-safe**

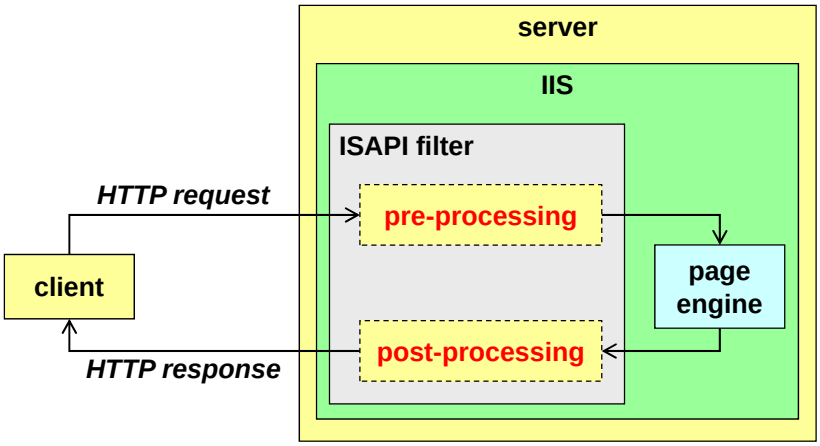
ISAPI

- **it is Microsoft's alternative to CGI**
- **CGI creates one process per each web request**
  - consumes a lot of resources (CPU and RAM) and processes have difficulties to communicate among themselves and with the web server
  - robust (crash of a process, not of the whole server)
- **ISAPI has better performance since:**
  - uses threads and synchronization mechanisms to better exploit resources
  - works in the same memory space of IIS
  - risk of blocking the whole IIS server

ISAPI applications: filters and extensions

- **ISAPI filters act on the HTTP channel:**
  - can perform request pre-processing
  - can perform response post-processing
  - e.g. compfilt.dll (HTTP compression), md5filt.dll (HTTP digest authentication), sspifilt.dll (SSL)
- **ISAPI extensions are associated to pages with a specific (file) extension:**
  - elaborate the page and return the resulting HTML code to the HTTP engine
  - e.g. asp.dll (ASP pages), ssinc.dll (SSI)

ISAPI filter



Usage of ISAPI filters

- **they can for example:**
  - redirect the request to balance the load among multiple servers
  - add security functionalities / log
  - adapt the response to client capabilities (supported HTML and script version)

Configuring ISAPI extensions

- **based on URL extension**
- **use MMC to manage a virtual directory of IIS**
- **in Properties / Home Directory / Application Settings / Configuration / Mappings, it is possible to associate:**
  - extensions (e.g. ".asp")
  - applications (e.g. asp.dll)
  - accepted HTTP commands (e.g. GET, HEAD, POST)
- **also possible to associate specific web pages for various application errors**

ASP

- **Active Server Pages**
- **extension of ISAPI (asp.dll, around 300 KB) associated by default to files with ".asp" extension**
- **allows inserting within the HTML page:**
  - server-side scripts in various interpreted languages (default: VBscript; JS also possible)
  - some IIS variables
  - interaction with built-in ASP objects

ASP (Active Server Pages)

ASP is a technology
(not a scripting languge)
proprietary to Microsoft
that allows interpreting
server-side scripts

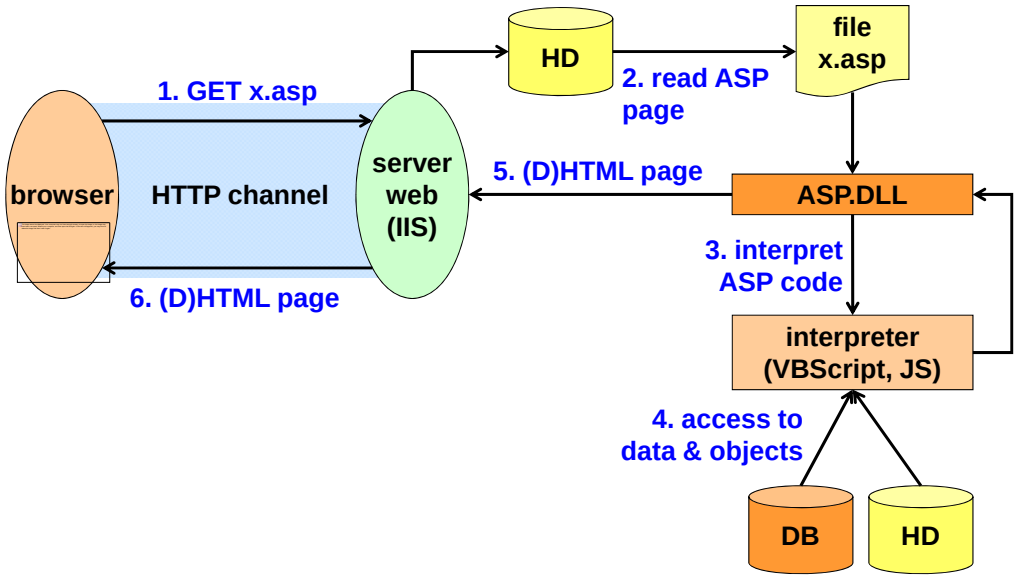
ASP

- **ASP is a technology:**
  - provided by Microsoft Internet Information Server (IIS)
  - for server-side scripting to develop dynamic web applications
- **ASP pages contain scripts which are elaborated by the ASP ISAPI extension in the web server**
- **the result of the elaboration is sent to the client**
- **ASP is independent from the scripting language**

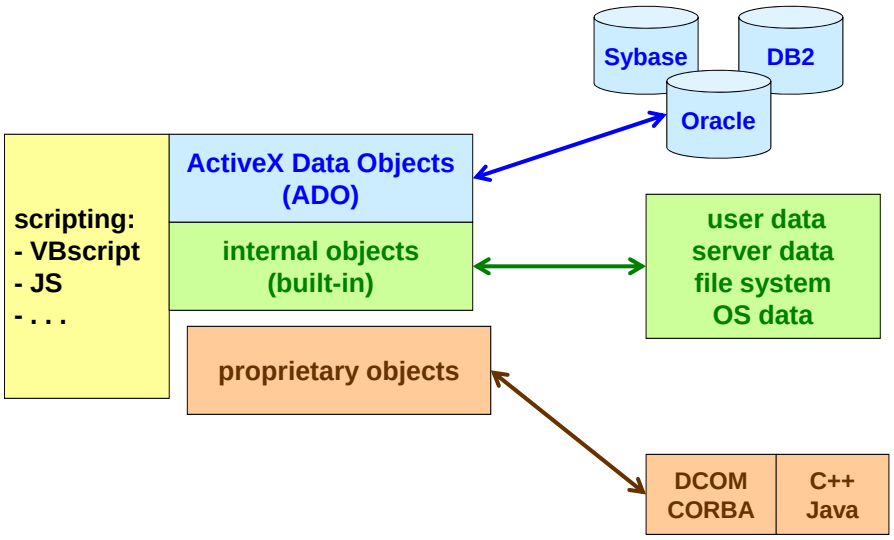
ASP: architecture

- **ASP engine:**
  - ASP.DLL
  - ISAPI extension interprets the files .asp
  - multithreaded service (ISAPI extension)
- **ASP file:**
  - text file with .asp extension
  - consists of standard HTML and script code enclosed within the special characters "<%" and "%>"

ASP: architecture



ASP: data access



Scripting languages

- **IIS natively interprets two languages:**
  - JScript / JavaScript
  - VBScript (default language)
- **possible to add PerlScript, Python, REXX and others**
- **to specify the interpreter use:**

```
<%@ LANGUAGE="JavaScript" %>
```

```
<%@ LANGUAGE="VBScript" %>
```

Example of ASP file (with JS)

```
<html>
<head>
  <title>Greetings</title>
</head>
<body>
<%@ LANGUAGE="JavaScript" %>
<%
for (var i=1; i<=5; i++) {
  Response.write ("<h"+i+">Ciao!</h"+i+">");
}
%>
</body>
</html>
```

Computed result

```
<html>
<head>
  <title>Greetings</title>
</head>
<body>
<h1>Ciao!</h1>
<h2>Ciao!</h2>
<h3>Ciao!</h3>
<h4>Ciao!</h4>
<h5>Ciao!</h5>
</body>
</html>
```

dynamically generated part

JS: Enumerator object

- to loop on a ASP Collection (= associative array), we cannot use the “for-in” JS control
- we must use the Enumerator object (MS-specific)

```
e = new Enumerator(collection)
e.moveFirst();
while (!e.atEnd())
{
     Response.write(e.item());
     e.moveNext();
}
```

JS: Enumerator object, methods

- **atEnd( )**
  - returns a Boolean value indicating if the end of the collection has been reached
- **moveFirst( )**
  - updates the current element to point to the first element
- **moveNext( )**
  - moves the current element to the next one inside the collection
- **item( )**
  - returns the current element

JS: example of Enumerator

```
var e = new Enumerator(Request.ServerVariables);
e.moveFirst();
while (!e.atEnd())
{
     Response.Write(e.item()+"<BR>");
     e.moveNext();
}
```

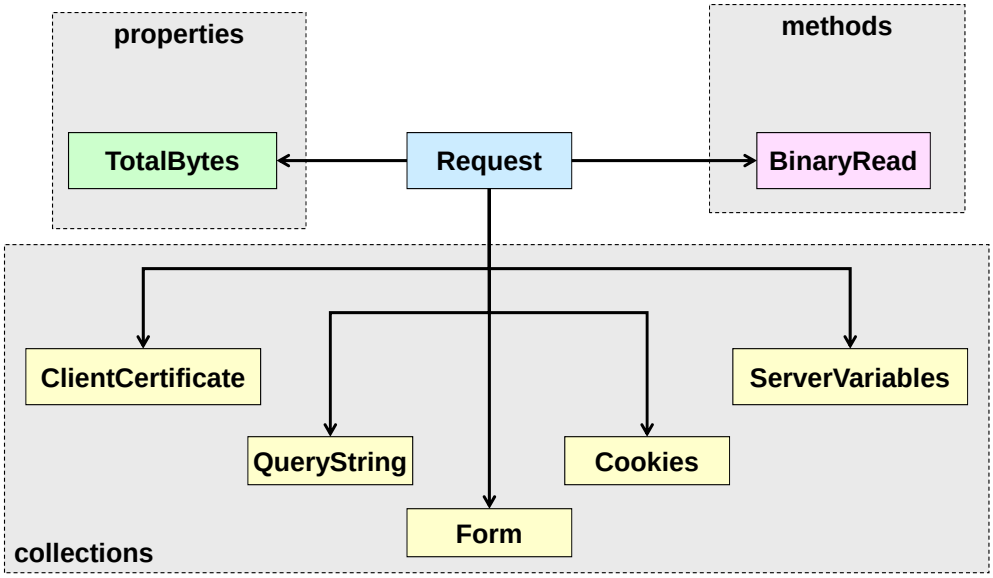
Internal ASP objects

- **objects that do not require to be instantiated**
- **internal objects:**
  - Request
  - Response
  - Application
  - Session
  - Server
- **they are ASP objects, available in both scripting languages (Javascript and VBscript) but unfortunately MS documentation is almost exclusively for VBScript …**

Request

- **information received from the client:**
  - form content sent with GET/POST
  - HTTP protocol headers
  - cookies (values sent by the browser)

Request



Request collections

- **ClientCertificate**
  - extracts values from the extensions of the X.509 digital certificate sent by the client
- **QueryString**
  - extracts the values of parameters sent via GET
- **Form**
  - extracts the values of parameters sent via POST
- **Cookies**
  - extract the values of application cookies

```
<% user = Request.Cookies("username") %>
```

Request.QueryString: example

```
<form action="http://a.b.com/x.asp" method="get">
  <input type="text" name="name">
  <input type="Submit">
</form>
```

↓

```
GET /x.asp?name=MARA HTTP/1.1
Host: a.b.com
```

↓

```
(x.asp)

. . .
n = Request.QueryString("name")
```

MARA

Request collections

- **ServerVariables**
  - extracts the values of the HTTP headers
  - the following examples return the browser type and the DNS name of the server (as written in the URL)

```
<% b = Request.ServerVariables("HTTP_USER_AGENT")%>
```

```
<% serv = Request.ServerVariables("HTTP_HOST")%>
```

Server Variables: example

```
<table border=1>
<tr>
 <td><b>Server variable</b></td>
 <td><b>Value</b></td>
</tr>
<%
e = new Enumerator(Request.ServerVariables)
for ( ; !e.atEnd(); e.moveNext()) {
%>
<tr>
<td><%= e.item() %></td>
<td><%= Request.ServerVariables(e.item()) %></td>
</tr>
<% } %>
</table>
```

JS

Request: properties

- **TotalBytes**
  - read-only
  - specifies the number of bytes sent by the client within the request body

```
<% bytecount = Request.TotalBytes %>
```

Request: methods

- **BinaryRead**
  - reads data sent by the client with POST

Important: form parameters in ASP

- the form fields extracted on the server side via Request.QueryString or Request.Form ...
- ... are not strings (as instead happens when reading them inside client-side scripts)
- ... but they are "ASP objects"
- they should be automatically converted to the required type for the given operation, but sometimes the automatic mechanism fails and the result is not the expected one

(suggestion) always convert explicitly form fields to the desired object type

Form parameters in ASP: example

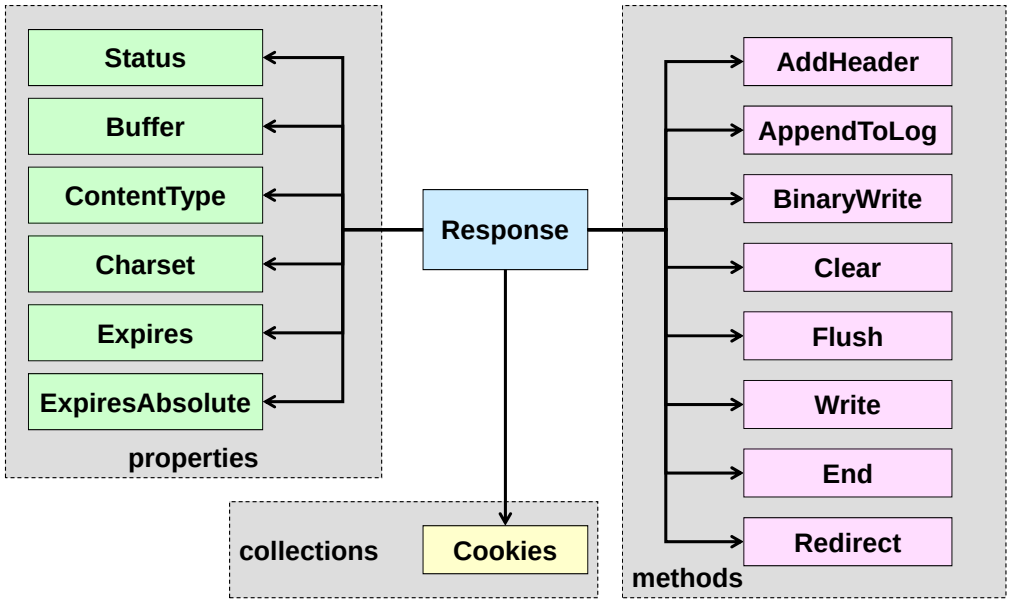
```
<form action="http://a.b.com/x.asp" method="get">
  <input type="text" name="name">
  <input type="text" name="age">
  <input type="Submit">
</form>
```

```
. . .
var n = String( Request.QueryString("name") )
var a = Number( Request.QueryString("age") )
. . .
```

Response

- sends information to the client
- configures cookies through the collection Cookies

Response



Response: properties

- **Boolean Buffer**
  - if set to TRUE, the server does not send data to the client until the script computation is fully terminated
- **String ContentType**
  - sets the client MIME type (e.g. “text/html”)
- **String Charset**
  - sets the response charset (e.g. “iso-8859-1”)

Response: properties

- **Int Expires**
  - sets the validity time (in minutes) of the page in client cache (default = 10)
- **Date ExpiresAbsolute**
  - sets the absolute validity time (i.e. expiration date and time) of the page in client cache
- **String Status**
  - sets the HTTP status sent by the server to the client
  - must contain both the numeric code and the comment (e.g. “401 Unauthorized”)

Response: methods

- **AddHeader (String HeaderName, String HeaderValue)**
  - adds the given HTTP header
- **AppendToLog (String logText)**
  - adds a line in the log file of the web server
- **BinaryWrite (Array Data)**
  - sends binary data to the client, useful for example to send images or Word files
- **Clear**
  - empties the output buffer

Response: methods

- **End**
  - terminates the script
- **Flush**
  - sends the content of the output buffer to the client
- **Redirect (String URI)**
  - redirects the client to the given URL
- **Write (data)**
  - write data to the HTML stream sent to the client
  - data must not contain “%>”, to be replaced with “%\>”

Response: methods

- **the following two constructs are equivalent**

```
<% Response.write("Ciao"); %>

<% ="Ciao" %>
```

Response: Cookies collection

- **to create a cookie with the given name and value:**
  - Response.Cookies("*cookiename*") = "*cookievalue*"
- **instead of creating multiple cookies, you can insert multiple values in the same cookie specifying a set of "keys" upon its creation:**
  - Response.Cookies("*cookiename*")("*key*") = "*keyval*"
  - the key:value pairs will be inserted within the cookie using the urlencoded encoding
- **keys are a Collection themselves**
- **to know if there are any keys, use the property:**
  - HasKeys
    - (read-only) returns the number of keys

Properties of cookies in ASP

- **Expires = *vardate***
  - expiration date and time for the cookie
  - if not set, the cookie is "volatile"
  - attention! set it through setVarDate(*date*)
- **Secure = true | false**
  - transmitted only inside secure channels (SSL, TLS)
- **Path = *pathprefix***
  - transmitted only to pages with the specified prefix
- **Domain = *domain***
  - transmitted only to pages in the specified domain
- **NOTE: properties of the cookie, not of single keys**

Response: example of cookie configuration

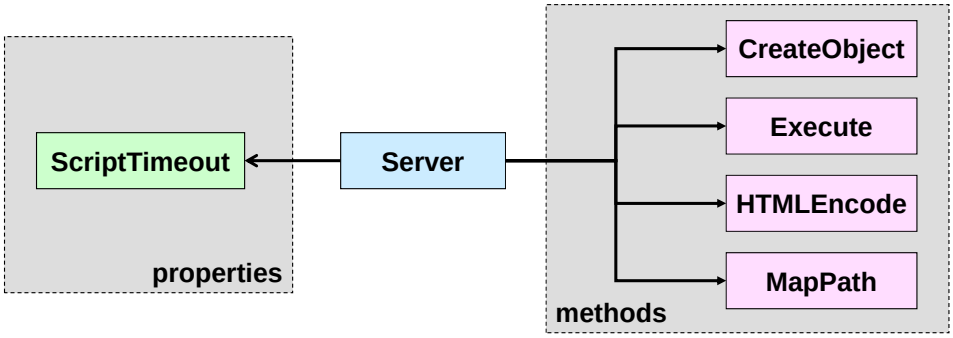
```
var Nome = Request.Form("yourname");
var Cognome = Request.Form("yourfamilyname");

Response.Cookies("myapp")("nome") = Nome;
Response.Cookies("myapp")("cognome") = Cognome;

var expire = new Date();
expire.setMonth(expire.getMonth()+2);

Response.Cookies("myapp").Expires =
  expire.setVarDate();

Response.Cookies("myapp").Domain = "polito.it";
```

Example: list the parameters of a form

```
// independent by GET or POST
// lists name and value of every field

var m = Request.ServerVariables("REQUEST_METHOD")

if (m == "GET")
 var form_data = Request.QueryString
else // POST
  var form_data = Request.Form

var x = new Enumerator(form_data)
for ( ; !x.atEnd(); x.moveNext())
  Response.write(x.item()+"="+form_data(x)+"<br>")
```

Server

- **provides methods and properties to access server's resources**
- **used to instantiate components**
  - components are object packages



Server: properties

- **Int ScriptTimeout**
  - sets a timeout (in seconds) for executing the script

Server: methods

- **Execute (String)**
  - executes the .asp file located at the given string (relative or absolute path; if absolute, the script must belong to the same application)
- **Component CreateObject (String)**
  - instantiates the given component (can be any component installed on the server, e.g. ActiveX)

```
<% MyAd =
new Server.CreateObject("MSWC.AdRotator"); %>
```

JS

Server: methods

- **String HTMLEncode (String)**
  - encodes the given string into HTML using the appropriate escape characters (e.g. è)
- **String MapPath (String)**
  - maps the given virtual directory to the corresponding physical directory of the server (important for selecting files or DBs)
- **String URLEncode (String)**
  - encode the given string in the way appropriate to be used as URL (e.g. %20)

Server objects - examples

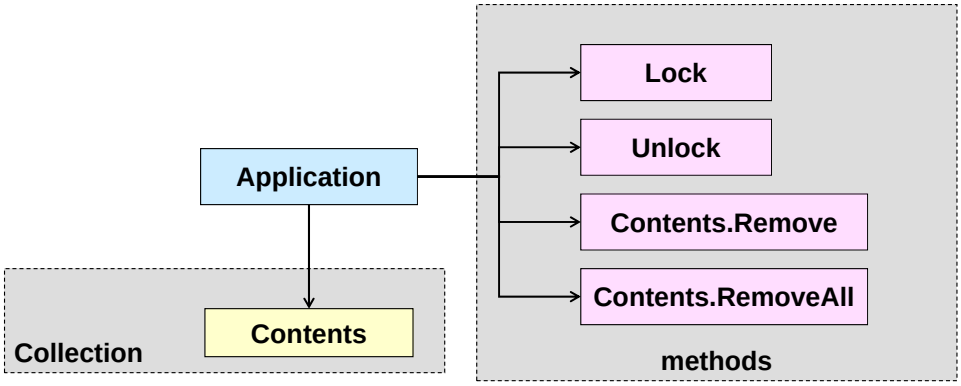
- **automatic computation of the date of last modification of a file:**

```
<%
var fso =
  Server.CreateObject("Scripting.FileSystemObject")
var file = fso.GetFile(
  Server.MapPath("avvisi.txt"))
var date = new Date(
  Date.parse(file.DateLastModified))

Response.write (
  "Document: " + file.name +
  " / Last update: " + date.toGMTString())
%>
```

Application

- an application is a set of IIS resources, configurable by the administrator
- by default, there is one single application that include all the ASP pages
- object shared by all the users (=browsers connecting to any ASP page included in the application)
- information lives until the IIS server remains active
- used to share information among different clients requesting resources belonging to the same application

Application



Application: collections

- **Contents**
  - collection of the variables of the application

```
<% Application("visitors") = 0 %>
```

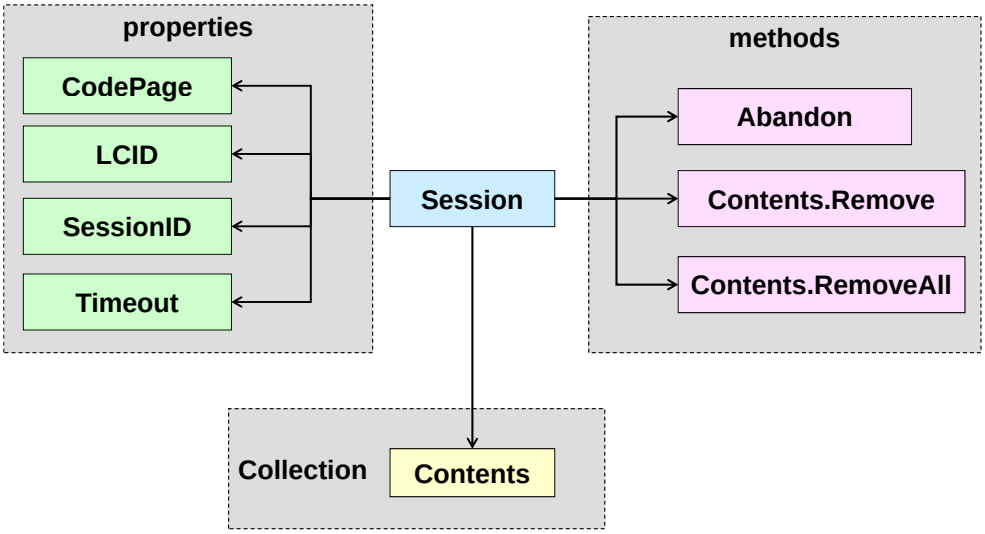
Application: methods

- **Lock**
  - blocks writing on the collection (synchronisation)
- **Unlock**
  - unlocks writing on the collection
- **Contents.Remove (variable_name)**
  - removes the variable from the collection
- **Contents.RemoveAll**
  - removes all the variables from the collection

Session

- stores information on the active session for a specific client
- every connecting client generates automatically a new instance of Session
- managed through the volatile cookie ASPSESSIONID (index to the session data available inside IIS server's RAM)

Session



Session: collections

- **Contents**
  - collection of the variables for the session

```
<%  Session("name") = "Antonio"  %>
```

Session: properties

- **SessionID**
  - contains the session identifier (uint32)
- **Int Timeout**
  - specifies the idle time (in minutes) for the session (default: 10')
  - too short values (e.g. less than 4') make you lose the state information
  - too long values (e.g. greater than 20') overload the server that is required to store a lot of active sessions in memory
  - set to the maximum time required by the user to switch from a page to the next one

Session: methods

- **Abandon**
  - destroys the session (and thus deletes all the associated Contents)
- **Contents.Remove (variable_name)**
  - removes the given variable from the collection
- **Contents.RemoveAll**
  - removes all the variables from the collection

File Global.asa

- **the file Global.asa contains events related to applications and sessions**
- **on starting a new session, the server runs the procedure Session_OnStart**
- **on closing a session, it runs the procedure Session_OnEnd**
- **on starting an application (after the IIS server has restarted), it runs the procedure Application_OnStart**
- **on closing an application, it runs the procedure Application_OnEnd**

File Global.asa

```
<script language="JScript" runat="server">

function Application_OnStart(){
      Application("visitors")=0;
}
function Application_OnEnd(){
}
function Session_OnStart(){
      Application.Lock();
      Application("visitors")=Application("visitors")+1;
      Application.UnLock();
}
function Session_OnEnd(){
      Application.Lock();
      Application("visitors")=Application("visitors")-1;
      Application.UnLock();
}
</script>
```

JS

ASP @ directives

- **@LANGUAGE**
  - sets the scripting language
- **@ENABLESESSIONSTATE**
  - set to FALSE to disable ASP sessions (to save execution time and memory)
- **@CODEPAGE**
  - sets the default codepage
- **@LCID**
  - sets the format to show date and time
- **@TRANSACTION**
  - sets the transaction support required/used

#include

- ASP understands a single SSI directives: #include
- the external file is included before passing the page to the ASP interpreter: it must be located within the HTML part but can contain both HTML and ASP code
- with the tag "virtual", use absolute pathnames, with / corresponding to the web server root
- with the tag "file", use relative pathnames starting from the directory containing the file with #include
- syntax:

```
<!--#include virtual="pathname_assoluto" -->
<!--#include file="pathname_relativo" -->
```

<script> in ASP

- instead of using <% and %> you can delimit the ASP code and specify that it is scripting code:
  - to be executed on the server-side
  - with the interpreter for the given language
- IIS5 introduced the SRC parameter to include ASP code from external files
  - VERY useful to include external JS functions
  - http://support.microsoft.com/kb/224963
- syntax:

```
<script language="javascript" runat="server"
  [ src="..." ] >
```

ASP example (with JS)

```
<form method="post" action="e1.asp" name="F1">
Nome: <input type="text" name="nome"><br>
Email: <input type="text" name="email"><br>
<input type="submit" name="Submit" value="Invia">
</form>
```

form.html

```
<%@ LANGUAGE="JavaScript"%>
<%
if (Request.Form("nome")=="" || Request.Form("email")==="") {
  Response.Redirect("form.html");
}
else {
  Response.Write ("Nome: " + Request.Form("nome")
    + "<br> E-mail: " + Request.Form("email"));
}
%>
```

e1.asp

References for ASP

http://msdn.microsoft.com/library/default.asp?
  url=/library/en-us/iissdk/iis/iis_web_pages.asp

http://aspjavascript.com

http://www.w3schools.com/asp/
(attention: in VBScript)

http://www.comptechdoc.org/
independent/web/cgi/javamanual/