---ꕥ📖ꕥ---

Giáo trình

ASP.NET

CÁC PHƯƠNG PHÁP LƯU TRỮ TRẠNG THÁI.

Nội dung:

- Tổng quan về các phương pháp lưu trữ trạng thái trong ASP.NET.
- ASP.NET Application.
- Managing Application State.
- Managing Session State.
- ViewState.
- Cách cấu hình thẻ <sessionState> trong file web.config.
- Cấu hình web server chạy trên nhiều server (đáp ứng yêu cầu của nhiều client truy cập cùng lúc).

I. Tổng quát về các phương pháp lưu trữ trạng thái trong Web:
I.1 Các tuỳ chọn phía Client:
I.1 Các tuỳ chọn phía server:
II. ASP.NET Application:
II.1 ASP.NET Application là gì?
II.1 Tạo một Application:
II.1 Thời gian sống của một Application:
II.1 Các chú ý về luồng (thread):
III. Managing Application State:
III.1 Application State:
III.1 Sử dụng:
IV. Managing Session State:
IV.1 Xác định một session:
IV.1 Lưu trữ Session State:
IV.1 Sử dụng:
V. Managing Client-side Cookies:
VI. ViewState:
VI.1 Khái niệm:
VI.1 Sử dụng:
VI.1 Các hạn chế của ViewState:
VI.1 Làm tăng hiệu suất khi sử dụng ViewState:
VI.1 Disabling ViewState:
VII. Cấu hình tag <sessionState> trong file web.config:
VII.1 Khái niệm:
VII.1 Sử dụng:
VIII. Cách cấu hình cho web server chạy trên nhiều server:

1 Tổng quát về các phương pháp lưu trữ trạng thái trong Web:

- Ta cần lưu trữ các thông tin có trong trang web do người sử dụng cung cấp để sử lí trong một số trường hợp cần thiết. ASP.NET cung cấp rất nhiều phương pháp để lưu trữ data trong quá trình hoạt động của trang web. Việc chọn lựa phương

pháp nào là tuỳ thuộc vào ứng dụng web của ta và nên dựa vào một số tiêu chí đánh giá sau:

+ Số lượng thông tin cần lưu trữ nhiều hay ít?

+ Các client có chấp nhận các cookies (có thể chiếm chổ trong bộ nhớ của họ) không?

+ Người phát triển trang web muốn lưu thông tin ở client hay server?

+ Hiệu suất của trang web mà bạn mong đợi (tốc độ).

ASP.NET cung cấp rất nhiều tuỳ chọn cho việc lưu trữ data ở các phía client và server.

- Các tuỳ chọn phía Client:
  + Sử dụng các thuộc tính **ViewState**.
  + Sử dụng các trường ẩn (**hidden field**).
  + Sử dụng **Cookies**.
  + Sử dụng QueryString
- Các tuỳ chọn phía server:
  + Application State
  + Session state
  + Database

Sau đây là các mô tả về các tuỳ chọn trên.

1.1 Các tuỳ chọn phía Client:

Việc lưu trữ data ở phía client không làm tốn nhiều tài nguyên của server. Do lưu trữ ở client nên sẽ không được an toàn về mặt data nhưng hiệu suất của server sẽ cao do các yêu cầu về tài nguyên sẽ không quá cao. Tuy nhiên do phải gửi data về client để lưu nên sẽ bị hạn chế về số lượng data mà ta gửi đi.

View State:

- Mỗi trang web trong ASP.NET tồn tại sẵn một thuộc tính là ViewState dùng để chứa các giá trị của một trang. Ta có thể dùng ViewState để lưu một giá trị nào đó của một trang.
- Ưu điểm:
  + Không yêu cầu tài nguyên của server.
  + Cài đặt đơn giản.
  + Tự động ghi nhớ trạng thái trang và control.
  + Làm tăng khả năng bảo mật do dữ liệu được ghi đã được mã hoá.
- Khuyết điểm:
  + Hiệu suất thấp, bởi vì các giá trị được lưu giữ trong chính trang đang sử dụng nó do đó nếu data lớn sẽ làm cho trang bị chậm khi hiển thị hay trao đổi data với server.
  + Bảo mật. ViewState được lưu trữ trong các field ẩn trong trang. Mặc dù đã được biến đổi định dạng (mã hoá) nhưng nó vẫn có thể bị phá hoại.

Hidden fields:

- Ta có thể lưu trữ thông tin trên các trường ẩn của trang web. Nên sử dụng các trường ẩn khi có nhu cầu lưu các thông tin nhỏ nhưng có tần suất thay đổi nhiều từ phía client. ASP.NET cung cấp control HtmlInputHiden cho các trường này.
- Khi sử dụng các trường ẩn thì phải dùng phương thức Post để chuyển dữ liệu
- Ưu điểm:
  + Không yêu cầu tài nguyên từ server.
  + Được hỗ trợ rộng rãi từ hầu hết các trình duyệt (browser) và client.
  + Cài đặt đơn giản.
- Khuyết điểm:
  + Tính bảo mật. Các trường ẩn có thể bị nhìn thấy khi view source của trang.
  + Bị hạn chế về cấu trúc lưu trữ. Không thể lưu trữ một cấu trúc trong các field ẩn trừ khi ta phải ghép lại thành một chuỗi và có cơ chế phân tích ngược trở lại thích hợp.
  + Hiệu suất. Giống như View State do lưu trữ cùng với trang.

Cookies:

- Cookies được dùng để lưu trữ các thông tin nhỏ thường xuyên thay đổi phía client. Các thông tin này sẽ được truyền cùng với yêu cầu về cho server.
- Ưu điểm:
  + Không yêu cầu tài nguyên phía server. Cookies được lưu trữ phía client và được đọc từ server khi yêu cầu gửi đi
  + Đơn giản. Cookies có cấu trúc dựa trên text và các khoá là các text.
  + Có thể cấu hình thời gian tồn tại. Cookies có thể mất khi phiên trình duyệt kết thúc hay có thể tồn tại trên máy client.
- Khuyết điểm:
  + Hạn chế về kích thước. Hầu hết các trình duyệt hạn chế kích thước của Cookies là 4096 bytes, nhưng cũng có một số trình duyệt mới hỗ trợ 8192 bytes.
  + Client có thể cấu hình để không cho phép các cookies ghi xuống máy của họ. Do đó ta không thể sử dụng Cookie.
  + Tính bảo mật. Cookies là mục tiêu để phá hoại. Người sử dụng có thể chỉnh sửa cookies trên máy của họ dẫn đến các khả năng về bảo mật và các ứng dụng dựa trên cookies có thể bị hỏng. More Introduction to Web Application Security
  + Tính lâu dài. Các cookies thường được dùng cho từng user, nội dung sẽ khác nhau đối với các user. Trong hầu hết trường hợp việc định danh (*identification)* sẽ được chú trọng hơn là định quyền (*authentication)*, do đó chỉ lưu trữ một số thông tin như user name, acount name hoặc GUID để xác định các user.

Query String:

- Query String là các thông tin được nối vào sau chuỗi URL của trang web. Ta có thể dùng query string để submit dữ liệu về một trang nào đó thông qua URL.

Query String cung cấp cách thức đơn giản để truyền thông tin giữa các trang với nhau nhưng không đọc các thông tin này từ một trang được submit cho server.

- Ưu điểm:
  + Không yêu cầu tài nguyên server.
  + Được hỗ trợ rộng rãi.
  + Cài đặt đơn giản. ASP.NET cung cấp các phương thức hỗ trợ đầy đủ để làm việc với Query String: HttpRequest.Params.
- Khuyết điểm:
  + Tính bảo mật. Thông tin trong query string hiện trực tiếp lên giao diện của trình duyệt (đi kèm với địa chỉ URL).
  + Hạn chế về kích thước. Hầu hết trình duyệt chỉ hỗ trợ tối đa 255 kí tự cho URL.

Tuỳ chọn	Dùng khi
View State	Ta cần lưu trữ thông tin được giữ lại cho chính trang đó.
Hidden fields	Ta cần lưu trữ thông tin có kích thước nhỏ cho một trang để gửi lại cho trang đó hay cho các trang khác và vấn đề bảo mật không được chú ý. Ta chỉ có thể dùng hidden field cho các trang được submit về server.
Cookies	Ta cần lưu trữ các thông tin nhỏ trên máy client và vấn đề bảo mật không được chú ý nhiều.
Query string	Dùng khi chuyển một lượng nhỏ thông tin giữa các trang và vấn đề bảo mật không được quan tâm.

1.2 Các tuỳ chọn phía server:

Việc lưu trữ thông tin phía server làm tăng khả năng bảo mật hơn phía server nhưng nó sẽ yêu cầu nhiều hơn tài nguyên của server có thể dẫn tới nhiều vấn đề khi thông tin lưu trữ lớn.

ASP.NET cung cấp một vài các tuỳ chọn để lưu trữ data trên server.

Application State:

- ASP.NET cung cấp các phương thức lưu trữ thông tin toàn cục cho toàn ứng dụng thông qua lớp HttpApplicationState. Các biến Application state là toàn cục cho ứng dụng ASP.NET.
- Ta có thể lưu trữ các thông tin lên các biến Application state để sau đó xử lý bởi server.
- Dữ liệu trong các biến Application State được chia sẽ cho nhiều session và thường không đổi.
- Ta cần phải có các chuyển đổi kiểu thích hợp trước khi sử dụng các giá trị trong các biến application.
- Ưu điểm:
  + Dễ cài đặt.

+ Có phạm vi toàn cục. Các biến Application được truy suất trong tất cả các trang của ứng dụng do đó giá trị lưu trữ trong các biến application là duy nhất cho toàn ứng dụng trái với việc lưu trữ thông tin trong session state hay trong các trang riêng lẽ.

- Khuyết điểm:

+ Phạm vi toàn cục. Tính toàn cục đôi khi cũng là một bất lợi. Các biến lưu trữ trên application state chỉ toàn cục cho process mà application đó đang chạy và mỗi process của application sẽ có một giá trị khác nhau. Do đó ta không thể dựa trên application state để lưu trữ các giá trị duy nhất hay cập nhật trên các ứng dụng được cấu hình chạy trên nhiều server hay nhiều process khác nhau.

+ Tính bền vững. Các thông tin lưu trữ sẽ bị mất nếu web server sử lí nó bị ngưng chạy hay trục trặc.

+ Yêu cầu tài nguyên. Application state dùng bộ nhớ của server nên có khả năng ảnh hưởng đến hiệu suất của server cũng như của ứng dụng.

- Việc thiết kế và cài đặt tốt các biến Application có thể làm tăng hiệu suất của ứng dụng. Ví dụ, đặt những thứ thường sử dụng, những dữ liệu tương đối tĩnh trong application state có thể làm tăng hiệu suất vì làm giảm một lượng lớn yêu cầu truy suất database. Tuy nhiên có một sự cân bằng về hiệu suất. Khi thông tin lưu trữ nhiều thì sẽ làm giảm hiệu suất của server và ứng dụng. Bộ nhớ sử dụng cho các biến Application sẽ không được giải phóng cho tới khi nào mà nội dung biến thay đổi hoặc bị xoá. Do đó các biến application chỉ nên dùng khi thao tác với các biến dữ liệu nhỏ và ít khi thay đổi.

Session State:

- ASP.NET cung cấp các phương thức để thao tác với các biến mức phiên (session) thông qua lớp HttpSessionState. Dữ liệu ở các biến này chỉ có nghĩa cho từng session.
- Dữ liệu lưu trữ trong session được quản lí bởi server, có thời gian sống ngắn và chỉ đại diện cho từng session riêng biệt.
- Ưu điểm:

+ Dễ dàng cài đặt.

+ Các event của session rõ ràng. Ta có thể bắt các event của session để sử lí trong chương trình.

+ Tính lâu bền. Dữ liệu đặt trên các biến session có thể vẫn tồn tại sau khi IIS restart hay tiến trình hiện tại restart bởi nếu data được lưu trữ ở một tiến trình khác (ta cần cấu hình lại file web.config để có được chức năng trên). Nhưng khi client đóng trình duyệt thì dữ liệu sẽ mất.

+ Khả năng chạy trên nhiều cấu hình khác nhau của web server. Có thể dùng trên web server cấu hình trên nhiều server hay nhiều tiến trình.

+ Session state làm việc với các trình duyệt không hỗ trợ cookies, mặc dù session state làm việc dựa trên cookies (để lưu trữ và truyền session ID giữa server và client).

- Khuyết điểm:

+ Hiệu suất. Các biến session state tồn tại trong bộ nhớ cho đến khi chúng bị thay thế hoặc xoá bỏ và do đó có thể làm giảm hiệu suất của server. Các biến session state chứa những khối data lớn có thể ảnh hưởng đến quá trình load dữ liệu của server.

Database:

- Trong một số trường hợp, ta có nhu cầu dùng các hỗ trợ database để quản lý các state trên trang web. Thông thường việc hỗ trợ database thường được dùng kết hợp với các cookies và các session state. Một vài web site thương mại điện tử sử dụng cơ sở dữ liệu quan hệ để lưu trữ các thông tin vì các lí do:
  + Tính bảo mật (***Security***)
  + Tính cá nhân (***Personalization***)
  + Tính nhất quán (***consistency***)
  + Tính khai thác dữ liệu (***Data mining***)

  Tính bảo mật (Security): Client cung cấp tên account và password để đăng nhập vào một site trên trang logon. Sau đó site sẽ tìm kiếm trên database xem cilent đó có được phép đăng nhập vào site hay không?. Nếu tìm thấy user đó trên database web site sẽ tạo một cookies chứa một số định danh duy nhất cho user đó trên máy của client. Site sẽ gán quyền truy cập cho client.

  Tính cá nhân (Personalization): với các thông tin bảo mật, site của chúng ta có thể phân biệt từng user bằng cách đọc các cookies trên máy client. Thông thường các thông tin trong database chứa các phần liên quan đến client (được xác định bởi một số định danh duy nhất ID). Mối liên hệ giữa ID trên cookies với database được coi như là tính các nhân. Site của ta có thể nghiên cứu các thông tin mà client thường quan tâm thông qua ID trên cookies và đặt các thông tin đó vào trang web khi user trở lại lần nữa.

  Tính nhất quán (consistency): nếu bạn tạo một web site thương mại, bạn cần theo dõi các lần giao dịch của hàng hoá và dịch vụ trên site của bạn. Các thông tin đó có thể được lưu một cách đáng tin cậy xuống database và được tham chiếu tới các user ID tương ứng. Các thông tin này có thể được dùng để xác định xem việc mua bán đã hoàn tất hay chưa, hay là nên huỷ bỏ. Các thông tin này cũng có thể được dùng để báo cho người sử dụng biết trạng thái của các đơn hàng của họ trên site của ta.

  Tính khai thác dữ liệu (data mining): các thông tin về site của ta, về các khách hàng viếng thăm, các lần giao dịch sản phẩm có thể được lưu một cách đáng tin cậy vào database. Phòng phát triển kinh doanh có thể dùng các thông tin thu thập được từ site để quyết định việc sản xuất các sản phẩm cho năm tới hay chính sách phân phối cho phù hợp. Phòng kinh doanh có thể tìm hiểu các thông tin về người sử dụng trên site của ta. Các kỹ sư và các phòng hỗ trợ có thể muốn xem các lần giao dịch và ghi chú các khu vực nào mà sản phẩm của ta có thể được phát triển (có số lượng sản phẩm tiêu thụ mạnh). Hầu hết các hệ quản trị cơ sở dữ liệu quan hệ như MS SQL đều hổ trợ một tập các công cụ để khai thác các dữ liệu đó.
- Ưu điểm:

+ Tính bảo mật. Việc truy cập database thông thường rất an toàn, yêu cầu rất khắc khe về việc thẩm định định quyền.

+ Dung lượng. Ta có thể lưu các thông tin tuỳ ý xuống database.

+ Sự ổn định. Các thông tin có thể lưu trữ bao lâu là do ta quyết định.

+ Toàn vẹn dữ liệu và an toàn. Các database bao gồm rất nhiều yếu tố để quản lý tốt dữ liệu bao gồm các trigger, các ràng buột toàn vẹn,… Bằng cách lưu giữ các thông tin về các giao dịch trên database, ta có thể dễ dàng phục hồi từ các lỗi dễ dàng.

+ Khả năng truy cập. Dữ liệu được lưu trữ trên database có thể được truy suất bằng các công cụ hổ trợ sẵn.

+ Hỗ trợ rộng rãi.

- Khuyết điểm:

+ Phức tạp. Sử dụng các hỗ trợ về database sẽ phức tạp hơn về việc cấu hình phần cứng và phần mềm

+ Hiệu suất. Việc xây dựng một CSDL không tốt có thể dẫn đến vô số vấn đề. Việc để quá nhiều truy vấn trên database có thể làm bất lợi đến hiệu suất của server.

Tuỳ chọn	Dùng khi
Application state	Lưu trữ các thông tin ít khi thay đổi, các thông tin toàn cục được dùng bởi nhiều user, vấn đề bảo mật không được quan tâm. Không nên lưu trữ một lượng lớn thông tin lên các biến Application.
Session state	Lưu trữ các thông tin có thời gian sống ngắn riêng biệt cho từng phiên và vấn đề bảo mật được quan tâm. Không lưu trữ dữ liệu lớn trên các biến session. Chú ý là các đối tượng session state được tạo và được duy trì trong suốt thời gian của mỗi session trong ứng dụng. Trong các ứng dụng có nhiều user, điều này có thể chiếm đáng kể tài nguyên của server.
Database support	Lưu trữ một lượng lớn thông tin, quản lí các giao tác (transaction), các thông tin cần lưu khi application và session restart. Việc khai thác dữ liệu đem lại lợi ích và bảo mật là một vấn đề.

Link : ms-help://MS.VSCC/MS.MSDNVS/vbcon/html/vbconChoosingServerStateOption.htm

2 ASP.NET Application:

- Đó là tổng quan về các tuỳ chọn ta có thể dùng để lưu trữ trạng thái của dữ liệu trên trang web. Sau đây ta chỉ tìm hiểu về các tuỳ chọn cho server.
- Trước khi tìm hiểu chi tiết về các cách thức lưu trữ thông tin của người sử dụng khi họ thao tác với các trang web của một site nào đó ta cần tìm hiểu các khái niệm như:

ASP.NET Application là gì?

Tạo một application.
Thời gian sống của một Application.
Các chú ý về luồng (thread)

2.1 ASP.NET Application là gì?

- ASP.NET định nghĩa một Application như là một tập các files, các pages, các handlers, các modules, và các executable code có thể được gọi để chạy trong phạm vi của một thư mục ảo (Virtual Directories) trên một Web application Server.

2.2 Tạo một Application:

- Khi ta tạo một ứng dụng Web bằng ASP.NET là ta đã bắt đầu tạo một Application.

File global.asax:

- Dùng để sử lí các event của đối tượng Application, và Session như: Start, End …
- File này được đặt ở thư mục gốc của Web Application trong IIS. Global.asax mở rộng từ HttpApplication và được ASP.NET phân tích và dịch tự động vào .NET framework lần đầu tiên khi có bất kì yêu cầu truy suất nào từ client.
- File Global.asax được cấu hình đặc biệt để ngăn không cho client truy suất trực tiếp nội dung file hay download hay view source của file đó.
- Môi trường soạn thảo ASP.NET đã hổ trợ sẵn các event của các đối tượng trên ta chỉ cần mở code của file Global.asax là có thể viết code cho các event đó.

2.3 Thời gian sống của một Application:

- Một application được tạo lần đầu tiên khi có một yêu cầu được gởi tới server. Khi đó event đầu tiên là Application_Start, các instances của HttpApplication được tạo để sử lí các yêu cầu đến khi instance cuối cùng được sử lí xong thì phát event Application_End.
- Chú ý các phương thức Init và Dispose của đối tượng HttpApplication được gọi mỗi khi có một yêu cầu client.

2.4 Các chú ý về luồng (thread):

- Nếu ta sử dụng các object mức Application thì cần phải lưu ý là ASP.NET sử lí các yêu cầu đồng thời do đó các object Application có thể được truy suất bởi nhiều client, và tại mỗi thời điểm một số command có thể được truy suất đồng thời bởi nhiều client điều này nguy hiểm nếu command đó liên quan đến việc cập nhật dữ liệu.

Ví dụ không nên viết

```
<%
 Application["counter"] = (Int32)Application["counter"] + 1;
%>
```

Nên dùng cơ chế lock và unlock của object Application

```
<%
          Application.Lock();
 Application["counter"] = (Int32)Application["counter"] + 1;
          Application.Unlock();
%>
```

- Như vậy sẽ an toàn hơn vì trước khi thao tác update thì biến này đã bị khoá và client khác không thể truy suất được, khi thao tác xong thì biến được mở khoá và các client khác có thể truy suất bình thường.

Tóm lại:

ASP.NET Application bao gồm mọi thứ trong một thư mục ảo của web server.

Thời gian sống của một ASP.NET application bắt đầu từ sự kiện Application_Start và kết thúc bằng sự kiện Application_End.

Việc truy cập vào một object mức Application phải đảm bảo an toàn cho việc truy suất đa luồng (multithread).

Link : http://localhost/quickstart/aspplus/

3 Managing Application State:

Việc quản lí trạng thái của trang web bao gồm các mục:

3.1 Application State:

- Các biến mức Application là các biến toàn cục được đưa ra bởi ASP.NET. Ta phải luôn luôn quan tâm đến tầm ảnh hưởng của việc lưu trữ dữ liệu lên các biến Application.
- Bộ nhớ dùng để lưu trữ các biến Application. Bộ nhớ được sử dụng bởi các biến này sẽ không bị giải phóng khi giá trị của chúng thay đổi hay được xoá đi. Do đó cần quan tâm đến việc sử dụng các biến Application, không nên sử dụng các biến có nhu cầu bộ nhớ lớn ví dụ dùng biến để lưu trữ một recordset có kích thước 10MB. Đối với các trường hợp này nên sử dụng Cache.
- Các tác động của việc truy cập và lưu trữ các biến Application đồng thời trong môi trường đa luồng. Trong môi trường đa luồng (nhiều client truy suất trang web cùng một lúc) ta phải chú ý là dữ liệu có thể được yêu cầu truy suất đồng thời cùng một lúc, do đó khi lập trình bạn phải đảm bảo được là không có sự đụng độ hay chồng chéo dữ liệu giữa các client với nhau, phải có cơ chế đồng bộ hoá dữ liệu với nhau. ASP.NET hỗ trợ 2 phương thức là : Lock và Unlock để giải quyết vấn đề này.
- Thời gian sống của Application cũng ảnh hưởng đến các thông tin lưu trong các biến mức Application. ASP.NET application hay web server có thể bị hỏng bất cứ lúc nào trong quá trình đang chạy (bị hỏng , do web server đang được update, ...). Các dữ liệu lưu trữ trong các biến Application là không bền vững, dữ liệu sẽ bị mất nếu host đang quản lí nó bị huỷ (ngưng chạy). Nếu muốn lưu trữ các giá trị này lâu dài phải lưu xuống database hay các thiết bị lưu trữ khác.
- Khi ta restart IIS thì các giá trị trong biến Application bị mất đi (nếu ta chạy trang nào có yêu cầu truy suất giá trị các biến này sẽ bị lỗi) điều này cũng xảy ra khi ta stop IIS và sau đó start lại IIS (trong quá trình stop thì giá trị vẫn truy suất được chỉ lỗi khi start trở lại)
- Các biến mức Application không được chia sẽ thông qua các ứng dụng web như: một web application chạy trên nhiều server, một web application được sử lí bởi nhiều tiến trình trên một server. Các biến toàn cục mức Application chỉ toàn cục trong ngữ cảnh của tiến trình đang chạy ứng dụng. Do đó ta không thể dựa trên

các biến toàn cục mức ứng dụng để lưu trữ các giá trị có tác dụng làm mốc chung cho toàn ứng dụng như: biến đếm … trên các loại ứng dụng web trên.

- Khi cần sử lí với các dữ liệu lớn nên sử dụng Cache sẽ có hiệu quả tốt hơn.

3.2 **Sử dụng:**

- Các biến Application được dùng trong các trang, các lệnh bắt sự kiện, file Global.asax

	Cú pháp	Mô tả	Ví dụ
Ghi giá trị lên biến	Application["var"] = value; Var: tên biến cần lưu xuống application. Value: giá trị cuả biến.	Dùng để ghi giá trị cho một biến mức application. Giá trị được lưu có thể là chuỗi, số hay là một lớp (đối tượng) nhưng đối tượng này phải có thuộc tính serializable.	`Application["Name"] = "Thanh";`
Lấy giá trị của một biến mức application.	Var1 = Application[“var2”]; Var1: tên biến chứa giá trị lấy được. Var2: tên biến chứa giá trị cần lấy.	Dùng lấy giá trị của một biến mức application, cần thực hiện một số việc ép kiểu cần thiết.	`string name;` `name = Application["Name"].ToString();`

- ASP.NET cung ấp một lớp để làm việc với các biến Application State: HttpApplicationState. Lớp này bao gồm các phương thức và các thuộc tính như sau:

Thuộc tính	Kiểu dữ liệu	Loại	Mô tả	Ví dụ
AllKeys	string[] (mảng chuỗi)	Get	Dùng lấy tất cả các tên biến mức application của một trang. Chỉ lấy các key	HttpApplicationState app = Application; string[] arrAppKey = new string[app.Count];

			dựa vào các key này ta sẽ lấy được giá trị.	arrAppKey = app.AllKeys;

Contents	HttpApplicationState	Get	Lấy về một đối tượng application của trang hiện tại.	HttpApplicationState app;
				app = Application.Contents
				string[] arrAppKey = new string[app.Count];
				arrAppKey = app.AllKeys;

Count	Int	Get	Lấy tổng số biến (đối tượng) đang được lưu trong Application. Mặc định là 0.	

Item	HttpApplicationState	Get/set	Dùng lấy về một biến (đối tượng) application và thiết lập giá trị cho biến. Trong C# tương đương toán tử []	string temp;
				//Get giá trị của biến Ho
				temp = Application["Ho"].ToString
				;
				//Let giá trị của biến Ho
				Application["Ho"] = "Pham

Keys	KeysCollection	Get	Lấy về một tập các khoá (tên biến) của đối tượng Application hiện tại. Phải tham chiếu đến namespace: `System.Collections.Specialized`	HttpApplicationState app = Application ;
				int n;
				//khai báo biến để chứa tập các key
				NameObjectCollectionBase KeysCollection keyCol;
				// lấy tập các key
				keyCol = app.Keys;
				// lấy tổng số key hiện có
				n = keyCol.Count;
				for(int i=0;i<n;i++)
				Response.Write (keyCol.Get(i) + "
				");

StaticObjects	HttpStaticObjectsCollection	get	Lấy tất cả các object được khai báo bởi tag <object> ở mức application trong file Global.asax.	HttpStaticObjectsCollection
				PageObjs = Application.StaticObjects;
				if(PageObjs.Count>0)
				{
				...
				}

IsReadOnly	Protected bool	Get/set	Xem tại : ms-help://MS.VSCC.2003/MS.MSDNQTR.2003APR.1033/cpref/html/frlrfsystemcollecti	

			onsspecializednameobjectcollectionbaseclassisreadonlytopic.htm	

Phương thức	Tham số	Kiểu trả về	Mô tả	Ví dụ

Add	String *name*,object *value*
	Name: tên (khoá) của biến (đối tượng) được thêm vào tập hợp.
	Value: giá trị của biến.
		Void	Thêm một biến (đối tượng) vào tập hợp application state hiện tại.	HttpApplicationState app Application;
				// ghi vào biến Ho
				app.Add("Ho","Pham");
				// ghi vào biến Ten
				app.Add("Ten","Thanh")

Clear	Void	Void	Xoá bỏ tất cả các biến hiện có trong tập hợp Application.	// tạo đối tượng Applicati
				HttpApplicationState app Application;
				// xoá bỏ tất cả các biến hiện có
				app.Clear();

Equals	Object obj Obj: đối tượng cần so sánh	Bool	So sánh đối tượng hiện tai với đối tượng khác. Phương thức này kế thừa từ đối tượng Object.	

Get	Int index
	Index: chỉ số của biến muốn lấy.
	String name
	Name: tên biến muốn lấy.
	Chỉ sử dụng một trong 2 tham số trên
		object	Trả về một biến trong tập các biến Application hiện tại. Ta có thể đưa vào tên biến hay chỉ số của biến.	// lấy đối tượng Applicati
				HttpApplicationState app Application ;
				object myObj;
				// gán giá trị cho một biến Ho
				app["Ho"] = "Pham";
				// lấy biến Ho vào biến myObj
				myObj = app.Get("Ho");
				// xuất ra
				Response.Write(myObj.T String() + "
				");
				// ---- output : Pham

GetEnumerator	Void	IEnumerator	Dùng liệt kê tất cả các khoá của các biến trong tập hợp Application. Ta không thể dựa vào các biến nhận được để thay đổi giá trị của các biến tương ứng	// lấy đối tượng Applicati
				HttpApplicationState app Application ;
				// biến giữ tập các khoá tr về

			trong tập Application. Phải tham chiếu đến namespace: `System.Collections.Specialized.`	IEnumerator iEnum;
				// khởi tạo giá trị cho 2 bi
				là Ho và Ten
				app["Ho"] = "Pham";
				app["Ten"] = "Thanh";
				// lấy các khoá có trong tậ
				hợp
				iEnum =
				app.GetEnumerator();
				while(iEnum.MoveNext(
				Response.Write(i
				um.Current.ToString() +
				"
				");
				// ---- > output : Ho
				// Ten

GetHashCode	Void	Int	Trả về mã băm (hash code) của đối tượng hiện tại. Xem thêm : ms-help://MS.VSCC.2003/MS.MSDNQTR.2003APR.1033/cpref/html/frlrfsystemobjectclassgethashcodetopic.htm	

GetKey	Int *index*
	Index: Chỉ số của biến application state cần lấy.
		String	Trả về tên của biến được lưu trong tập hợp Application State ứng với chỉ số của biến đó. Giá trị trả về không phải là giá trị của biến.	// lấy đối tượng Applicati
				HttpApplicationState app
				Application ;
				string nameObj;
				// khởi tạo giá trị cho 2 bi
				là Ho và Ten
				app["Ho"] = "Pham";
				app["Ten"] = "Thanh";
				for(int i=0 ; i<app.Count
				i++)
				{
				// lấy tên biến
				nameObj =
				app.GetKey(i);
				Response.Write(n
				meObj + "
				");
				}
				// ----- > output : Ho
				// Ten

GetObjectData	SerializationInfo *info*, StreamingContext *context*		Xem thêm tại : ms-help://MS.VSCC.2003/MS.MSDNQTR.2003APR.1033/cpref/html/frlrfsystemcollectionsspecialize	

			dnameobjectcollectionbaseclassgetobjectdatatopic.htm	

GetType	Void	Type	Trả về kiểu dữ liệu hiện tại của đối tượng Application state hiện hành.	// lấy đối tượng Applicati
				HttpApplicationState app
				Application ;
				// khai báo biến kiểu Typ
				Type typeName;
				// khởi tạo giá trị cho 2 biến_u108 ?à Ho và Ten
				app["Ho"] = "Pham";
				app["Ten"] = "Thanh";
				// lấy kiểu
				typeName = app.GetType();
				Response.Write(typeNam
				ToString() + "
				");
				// ---- > output: System.Web.HttpApplica
				onState

Lock	Void	Void	Khoá việc truy cập tới biến hiện tại. Sử dụng khi có nhu cầu cập nhật giá trị của biến. Sử dụng với UnLock()	
OnDeserialization	Object sender	Void	Xem thêm tại : ms-help://MS.VSCC.2003/MS.MSDNQTR.2003APR.1033/cpref/html/frlrfsystemcollectionsspecializednameobjectcollectionbaseclassondeserializationtopic.htm	
Remove	String name Name: tên của biến cần huỷ bỏ	Void	Huỷ bỏ một biến khỏi tập hợp Application State thông qua tên biến.	
RemoveAll	Void	Void	Huỷ bỏ tất cả các biến hiện có trong tập hợp Application State. Phương thức này sẽ gọi phương thức Clear().	
RemoveAt	Int *index* *Index*: chỉ số của biến cần huỷ bỏ	Void	Huỷ bỏ một biến khỏi tập hợp Application State thông qua chỉ số của biến.	
Set	String *name*, object *value* *Name*: tên của biến cần cập nhật giá trị.	Void	Cập nhật lại giá trị của một biến thông qua tên của biến.	

	Value: giá trị mới của biến			
ToString	Void	String	Kế thừa từ đối tượng object. Trả về chuỗi đại diện cho đối tượng hiện tại. Trong trường hợp này trả về kiểu dữ liệu của đối tượng Application State.	

Unlock	Void	Void	Cho phép truy cập trở lại biến application state đã bị khoá bằng phương thức Lock()	// lấy đối tượng Applicati
				HttpApplicationState app
				Application ;
				// khoá lại để cập nhật giá trị
				app.Lock();
				// cập nhật
				app["Ho"] = "Pham";
				app["Ten"] = "Thanh";
				// mở khoá
				app.UnLock();

- Ngoài ra còn có các phương thức để bắt các event được tạo sẵn cho đối tượng Application được định nghĩa trong tập tin Gobal.asax.cs.

Event	Tham số	Mô tả
`Application_Start`	`Object sender, EventArgs e` `Sender:` đối tượng kiểu System.Web.HttpApplicationFactory `E`: đối tượng kiểu System.EventArgs	Event này phát sinh khi trang web được gọi chạy lần đầu tiên. Trong event này các đối tượng Response, Session, Request chưa được khởi tạo bởi ASP.NET do đó nếu sử dụng sẽ bị lỗi, chỉ sử dụng được đối tượng Application.

`Application_BeginRequest`	`Object sender, EventArgs e` `Sender:` đối tượng kiểu ASP.Global_asax. `E`: đối tượng kiểu System.EventArgs	Event này phát sinh khi client gửi một yêu cầu đến server.
`Application_AuthenticateRequest`	`Object sender, EventArgs e` `Sender:` đối tượng kiểu ASP.Global_asax. `E`: đối tượng kiểu System.EventArgs	Event này phát sinh khi chứng thực client gửi yêu cầu có được cấp quyền hay không.
`Application_Error`	`Object sender, EventArgs e`	Event này chưa bắt được.
`Application_End`	`Object sender, EventArgs e`	Event này phát sinh khi ứng dụng kết thúc.Client cuối cùng thoát khỏi ứng dụng hay khi ta restart IIS.

Ví dụ:

4 Managing Session State:

4.1 Xác định một session:

- Mỗi một phiên trên ASP.NET được định danh và theo dõi bằng một chuỗi có 120 bits (16 kí tự)(Session ID) gồm các kí tự ASCII hợp lệ cho phép trong chuỗi URL. Session ID được tạo bằng một thuật toán đặc biệt để đảm bảo rằng nó được tạo ngẫu nhiên và không bị trùng nhau do đó những người sử dụng có ý đồ xấu sẽ không thể dựa vào một session ID để tính ra một session ID khác hiện có.
- Chuỗi Session ID được truyền giữa client và server thông qua cookies hoặc chuỗi URL được thay đổi để nhúng session ID kèm theo tuỳ theo ta cấu hình ứng dụng của mình. Việc cấu hình được thực hiện trong file Global.asax.cs trong tag <sessionState>. Khi ta đặt thuộc tính cookieless = "false" thì sessionID không hiện kèm theo địa chỉ URL, khi cookieless = "true" thì sessionID hiện kèm theo địa chỉ URL.

4.2 Lưu trữ Session State:

- ASP.NET cung cấp một mô hình đơn giản và dễ sử dụng để lưu trữ dữ liệu bất kì và các object trong nhiều yêu cầu web khác nhau.
- Thay vì giữ các object trực tiếp, chế độ .NET state server đơn giản lưu trữ các session state trong bộ nhớ. Trong chế độ này tiến trình (process) đang làm việc giao tiếp trực tiếp với State server. Trong chế độ SQL, các session state lưu trữ trong CSDL SQL và tiến trình làm việc giao tiếp trực tiếp với SQL. Các tiến trình làm việc của ASP.NET có thể cải tiến các dịch vụ lưu trữ đơn giản này bằng cách serializing và lưu tất cả obje ct trong một tập các session của client tại đầu cuối của mỗi yêu cầu. Khi client trở lại trang web lần nữa, các tiến trình ASP.NET có liên quan lấy các object này từ state server và de-serializes các object đó thành các instance sau đó đặt chúng vào các tập hợp session mới được đưa ra cho các yêu cầu.
- Các chế độ của session state bao gồm:

  + **Off**: không hỗ trợ session state.

  + **Inproc**: các session state được lưu trữ cục bộ. Đây là chế độ mặc định. Cách lưu trữ này giống với ASP. Các thông tin được giữ trong tiến trình của ứng dụng đang chạy, khi tiến trình khởi động lại thì các giá trị này sẽ mất. Ưu điểm của chế độ này là hiệu suất cao. Việc truy suất các giá trị của session sẽ nhanh hơn so với 2 chế độ bên dưới.

  + **StateServer**: các session state được lưu trữ trên server ở xa. Để chạy được chế độ này thì server phải chạy dịch vụ aspnet_state.exe. Và cuối cùng là phải thiết lập lại các cấu hình trong file Web.config ở tag <sessionState>. Các thông tin của session sẽ được lưu trong một tiến trình khác với tiến trình đang chạy ứng dụng ASP.NET nhưng cũng vẫn nằm trong bộ nhớ chứ không được lưu lên database do đó sẽ bị mất nếu restart lại máy. Các thông tin lưu trong session vẫn không bị mất khi ta restart hay stop IIS lại. Thông tin bị mất khi ta đóng trình duyệt và mở lại để truy cập vào các thông tin đã lưu trước đó.

  + **SQLServer**: để chạy được mode này trên server phải chạy thêm một đoạn script để tạo một database trong SQL, các session state được lưu trên database này trên SQL. Đối với bản .NET 2002 chỉ cung cấp đoạn script để tạo table trên database tempdb do đó không thể giữ lại các thông tin khi restart SQL Server. Bản .NET 2003 cung cấp đoạn script để tạo table trên database ASPState do đó sẽ không bị mất dữ liệu.
- Các mode trên được cấu hình trong file ***Web.config***. Mỗi mode sẽ có một số thuộc tính kèm theo để cấu hình cho đúng. Chi tiết sẽ được nêu trong phần sau.
- Bằng cách phân biệt rõ ràng dữ liệu trong các session của ứng dụng tạo ra nó (session), ASP.NET cung cấp rất nhiều điểm mới mà phiên bản ASP không h ỗ trợ:

  + Có thể phục hồi khi Application bị down, bởi vì bộ nhớ dùng cho các session state không nằm trong tiến trình của ASP.NET hay nằm trên cơ sở dữ liệu.

  + Phân vùng một ứng dụng thông qua nhiều tiến trình làm việc.

  + Phân vùng ứng dụng cho các ứng dụng được cấu hình chạy trên nhiều server (web farm).

+ Phần cấu hình chi tiết sẽ được trình bày ở phần sau.

4.3 Sử dụng:

- ASP.NET cung cấp lớp HttpSessionState để làm việc với các biến session trong một site.
- Lớp HttpSessionState gồm các phương thức và thuộc tính sau:

Thuộc tính	Kiểu giá trị	Loại	Mô tả	Ví dụ

CodePage	Int	Get/set	Lấy hoặc thiết lập mã trang cho phiên hiện tại. Thuộc tính này được cung cấp để tương thích với các phiên bản ASP trước, ASP.NET không lưu CodePage trong session state. Nên dung `Response.ContentEncoding.CodePage`	// lấy đối tượng Sesion
				HttpSessionState ses = Session;
				int n;
				// các biến này được dung toàn bộ các ví dụ bên dướ
				n = ses.CodePage;
				// hay n = Response.ContentEncodin odePage;
				Response.Write(n.ToStrin + "
				");
				// ---- > output: 65001

Contents	HttpSessionState	Get	Trả về đối tượng session hiện tại.	

Count	int	get	Trả về tổng số biến hiện có trong session. Mặc định là 0	ses["Ho"] = "Pham";
				ses["Ten"] = "Thanh";
				n = ses.Count;
				// hay n = Response.ContentEncodin odePage;
				Response.Write(n.ToStrin + "
				");
				// ---- > output: 2

IsCookieless	Bool	Get	Cho biết session hiện tại có ID được nhúng trong địa chỉ URL hay nhúng trong cookie. Liên quan đến việc ta đặt thuộc tính cookiesless trong tag <sessionState> ở file Global.asax. TRUE: được nhúng trong địa chỉ URL FALSE: trong cookie	bool ret; ret = ses.IsCookieless; Response.Write(ret.ToStri) + " "); // ---- > output: False do ta không đặt cookiesless = TRUE
IsNewSessio n	Bool	Get	Trả về giá trị cho biết session có được tạo với yêu cầu hiện tại hay không.	bool ret; ret = ses.IsNewSession; Response.Write(ret.ToStri

				) + " "); // ---- > output: TRUE
IsReadOnly	Bool	Get	Trả về giá trị cho biết session có phải là chỉ đọc hay không?. TRUE: read-only	bool ret; ret = ses.IsReadOnly; Response.Write(ret.ToStri) + " ");
IsSynchronized	Bool	Get	Trả về giá trị cho biết việc truy cập vào tập các biến trong session state có được đồng bộ hay không (an toàn luồng)? TRUE: thread safe	bool ret; ret = ses.IsSynchronized; Response.Write(ret.ToStri) + " "); // ---- > output: false
Item	Object	Get/set	Trả về một biến trong tập Sesion State. Trong C# sử dụng toán tử [] để thay thế.	

Keys	KeysCollection	Get	Trả về tập các khoá (tên biến) hiện có trong session state.	ses["Ho"] = "Pham";
				ses["Ten"] = "Thanh";
				n = ses.Keys.Count;
				for(int i=0;i<n;i++)
				Response.Write(se
				eys.Get(i).ToString() +
				"
				");
				// ---- > output: Ho
				// Ten

LCID	Int	Get/set	Lấy và trả về số chỉ thông tin về vùng của một session. Xem thêm: ms-help://MS.VSCC.2003/MS.MSDNQTR.2003APR.1033/cpref/html/frlrfsystemwebsessionstatehttpsessionstateclasslcidtopic.htm	n = ses.LCID; Response.Write(n.ToStrin + " "); // ---- > output: 1033 (tuỳ máy)
Mode	SessionStateMode	Get	trả về chế độ của session hiện tại do ta thiết lập trong tập tin cấu hình web.config ở tag <sessionState>. Khi ta thiết lập mode= Off thì không thể lấy đối tượng session để kiểm tra được thuộc tính này	Response.Write(ses.Mode String() + " "); // ---- > output: SQLServe (tuỳ thiết lập mode)
SessionID	String	Get	Trả về session ID của session hiện tại.	
StaticObjects	HttpStaticObjectsCollection	get	Trả về tập các object được khai báo trong tag <object Runat=”server” Scope=”Session”/> trong file Global.asax. Xem thêm : ms-help://MS.VSCC.2003/MS.MS	

			DNQTR.2003APR.1033/cpref/html/frlrfsystemwebsessionstatehttpsessionstateclassstaticobjectstopic.htm	
SyncRoot	Object	Get	Trả về đối tượng dung để truy suất toàn bộ các đối tượng có trong session State.	Response.Write(ses.SyncR .ToString() + " "); // ---- > output: System.Web.SessionState pSessionState

Timeout	Int	Get/set	Lấy hoặc thiết lập thời gian hết hạn của session (tính bằng phút). Thời gian hết hạn được tính bằng thời gian ngay sau khi client gửi một yêu cầu cộng thêm Timeout .	// thiết lập thời gian hết hạ
				12 phút sau mỗi lần gửi yê
				cầu
				ses.Timeout = 12;
				Response.Write(ses.Timeo
				ToString() + "
				");
				// ---- > output: 12

Phương thức	Tham số	kiểu trả về	Mô tả	Ví dụ
Abandon	Void	Void	Huỷ session hiện tại. Các dữ liệu trong session này sẽ bị mất. Gọi phương thức này sẽ làm phát sinh event Session_End trong file Global.asax. Chú ý: là event Session_End() chỉ phát sinh trong mode InProc không phát sinh trong mode StateServer và SQLServer.	

Add	String *name*, Object *value*
	Name: tên biến được thêm vào session.
	Value: giá trị của biến.
		Void	Thêm một biến mới mức session vào tập hợp session state hiện tại.	// thêm một biến Ho
				ses.Add("Ho","Pham");
				// thêm một biến Ten
				ses.Add("Ten","Thanh")
				// tương đương với
				// ses["Ho"] = "Pham";
				// ses["Ten"] = “Thanh";

Clear	Void	Void	Xoá tất cả các biến hiện có trong session state.	

CopyTo	Array *array*, Int index
	Array: mảng dung để chứa các khoá (tên
		Void	Chép tất cả các biến trong session state vào một mảng. Nếu số phần tử của mảng tính từ vị trí index nhỏ hơn số biến hiện có	string[] arrValue = new s
				// thêm một biến Ho
				ses.Add("Ho","Pham");
				// thêm một biến Ten
				ses.Add("Ten","Thanh")

	biến) trả về. *Index*: chỉ số để bắt đầu chép vào trên của mảng array (không phải chỉ số của các biến trong session state)		trên mảng sẽ gây ra lỗi.	ses.CopyTo(arrValue,0);
				n = ses.Count;
				for(int i=0;i<n;i++)
				Response.Write(
				arrValue.GetValue(i).To
				"
				");
				// ---- > output: Ho
				// Ten

Equals	Object	Bool	So sánh 2 object. Xem thêm : ms-help://MS.VSCC.2003/MS.MSDNQTR.2003APR.1033/cpref/html/frlrfsystemobjectclassequalstopic.htm	

GetEnumerator	Void	IEnumerator	Lấy tập các biến trong session state thông qua một giao tiếp IEnumerator	IEnumerator enumSes;
				// thêm một biến Ho
				ses.Add("Ho","Pham");
				// thêm một biến Ten
				ses.Add("Ten","Thanh")
				enumSes = ses.GetEnum
				while(enumSes.MoveNe
				Response.Write(e
				urrent.ToString() + "<br
				// ---- > output: Ho
				// T

GetHashCode	Void	Int	Trả về khoá băm của đối tượng. Phương thức kế thừa từ lớp Object. Xem thêm: ms-help://MS.VSCC.2003/MS.MSDNQTR.2003APR.1033/cpref/html/frlrfsystemobjectclassgethashcodetopic.htm	
GetType	Void	Type	Trả về kiểu của đối tượng.	Response.Write(ses.GetT tring() + " "); // ---- > output: System.Web.SessionStat ionState
Remove	String *name*. Name: tên biến cần xoá bỏ khỏi session state.	Void	Xoá bỏ một biến ra khỏi session state thông qua tên biến.	
RemoveAll	Void	Void	Xoá bỏ tất cả các biến ra	

			khỏi session state. Phương thức này sẽ gọi phương thức Clear()	
ToString	Void	String	Trả về một chuỗi đại diện cho object	Response.Write(ses.ToS " "); // ----> output: System.Web.SessionStat ionState
RemoveAt	Int *index*. Index: chỉ số của biến cần loại bỏ ra khỏi session state.	Void	Loại bỏ một biến ra khỏi session state thông qua chỉ số của biến.	

- Ngoài ra còn có các event do ASP.NET cung cấp được cài đặt sẵn trong file Global.asax.cs.

Event	Tham số	Mô tả
`Session_Start`	`Object sender, EventArgs e` `Sender:` System.Web.SessionState.SessionStateModule `E:` System.EventArgs	Event phát sinh khi có một client truy cập tới trang web.
`Session_End`	`Object sender, EventArgs e`	Event này phát sinh khi hết thời gian timeout đo ta thiết lập trong tag <sessionState>, hoặc ta gọi phương thức Abandon(). Chú ý: event này chỉ phát sinh khi mode=InProc. Còn các mode còn lại :StateServer, SQLServer không phát sinh event này.

Ví dụ:

5 Managing Client-side Cookies:

- Phương pháp lưu trữ bằng cookies là tương tự như của ASP. Các thông tin sẽ được lưu thành một file text trên máy của client gọi là cookies. Chỉ những site nào ghi cookies xuống thì mới có thể đọc cookies đó lên.
- ASP.NET cung cấp một lớp để làm việc với cookies là: HttpCookie với các phương thức và thuộc tính sau:

Thuộc tính	Kiểu	Loại	Mô tả	Ghi chú
Domain	String	Get/set	Dùng để thiết lập hoặc lấy về tên miền gán với cookies.	Thiết lập tên miền để hạn chế chỉ truyền các cookies của các client trong domain đó.
Expires	Datatime	Get/set	Dùng thiết lập hoặc lấy về ngày hết hạn của cookies.	Có thể thiết lập đến từng phút.
HasKeys	Bool	get	Xác định xem cookies có chứa nhiều item con không. True nếu có nhiều, false không chứa item con.	Mặc định là false.
Item	String	Get/set	Dùng để lấy hoặc thiết lập giá trị cho một item trong trong cookies. Trong C# dung toán tử [] thay thế.	
Name	string	Get/set	Dùng thiết lập hoặc đặt tên cho cookies.	Mặc định là rỗng trừ khi ta khởi tạo có tên.
Path	String	Get/set	Lấy hoặc thiết lập đường dẫn ảo để lưu cookies.	
Secure	Bool	Get/set	Lấy hoặc thiết lập thông tin cho biết việc truyền cookies có bảo mật hay không.	Mặc định là false
Value	String	Get/set	lấy hoặc đặt giá trị cho cookies	
Values	NameValueCollection	Get	lấy về một tập các item con của	Dùng để gán nhiều giá trị cho

			cookies hiện tại.	cookies.

Phương thức	Tham số	Trả về	Mô tả	Ví dụ
HttpCookie	String name *Name*: tên của cookies	Void	Phương thức khởi tạo của lớp. Dùng để khởi tạo một biến cookie.	// khởi tạo với 1 tham số là t cookie HttpCookie cok1 = new HttpCookie("test1");
HttpCookie	String *name*,string *value* *Name*: tên của cookies *Value*: giá trị	void	Phương thức khởi tạo cho lớp đồng thời gán giá trị cho biến	// khởi tạo với 2 tham số là t cookie và giá trị HttpCookie cok2 = new HttpCookie("test2","Value")
Equals	Object	Bool	Dùng so sánh 2 đối tượng. Xem thêm: ms-help://MS.VSCC.2003/MS.MSDNQTR.2003APR.1033/cpref/html/frlrfsystemobjectclassequalstopic.htm	
GetHashCode	Void	Int	trả về mã băm của đối tượng hiện tại	
GetType	void	type	trả về kiểu của đối tượng hiện tại. Phương thức này kế thừa từ lớp Object.	
ToString	Void	String	trả về chuỗi đại diện cho cookies	HttpCookie cok = new HttpCookie("test"); Response.Write(cok.ToStrin + " "); // ----> output: System.Web.HttpCookie

Ví dụ :

6 ViewState:

6.1 Khái niệm:

- Đây là một cơ chế của ASP.NET dùng để lưu lại các vết giá trị của các server control. Ví dụ: chuỗi text hiển thị trên label control được lưu mặc định trên viewstate. Ta có thể gán giá trị lúc thiết kế hoặc bằng code chỉ một lần lúc trang được load, và sau mỗi lần submit trang chuỗi text của label được tự động lấy ra từ ViewStates.
- ViewState là một trong các thuộc tính có sẵn của server control. Dùng khi thiết kế control trên web.
- ViewState là một hidden field được quản lí bởi ASP.NET framework. Khi ASP.NET thực thi một trang, giá trị của trang và của tất cả các control được thu thập và được định dạng thành một dòng được mã hoá và gán cho thuộc tính value

của control hidden đặc biệt trên form. Giá trị này được truyền đi theo mỗi yêu cầu của client cho server.

- Ta cần phân biệt giữa hidden field là control HTML, và hidden field lưu giá trị của ViewState.

Loại	ID/Name	Value	Ví dụ
ViewState	chỉ có một tên (name) duy nhất cho ViewState là : __VIEWSTATE	Thuộc tính Value được mã hoá một cách tự động mà không cần áp dụng bất cứ một phương pháp mã hoá nào.	`<input type="hidden" name="__VIEWSTATE" value="dDwyMDY1OTM4NTc3Ozs+Oh/5q+B/cE0xL4JyWhXlmtrKjM=" />`
HTML Hidden field	ID do người phát triển trang web đặt khi họ thiết kế trang. Có thể có nhiều tên khác nhau.	Thuộc tính Value không tự động mã hoá do đo ta có thể thấy được chính xác nội dung khi view source.	`<INPUT id="txtHTMLHidden" type="hidden" value ="HTML Hidden value"/>`

- Khi sử dụng ViewState ta phải chú ý một số điểm sau đây:

+ Bắt buột phải có thẻ form phía server (<form runat=”server”/>) trên trang aspx.

+ Một trang sẽ tự động lưu khoảng 20 bytes hoặc nhiều hơn các thông tin vào ViewState. Do đó nếu ta không cho phép chế độ ViewState của một trang hay của cả ứng dụng thì ta vẫn thấy một số giá trị trong thuộc tính value của tag ViewState.

+ Trong trường hợp trang không có nhu cầu trao đổi dữ liệu với server (trang tĩnh), ta có thể loại bỏ ViewState trong trang bằng cách bỏ đi tag <form>.

+ ViewState chỉ có tác dụng trên trang mà nó được sử dụng ta không thể truy cập giá trị của biến trên ViewState từ một trang khác.

6.2 **Sử dụng:**

	Cú pháp	Ví dụ
Ghi giá trị xuống ViewState	ViewState[“var”] = value; Var: tên biến để chứa giá trị. Value: giá trị cần lưu	ViewState["T1"] = "test1";
Lấy giá trị ra	Var = ViewState[“Var2”]; Var: tên biến cần lưu giá trị. Var2: tên biến chứa giá trị cần lấy ra trong ViewState.	string str; str = ViewState["T1"].ToString();

Ngoài ViewState còn được sử dụng khi thiết kế control cho web. Phần này sẽ nói đến khi thiết kế web control.

6.3 **Các hạn chế của ViewState:**

- Trong một số trường hợp sử dụng ViewState không phải là lựa chọn tốt vì một số lý do:

+ Khả năng lưu trữ dữ liệu lớn. Khi đó sẽ làm cho kích thước trang và kích thước của form tăng lên khi gửi cho trình duyệt.

+ Sự an toàn dữ liệu. Mặc dù dữ liệu trong ViewState đã được mã hoá nhưng sẽ an toàn hơn nếu dữ liệu đó client không thể xem được.

+ Hạn chế các kiểu dữ liệu ghi trong ViewState. ViewState chỉ chứa được các kiểu dữ liệu cơ bản. Các kiểu có cấu trúc không lưu được.

6.4 Làm tăng hiệu suất khi sử dụng ViewState:

- Khi sử dụng ViewState nên lưu ý một số điểm sau:

+ Disable ViewState khi ta không có nhu cầu sử dụng. Phần sau sẽ nói rõ một số cách để disable ViewState.

+ Không sử dụng nhiều biến trong ViewState.

+ Chỉ sử dụng ViewState trong các trường hợp cần thiết. Ví dụ ta có một trang chỉ hiển thị một lưới chứa dữ liệu thì không nên dùng ViewState vì thông thường kích thước dữ liệu của một lưới sẽ rất lớn do đó sẽ làm tăng kích thước của trang lên rất nhiều.

6.5 Disabling ViewState:

Trong một số trường hợp không dùng ViewState cho một số đối tượng thì ta phải disable thuộc tính ViewState cho đối tượng đó.

Phạm vi	Nơi cấu hình	Mô tả	Ví dụ
Đối với từng control trên form	Trong thẻ định nghĩa control đó ở file aspx(html). Mở form ở chế độ thiết kế (design), chọn phần xem mã html (nhấn Ctrl + PageDown), tìm đến tag định nghĩa control cần cấu hình, thêm thuộc tính `EnableViewState=false`.	Tắt chế độ ViewSate của control hiện tại. Các giá trị của control sẽ không được tự động lưu lại trong ViewState cuả form.	`<asp:datagrid EnableViewState="false" … />`
Đối với một trang.	Trong thẻ cấu hình trang. Cũng mở form ở chế độ HTML như trên, tìm đến thẻ <%@Page ... > (thường ở đầu file) , thêm thuộc tính `EnableViewState=false`.	Tắt chế độ ViewState cho toàn trang.	`<%@ Page EnableViewState="False" … %>`
Đối với toàn ứng dụng.	Trong file web.config, mở file ở chế độ soạn thảo, tìm tag <pages> (nếu không thấy có thể thêm vào), tìm thuộc tính enableViewState (nếu chưa có thì thêm vào) đặt giá trị = false	Tắt chế độ ViewState cho toàn ứng dụng.	`<pages enableViewState="false"/>`

	(enableViewState=false).		

7 Cấu hình tag <sessionState> trong file web.config:

7.1 Khái niệm:

- Việc chọn lưu các thông tin trong trang web ở đâu là do ta cấu hình trong file web.config.
- Mở tập tin web.config ở chế độ soạn thảo trong IDE. Tìm tag <sessionState> (nếu chưa có thì thêm vào). Sau đó ta có thiết lập các thông số sau:

Thuộc tính	Giá trị	Ý nghĩa
mode		Chỉ nơi để lưu giữ các thông tin về session Có thể là client hay server. Thuộc tính này bắt buộc phải có nếu ta cấu hình tag <sessionState>
	Off	Không lưu lại các session state trên trang web.
	InProc	Session state được lưu trên máy client. Mode này có lợi thế là tốc độ nhanh hơn v không phải kết nối database hay kết nối vớ máy khác. Nhưng chỉ nên dùng để lưu những dữ liệu nhỏ và không có nhu cầu lư giữ lại.
	StateServer	Session state được lưu trên server, có thể nằm cùng với server đang chạy ứng dụng hay là một server nào đó trong hệ thống củ ta.
	SQLServer	Session state được lưu trên SQL server. Mode này sẽ chậm hơn 2 mode trên nhưng có lợi thế là thông tin lưu trữ lớn và có tính lâu dài.
cookieless		Xác định xem session có sử dụng cookie đ lưu thông tin session ID hay không.
	true	Session không sử dụng cookie, do đó session ID sẽ đi kèm với địa chỉ URL. Đối với session không sử dụng Cookies (cookieless = true) thì các đường dẫn của t tin trong trang web phải sử dụng đường dẫ tương đối không sử dụng được đường dẫn tuyệt đối.
	false	Session sử dụng cookie. Đây là giá trị mặc định.
timeout	Số nguyên	Xác thời gian hết hạn của một session. Nếu khoảng thời gian giữa 2 lần gửi yêu cầu củ

		client lớn hơn timeout thì session sẽ tự độ phát sinh event session_end. Tính bằng ph Mặc định là 20 phút. Sau mỗi lần client gử yêu cầu thì thời gian timeout sẽ được tính lại.
stateConnectionString	chuỗi chứa thông tin kết nối. Cú pháp: "tcpip=server:port" server: tên server sẽ chứa các thông tin về session (có thể là tên máy hoặc địa chỉ IP của máy). Port: cổng kết nối. Mặc định là 42424	Chỉ đến server và cổng để chứa các sessio state. Thuộc tính này có tác dụng nếu mode=StateServer.
sqlConnectionString	chuỗi chứa thông tin kết nối. Cú pháp: Giống như khi ta thiết lập một chuỗi connectionString để kết nối đến SQL	Chuỗi chứa thông tin kết nối đến database chứa session state. Thuộc tính này có tác dụng khi mode = SQLServer.
stateNetworkTimeout	Số nguyên.	Dùng khi mode = StateServer. Xác định s giây kết nối nối mạng giữa web server và state server (server giữ trạng thái) có thể rãnh trước khi session bị huỷ. Mặc định là 10 giây.

7.2 **Sử dụng:**

- Đối với mode **Off** và **InProc** thì ta không cần hiệu chỉnh tới file **web.config** vì đây là các thiết lập mặc định. Do đó cơ chế lưu giữ trạng thái sẽ giống như ASP.

- Đối với mode **StateServer**:
  + Để chạy được chế độ này ta trước tiên ta phải cho chạy service "***ASP.NET State Service***" (đây là một dịch vụ khi cài .NET Framework sẽ có). Cách chạy: Mở cửa sổ quản lý services trên máy sẽ chứa các thông tin về session (vào menu **Start** -> **Programs** -> **Administrative Tools** -> **Services** -> Enter). Tìm service "***ASP.NET State Service***" nếu service này chưa được Start thì Start service đó.
  + Tiếp theo là thiết lập cấu hình trong file **web.config**: mở file web.config -> tìm tag <sessionState> (nếu chưa có thì thêm vào) -> thiết lập thuộc tính **mode=StateServer** và thiết lập thuộc tính **stateConnectionString**. Ví dụ stateConnectionString="tcpip=azweb80:42424".
  + Ví dụ:

```
<sessionState
      mode="StateServer"
      stateConnectionString="tcpip=127.0.0.1:42424"
      cookieless="false" timeout="20"
/>
```

- Đối với mode **SQLServer**:
  + Để chạy mode này thì không cần phải chạy service giống như trên.
  + Ta cần tạo một database trên SQL server ở máy cần lưu các thông tin. Trước khi tạo database này ta cần khởi động service "***SQL Server Agent***" vì service này

quản lý một số dịch vụ của SQL cung cấp việc thực hiện các job. Mở SQL Query Analyzer. Sau đó mở đoạn script **InstallSqlState.sql** (đối với .NET framework v1.0) hoặc **InstallPersistSqlState.sql** (đối với .NET framework v1.1) sau đó cho thực thi (nhấn F5).

+ Lưu ý:

+ Cả 2 đoạn script trên mặc định nằm ở vị trí: <Tên ổ đĩa>:\[Winnt]\Microsoft.NET\Framework*version (số phiên bản của .NET framework)* \ .

+ Khi chạy đoạn script **InstallSqlState.sql** sẽ tạo một database **ASPState** nhưng các table để chứa thông tin sẽ được tạo trong database **Tempdb** do đó các thông tin sẽ mất nếu ta khởi động lại SQL Server.

+ Khi chạy đoạn script **InstallPersistSqlState.sql** cũng tạo database **ASPState** nhưng các table sẽ được tạo trên database này do đó các thông tin sẽ không bị mất.

+ Ta cũng có thể xoá database ASPState bằng cách chạy các đoạn script **UninstallSqlState.sql** và **UninstallPersistSqlState.sql** (tìm thấy cùng vị trí với các file script trên) tương ứng. Nhưng trước khi chạy phải stop service "ASP.NET State Service" nếu không sẽ bị lỗi.

+ Khi thao tác với file web.config thì các tag và các thuộc tính đều phân biệt chữ hoa và chữ thường do đó phải gõ chính xác.

+ Ví dụ:

```
<sessionState
      mode="SQLServer"
      sqlConnectionString="data source=127.0.0.1;user
id=sa;password="
      cookieless="false" timeout="20"
/>
```

8 Cách cấu hình cho web server chạy trên nhiều server:

Phần này chưa tìm được tài liệu nên chứa trình bày được.

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Báo cáo nghiên cứu các vấn đề xung quanh ASP.NET: Ưu điểm, HighPerformace…
Trực tiếp nghiên cứu : Trương Hiền.
Module: Hệ thống, C#.
Ngày bắt đầu:18/12/2003
Ngày hoàn thành: 31/12/2003
Ngày cập nhật: 06/01/2003

I./ Các đặc điểm mới của ASP.NET so với ASP:

ASP.NET là phiên bản mới của ASP,ngoài những ưu điểm của ASP, ASP.NET có những đặc điểm khác và mạnh hơn so với ASP:

- Pages: sử dụng các thành phần điều khiển có khả năng hoạt động và tương tác với nhau ngay trên trình chủ Server. Đặc điểm này giảm thiểu thời gian viết code tương tác giữa các trang. Lập trình trong môi trường ASP.NET tương tự như lập trình trong Form, do đó các ứng dụng của ASP.NET còn được gọi là Web Form.

- HTML Server Side Controls: các thành phần điều khiển HTML có khả năng xử lý ngay trên trình chủ(Server) dựa trên thuộc tính và phương thức tương tự như cách thức hoạt động của trình khách(client). Những thành phần điều khiển này còn cho phép ta kết hợp mã xử lý của pages ASP.NET với một sự kiện nào đó phát sinh phía client được xem như đang diễn ra trên trình server(mô hình chuyển giao- deligate).
- Rich controls: tập các điều khiển đa năng.Các Rich Controls chạy trên server và có thể tạo ra các phần tử cũng như các đối tượng HTML phức hợp trên client (Grid, calendar,table, view,…). Rich controls còn cho phép bạn ràng buộc dữ liệu và xử lý dữ liệu tương tự như bạn đang viết một ứng dụng desktop thật sự. Xoá đi biên giới mô hình Client/server.
- Web Service: các dịch vụ Web. Trang ASP.NET có thể không cần hiển thị kết xuất cho client. Chúng hoạt động như những chương trình xử lý yêu cầu ở hậu cảnh. VD trang ASP.NET nào đó có thể là đối tượng cung cấp phương thức trả về giá trị nào đó khi nhận được yêu cầu từ client.
- Cấu hình & phân phối: đơn giản và dễ dàng với các file cấu hình theo định dạng văn bản của XML.Không cần phải đăng ký hệ thống khi sử dụng nữa(Quên đi regsrv32.exe!). Chúng ta chỉ cần copy các trang ASP.NET hay các đối tượng lên máy chủ, chỉ ra vị trí của chúng và thế là chương trình cũng như dịch vụ của chúng ta đã sẵn sàng!.
- Tự động quản lý trạng thái của đối tượng Session hay Aplication. Chúng ta có thể lưu nội dung của Session hay Aplication của một ứng dụng đặc thù nào đó xuống một file trên đĩa sau đó dùng lại.
- Debug, Tracing: Các công cụ debug được nâng cấp đáng kể. Mỗi trang tài liệu có thể sử dụng một trang xử lý lỗi riêng biệt và kết xuất nội dung của biến để theo dõi ngay trong quá trình thực thi trang. Các trình Debug được tích hợp sẵn trong môi trường đa ngôn ngữ VB.NET,C++,C#.
- Security management: Chúng ta có thể tận dụng các dịch vụ đăng nhập(login) tuỳ biến cho trang tài liệu ASP.NET theo phong cách của Web hoặc cơ chế đăng nhập và dựa trên hệ thống bảo mật của HĐH.
- Tùy biến vùng đệm trên Server (Customer Server Caching): vùng đệm của kiến trúc ASP.NET được quản lý rất linh động. Chúng ta có thể tạo ra các vùng đệm riêng biệt chứa một kiểu giá trị và đối tượng trong quá trình hoạt động của trang nhằm tăng tốc cho ứng dụng.
- Một tập các đối tượng phong phú: ASP.NET hỗ trợ một tập phong phú các thư viện và các đối tượng phục vụ hầu hết những gì mà những nhà ứng dụng cần đến. Bằng những thư viện này công việc viết ứng dụng cho Web trở nên dễ dàng hơn.VD: bạn có thể sử dụng các thành phần đối tượng "Send mail " để gửi nhận thư, Đối tượng mã hoá để giải mã thông tin, Web Counter(đếm số người truy cập),ADO.NET,…

Các đối tượng nội tại khác như Request, Response, Form, Cookies, ServerVaribles, đều được giữ lại và hoàn toàn tương thích với ASP. Tuy nhiên, ASP.NET đã cung cấp thêm cho những đối tượng này rất nhiều những thuộc tính và phương thức mới giúp nâng cao khả năng xử lý các ứng dụng.

Một ứng dụng được triển khai bằng ASP.NET sẽ thừa hưởng được các thế mạnh của ASP.NET như: tốc độ nhanh, linh động, an toàn và có tính thực thi cao. Điều này dựa trên các ưu điểm nổi bật sau của ASP.NET(đứng ở góc độ lập trình.):

1. Thành phần điều khiển đóng gói các chức năng thường xuyên sử dụng rất tiện lợi như: quản lý trạng thái (State), kiểm tra dữ liệu nhập(validate), … những công việc này trước đây thường phải viết rất thủ công.
2. Mã nguồn dễ dùng, dễ đọc: Mọi công việc thao tác hay xử lý không cần thiết trước đây(như kiểm tra tính hợp lệ của dữ liệu, bảo vệ trạng thái của session,…) đều được chuyển giao cho kiến trúc và bộ khung ASP.NET xử lý. Trang ASP.NET viết mã ít hơn, ngắn gọn hơn đồng thời cũng thực thi nhanh hơn trang ASP do trang ASP.NET đã được biên dịch và được đưa vào vùng đệm bộ nhớ trong suốt quá trình thực thi.
3. Với trang ASP.NET chúng ta có thể tạo trang tài liệu HTML kết xuất phía đầu cuối đẹp mắt bằng một tập phong phú các thành phần điều khiển giao diện thân thuộc trong Windows đã được xây dựng lại.

4. Trong ASP.NET không còn phụ thuộc vào ngôn ngữ phi định kiểu như VBScript nữa, mà nó cho phép sử dụng ngôn ngữ trung lập.Trang ASP.NET có thể viết bằng rất nhiều ngôn ngữ lập trình hiện đại: C++, C#, VB.NET, Perl,…

Ngoài ra, còn có những ưu điểm khác so với ASP thường: Ở trang ASP mã lệnh và giao diện trộn lẫn với nhau. Khi phát triển những ứng dụng Web lớn, thường các dự án cần được tách ra làm 2 phần. Một nhóm thiết kế giao diện(Web Designer) và một nhóm viết lệnh lập trình(coder). Kết quả cuối cùng thường là một sự trộn lẫn giữa phần thiết kế giao diện và mã lệnh ASP để tạo thành một file chương trình duy nhất. Các trang ASP của ứng dụng đó rất khó bảo trì khi bạn muốn thêm vào các mã lập trình mới hay thay đổi giao diện. ASP.NET cho phép tách rời giữa mã lập trình và nội dung tài liệu.

Ngày nay, các thiết bị cầm tay: ĐTDĐ, máy Palm,… thay đổi rất nhanh, đòi hỏi nhu cầu sử dụng Internet ngày càng cao. Vấn đề là các trang tài liệu thiết kế cho những thiết bị này yêu cầu phải nhỏ gọn và không thể sử dụng cách định dạng cho tài liệu như trên những trình duyệt hiện đại. Một trong những cách giải quyết vấn đề trên đó là ta sẽ xây dựng 2 site khác nhau để hướng đến 2 ứng dụng khác nhau hoặc là trong cùng một site chúng ta code nhận dạng từng loại thiết bị rồi viết mã từng site cho phù hợp(Select case …) Cách này lập trình rất khó, chi phí cao và hiện đang được sử dụng nhiều nhất. Tuy nhiên, với ASP.NET chúng ta không cần viết mã lệnh, các thành phần điều khiển hoạt động trên Server có khả năng nhận dạng và phát sinh mã tuỳ theo yêu cầu sử dụng cuối cùng của Client…..

II./ Các vấn đề về nâng cao tốc độ thực thi một trang Web viết bằng ASP.NET:

Cũng giống như những mô hình lập trình khác, khi viết code trong một ứng dụng ASP.NET cũng có một số vấn đề cần lưu ý, đôi khi rất nguy hiểm cho ứng dụng của chúng ta:

1. Cần **Disable Session state** khi không dùng nó nữa.

Trong ứng dụng sẽ có trang dùng , trang không dùng các session. Vì vậy chúng ta cần disable các session lại khi không dùng chúng nữa.

```
Cú pháp:    <%@ Page attribute="value" [attribute="value"…] %>
Ví Dụ:      <%@ Page EnableSessionState="false" %>
```

Nếu một trang nào đó đòi hỏi cần truy cập vào một biến session nào đó mà không cần ***modify*** hay ***create*** nó, ta `set EnableSessionState` trực tiếp vào trong `@Page` với thuộc tính ***ReadOnly***.

Session state cũng có thể Disable ở những phương thức dịch vụ Web XML, để biết thêm nhiều chi tiết các bạn hãy tham khảo ở trang:XML Web Services Created Using ASP.NET and XML Web Service Clients.

Một cách khác để Disable Session state trong ứng dụng, trong file **Web.config** , ta **set mode** của **sessionstate** sang **off**. VD: `<sessionstate mode="off" />`.

2. Chọn cách cấp session–state một cách cẩn thận. ASP.NET cung cấp 3 cách khác nhau để lưu trữ dữ liệu session trong một ứng dụng: ***in-Process session-state***, ***out of Process session-state*** trong dịch vụ Window (Window sercice) và ***out of Process session-state*** trong SQL Server Database. Mỗi cách đều có ưu điểm riêng, tuy nhiên ***in-Process session-state*** là giải pháp nhanh nhất. Nếu chúng ta chỉ lưu trữ một số ít dữ liệu lưu động trong session state thì nên dùng ***in-Process session-state*** ,còn ***out-Process***

session-state thường được dùng trên môi trường nhiều máy tính hoặc các loại dữ liệu hay bị mất đi khi chúng ta restart.

Các bạn có thể tham khảo thêm :
mshelp://MS.VSCC/MS.MSDNVS/cpguide/html/cpconaspstatema♦nagement.htm

3. Tránh những kết nối vòng (Round Trips) không cần thiết đến Server. Khi một ứng dụng Web được triển khai, vấn đề thời gian cũng là một phần rất quan trọng. Thông thường, khi ứng dụng của chúng ta truy cập hay lưu trữ cơ sở dữ liệu thì ta mới kết nối tới Server. Hầu hết các thao tác dữ liệu đều thực hiện trên Client, VD các trường hợp nhập dữ liệu, ta có sự kiện Validation ở Form Input (Client) để kiểm tra dữ liệu nhập có hợp lệ hay không trước khi Submit…, tức là những gì xử lý được ở Client thì nên xử lý, điều này sẽ giảm thời lượng đáng kể.

Ta nên dùng sự kiện Page.IsPostBack để ngăn chặn thực thi các tiến trình không cần thiết trên một Round Trip. Có những tình huống chỉ cần thực hiện một thao tác nào đó ở lần chạy đầu tiên, các lần sau không cần biên dịch lại, ta dùng thuộc tính Page.IsPostBack bẫy chúng lại.VD :

```
void Page_Load(Object sender, EventArgs e) {
        // ...Set up a connection and command here....
        if (!Page.IsPostBack) {
            String query = "select * from Authors where FirstName
like '%JUSTIN%'";
            myCommand.Fill(ds, "Authors");
            myDataGrid.DataBind();
        }
}
```

Ở sự kiện trên, ta thấy trước hết nó sẽ kiểm tra thuộc tính Page.IsPostBack tra nếu False thì sẽ tiếp tục thực thi các câu lệnh bên trong, ngược lại thì không. Tham khảo thêm:
Ms-help://MS.VSCC/MS.MSDNVS/cpref/html/frlrfSystemWebUIPageClassIsPostBackTopic.htm

4. Sử dụng Server Controls đúng lúc, đúng nơi. Xem xét kỹ ứng dụng của chúng ta, thật sự có cần thiết hay không khi sử dụng ASP.NET Server Control, đừng nên thấy chúng dễ dùng mà "lạm dụng" nó,chỉ khi nào có sử dụng đến tài nguyên Server. Một số trường hợp hiển thị dữ liệu hay binding data đơn giản thì chúng ta nên sử dụng chúng. Trong ví dụ sau, chúng ta sẽ thấy việc dùng Server Control sẽ không phải là phương pháp hiệu quả để đưa những giá trị vào trang HTML(Client). Mỗi phương thức gửi một đường dẫn của một image từ Server đến Client bằng việc lựa chọn Server Control sẽ làm chậm quá trình xử lý của ứng dụng.

```
<script language="C#" runat="server">
    public String imagePath;
    void Page_Load(Object sender, EventArgs e) {
        //...Retrieve data for imagePath here...
        DataBind();
    }
</script>

<%-- The span and img server controls are unecessary...--%>
The path to the image is: <span innerhtml='<%# imagePath %>'
runat="server"/><br>
<img src='<%# imagePath %>' runat="server"/>

<br><br>
```

```
<%-- Use data binding to substitute literals instead...--%>
The path to the image is: <%# imagePath %><br>
<img src='<%# imagePath %>' />

<br><br>

<%-- Or use a simple rendering expression...--%>
The path to the image is: <%= imagePath %><br>
<img src='<%= imagePath %>' />
```

Tuy nhiên, nếu biết vận dụng, khai thác các thuộc tính của Server Control, nắm bắt các sự kiện, các ưu điểm của việc lưu trữ ViewState thì Server Control vẫn là cách lựa chọn thích hợp.

5. Chỉ lưu những Server Control ViewState khi nào cần. Tự động quản lý viewstate là một điểm đặc biệt của ServerControls để phục hồi những giá trị thuộc tính trên một Round Trip mà không cần viết code.Thuộc tính Enable Viewstate luôn mặc định True. Tuy nhiên, chúng ta cần biết khi nào nó lợi và khi nào nó gây cản trở cho ứng dụng của chúng ta. Ví dụ, khi ta binding data vào một Server Controls trên mỗi Round Trip, các giá trị của ViewState sẽ được lưu những giá trị mới từ thao tác data-binding. Trong trường hợp này ta nên Disable quá trình Saves viewstate. Mặc định tất cả các Server control đều set ebnable = true, để disable chúng ta có các phương pháp sau. VD ở DataGrid Server Controls như sau:

```
<asp:datagrid EnableViewState="false" datasource="..."
runat="server"/>
```

Disable viewstate toàn bộ một trang ta dùng như sau :

```
<%@ Page EnableViewState="false" %>
```

Chú ý thuộc tính viewstate có thể gán trực tiếp vào trong controls :

```
<%@ Control attribute="value" [attribute="value" … ] %>
```

6. Dùng phương thức **Response.Write** cho những chuỗi kết nối (Concat).

Nên dùng HttpResponse.Write trong một trang hay trên các User Control cho chuỗi kết nối, phương pháp này cung cấp dịch vụ đệm và kết nối rất hiệu quả. Tuy nhiên, nếu thực thi việc kết nối chuỗi ở diện rộng nên dùng nhiều câu lệnh **Response.Write** VD:

```
Response.Write("a");
Response.Write(myString);
Response.Write("b");
Response.Write(myObj.ToString());
Response.Write("c");
Response.Write(myString2);
```

7. Đừng quá tin tưởng vào công việc bẫy lỗi Exceptions của hệ thống. Khi chương trình quá lớn, phức tạp , chúng ta sẽ rất khó quản lý được chúng. Nếu có thể, ta nên detect những điều kiện gây ra exception. Khi ta biết được các điều kiện gây ra exception, đừng nên để tự nó catch exception, phải chủ động bẫy lỗi theo cách của chúng ta. Thông thường nhất là các sự kiện Check Null, ép kiểu từ chuỗi sang số(parsed), hay kiểm tra một giá trị đặc biệt nào đó trước khi thực hiện một phép toán,…VD:

```
// Consider changing this...
try {
   result = 100 / num;
}
catch (Exception e) {
  result = 0;
}
```

```
   // to this...
   if (num != 0)
      result = 100 / num;
   else
  result = 0;
```

Hai ví dụ trên sẽ cho ra cùng một kết quả, tuy nhiên ở VD sau chúng ta có phần chủ động hơn trong việc catch lỗi.

8. Dùng store procedure để truy cập data. Trong tất cả những phương thức truy cập dữ liệu mà .Net FrameWork cung cấp thì truy cập bằng SQL Server có tính thực thi cao, nhất là các ứng dụng Web. Tuy nhiên, khi đã quản lý CSDL bằng SQL Server truy cập dữ liệu chúng ta nên dùng Store Procedure thay vì dùng những câu truy vấn. Khi dùng Store procedure, các câu lệnh được biên dịch lần đầu rồi lưu vào bộ nhớ do đó những lần sau sẽ truy cập nhanh hơn. Đồng thời, Store procedure được thực thi ở Server sẽ nhanh hơn rất nhiều so với những câu lệnh SQL thực thi từ Client. Điều này sẽ giảm chi phí về thời gian thực thi rất đáng kể.

9. Dùng Class SlqDataReader cho những loại dữ liệu Forward - Only. Class SlqDataReader cung cấp một phương tiện để đọc các luồng dữ liệu Forward – Only truy cập từ SQL Server- DataBase. Nếu những tình huống nảy sinh khi chúng ta tạo một ứng dụng bằng ASP.NET mà có dùng Forward – Only data, thì SlqDataReader có tính thực thi nhanh hơn DataSet Class, bởi vì Class SlqDataReader dùng các định dạng truyền tải dữ liệu qua mạng có sẵn của SQL Server để đọc trực tiếp dữ liệu từ sự kết nối Database. Bên cạnh đó, Class SlqDataReader thực hiện đầy đủ các chức năng giao diện IEnumerable cho phép chúng ta bind data đến Server rất tốt.

10. Cache data và Page Output khi có thể. ASP.NET có rất nhiều kỹ thuật để caching page output và data không cần thiết cho những trang, hay những sự tính toán động nào đó. Tuy nhiên, nếu thiết kế page hay data, đặc biệt là những Areas có sự truyền tải dữ liệu rất lớn, mà cache không đúng lúc, đúng nơi sẽ rất nguy hiểm. Vì vậy, nếu biết cách Cache sẽ tăng khả năng thực thi rất lớn cho Web Site, đồng thời giảm chi phí đáng kể.

11. Phải đảm bảo disable Debug mode. Luôn nhớ là phải disable Debug Mode trước khi triển khai một ứng dụng hay kiểm tra thực thi một phần đo lường nào đó.

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Báo cáo nghiên cứu : những điểm khác biệt giữa C++ và C#

Ở đây tôi trình bày trên cơ sở chúng ta đã biết C++

==========

I./ Đối tượng và Lớp :

- Thế giới thực:
  - Các ví dụ về đối tượng: Con người (anh là một đối tượng người , tôi là một đối tượng người), Con chó, Con mèo, Computer, Xe máy, Máy bay …
  - Lớp: khi nhiều đối tượng cùng loại và có cùng tính chất với nhau sẽ tạo thành một lớp đối tượng. Ví dụ: Máy tính PIII, máy tính PIV... sẽ hợp thành một lớp các đối tượng máy tính…
- Thế giới tin học (cụ thể là C#):
  - Một lớp trong C# được xem là những khái niệm cơ sở của ngôn ngữ. Điều này có nghĩa là mỗi khi viết một chương trình bằng C# bạn sẽ tạo ra các

lớp để cấu thành một chương trình. Chúng ta sử dụng lớp như là khuôn mẫu để đặt tất cả những thuộc tính và những chức năng hay hành vi vào trong một khối cho một nhóm đối tượng nào đó, sau đó chúng ta sẽ dùng lớp đó để tạo những đối tượng mà chúng ta cần.

- Có 2 loại lớp: loại dựng sẵn của .NET Framework (Framework Class Library), và loại do người dùng định nghĩa.
- Lớp chứa dữ liệu(dưới dạng biến và thuộc tính) và các hành vi (dưới dạng phương thức để xử lý dữ liệu đó). Khi chúng ta khai báo một biến trong một lớp ta gọi đó là một biến thành viên(member data).
- Khi tạo một đối tượng , chúng ta sẽ thiết lập các thuộc tính cho đối tượng đó. Nghĩa là nếu tôi là một đối tượng, bạn cũng là một đối tượng thì tôi và bạn sẽ có những thuộc tính khác nhau(màu mắt , màu tóc…).

Bây giờ chúng ta sẽ tìm hiểu về **Thuộc tính**(Properties) và **Biến**(Variable).

Biến được khai báo trong một lớp chứa dữ liệu cho từng đối tượng cụ thể. Điều này có nghĩa là khi bạn tạo một đối tượng từ một lớp, đối tượng sẽ cấp phát một vùng nhớ để chứa dữ liệu của biến đó. Ví dụ:

```
class Nguoi
{
        public int tuoi;
        public string mautoc;
}
```

Ta đã có một lớp Nguoi có 2 biến **tuoi**, **mautoc**.Ta sẽ tạo một vài đối tượng trên Class đó

```
static void Main(string[] args)
{       Nguoi Thanh =new Nguoi();
        Nguoi Hien =new Nguoi();
        // Thiết lập thuộc tính cho các đối tượng vừa tạo.
        Thanh.tuoi=21;
        Thanh.mautoc="Vàng";
        Hien.tuoi=22;
        Hien.mautoc="Nâu";
        // Xuất ra màn hình.
        System.Console.WriteLine("Tuổi của Thạnh:{0},và của Hiền
là:{1}",Thanh.tuoi,Hien.tuoi);
        Console.ReadLine();
}
```

Trong ví dụ trên ta đã tạo ra 2 đối tượng Thanh, Hien thuộc lớp **Nguoi**.Khi tạo đối tượng Thanh bộ nhớ sẽ cấp phát một vùng nhớ cho 2 biến thành viên(tuoi, mautoc), tương tự như đối tượng Hien.Bây giờ mỗi đối tượng đều có dữ liệu khác nhau và có thể gán bất kỳ giá trị nào nếu chúng ta muốn. Một vấn đề đặt ra là nếu chúng ta gán giá trị cho biến tuoi là 315, hay 450 thì sao?. Để giải quyết vấn đề này, chúng ta sử dụng thuộc tính.

Thuộc tính là cách để truy xuất biến của lớp một cách an toàn.Chúng ta tiến hành xét đến VD:

```
class Nguoi
{
// Đặt Properties cho tuoi
        public int tuoi
        {
```

```
                get
                {
                        return tuoi;
                }
                set
                {
                        if (value >=18 && value<=65)
                        {
                                tuoi=value;
                                }
                        else
                        {
                                tuoi=18;
                        }
                }
        }
//Đặt Properties cho mautmautoc
        public string mautoc
        {
                get
                {
                        return mautoc;
                }
                set
                {
                        mautoc=value;
                }
        }
```

Đoạn code trên đã được thay đổi, hãy chú ý đến các thuộc tính mà chúng ta vừa tạo ra ở trên. Một thuộc tính bao gồm 2 phương thức truy xuất. Phương thức get có nhiệm vụ lấy giá trị của biến, còn phương thức set có nhiệm vụ thay đổi giá trị của biến. Do đó đoạn mã cho phương thức get khá đơn giản, chúng ta sử dụng từ khoá return với tên biến mà cần lấy giá trị. Sau khi hoàn thành phần thuộc tính, ta đặt chúng vào chương trình như sau:

```
static void Main(string[] args)
        {
                Nguoi Thanh =new Nguoi();
                Nguoi Hien =new Nguoi();
                // Thiết lập thuộc tính cho các đối tượng vừa tạo.
                Thanh.tuoi=21;
                Thanh.mautoc="Vàng";
                Hien.tuoi=22;
                Hien.mautoc="Nâu";
                // Xuất ra màn hình.
                System.Console.WriteLine("Tuổi của Thạnh:{0},và của Hiền
là:{1}",Thanh.tuoi,Hien.tuoi);
                Console.ReadLine();
        }
```

Ở đây các bạn thấy rất giống với đoạn code trước, tuy nhiên về bản chất chúng khác nhau hoàn toàn. Ở đoạn trên chúng ta truy xuất trực tiếp đến các biến, còn đoạn dưới, chúng ta truy xuất thông qua các thuộc tính của các đối tượng.

Khi bạn gán một giá trị vào thuộc tính, C# sẽ dùng phương thức **set.** Ưu điểm của việc dùng phương thức này là chúng ta có thể điều khiển được các giá trị gắn vào và kiểm tra xem có hợp lệ hay không, thậm chí trong vài trường hợp chúng ta có thể thay đổi giá trị đó. Khi bạn gán một giá trị vào thuộc tính, C# thay đổi giá trị trong biến và bạn có thể truy xuất giá trị của biến bằng cách sử dụng từ khoá **value** như đã dùng ở trên:

```
set
{
        if (value>=18 && value<=65)
        {
                tuoi=value;
                }
        else
        {
                tuoi=18;
        }
}
```

Ta thấy trong đoạn code trên sử dụng câu lệnh if để kiểm tra giá trị được gán vào vì trong một lý do nào đó chúng ta muốn bất kỳ đối tượng nào đó của lớp người đều phải có tuổi từ 18->65. Ở đây ta kiểm tra giá trị và nếu chúng nằm trong khoảng quy định, ta sẽ lưu vào biến tương ứng, nếu không chúng sẽ đổi thành 18. Đây chỉ là một VD đơn giản về thuộc tính.

Cách tạo Lớp và Đối tượng:

Dùng từ khoá class tương tự như ví dụ trên. Còn ở đối tượng, ngoài cách trên chúng ta có thể dùng phương thức khởi dựng đối tượng gán giá trị mặc định cho đối tượng:

```
Nguoi Thanh =new Nguoi(20,"Den");
```

Ở đây chúng ta khai báo các giá trị của biến trong khi tạo đối tượng. Nhưng để đoạn code này làm việc, chúng ta cần khai báo **constructor** trong lớp **Nguoi**, vấn đề này khá đơn giản các bạn có thể tham khảo trong **MSDN**.

9 ms-help://MS.VSCC/MS.MSDNVS/csref/html/vclrfpropertiespg.htm

10

11 II./ Các phương thức trong C# :

*Các chương trình máy tính luôn tồn tại để giải quyết một vấn đề nào đó, và luôn có **phương thức** để giải quyết.*

Tất cả các chương trình đều được xây dựng từ một vài lớp. Và những lớp này đều chứa những phương thức để giải quyết những vấn đề của chương trình.VD phương thức chuyển chuỗi thành số :

```
private int[] toARR(string s)
        {
                int o=s.Length();
                int[] iarray=new int[o];
                for (int i=0 ;i<s.Length;i++)
                {
                        iarray[i]=Convert.ToInt32( s.Substring(i,1));
```

```
            }
            return iarray;

        }
```

Như vậy với phương thức toARR() vấn đề của chúng ta đã được giải quyết. Các bạn cần lưu ý chúng ta đang sử dụng các phương thức của .NET Framework Class Library(FCL) như ToInt32(), subString() để tạo ra những phương thức mới. Điều này làm cho công việc lập trình trở nên nhẹ nhàng hơn.

Khi bạn khai báo một biến trong phương thức thì nó là biến cục bộ của phương thức đó. Và không có phương thức nào có thể truy xuất đến biến cục bộ của phương thức đó.

Các phương thức rất có ích khi chương trình của bạn lặp đi lặp lại nhiều lần. Khi đó chúng ta có thể biến đoạn mã đó thành phương thức và gọi phương thức mỗi khi sử dụng.

Về cơ bản ta đã hiểu phương thức là gì, các tham số (parameter), và trị trả về (return value), thế còn trong C# thì sao. Giả sử chúng ta có một đoạn mã và cần đặt chúng vào một phương thức : Console.WriteLine(Math.Sqrt(9));.

Dòng mã này dùng phương thức Sqrt() của lớp Math để lấy căn bậc 2 của 9. Chúng ta cần viết một phương thức để ghi ra màn hình console căn bậc 2 của 9 :

```
static  void sqrtConsole(double x)
{
        Console.WriteLine(Math.Sqrt(9));
}
```

Để hiểu rõ vấn đề chúng ta đi tìm hiểu khái niệm tham số và trị tham số . Hầu hết các phương thức đều cần có một vài thông tin gì đó để hoàn thành công việc. VD: toARR() thì cần một tham số kiểu string. Như vậy tham số sẽ đại diện cho giá trị thực sự mà chúng ta truyền vào cho phương thức mỗi khi được gọi. Và những giá trị truyền vào đó được gọi là trị tham số. Mỗi tham số đều thuộc một kiểu dữ liệu nào đó và các trị tham số truyền vào phải cùng kiểu dữ liệu của tham số đó. Sau mỗi phương thức, chúng ta đều nhận được giá trị của phương thức đó, gọi là trị trả về VD: return iarray;

Cuối cùng ta đi đến một khái niệm phương thức : *"Tập hợp gồm Tên phương thức, danh sách các đối số, kiểu trị trả về gọi là khai báo phương thức đó".*

Chúng ta đều biết phương thức Main là ngõ nhập chính của tất cả các chương trình C#. Đó là phương thức đầu tiên mà trình biên dịch sẽ gọi. Do đó, tất cả các phương thức (do người dùng tạo & FCL) sẽ được gọi trong phương thức Main .VD: Giả sử ta có 2 phương thức sau :

```
1.  public void A()
{
        //Coding for A...
}

2.public void B( int x)
{
        //Coding for B...
}
```

Và từ phương thức Main() ta gọi chúng như sau :

```
static void Main(string [] args)
```

```
        {
                A();
                B(1001);
                //.....
        }
```

III./ Cấu trúc chương trình :

C# là một ngôn ngữ lập trình mang đặc điểm của C++, phong cách của Java và có mô hình ứng dụng của Visual Basic. Nếu đã biết C++ thì bạn sẽ mất khoảng 1 giờ để tìm hiểu cú pháp của C#. Cấu trúc chương trình Java, khái niệm về gói(Package), garbage collection… chắc chắn sẽ giúp bạn học C# nhanh hơn.

1. Cấu trúc một chương trình C#

C# là một chương trình thuộc dạng case-sensitive(phân biệt chữ hoa, chữ thường), tương tự như cú pháp của C++. Tuy nhiên, C# không giống như C++ ở chỗ, trong C# không chó sự phân chia giữa phần khai báo(header) và phần hiện thực (cpp).Mọi đoạn mã (class và hiện thực) đều được đặt trong một file có phần mở rộng .cs. Chúng ta xét một ví dụ quen thuộc với tất cả các ngôn ngữ lập trình - Hello:

```
namespace WindowsApplication1
{
  public class Hello
        {
        static void Main(string [] args)
                {
                        Console.WriteLine("Hello Everybody!!!");

                }
        }
}
```

Trong C# tất cả các câu lệnh, phương thức… được "gói" trong một class, tất cả các class được "gói " trong một **namespace**.Cũng giống như C++, có một chương trình chính chứa những điểm nhập cho chương trình nhập của bạn – đó là "Main".

Không cần đặt dấu chấm phẩy sau một khối class hay sau định nghĩa struct. Đó là quy định của C++, C# không cần.

2. namespace

Mỗi class được gói trong một **namespace**. Thật ra **namespace** là một khái niệm trong C++, nhưng trong C# chúng ta dùng thường xuyên hơn. Các bạn có thể truy cập một class trong **namespace** thông qua toán tử dấu chấm(.).

Bây giờ chúng ta sẽ viết lại chương trình Hello bằng cách truy xuất lớp Hello từ một lớp khác trong một **namespace** khác:

```
namespace AnotherNamespace
{
        class Anotherclass
        {
                public void Func()
                {
                        Console.WriteLine("Hello Everybody!!!");
                }
```

```
        }
}
```

Bây giờ từ lớp Hello , chúng ta có thể truy xuất tới nó:

```
namespace Mynamespace
{
        class Hello
        {
                static void Main(string [] args)
                {
                Anotherclass obj =new Anotherclass();
                        obj.Func();
                        Console.ReadLine();

                }
        }
}
```

Trong thư viện .NET, system là **namespace** cấp cao nhất trong các **namespace.**Trong C#, bằng cách mặc định tồn tại một **namespace** toàn cục, một class định nghĩa bên ngoài một **namespace** sẽ được lưu trực tiếp trong **namespace** toàn cục này và do đó có thể truy xuất lớp này mà không cần bất kỳ một **qualifier** (bổ từ) nào. Chúng ta cũng có thể định nghĩa những **namespace** lồng nhau .

ms-help://MS.VSCC/MS.MSDNVS/csref/html/vclrfNamespace.htm

3. Biến (Variables)

Biến trong C# hầu hết đều giống như C++, ngoại trừ những điểm khác biệt sau:

- Biến trong C# luôn cần khởi tạo trước khi bạn truy xuất tới nó, nếu không bạn sẽ bị báo lỗi trong khi biên dịch. Do đó bạn cần nhớ rằng không thể truy xuất một biến chưa được khởi tạo.
- Bạn không thể truy xuất một con trỏ không trỏ vào đâu cả trong C#.
- Một biểu thức không thể gọi một phần tử của mảng (array) mà chỉ số của nó vượt khỏi kích thước của mảng.
- Không có khái niệm biến toàn cục hay hàm toàn cục trong C# và những khái niệm về toàn cục được biểu diễn thông qua hàm và biến **static.**

ms-help://MS.VSCC/MS.MSDNVS/csref/html/vclrfStaticPG.htm

4. Kiểu dữ liệu (Data structs)

Tất cả các những kiểu dữ liệu của C# được định nghĩa trong **Object.** Có 2 loại kiểu dữ liệu:

- Kiểu dữ liệu cơ bản (dựng sẵn) .
- Kiểu dữ liệu do người dùng định nghĩa.

Dưới đây là những kiểu dữ liệu dựng sẵn:

Kiểu	Kích thước	Mô tả
Byte	1	Unsigned byte
Sbyte	1	Signed byte
Short	2	Signed short
Ushort	2	Unsigned short
Int	4	Signed integer

Uint	4	Unsigned integer
Long	8	Signed Long
Ulong	8	Unsigned Long
Float	4	Float pointing
Double	8	Double precision
Decimal	8	Fixed precision
String		Unicode string
Char		Unicode char
Bool	true, false	Boolean

CHÚ Ý : Các kiểu dữ liệu trong C++ và C# không giống nhau, ví dụ: kiểu long trong C++ là 4 bytes, trong C# là 8 bytes. Kiểu bool và string cũng khác, kiểu bool trong C# chỉ chấp nhận giá trị true, false không chấp nhận kiểu integer.

Kiểu dữ liệu do người dùng định nghĩa bao gồm:

- Class.
- struct.
- interface.

Nếu phân chia theo kiểu dữ liệu theo sự cấp phát bộ nhớ thì ta có thể chia làm hai loại:

- Kiểu giá trị.
- Kiểu tham khảo.

Kiểu giá trị: là những dữ liệu được cấp phát bộ nhớ trong stack. Các loại dữ liệu này bao gồm :

+ Tất cả những kiểu dữ liệu dựng sẵn, ngoại trừ kiểu string
+ Struct
+ Kiểu liệt kê.

Kiểu tham khảo: được cấp phát bộ nhớ trên heap và sẽ trở thành rác khi chúng không còn sử dụng nữa. Để khai báo kiểu dữ liệu loại này chúng ta dùng từ khoá new, và không như C++, không có từ khoá delete. Trong C#, chúng tự động gom lại nhờ công cụ "garbage collection". Kiểu tham khảo bao gồm:

+ class
+ Interface
+ Kiểu tập hợp như mảng.
+ String.

Kiểu liệt kê (Enumeration): Kiểu liệt kê trong C# hoàn toàn giống C++, chúng được định nghĩa thông qua từ khoá enum.VD:

```
enum Weekdays
{
        Sat,Sun,Mon,Tue,Wed,Thur,Fri
}
```

ms-help://MS.VSCC/MS.MSDNVS/csref/html/vcrefintegraltypes.htm

5. Class & Struct

Class & Struct cũng tương tự như trong C++ chỉ khác nhau về sự cấp phát bộ nhớ. Class được cấp phát bộ nhớ trong heap thông qua cách dùng new, Struct được cấp phát

bộ nhớ trong stack. Struct trong C# là kiểu dữ liệu rất nhẹ và nhanh. Do đó, với những kiểu dữ liệu nặng, chúng ta nên khai báo class.Xét VD:

```
struct Date
{
        int day;
        int month;
        int year;
}// không có dấu chấm phẩy ở đây.

class Date
{
        int day;
        int month;
        int year;
        string Weekday;
        string monthName;
        public int Getday()
        {
                return day;
        }
        public int Getmonth()
        {
                return month;
        }
        public int Getyear()
        {
                return year;
        }
        public int SetDay(int Day)
        {
                day=Day;
        }
        public int SetMonth(int Month)
        {
                month=Month;
        }
        public int SetYear(int Year)
        {
                year=Year;
        }
        public void SetDate(int day, int month, int year)
        {
                ……
        }
}
```

class: ms-help://MS.VSCC/MS.MSDNVS/csref/html/vcrefTheClassType.htm
struct:ms-help://MS.VSCC/MS.MSDNVS/csref/html/vcrefStructTypes.htm

6. Properties

Nếu các bạn đã quá quen thuộc với hướng đối tượng trong C++, chắc chắn bạn sẽ am hiểu về properties(thuộc tính). Trong C# cung cấp những cách thuận tiện, đơn giản và trực tiếp để truy xuất những thuộc tính. Ta viết lại VD trên như sau:

```
class Date
{
	public int Day
	{
		get
		{
			return Day;
		}
		set
		{
			Day=value;
		}
	}
	int day;
	public int Month
	{
		get
		{
			return Month;
		}
		set
		{
			Month=value;
		}
	}
	int month;
	public int Year
	{
		get
		{
			return Year;
		}
		set
		{
			Year=value;
		}
	}
	int year;
	public bool IsLeapYear(int year)
	{
		return year %4==0? true:false;
	}

	public void SetDate(int day, int month, int year)
	{
		this.day=day;
		this.month=month;
		this.year=year;
	}
}
```

Và cách thiết lập thuộc tính:

```
class User
{
```

```
        public void Main(string[] args)
        {
                Date date=new Date();
                date.Day=27;
                date.Month=12;
                date.Year=2003;
                Console.WriteLine("Date is : {0}/{1}/{2}" ,date.Day,date.Month,date.Year);
                Console.ReadLine();
        }
}
```

ms-help://MS.VSCC/MS.MSDNVS/csref/html/vclrfpropertiespg.htm

7. Modifier (Bổ từ) (ms-help://MS.VSCC/MS.MSDNVS/csref/html/vclrfcsharpkeywords_pg.htm)

Trong C++ chúng ta đã quá quen thuộc các bổ từ thông dụng như: public, private, protected. Chúng ta có một số từ mới trong C#:

* readonly: chỉ được dùng cho những dữ liệu thành viên của class. Dữ liệu kiểu readonly chỉ có thể đọc khi chúng đã được khởi tạo trực tiếp hay gán giá trị cho chúng trong **constructor**.Sự khác nhau giữa dữ liệu kiểu readonly và const là khi khai báo hằng, ta phải khởi tạo giá trị cho nó một cách trực tiếp.VD:

```
class Myclass
{
        const int constInt=100; // khai báo trực tiếp.
        readonly int myInt1;
        public Myclass()
        {
                myInt1=13;
        }
        public void Func()
        {

                Console.WriteLine(myInt1.ToString());
        }
}
```

* sealed: Khi sử dụng từ khoá này để khai báo một class sẽ không cho phép bạn lấy bất kỳ một class nào từ nó. Do đó chúng ta nên sử dụng sealed cho những lớp mà chúng ta không muốn những lớp con thừa kế chúng.

```
        sealed class pppp
        {
                int d=1000;
        }
```

* unsafe: Chúng ta có thể định nghĩa một ngữ cảnh không an toàn trong C# bằng từ khoá unsafe. Trong ngữ cảnh không an toàn chúng ta có thể viết một đoạn mã không an toàn, VD con trỏ trong C++ chẳng hạn.VD:

```
class UnsafeTest
{
        unsafe static void SquarePtrParam (int* p)
                // unsafe method: takes pointer to int
        {
                *p *= *p;
```

```
        }

        unsafe public static void Main()
                // unsafe method: uses address-of operator (&)
        {
                int i = 5;
                SquarePtrParam (&i);
                Console.WriteLine (i);
        }
}
```

ms-help://MS.VSCC/MS.MSDNVS/csref/html/vclrfUnsafe.htm

*interface: Nếu bạn đã có khái niệm về COM, bạn sẽ dễ dàng hiểu được nội dung của phần này. một interface là một lớp trừu tượng cơ bản, trong đó chỉ chứa những ký hiệu của hàm, sự hiện thực những hàm này được cung cấp bởi những lớp con. Trong C# bạn có thể định nghĩa những class như những interface thông qua từ khoá interface. .NET có nền tảng từ nhiều interface. Trong C# bạn không thể dùng nhiều lớp thừa kế, điều mà trong C++ cho phép, nhưng thực ra bản chất của sự thừa kế được thực hiện thong qua interface. Những lớp con của bạn cũng có thể thực hiện đa interface.

```
interface myDrawing
{
        int originx
        {
                get;
        set;
        }
        int originy
        {
                get;
                set;
        }
        void Draw(object shape);
}

class shape:myDrawing
{
        int Orix;
        int Oriy;
        public int originx
        {
                get{
                        return Orix;
                }
                set
                {
                        Orix=value;
                }
        }
        public int originy
        {
                get
                {
                        return Oriy;
                }
```

```
                set
                {
                        Oriy=value;
                }
        }
        public void Draw(object shape)
        {
        //.....
        }
}
```

ms-help://MS.VSCC/MS.MSDNVS/csref/html/vcrefTheInterfaceType.htm

8. Array

Mảng trong C# có nhiều tính năng vượt trội hơn C++ rất nhiều. Mảng được cấp phát bộ nhớ trong heap và do đó nó được truyền bằng tham khảo. Chúng ta không thể truy xuất một phần tử vượt ngoài giới hạn trong một mảng(Có chỉ số lớn hơn số phần tử trong mảng). Do đó C# đã khắc phục điều này. Ngoài ra C# còn cung cấp một số hàm hổ trợ giúp để xử lý các phần tử trong mảng. Ta có thể thấy rõ sự khác nhau giữa cú pháp của mảng trong C# và C++:

+ Dấu [] được đặt sau tên kiểu chứ không phải sau tên biến.
+ Bạn có thể tạo vùng nhớ cho phần tử trong mảng bằng cách dùng từ khoá new.

Ngoài ra C# còn hỗ trợ việc hiện thực mảng một chiều (single dimension), đa chiều (multi dimension) và mảng của mảng (**jagged** dimension).

```
// Mảng một chiều
int[] array=new int[10];
for(int i=0;i<array.Length;i++)
        array[i]=i;

// Mảng 2 chiều
int[,] array2=new int[5,10];
array2[2,10]=1234;

// Mảng 3 chiều
int [,,] array3=new int[2,3,6];
array3[1,1,5]=51;

// mảng của mảng
int [][]
        arrayOfarray=new int[2];
        arrayOfarray[0]=new int[4];
arrayOfarray[0]=new int[] {1,2,6};
```

ms-help://MS.VSCC/MS.MSDNVS/csref/html/vclrf[]Operator.htm

9. Indexer

indexer được dùng để viết một phương thức truy xuất trực tiếp một phần tử từ tập hợp bằng cách dùng dấu [], như trong mảng. Việc bạn cần làm là chỉ rõ chỉ số cần truy xuất một phần tử. Cú pháp của indexer cũng giống như của thuộc tính một class, ngoại trừ chúng cần một thông số nhập, đó chính là chỉ số của phần tử cần truy xuất.

Chúng ta tiếp tục xét VD, và sẽ gặp lớp CollectionBase, đó là một lớp thư viện dùng để tạo ra những tập hợp. Danh sách (list) là một protected member của lớp CollectionBase, trong đó lưu trữ tập hợp các danh sách.

```
class shape:CollectionBase
{
        public void add(Shape shp)
        {
                List.Add(shp);
   }

        //indexer
        public Shape this[int index]
        {
                get{
                        return (Shape) List[index];
                }
                set{
                        List[index]=value;
                }
        }
}
```

ms-help://MS.VSCC/MS.MSDNVS/csref/html/vclrf[]Operator.htm

10. Boxing & Unboxing

Boxing là một khái niệm mới trong C#. Như đã đề cập ở trên, mọi kiểu dữ liệu do người dùng hay dựng sẵn đều được lấy từ một lớp cơ bản là Object trong **namespace system**. Do đó việc đóng gói những kiểu dữ liệu căn bản hay nguyên thuỷ vào trong class Object được gọi là Boxing, và thao tác ngược lại gọi là unboxing. VD:

```
class Test
{
        static void Main()
        {
                int myInt=2;
                // boxing
                object Obj=myInt;
                // unboxing
                int myInt2=(int) Obj;
        }
}
```

Trong ví dụ trên cho ta thấy hai thao tác boxing và unboxing. Một giá trị int có thể được chuyển thành kiểu Object và được chuyển đổi ngược lại kiểu int. Khi một kiểu dữ liệu của một biến cần được chuyển thành một kiểu truyền bằng tham khảo, một kiểu box được tạo ra để chứa giá trị, và giá trị được lưu trong box. Unboxing chỉ là quá trình ngược lại. Khi một object box được trả về kiểu nguyên thuỷ, giá trị sẽ được chuyển từ box sang ô nhớ lưu trữ ban đầu.

ms-help://MS.VSCC/MS.MSDNVS/cpref/html/frlrfSystemReflectionPointerClassBoxTopic.htm

11. Các thông số của hàm

Trong C# có 3 loại thông số:

1. Thông số in/ truyền bằng trị.
2. Thông số in-out / truyền bằng tham khảo.
3. Thông số out.

Nếu bạn đã hiểu về COM, interface và những kiểu thông số của nó, bạn sẽ dễ dàng hiểu được các thông số của C#.

Thông số in/ truyền bằng trị: Khái niệm thông số truyền bằng trị cũng tương tự như C++. Giá trị truyền được chép vào một ô nhớ và được truyền vào hàm.VD:

```
setDay(6);
void setDay(int day)
{
        //.....
}
```

Thông số in-out/ truyền bằng tham khảo: Thông số truyền bằng tham khảo trong C++ được truyền thông qua con trỏ hay toán tử &. Trong C#, thông số truyền bằng tham khảo còn được gọi là in-out, vì khi bạn truyền một địa chỉ tham khảo của một ô nhớ, bạn đã truyền một giá trị nhập và lấy một giá trị xuất từ hàm đó. Chúng ta không thể truyền một thông số chưa khởi tạo từ một hàm.C# dùng từ khoá ref để chỉ thông số truyền bằng tham khảo. Chúng ta cũng có thể dùng từ khoá ref cho một đối số trong khi truyền nó vào một hàm có thông số truyền bằng tham khảo. VD:

```
int a=5;
FunctionA(ref a);
void FunctionA(ref int val)
{
        //.....
}
```

Thông số out: Là thông số chỉ trả về giá trị là kết quả của một hàm, không đòi hỏi giá trị nhập.C# dùng từ khoá out cho loại tham số này.VD:

```
int val;
getVal(val);
bool getVal(out int val)
{
        //.....
}
```

11. Số lượng các thông số và mảng

Để truyền thông số là một mảng trong C#, người ta dùng từ khoá **params**. Chỉ có thể có một đối số kiểu mảng. Bạn có thể truyền phần tử như là một đối số của mảng đó. VD:

```
void Func(params int[] array)
{
        Console.WriteLine("Số phần tử của mảng là : {0}",array.Length);
}
Func();
Func(5)
Func(9);
Func(new int[] {3,8,7});
int [] array=new int[8]; {1,2,3,4,5,7,2,8};
Func(array);
```

12. Toán tử và biểu thức

Hầu hết các toán tử trong C# đều giống như trong C++. Tuy nhiên C# còn bổ sung thêm một số mới và hữu ích.

Toán tử is : Dùng để kiểm tra xem kiểu của các toán hạng có tương đương nhau không. Toán tử is thường dùng trong kịch bản đa ngữ cảnh. Toán tử này có 2 toán hạng và trả về kiểu bool.

```
void testis (object param)
{
        if ( param is ClassA)
                //.....doing ...
        else if (param is myStruct)
                        //.....doing something

}
```

Toán tử as : Kiểm tra xem kiểu các toán hạng có khả đổi hay không, nếu có thì kết quả trả về là một đối tượng đã được chuyển đổi hay được box. Nếu đ.tượng không được chuyển đổi hay box được, kết quả trả về là Null. VD:

```
Shape shp=new Shape();
Vehicle veh=new Vehicle();
Circle cir=new Circle();
Shape shp=cir;
Circle cir2=shp as Circle;
object[] objects=new object[2];
objects[0]="Aisha";
objects[1]=new Shape();

string str;
for (int i=0; i&< objects.Length;i++)
{
        str=objects[i] as string;
        if (str==null)
                Console.WriteLine("Không thể Convert !!");
        else
                Console.WriteLine("{0}",str);
}
```

ms-help://MS.VSCC/MS.MSDNVS/tsqlref/ts_oa-oz_3qpf.htm

13. Câu lệnh (Statement)

Đa số các câu lệnh đều giống C++, ngoài ra còn có một số câu lệnh mới bổ sung, và có một số sửa đổi cho câu lệnh cũ. Sau đây là một số câu lệnh mới:

foreach: Dùng cho việc thực hiện vòng lặp cho tập hợp như mảng (ý nghĩa tương tự như for each .. trong VB).VD:

```
foreach(string s in array)
Console.WriteLine(s);
```

lock: Dùng để bao một đoạn code thành một session.

checked/unchecked: Dùng để kiểm tra tràn trong các toán tử có đối số là số. VD:

```
int x=Int32.MaxValue;x++;
{
```

```
x++;
}
unchecked
{
        x++;
}
```

Một số thay đổi :

-Switch: Trong C# switch có một số thay đổi như sau:

1. Sau khi thực thi một câu lệnh case, chương trình sẽ không nhảy đến câu lệnh kế tiếp .VD:

```
int var=100;
switch (var)
{
case 100:
        Console.WriteLine("<Value is 100>");
// Không dùng break;
case 200:
        Console.WriteLine("<Value is 200>");
case ...:
break;// kết thúc .

}
```

Trong C++, kết quả sẽ là : <Value is 100> , <Value is 200> tuy nhiên trong C#,ta sẽ nhận được thông báo lỗi như sau:

Error CS0163: Control cannot fall through from one case lable (‘case 100:’) to another

Tuy nhiên, chúng ta sẽ làm lại như sau:

```
int var=100;
switch (var)
{
case 100:
case 200:
        Console.WriteLine("100 or 200 <Value is 200>");
case ...
break;// kết thúc .

}
```

2. Chúng ta cũng có thể dùng hằng trong giá trị case.VD:

```
const string Weekend="Sun";
const string Weekday1="Mon"; //.....
string Weekday=Console.ReadLine();

switch (Weekday)
{
case Weekend:
        Console.WriteLine("Hôm nay là cuối tuần ");
        break;
case Weekday1:
        Console.WriteLine("Hôm nay là thứ hai");
case ...
break;// kết thúc .
```

```
}
```

-deletgate: Cho phép chúng ta lưu sự tham khảo hàm vào một biến.Trong C++ việc này giống như dùng và lưu con trỏ hàm và chúng ta hay dùng **typedef**.VD:

```
delegate int Operation(int val1, int val2)
public int Add(int val1, int val2)
{
        return val1+val2;
}
public int Sub(int val1, int val2)
{
        return val1-val2;
}
public void perform()
{
        Operation Oper;
        Console.WriteLine("Nhập vào + hay -");
        string op=Console.ReadLine();
        Console.WriteLine("Nhập vào số thứ nhất :");
        string val1=Console.ReadLine();
        Console.WriteLine("Nhập vào số thứ hai :");
        string val2=Console.ReadLine();
        val1=Convert.ToInt32(val1);
        val2=Convert.ToInt32(val2);
        if (op=="+")
                Oper = new Operation(Add);
        else
                Oper = new Operation(Sub);
        Console.WriteLine("Result is : {0}",Oper(val1,val2));
        Console.ReadLine();
}
```

ms-help://MS.VSCC/MS.MSDNVS/csref/html/vclrfStatements.htm

13. Tính thừa kế và tính đa hình

Trong C# chỉ cho kế thừa đơn. Nếu muốn đa kế thừa chúng ta dùng interface.VD:

```
class cha
{
}

class con: cha
```

ms-help://MS.VSCC/MS.MSDNVS/midl/mi-laref_1hut.htm

14. Hàm ảo

Từ khái niệm hàm ảo đến hiện thực khái niệm đa hình trong C# là như nhau, ngoại trừ việc dùng từ khoá override đối với việc hiện thực hàm ảo trong class con. Lớp con vẫn sử dụng từ khoá virtual. Lớp nào override phương thức ảo cũng sử dụng từ khoá override.

```
class Shape
{
        public virtual void Draw ()
        {
```

```
        Console.WriteLine("Shape.Draw");
        }
}
class Rectangle:Shape
{
        public override void Draw()
        {
                Console.WriteLine("Rectangle.Draw");
        }
}
class Square:Rectangle
{
        public override void Draw()
        {
                Console.WriteLine("Square.Draw");
        }
}
class MainClass
{
                static void Main()
                {
                        Shape[] shp=new Shape[3];
                        Rectangle rect=new Rectangle();
                        shp[0]=new Shape();
                        shp[1]=rect;
                        shp[2]=new Square();
                        shp[0].Draw();
                        shp[1].Draw();
                        shp[2].Draw();
                }
}
```

Kết quả như sau:

```
Shape.Draw
Rectangle.Draw
Square.Draw
```

Sau đây chúng ta làm ẩn đi lớp cha bằng cách dùng "new". Trong một lớp con, chúng ta có thể định nghĩa một hàm mới, ẩn với lớp cha, bằng cách dùng từ khoá new.Ở VD trên, chúng ta đổi từ khoá override thành new trong lớp Rectangle.

```
class Shape
{
        public virtual void Draw ()
        {
        Console.WriteLine("Shape.Draw");
        }
}
class Rectangle:Shape
{
        public new void Draw() // được đổi từ override ->new
        {
                Console.WriteLine("Rectangle.Draw");
        }
}
class Square:Rectangle
```

```
{
        public new void Draw()
        {
                Console.WriteLine("Square.Draw");
        }
}
class MainClass
{
                static void Main(string[] args)
                {
                        Console.WriteLine("Using polymorphism: ");
                        Shape[] shp=new Shape[3];
                        Rectangle rect=new Rectangle();
                        shp[0]=new Shape();
                        shp[1]=rect;
                        shp[2]=new Square();
                        shp[0].Draw();
                        shp[1].Draw();
                        shp[2].Draw();
                        Console.WriteLine("Using without polymorphism: ");
                        Square sqr=new Square();
                        sqr.Draw();
                }

}
```

từ Kết quả trả về:

```
Using polymorphism:
Shape.Draw
Shape.Draw
Shape.Draw
Using without polymorphism:
Square.Draw
```

Trong VD trên, phương thức Draw() của lớp Rectangle không phải là dạng đa hình của phương thức Draw() trong lớp Shape. Thay vì vậy, nó được xem là một phương thức khác. Do đó để tránh sự trùng lặp giữa lớp cha và lớp con ta nên dùng từ khoá new. Lưu ý : Chúng ta không nên dùng 2 phương thức cùng tên trong một lớp nếu một phương thức dùng từ khoá new, phương thức kia dùng từ khoá override hay virtual. Do đó, trong lớp Square không thể override phương thức Draw của lớp Shape.

Nếu lớp con có dữ liệu member cùng tên với dữ liệu đó trong lớp cha, để tránh bị trùng tên, dữ liệu và hàm member của lớp cha được truy xuất thông qua từ khoá base. Trong VD sau, hãy xem cách constructor của lớp cha được gọi và dữ liệu member được dùng.VD:

```
        public Child(int Val):base(val)
                {
                        myval=65;
                        base.myval;
                }
```

hay

```
        public Child(int Val)
                {
                        base(val);
```

```
                myval=65;
                base.myval;
        }
```

ms-help://MS.VSCC/MS.MSDNVS/vclang/html/_pluslang_Virtual_Functions.htm

Trên đây chỉ là những kiến thức cơ bản trong ngôn ngữ lập trình C#. Các bạn có thể tìm hiểu thêm trong MSDN…..

==========Hết==========

Các cách hiển thị dữ liệu trong ASP.NET

Như chúng ta đã biết Web form là một control hiển thị dữ liệt trên ASP.NET. Ngoài web form ta còn có các control khác để hiển thị dữ liệu một các linh hoạt tuỳ vào yêu cầu của từ website nhất định mà ta có thể quyết định sử dụng loại control nào.

I. Tạo Custom Tag (Web Custom Control)

Một trong những cách hiển thị và xử lý dữ liệu trong web là tạo ra các customTag. Ta có thể xem các đối tượng UI trong web form là các customTag do ASP.Net hỗ trợ.

Tạo một customTag:

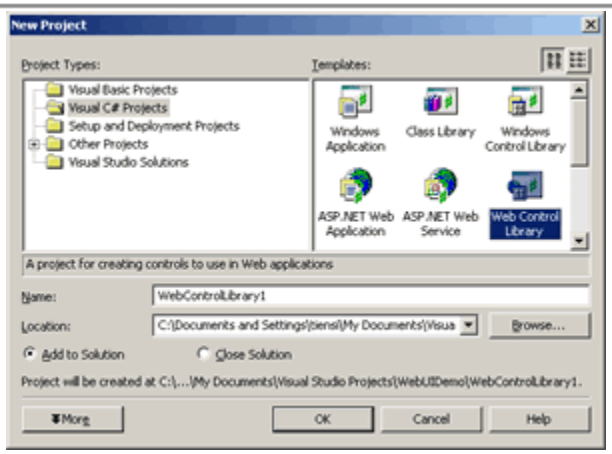
- Trong C# customTag có tên là web Custom control.
- Để tạo 1 web Custom control ta phải tạo 1 project có tên web Control library.
- Sau khi web Control library được tạo, thì web Custom control được tạo mặc định. Dịch file (.cs) thành (.dll).
- Để sử dụng dll. tạo 1 project mới (webApp).

Đăng ký 1 custom Control.

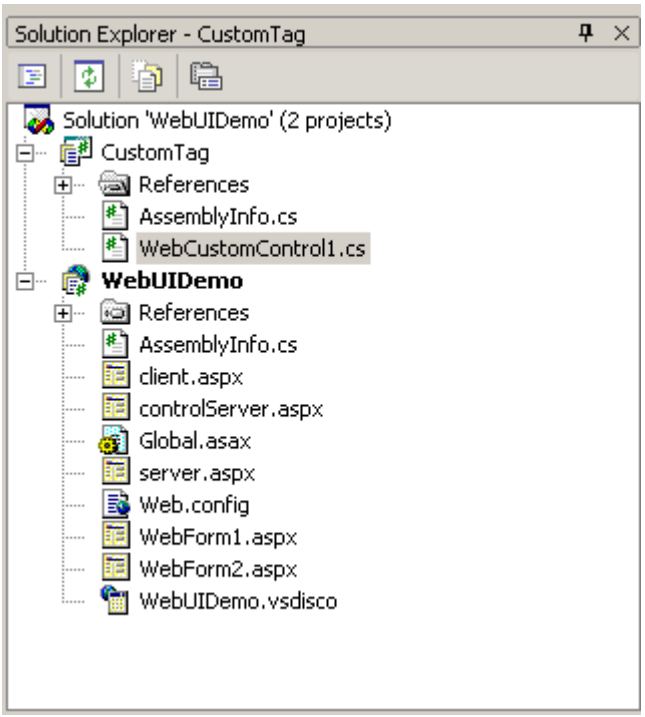
- Từ project vào menu Tool\customize Toolbox…
- Trong tab .NET chọn brower. Chọn file dll vừa dịch, ấn OK.
- Lúc này, ta đã có 1 tool mới trên toolbar. Kéo tool này vào document.

1. Tạo project Web Control Library.

- Vào menu File\New\Project…
- Trong hộp thoại New Project:
-



- Chọn Web Control Library.
- Nhập tên project mới (**ta nhập CustomTag**). (tên của project là tên namepase của các đối tượng sau này).
- Nếu project chính đã được tạo và đang mở thì chọn Add to Solution. Nếu chưa có project chính thì chọn Close Solution.
- Click vào OK. Dialog project được đóng, trong cửa sổ Solution Explorer xuất hiện một project mới.



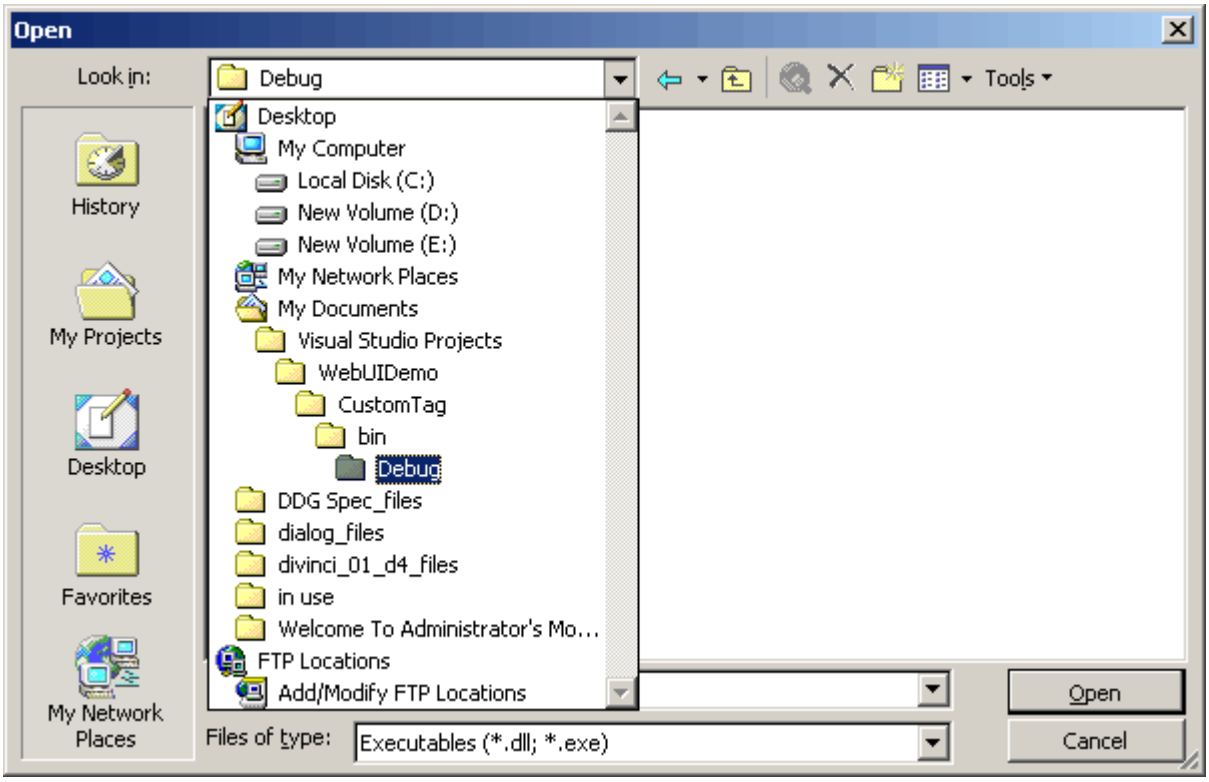
- Một control được tạo có tên WebCustonControl1.cs.

2. Dịch control WebCustonControl1.

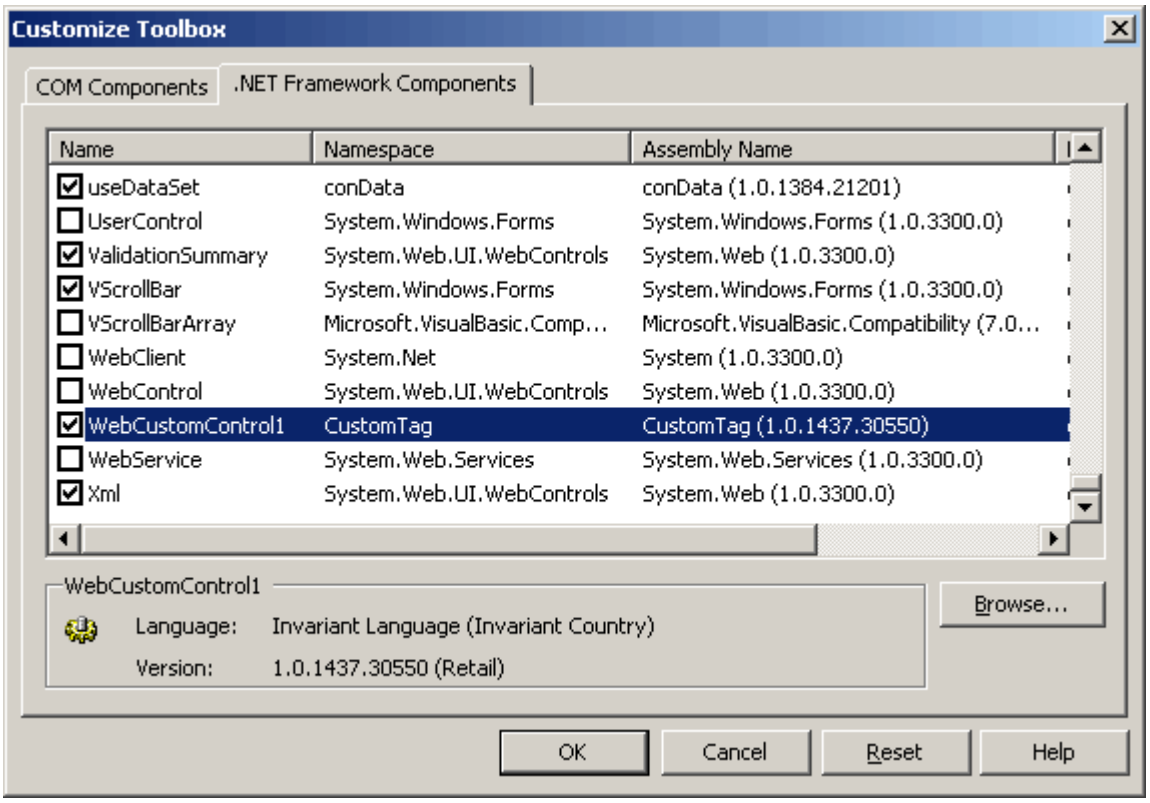
- Vào menu Build\Build CustomTag.
- Lúc này CustomTag.dll đã được tạo trong thư muc bin của project.

3. Đăng ký một Custom Control.

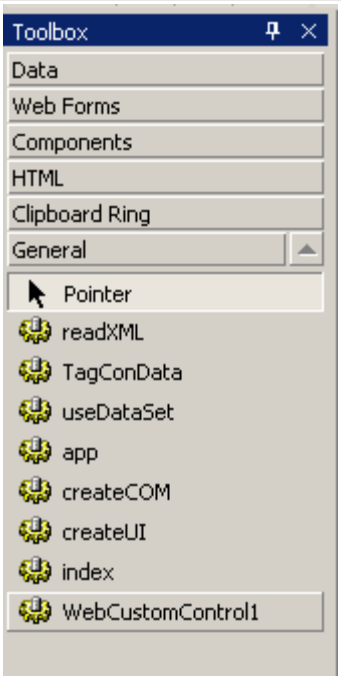
- Vào menu Tool\Customize ToolBox… hoặc ấn chuột phải lân toolbar chọn Customize ToolBox… .
- Hộp thoại Customize ToolBox xuất hiện chọn .NET Framework Components.
- Trong tab .NET Framework Components. Chọn nút Browse…
- Thoại Open xuất hiện, chọn đường dẫn đến tập tin CustomTag.dll (thường tập tin này nằm trong Solution chứa project\tên project\bin\Debug).



- Click vào Open.
- Trong hộp thoại Customize ToolBox có thêm dòng WebCustomControl1 được đánh dấu.



- Click OK. Bậy giờ chúng ta để ý trên thanh ToolBar xuất hiện một control mới có tên WebCustomControl1.



4. Sử dụng một CustomControl.

- Nếu muốn sử dụng control này ta cứ việc kéo control này vào WebForm.
- Nhưng khi chúng ta sử dụng control này cần chú ý đến vị trí đặt của control, vì control được đặt đâu trong code HTML thì dữ liệu được in ở chỗ đó. (để biết cách in dữ liệu trong web Custom, xen tiếp phần Nội dung của web Custom Control).
- Từ cửa sổ HTML code ta thất xuất hiện dòng code được tô vàng:

```
<%@ Register TagPrefix="cc1" Namespace="CustomTag"
Assembly="CustomTag" %>
```

- Đây là dòng đăng ký sử dụng dll CustomTag.
- Ngoài dòng code để đăng ký sử dụng dll, ta còn có một dòng code khác, thường dòng code này được đặt trong nhãn <Form>:

```
<cc1:WebCustomControl1 id="WebCustomControl11"
  style="Z-INDEX: 101; LEFT: 48px; POSITION: absolute;
  TOP: 280px" runat="server"></cc1:WebCustomControl1>
```

- Đây là dòng quyết định sử dụng control nào trong project đã được đăng ký.

5. Nội dung của một Web Custom Control.

- Một class Web Custom Control là class kế thừ lớp `System.Web.UI.WebControls.WebControl`. Khi ta kế thừa class WebControl thì từ class này ta cò truy xuất đến các đối tượng của web thông qua object Page.

Ví dụ: `this.Page.Request` Là truy xuất đến đối tượng request của webPage.

- Các lớp được sử dụng:

```
 using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.ComponentModel;
```

- Cũng giống như những class bình thường đầu tiên vẫn là khai báo namespace.

```
namespace CustomTag
```

- Rồi đến khai báo class.

```
public class WebCustomControl1 :
System.Web.UI.WebControls.WebControl
```

- Cuối cùng là phương thức:

```
protected override void Render(HtmlTextWriter output)
```

- Đây là phương thức default của một Web Custom Control. Nó sẽ được chạy khi Web Custom Control đươc sử dụng và nhận vào một HtmlTextWriter (output). Output này dùng để in nội dụng ra khi sử dụng Web Custom Control.
- Nếu như trong phương thức ta có dòng:

```
output.Write("AZ Solution");
```

- Thì khi chay trang web có sử dụng Web Custom Control này sẽ in dòng "AZ Solution" ngay vị trí đặt Web Custom Control.

Xây dựng thuộc tính cho Web Custom Control.

- Trước tiên ta xây dựng một thuộc tính để chứa dữ liệu.

```
 private string text;
```

- Khai báo một thuộc tính.

```
public string Text
        {
                get
                {
                        return text;
                }

                set
                {
                        text = value;
```

```
            }
    }
```

- Từ đây trong control ta có thể truy xuất đến text hay Text đều được.
- Trong lúc khởi tạo control trong HTML ta gán thuộc tính cho control.

```
<cc1:WebCustomControl1 id="WebCustomControl11" runat="server"
    Text="AZ Solution"></cc1:WebCustomControl1>
```

Xem ví dụ:

http://azweb05/WebUIDemo/useCustomTag.aspx.

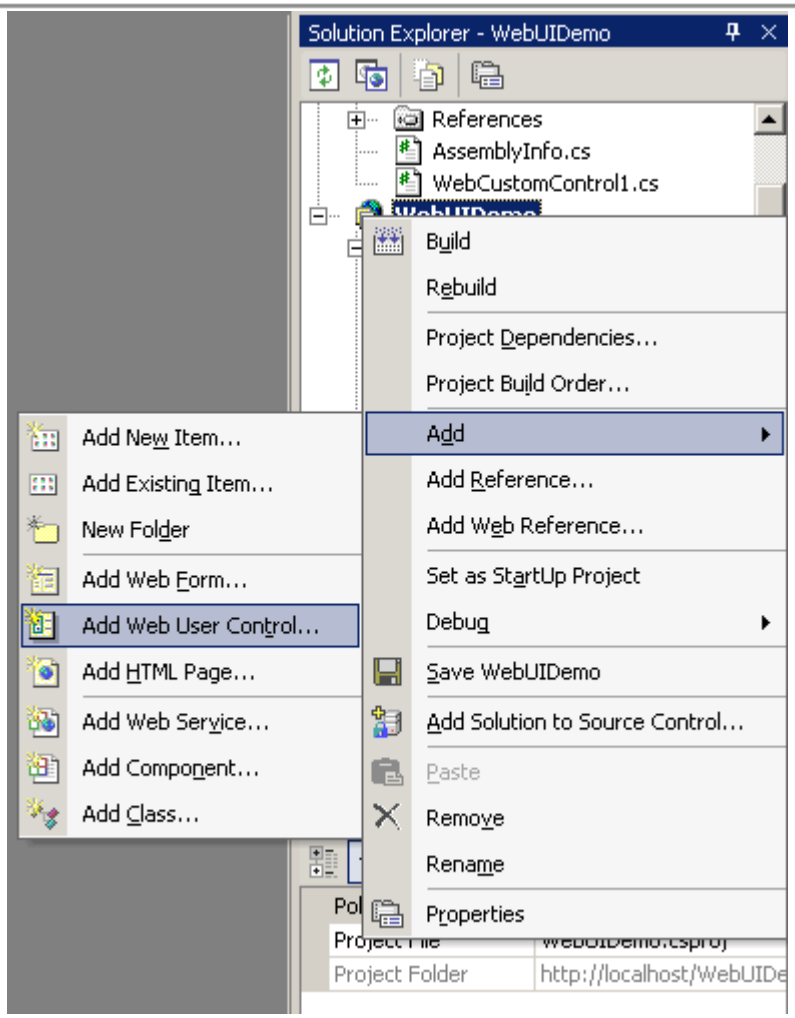
Xem document:

ms-help://MS.VSCC/MS.MSDNVS/vbcon/html/vbwlkwalkthroughcreatingcustomwebcontrols.htm

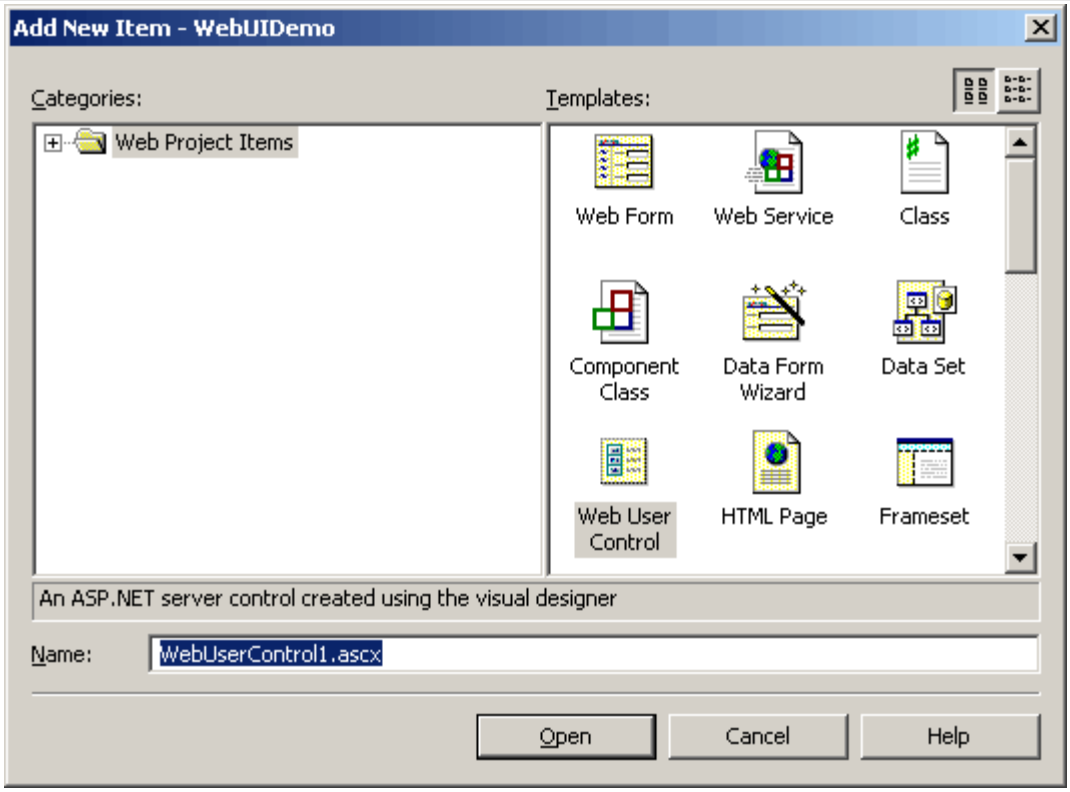
II. Tạo một Web User Control.

1. Tạo một Web User Control.

- Tạo một Web User Control ta không cần phải tạo một project mới như Web Custom Control, mà ta có thể sử dụng project chứa trong aspx, vì thực chất các Web User Control có thể được xem như là một web form.
- Từ Solution Explorer click phải vào project chọn Add\Add Web User Control.



- Hộp thoại Add New Item xuất hiện, nhập tên control vào textbox Name, click Open.



- Lúc này trong cửa sổ Solution Explorer xuất hiện control WebUserControl1.ascx.
- Đối tượng này gồm có hai file, WebUserControl1.ascx và WebUserControl1.ascx.cs.
- Trong file WebUserControl1.ascx ta có thể tạo các control của web form hay HTML.
- Trong file WebUserControl1.ascx.cs ta có thể tạo các thuộc tính và gán dữ liệu cho các control trên form.

2. Bắt đầu tạo một control cho Web User Control.

- Từ form control (WebUserControl1.ascx) ta tạo một text box từ toolbar.
- Trong code HTML xuất hiện dòng:

```
<asp:TextBox id=TextBox1 runat="server">
```

- Trong file WebUserControl1.ascx.cs. Đây là một class kế thừa lớp `System.Web.UI.UserControl` .
- Để truyền dữ liệu cho textbox vừa tạo ta khai báo một thuộc tính value.

```
public string values;
```

- Sau đó trong sự kiện load của control ta gán giá trị value này cho textbox.

```
private void Page_Load(object sender, System.EventArgs e)
        {
              this.TextBox1.Text = values;
 }
```

3. Sử dụng Web User Control vào Web Form.

- Từ web form ta kéo Web User Control vừa tạo từ cửa sổ Solution Explorer vào web form.
- Để gán giá trị cho textbox trên Web User Control ta chỉ cần gán giá trị value của control vừa được tạo trên web form trong sự kiện load của web form

```
protected WebUserControl1 Menu1;
       private void Page_Load(object sender, System.EventArgs e)
       {
             Menu1.values="AZ Solution";
 }
```

- Đối tượng Menu1 là control vừa được tạo từ trên. Nếu VS.NET không tự khởi tạo đối tượng thì ta phải tự khởi tạo bằng tay.
- Bây giờ chạy trang web thì Web User Control là một textbox có chứa dòng chữ "AZ Solution".

Xem ví dụ:

http://azweb05/WebUIDemo/use_userControl.aspx

Xem document:

ms-help://MS.VSCC/MS.MSDNVS/vbcon/html/vbwlkwalkthroughcreatingwebusercontrols.htm

Kết luật:

Trong lập trình web, Web Custom Control là một công cụ không thể thiếu. Ta có thể viết một lần và sử dụng cho tất cả các trang web, chỉ cần đăng ký và sử dụng.

Web Custom Control có thể được xem là một control trong web form mà do người lập trình định nghĩa.

Ngoài Web Custom Control, MS còn hỗ trợ một công cụ khác nữa đó là Web User Control. Công cụ này dễ sử dụng hơn, nó được sử dụng giống như web form và có thể viết code HTML, tương tác đến sự kiện của các control nhưng không thể sử xây dựng thành một công cụ như Web CusTom Control vì nó được dịch trong lúc chạy.

Với những khả năng và giới hạn của từng control mà có cách sử dụng tốt nhất như:

- Xây dựng Web Custom Control là có thể sử dụng không những cho website hiện tại mà có thể sử dụng cho các website khác sau này.
- Còn sử dụng Web User Control là có thể sử dụng cho website hiện tại. Trong khi đó Web Form chỉ chó thể sử dụng cho trang hiện tại. Vì thế, một Web Form có thể conver thành một Web User Control một các tự động nhờ vào cộng cụ của VS.NET (xem ms-help://MS.VSCC/MS.MSDNVS/vbcon/html/vbwlkwalkthroughconvertingwebformtousercontrol.htm. trong document).

Xem tài liệu tổng quan:

ms-help://MS.VSCC/MS.MSDNVS/vbcon/html/vboriWebUserControls.htm

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Các đối tượng cần biết trong WEB.

I. Page.

Page là một đối tượng Servers. Nó chứa tất cả các control server page.

Các thuộc tính cần quan tâm của đối tượng Page:

- Application:
- Request:
- Response:
- Session:
- Server:

Xem Document:

ms-help://MS.VSCC/MS.MSDNVS/cpref/html/frlrfsystemwebuipagememberstopic.htm

II. Response:

1. Lưu và đọc một cookie:

```
//Tạo thời gian lưu trữ
          DateTime dt = DateTime.Now;
          TimeSpan ts = new TimeSpan(0,0,10,0);
          //Cấp phát một Cookie.
          HttpCookie MyCookie = new HttpCookie("name");
```

```
MyCookie.Value = "AZ Solution";
MyCookie.Expires = dt.Add(ts);
//Lưu giá trị Cookie
Response.Cookies.Add(MyCookie);
//Đọc giá trị Cookie
String valueCookie = Response.Cookies["name"].Value;
//In giá trị Cookie ra trang web
Page.Response.Write(valueCookie);
```

2. Phương thứa write();

Ví dụ: ta muốn in một câu thông báo hay một đoạn script nào đó trên trang web bằng các gọi một phương thức đã được định nghĩa trong code của web form. Ta thực hiện như sau:

- Tạo một phương thức có tên print_text();

```
public void print_text(String text)
{
      Page.Response.Write(text);
}
```

- Trong code HTML ta gọi phương thức print_text() vừa tạo.

```
<%
      print_text("AZ Solution");
%>
```

* Nội dung được in đúng ngay vị trí lúc gọi phương thức print_text().

Xem Document:

ms-help://MS.VSCC/MS.MSDNVS/cpref/html/frlrfsystemwebhttpresponsememberstopic.htm

III. Request:

1. Đọc tất cả các giá trị có trong đối tượng headers.

```
public void printHeaders()
      {
            int loop1, loop2;
            System.Collections.Specialized.NameValueCollection
coll;
            // Load Header collection vào đối tượng
NameValueCollection.
            coll=Request.Headers;
            // Đọc tên tất cả các thành phần vào mảng chuỗi.
            String[] arr1 = coll.AllKeys;
            for (loop1 = 0; loop1<arr1.Length; loop1++)
            {
                  Response.Write("Key: " + arr1[loop1] + "<br>");
                  // Lấy tất cả các giá trị của tên.
                  String[] arr2=coll.GetValues(arr1[loop1]);
                  for (loop2 = 0; loop2<arr2.Length; loop2++)
```

```
                        {
                              Response.Write("Value " + loop2 + ": " +
arr2[loop2] + "<br>");
                        }
                  }
            }
```

2. Đọc tất cả các giá trị có trong QueryString.

```
      public void printQueryString()
            {
                  int loop1, loop2;
                  // Load đối tượng NameValueCollection.
                  System.Collections.Specialized.NameValueCollection
coll=Request.QueryString;
                  // Đọc tên tất cả các thành phần vào mảng chuỗi.
                  String[] arr1 = coll.AllKeys;
                  for (loop1 = 0; loop1 < arr1.Length; loop1++) {
                        Response.Write("Key: " + arr1[loop1] + "<br>");
                        String[] arr2 = coll.GetValues(arr1[loop1]);
                        for (loop2 = 0; loop2 < arr2.Length; loop2++) {
                              Response.Write("Value " + loop2 + ": " +
arr2[loop2] + "<br>");
                        }
                  }
            }
```

QueryString là dữ liệu được truyền từ trang này qua trang khác bằng method get. Tức dữ liệu truyền đi được hiển thị trong URL.

Ví dụ: một cách truyền dữ liệu theo QueryString.

http://azweb05/WebUIDemo/WebForm1.aspx?name=AZ Solution&value=1

Để lấy giá trị của name và value. Ta viết:

```
      Page.Response.Write(Request.QueryString["name"]);
                  Page.Response.Write(Request.QueryString["value"]);
```

3. Đọc tất cả các đối tượng trong Form.

```
      public void printForms()
            {
                  int loop1;
                  System.Collections.Specialized.NameValueCollection
coll;
                  //Load Form variables vào đối tượng
NameValueCollection.
                  coll=Request.Form;
                  // Đọc tên tất cả các thành phần vào mảng chuỗi.
                  String[] arr1 = coll.AllKeys;
                  for (loop1 = 0; loop1 < arr1.Length; loop1++) {
                        Response.Write("Form: " + arr1[loop1] +
"<br>");
                  }
            }
```

Trong ASP.NET, đối tượng Form là tập hợp các dữ liệu được truyền trong method Post.

Ví dụ: ta có một form gồm một textbox và một nút submit. Thuộc tính action là liên kết đến trang hiện tại (WebForm1.aspx). method = post.

```
<form action="WebForm1.aspx" method="post" runat="server">
          <asp:TextBox id="name" runat="server"></asp:TextBox>
    <asp:button id="Button1" runat="server"
Text="Submit"></asp:button>
</form>
```

Khi nút submit được click, dữ liệu trong text được truyền đến trang WebForm1.aspx. Lúc này để lấy dữ liệu trong textbox ta phải dùng qua đối tượng Form.

```
Page.Response.Write(Request.Form["name"]);
```

* name là id của textbox.

4. Đọc tất cả các biến server.

```
public void printSerVerVariables()
        {
              int loop1, loop2;
              System.Collections.Specialized.NameValueCollection
coll;
              // Load ServerVariable collection vào đối tượng
NameValueCollection.
              coll=Request.ServerVariables;
              // Đọc tên tất cả các thành phần vào mảng chuỗi.
              String[] arr1 = coll.AllKeys;
              for (loop1 = 0; loop1 < arr1.Length; loop1++) {
                    Response.Write("Key: " + arr1[loop1] + "<br>");
                    String[] arr2=coll.GetValues(arr1[loop1]);
                    for (loop2 = 0; loop2 < arr2.Length; loop2++) {
                          Response.Write("Value " + loop2 + ": " +
arr2[loop2] + "<br>");
                    }
              }
        }
```

Với đối tượng Request ta có thể lấy tất cả các biến mội trường, thành phần dữ liệu, dữ liệu truyền từ các trang khác.

Từ những ví dụ trên ta có thể lấy một thành phần dữ liệu khi ta biết được tên hay vị trí (index) bằng cách truy xuất trực tiếp với tên hay index.

```
Page.Response.Write(Page.Request.Params["name"]);
```

IV. Application

Các biến được lưu trong application có giá trị cho cả website. Tức các trang trong website đều có thể truy xuất đến các biến này.

1. Gán môt biến vào Application.

```
Page.Application.Set("name","AZ Solution");
```

Dữ liệu trong Application là một object.

2. Đọc dữ liệu trong Application.

```
String valueApp = (String)Page.Application.Get("name");
```

Ngoài ra ta còn có các phương thức và các thuộc tính khác của Application.

V. Session.

Các biến trong Session có giá trị đối với từng user một.

1. Gán một biến vào Session.

```
Page.Session.Add("name","AZ Solution");
```

Trong Session không có phương thức Set giống như Application mà sử dụng phương thức Add.

2. Đọc dữ liệu trong Session.

```
String valueSes = (String)Page.Session["name"];
```

Trong Session cũng không có phương thức Get mà các phần tử trong Session là các item. Các item này được sử dụng như các một mảng.

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

Read XML

```
private void Page_Load(object sender, System.EventArgs e)
		{
			XmlDocument doc = new XmlDocument();
			doc.Load(Server.MapPath("application.xml"));
			XmlElement root = doc.DocumentElement["information"];
			Page.Response.Write(root.GetAttribute("id"));
			root.SetAttribute("id","AZ Solution");
			Page.Response.Write(root.GetAttribute("id"));
			//write xml.
			XmlTextWriter writer = new
XmlTextWriter(Server.MapPath("application.xml"),System.Text.UTF8Encoding.UTF8);
			writer.Formatting= Formatting.Indented;
			doc.WriteTo(writer);
			writer.Flush();
			Console.WriteLine();
			writer.Close();
		}
```

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&