How To Save ASP.NET View And Control State In Cache On The Server Instead Of Using Hidden Fields

Jürgen Bäurle

September 2006

Parago Media GmbH & Co. KG

Introduction

In the area of web development it's a matter of common knowledge that sequenced HTTP requests are stateless. To maintain state between HTTP requests, ASP.NET introduced the concept of page state, also known as "View State" to store data between page requests. The view state consists of control and page settings and data that make up the web form itself.

With ASP.NET 2.0, Microsoft introduced an additional concept, called "Control State". The control state is similar to view state but works independent of it. View state can be disabled by developers, control state not. That implies, data stored as control state should be essential and critical for control functionality and data that must be available on postbacks.

In order to save respectively load view state and control state the ASP.NET storage mechanisms calls the method SavePageStateToPersistenceMedium respectively LoadPageStateFromPersistenceMedium on the Page class. Once you analyze the Page class with Lutz Roeder's ".NET Reflector" you see both of the methods will get their concrete PageStatePersister instance from the property with same name. The code for property PageStatePersister shows that the default implementation creates an instance of type HiddenFieldPageStatePersister to save and load view and control state in a hidden form field. The parameter passed to SavePageStateToPersistenceMedium is of type Pair in .NET 2.0, containing both view state in property Second and control state in property First. Prior to version 2.0, just the view state itself is passed.

Knowing this procedure, there are basically two ways to implement your own storage mechanisms. This article uses the server-side cache as page state storage, described in the following sections.

Implementation I

One implementation of the server-side cache storage is to overide the methods SavePageStateToPersistenceMedium and LoadPageStateFromPersistenceMedium of the Page class. For this purpose we will create a class BasePageSimple which inherits from Page class. Here the complete sample code:

```
public class BasePageSimple : Page {

  protected override void SavePageStateToPersistenceMedium(object state) {

    StringBuilder key=new StringBuilder();

    // Generate a unique cache key
    key.Append("VIEWSTATE_")
       .Append(Session!=null?Session.SessionID:Guid.NewGuid().ToString())
       .Append("_")
       .Append(Request.RawUrl)
       .Append("_")
       .Append(DateTime.Now.Ticks.ToString());

    // Add view state and control state to cache
    Cache.Add(
       key.ToString(),
       state,
       null,
       DateTime.Now.AddMinutes(15),
       Cache.NoSlidingExpiration,
       CacheItemPriority.Default,
       null
       );
```

```
    // Register hidden field to store cache key in
    ClientScript.RegisterHiddenField("__VIEWSTATE_CACHEKEY", key.ToString());

  }

  protected override object LoadPageStateFromPersistenceMedium() {

    // Get the cache key from the web form data
    string key=Request.Params["__VIEWSTATE_CACHEKEY"] as string;

    // Abort if cache key is not available or valid
    if(String.IsNullOrEmpty(key)||!key.StartsWith("VIEWSTATE_"))
      throw new ApplicationException("Missing vaild __VIEWSTATE_CACHEKEY");

    // Return cached view state and control state
    return Cache[key];

  }

}
```

A web form will inherit it's base functionality from BasePageSimple class instead directly from Page. The solution will save a page specific cache key in a hidden field, called "`__VIEWSTATE_CACHEKEY`". The cache key will be used to save and load the view state and control state to and from the server-side cache.

The SavePageStateToPersistenceMedium method generates an unique cache key containing among others the URL of the web form:

```
key.Append("VIEWSTATE_")
   .Append(Session!=null?Session.SessionID:Guid.NewGuid().ToString())
   .Append("_")
   .Append(Request.RawUrl)
   .Append("_")
   .Append(DateTime.Now.Ticks.ToString());
```

After adding the passed view state (parameter state) to the cache using the created key, the next step will register a hidden field to save the key within the web form:

```
ClientScript.RegisterHiddenField("__VIEWSTATE_CACHEKEY", key.ToString());
```

The LoadPageStateFromPersistenceMedium method is requesting the above registered hidden field to get the cache key and receive the view state and control state:

```
string key=Request.Params["__VIEWSTATE_CACHEKEY"] as string;
…
return Cache[key];
```

That's it.

Implementation II

The other way to realize a server-side page state storage is to provide a concrete implementation of the abstract class PageStatePersister. In our case that will be the CachePageStatePersister class. In order to use the storage mechanisms we also need to create a new base class named BasePage derived from Page class. The BasePage class is overriding now just the property PageStatePersister of the Page class. Here the code for BasePage:

```
public class BasePage : Page {

  CachePageStatePersister cachePageStatePersister;

  public BasePage()
    : base() {
    cachePageStatePersister=new CachePageStatePersister(this);
  }

  protected override PageStatePersister PageStatePersister {
    get {
      return cachePageStatePersister;
    }
  }

}
```

The BasePage constructor is initialzing a new instance of the CachePageStatePersister class and saves it in a private member field. The object instance is returned when the overridden Page class property PageStatePersister is called by the method SavePageStateToPersistenceMedium and LoadPageStateFromPersistenceMedium. These methods are using the property PageStatePersister to get an actual implementation of the PageStatePersister class.

Microsoft's default implementation for the Page.PageStatePersister property looks as follows:

```
protected virtual PageStatePersister PageStatePersister {
  get {
    if(_persister==null) {
          PageAdapter adapter=PageAdapter;
      if(adapter!=null)
            _persister=adapter.GetStatePersister();
      if(_persister==null)
        _persister=new HiddenFieldPageStatePersister(this);
    }
    return _persister;
  }
}
```

The getter method is first checking if the private member field _persister has been set, if not the logic tries to create an instance of the PageStatePersister by calling PageAdpater's GetSatePersister method. The default implementation of GetSatePersister is returning a HiddenFieldPageStatePersister instance. If all fails the PageStatePersister property will always return a new HiddenFieldPageStatePersister instance.

Now that we have integrated ourselves into the page logic we still need to look at the CachePageStatePersister class itself. The CachePageStatePersister is drived from the abstract PageStatePersister class. Here the sample code:

```
public class CachePageStatePersister : PageStatePersister {

  public CachePageStatePersister(Page page)
    : base(page) {
  }

  public override void Load() {

    // Get the cache key from the web form data
    string key=Page.Request.Params["__VIEWSTATE_CACHEKEY"] as string;

    // Abort if cache key is not available or valid
    if(String.IsNullOrEmpty(key)||!key.StartsWith("VIEWSTATE_"))
      throw new ApplicationException("Missing vaild __VIEWSTATE_CACHEKEY");

    Pair state=Page.Cache[key] as Pair;

    // Abort if cache object is not of type Pair
    if(state==null)
      throw new ApplicationException("Missing vaild __VIEWSTATE_CACHEKEY");

    // Set view state and control state
    ViewState=state.First;
    ControlState=state.Second;

  }

  public override void Save() {

    // No processing needed if no states available
    if(ViewState==null&&ControlState!=null)
      return;

    StringBuilder key=new StringBuilder();

    // Generate a unique cache key
    key.Append("VIEWSTATE_")
       .Append(Page.Session!=null?Page.Session.SessionID:Guid.NewGuid().ToString())
       .Append("_")
       .Append(Page.Request.RawUrl)
       .Append("_")
       .Append(DateTime.Now.Ticks.ToString());

    // Save view state and control state separately
    Pair state=new Pair(ViewState, ControlState);
```

```
    // Add view state and control state to cache
    Page.Cache.Add(
      key.ToString(),
      state,
       null,
       DateTime.Now.AddMinutes(15),
       Cache.NoSlidingExpiration,
       CacheItemPriority.Default,
       null
    );

    // Register hidden field to store cache key in
    Page.ClientScript.RegisterHiddenField("__VIEWSTATE_CACHEKEY", key.ToString());

  }

}
```

The code is similar to the first implementation, but distinguishs view state and control state to save it separately in the corresponding properties provided by the base class PageStatePersister. Therefore it caches the state information using an instance of type Pair. The class Pair offers just two public fields, First and Second. These properties are of type object.

Summary

Microsoft has done a great job in the area of extending ASP.NET. Implementing own PageStatePersisters is a rarely used feature, but it can be very helpful in avoiding transferring a lot of overhead between server round trips. The sample is using the server-side cache, but you could also save state information in a database or on the file system. To serialize and deserialize states you can easily use the StateFormatter property of the Page class.

The source code for the CachePageStatePersister is available for download on my homepage.

Contact Information

If you have any feedback or suggestions, please feel free to contact me:

Jürgen Bäurle
jbaurle@parago.de
http://www.parago.de/jbaurle

Parago Media GmbH & Co. KG
Im Wengert 3 | 71336 Waiblingen, Germany | Phone +49.7146.861803 | Internet http://www.parago.de