Share

Efficient Server-Side View State Persistence

By datacop, 8 Aug 2008

4.95 (27 votes) Rate:

Download demo - 10.39 KB

Introduction

View State is a mechanism employed by ASP.NET web pages to persist the state of the page itself, individual controls, objects, and data that are housed on that particular ASP.NET web page. View State is a double edged sword in that using it properly allows the developer to build full and robust web applications that seem to overcome the stateless nature of the web.

As the complexity of our web applications grow, more and more controls get added to the screen, more and more data needs to be persisted... View State grows. There's no way to avoid it. Even if we are diligent in our efforts in only making sure that those things that need View State are the only ones with it turned on, View State can grow to several kilobytes in size. It's not uncommon to have View State sizes that run into the 50K to 100K range on custom intranet/extranet applications.

Background

Scott Mitchell wrote an excellent article on View State some years ago that was published on MSDN. I won't go into the details on View State that he does (since he's already done an excellent job), but I will mention a few things:

1. **To much of a good thing...** By default, every control that you add to an ASP.NET web page has View State enabled. Every button, label, grid, drop down.. everything. That means, every control on your page makes the size of the View State grow.
2. **View State is persisted in the HTML markup**. That's right boys and girls. Every control on your page that makes use of View State makes that View State bigger.. makes your overall page size bigger as well.
3. **View State travels both ways**. Since View State is used by ASP.NET, and ASP.NET only exists on the server, the View State that was sent down to the browser has to be sent **back** to the server for it to be used. Yes, that includes partial page updates with AJAX as well.
4. **View State is ONLY used on the server**. I know I said this in #3, but it bears repeating here... the only place that View State is **ever** used is on the server, by the ASP.NET runtime...

Requirements

1. The View State needs to be persisted on the server.
2. The View State persistence mechanism needs to be identified by a specific user session.
3. The persisted View State artifact must not be allowed to remain forever.
4. The persisted View State should be able to be enabled and disabled on a page by page bases.
5. Different persistence mechanisms should be able to be used.
6. Page development and structure should not be modified.

Control Adapters to the Rescue

Since the ASP.NET `Page` object, at its core, is a control (granted.. it is ***the*** control.. but a control nonetheless), we're able to modify its behavior with a simple control adapter. Typically, when speaking about control adapters, the development community at large thinks about CSS Control Adapters.

MSDN defines control adapters thusly: Control adapters are components that override certain `Control` class methods and events in its execution lifecycle to allow browser or markup-specific handling. The .NET Framework maps a single derived control adapter to a `Control` object for each client request.

I discovered this rather excellent article by Robert Boedigheimer on using Server Side View State in ASP.NET using the `SessionPageStatePersister`. I knew then that this was the road I wanted to go down... although using the `SessionPageStatePersister` would run into the same problems with finite memory resources.. so that as a brush stroke solution wouldn't do...

My solution actually contains two parts. The first part is the control adapter `PageStateAdapter`, and the second is

About Article

Increasing the performance of your ASP.NET website by reducing the download footprint of your pages.

Type	Article
Licence	Ms-PL
First Posted	5 Aug 2008
Views	59,696
Downloads	788
Bookmarked	60 times

C#2.0 C#3.0 C# ASP.NET XML Architect Dev

Bookmark Print Email

Related Articles

How to Improve ASP.NET Performance
Everything you wanted to know about State Management in ASP.NET
Beginner's Guide To View State
Disconnected Client Architecture
Essential ASP.NET 2.0, 2nd Edition: Chapter 4: State Management

the custom persistence mechanism CachePageStatePersister.

PageStateAdapter

To solve my requirements #4 and #5, I decided to go with a simple attribute scheme, where basically, the developer can decide to use a different View State persistence scheme simply by putting an attribute at the top of the page's class definition. The attribute class and supporting enum are both defined in the PageStateAdapter class.

Collapse | Copy Code

```
public enum StateStorageTypes { Default, Cache, Session, InPage }

[AttributeUsage(AttributeTargets.Class, AllowMultiple = false, Inherited = true)]
public class PageViewStateStorageAttribute : Attribute
{
    private readonly StateStorageTypes storageType = StateStorageTypes.Default;

    public PageViewStateStorageAttribute(StateStorageTypes stateStorageType)
    {
        storageType = stateStorageType;
    }

    internal StateStorageTypes StorageType
    {
        get { return storageType; }
    }
}
```

Now, all the PageStateAdapter has to do is implement the virtual method GetStatePersister.

Collapse | Copy Code

```
public override PageStatePersister GetStatePersister()
{
    PageViewStateStorageAttribute psa =
        Attribute.GetCustomAttribute(Page.GetType(),
           typeof(PageViewStateStorageAttribute), true) as
           PageViewStateStorageAttribute ??
           new PageViewStateStorageAttribute(StateStorageTypes.Default);

    PageStatePersister psp;
    switch (psa.StorageType)
    {
        case StateStorageTypes.Session:
            psp = new SessionPageStatePersister(Page);
            break;
        case StateStorageTypes.InPage:
            psp = new HiddenFieldPageStatePersister(Page);
            break;
        default:
            psp = new CachePageStatePersister(Page);
            break;
    }
    return psp;
}
```

If a developer wishes to override the (now) default View State persistence method of CachePageStatePersister, they can do so by applying a simple attribute to the page class declaration.

Collapse | Copy Code

```
[PageStateAdapter.PageViewStateStorage(PageStateAdapter.StateStorageTypes.InPage)]
public partial class ViewStateInPage : System.Web.UI.Page
```

CachePageStatePersister

The CachePageStatePersister inherits from PageStatePersister. So, all it has to do is implement the two virtual methods Load() and Save().

Collapse | Copy Code

```
public override void Save()
{
    if (ViewState != null || ControlState != null)
    {
        if (Page.Session == null)
            throw new InvalidOperationException(
                "Session is required for CachePageStatePersister (SessionID -> Key)");

        string vsKey;
        string cacheFile;
        // create a unique cache file and key based on this user's
        // session and page instance (time)
        if (!Page.IsPostBack)
        {
            string sessionId = Page.Session.SessionID;
            string pageUrl = Page.Request.Path;
            vsKey = string.Format("{0}{1}_{2}_{3}", VSPREFIX, pageUrl, sessionId,
                DateTime.Now.Ticks);

            string cachePath = Page.MapPath(CACHEFOLDER);
            if (!Directory.Exists(cachePath))
                Directory.CreateDirectory(cachePath);
            cacheFile = Path.Combine(cachePath, BuildFileName());
        }
        // get our vs key from the page, re use it, and the cache
        // file (pulled from page.cache)
        else
        {
            vsKey = Page.Request.Form[VSKEY];
            if (string.IsNullOrEmpty(vsKey)) throw new ViewStateException();
            cacheFile = Page.Cache[vsKey] as string;
            if (string.IsNullOrEmpty(cacheFile)) throw new ViewStateException();
```

View States in .NET
StateManagement Suggestions : A Beginner's View
Differences between MVC and MVP for Beginners
Let's Talk About Push
Architecting .NET Web Applications for Scale & Performance (A Practical Guide)
ASP .NET MVC for Beginners in Web Development
Persisting the state of a web page
A Preview of Active Server Pages+ - Chapter 1: Introducing ASP+
C# and Asp.Net Question and Answers(All in one)
ASP.NET Web Site Performance Improvement
Reduce ASP.NET Page Output Size
State Management
Persistence Point in BizTalk 2004
ViewState Provider - an implementation using Provider Model Design Pattern
State Management in ASP.NET - Introduction

```
        }

        IStateFormatter frmt = StateFormatter;
        string state = frmt.Serialize(new Pair(ViewState, ControlState));
        using (StreamWriter sw = File.CreateText(cacheFile))
            sw.Write(state);

        Page.Cache.Add(vsKey, cacheFile, null, DateTime.Now.AddMinutes(
                       Page.Session.Timeout),
                       Cache.NoSlidingExpiration, CacheItemPriority.Low,
                       ViewStateCacheRemoveCallback);
        Page.ClientScript.RegisterHiddenField(VSKEY, vsKey);
    }
}
```

In our Save method, you can see we're doing a number of things.. first is generating a unique View State key (if this is a new page request) based on the page that's being requested, the user's session ID and the time represented in ticks. Then, we're serializing **both** the View State and the control state to a physical file (in this case, we're storing the file in the *~/App_Data/Cache* directory). After the file is created, we store the View State key and the path to the file in the Page.Cache, and save the View State key to a hidden field in the page.

If the page is requested from a POST verb, then we know that we've already got a unique key for this page instance. We extract that key from the page, and the file path from the cache and re use those settings to persist the view state and control state

So, we are mindful of finite server resources by only storing the file path in cache, we are mindful of download page size by only storing the unique key in the page (instead of the entire View State), and by using the Page.Cache object to tie everything together, we're giving our persisted View State files a life time. Notice on the last bit of the Page.Cache.Add() method, we're defining ViewStateCacheRemoveCallback as our callback method when an item is removed from cache.

Collapse | Copy Code

```
public static void ViewStateCacheRemoveCallback(string key,
       object value, CacheItemRemovedReason reason)
{
    string cacheFile = value as string;
    if (!string.IsNullOrEmpty(cacheFile))
        if (File.Exists(cacheFile))
            File.Delete(cacheFile);
}
```

When the cache makes the callback, it passes on what that cache object contained, which we know is the file path to the persisted View State object. All we have to do when we receive the callback is to delete the physical file.

The Load method basically works in reverse of the Save method..

Collapse | Copy Code

```
public override void Load()
{
    if (!Page.IsPostBack) return;
    // We don't want to load up anything if this is an inital request

    string vsKey = Page.Request.Form[VSKEY];

    // Sanity Checks
    if (string.IsNullOrEmpty(vsKey)) throw new ViewStateException();
    if (!vsKey.StartsWith(VSPREFIX)) throw new ViewStateException();

    IStateFormatter frmt = StateFormatter;
    string state = string.Empty;

    string fileName = Page.Cache[vsKey] as string;
    if (!string.IsNullOrEmpty(fileName))
        if (File.Exists(fileName))
            using (StreamReader sr = File.OpenText(fileName))
                state = sr.ReadToEnd();

    if (string.IsNullOrEmpty(state)) return;

    Pair statePair = frmt.Deserialize(state) as Pair;

    if (statePair == null) return;

    ViewState = statePair.First;
    ControlState = statePair.Second;
}
```

Some points of interest here are... if the page is not working as a post back, we don't want to load our View State from persistence. This solves the problem of a user loading stale data. The Load() method gets its View State key from the page, then gets the path to the persisted View State file from the Page.Cache using that View State key. The file is then read, de-serialized into a Pair object, then ViewState and ControlState are loaded from the Pair object.

The Browsers File

The last point of order is to wire up the PageStateAdapter for use with a simple *.browser* file located in *App_Browsers*:

Collapse | Copy Code

```
<browsers>
  <browser refID="Default">
    <controlAdapters>
      <adapter controlType="System.Web.UI.Page" adapterType="PageStateAdapter" />
    </controlAdapters>
  </browser>
</browsers>
```

Conclusion

View State, while a powerful tool, can very quickly become the demise of your users in experiencing your website to the fullest. By making use of built-in ASP.NET technologies like Cache and Control Adapters, we can effectively and accurately persist View State on the server instead of streaming it down to the user. Using the outlined `PageStateAdapter` has enabled our developers to make full use of our existing ASP.NET skill set, including our heavy use of ASP.NET AJAX, without having to re-design our entire page framework; it is truly a drop in solution. Our `PageStateAdapter` method could be very easily modified to persist View State files to a common location for a multi-homed web environment, and the `Page.Cache` replaced with a common mechanism in that same environment. We've been using this technique in our training environment under both real world load and extreme test load, with very impressive results.

In our code base, we've also added to the global `Application_Start` and `Application_End` events to delete all the *.cache* files that might be present. This takes care of any artifacts that might be around when the server reboots, or that might have been missed in the cache remove callback.

Points of Interest

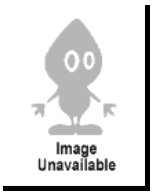
When we first started down the path of persisting View State on the server, we were stuffing the entire thing into `Page.Cache`. It worked fine on our workstations and our development server. It even worked find on our training server.. until that training server went under load. 100 users and three hours later, the sever came crashing down, falling flat on its face after running out of memory. Point of Interest: `Page.Cache`.. is in memory.

History

- 2008-08-06 - Modified to make Persister Page Instance aware: Efficient Server-Side View State Persistence, Two Dot DOH!!
- 2008-08-05 - Created initial article based on my blog post: Efficient Server-Side View State Persistence

License

About the Author

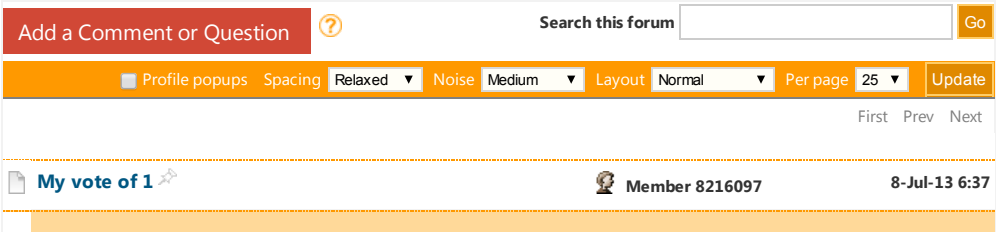


datacop
Team Leader ICAN
United States

Jason Monroe is a Project Manager / Architect with the ICAN grant for the Indiana Department of Education. He has been a professional software developer for 13 years, with the last 8 years focusing on Microsoft technologies.

Article Top

Comments and Discussions



Issues with UpdatePanels and partial postbacks? [modified] Ivan Mitev 3-Aug-12 5:14

It seems that the default file caching of viewstate doesn't work well with partial postbacks. At least in couple of cases, it broke the logic in our existing pages.

Probably we could just use different viewstate handling for those particular pages... but I wondered if there is some generic way to seamlessly handle these scenarios.

It seems that the problem was with failing to send back the hidden field. This could be addressed by:

```
// register the hidden field
ScriptManager scriptManager = ScriptManager.GetCurrent(Page);
if (scriptManager != null && scriptManager.IsInAsyncPostBack)
{
    // the postback is from UpdatePanel
    ScriptManager.RegisterHiddenField(Page, VSKEY, vsKey);
}
else
{
    // normal postback
    Page.ClientScript.RegisterHiddenField(VSKEY, vsKey);
}
```

Ivan Mitev

modified 6-Aug-12 3:09am.

Reply · Email · View Thread · Permalink · Bookmark

Issues when using the browser Back button?	**Ivan Mitev**	**30-Jul-12 3:28**
Re: Issues when using the browser Back button? new	xMANIGHTx	20hrs 42mins ago
Thank you	**Prashant Trambake**	**20-Jun-12 5:43**
Thanks a million!	**Ben Lenihan**	**30-May-12 12:04**
Works great! Just one problem we are seeing...	**Polleitor**	**28-Mar-12 17:33**
Re: Works great! Just one problem we are seeing...	Ivan Mitev	26-Jul-12 10:51
Great! A question too.	**navinmishra**	**13-Sep-11 13:18**
Broken link	**druffyw**	**8-Aug-11 8:56**
Random errors using page.cache but not Session or HttpRuntime.Cache	**maurolo**	**10-Feb-11 3:01**
Re: Random errors using page.cache but not Session or HttpRuntime.Cache	Kalpesh Desai	29-Oct-11 1:21
I wanted to make a note of a security flaw in your implemention.	**squareplanet**	**11-Jan-10 10:21**
Re: I wanted to make a note of a security flaw in your implemention.	MarjaR	11-Mar-11 9:17
Good solution	**Shelly Saunders**	**20-Nov-09 1:29**
Superb!	**inharry**	**23-Sep-09 14:36**
great solution	**sistec**	**21-Aug-09 12:41**
Is it work with .Net 2.0	**maxcom**	**4-Dec-08 8:59**
Re: Is it work with .Net 2.0	datacop	23-Dec-08 7:48

Last Visit: 20-Jun-14 8:46 Last Update: 22-Jun-14 22:06 Refresh 1

General News Suggestion Question Bug Answer Joke Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.