Software Craftsmanship Guild

526 S. Main St, Suite 609

Akron, OH 44311

SOFTWARE
CRAFTSMANSHIP GUILD

Techniques for Temporary Storage in ASP.NET

Software Craftsmanship Guild

Why do we care?

The web is stateless, which means that every request a whatever data we were working with is destroyed once it is sent to the browser.

Sometimes though, we have data that we want to keep around.

Ways to keep things around

1. Session
2. Cookies
3. Cache
4. TempData

Session

Session is a dictionary object available in all controllers. The session key is a String, and the value is an Object, so when you remove things from session you must cast it.

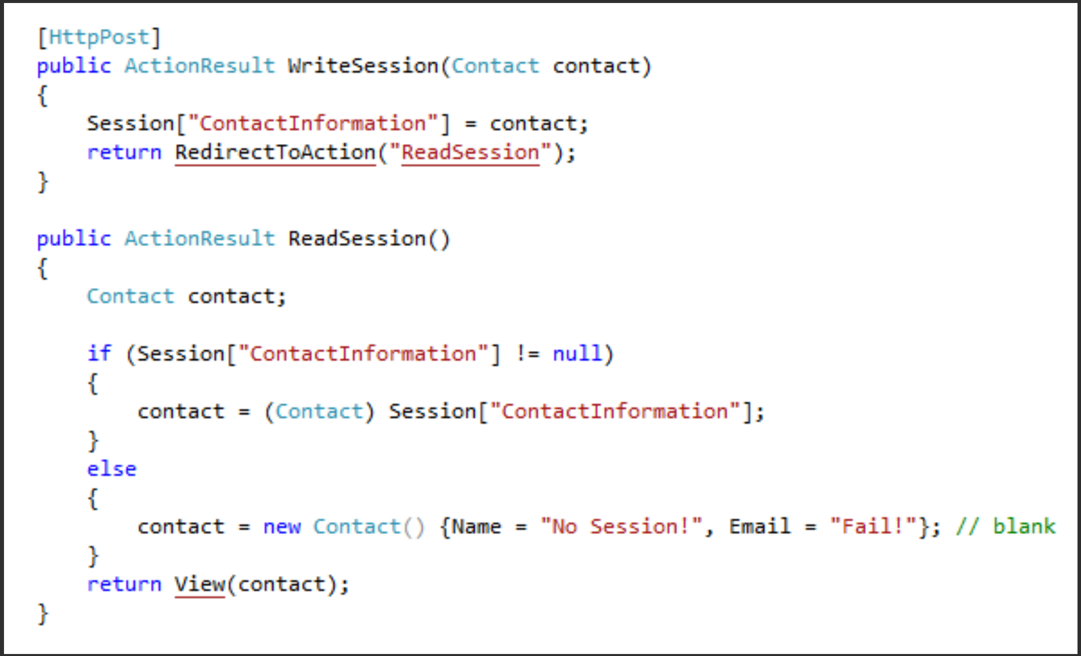
Session is per user, so each unique visitor to your website gets their very own copy of session.

Advantages:

1. We can store user data here conveniently.
2. It is on the server, so it is relatively secure

Disadvantages:

1. It doesn't scale well
2. If the web server resets for any reason, session is destroyed, so null checking is required
3. It will eventually time out and be destroyed, so it doesn't last between visits
4. Difficult to use in web-farm scenarios



```
[HttpPost]
public ActionResult WriteSession(Contact contact)
{
    Session["ContactInformation"] = contact;
    return RedirectToAction("ReadSession");
}

public ActionResult ReadSession()
{
    Contact contact;

    if (Session["ContactInformation"] != null)
    {
        contact = (Contact) Session["ContactInformation"];
    }
    else
    {
        contact = new Contact() {Name = "No Session!", Email = "Fail!"}; // blank
    }
    return View(contact);
}
```

Verdict: avoid using when possible

Cookies

Cookies are stored in the browser client of the website visitor. The Cookie object is a dictionary that can read and write data from the browsers cookies.

All cookies for a site are automatically sent with each request and you can tell the browser how long to keep cookie data around.

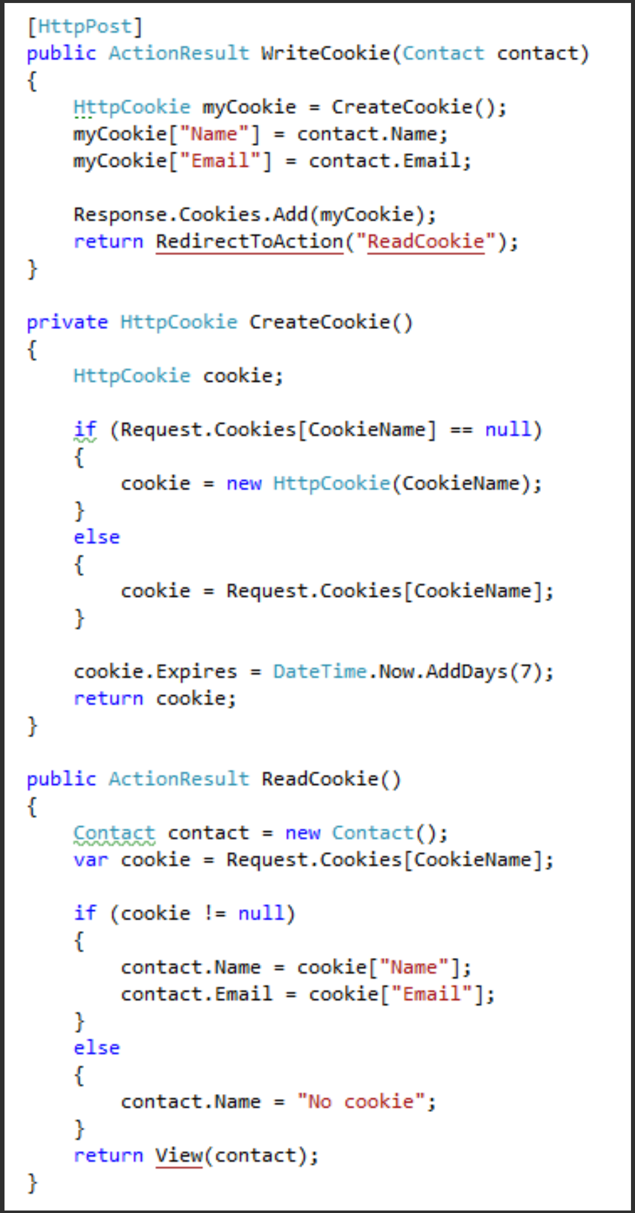
Advantages:

1. Data can be stored between website visits.

Disadvantages:

1. It lives on the client, so it should not store secure data.
2. User can remove cookies, so you can't depend on it being there

In general, cookies are good for storing settings and login tokens.



```
[HttpPost]
public ActionResult WriteCookie(Contact contact)
{
    HttpCookie myCookie = CreateCookie();
    myCookie["Name"] = contact.Name;
    myCookie["Email"] = contact.Email;

    Response.Cookies.Add(myCookie);
    return RedirectToAction("ReadCookie");
}

private HttpCookie CreateCookie()
{
    HttpCookie cookie;

    if (Request.Cookies[CookieName] == null)
    {
        cookie = new HttpCookie(CookieName);
    }
    else
    {
        cookie = Request.Cookies[CookieName];
    }

    cookie.Expires = DateTime.Now.AddDays(7);
    return cookie;
}

public ActionResult ReadCookie()
{
    Contact contact = new Contact();
    var cookie = Request.Cookies[CookieName];

    if (cookie != null)
    {
        contact.Name = cookie["Name"];
        contact.Email = cookie["Email"];
    }
    else
    {
        contact.Name = "No cookie";
    }
    return View(contact);
}
```

SOFTWARE CRAFTSMANSHIP GUILD

Cache

The Cache is a dictionary object available in all controllers. The cache key is a String, and the value is an Object, so when you remove things from cache you must cast it.

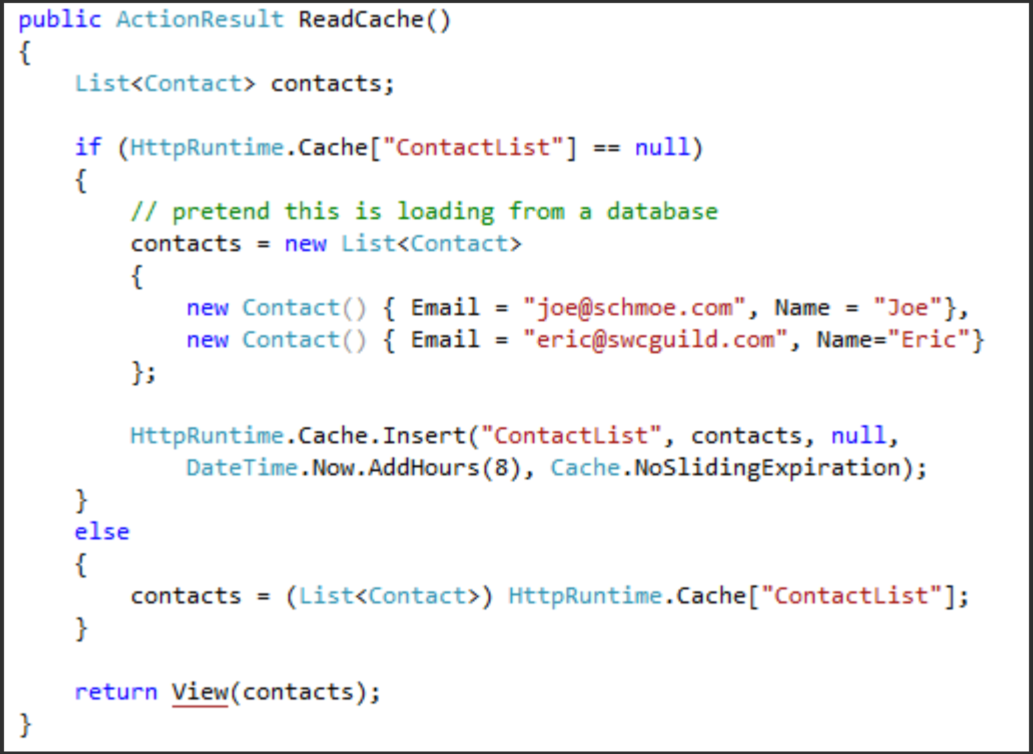
There is only one shared cache for all requests.

Advantages:

1. Common data that does not change frequently is great for cache (ex: list of US states)
2. Takes load off your database by storing repeated requests
3. Can be offloaded to a “cache” server, which can make it high performance

Disadvantages:

1. Cache must be invalidated if any cached data changes, which adds complexity to management.



```
public ActionResult ReadCache()
{
    List<Contact> contacts;

    if (HttpRuntime.Cache["ContactList"] == null)
    {
        // pretend this is loading from a database
        contacts = new List<Contact>
        {
            new Contact() { Email = "joe@schmoe.com", Name = "Joe"},
            new Contact() { Email = "eric@swcguild.com", Name="Eric"}
        };

        HttpRuntime.Cache.Insert("ContactList", contacts, null,
            DateTime.Now.AddHours(8), Cache.NoSlidingExpiration);
    }
    else
    {
        contacts = (List<Contact>) HttpRuntime.Cache["ContactList"];
    }

    return View(contacts);
}
```

TempData

TempData is specific to ASP.NET MVC. It is a dictionary with a key as a String, and the value is an Object, so when you remove things from TempData you must cast it.

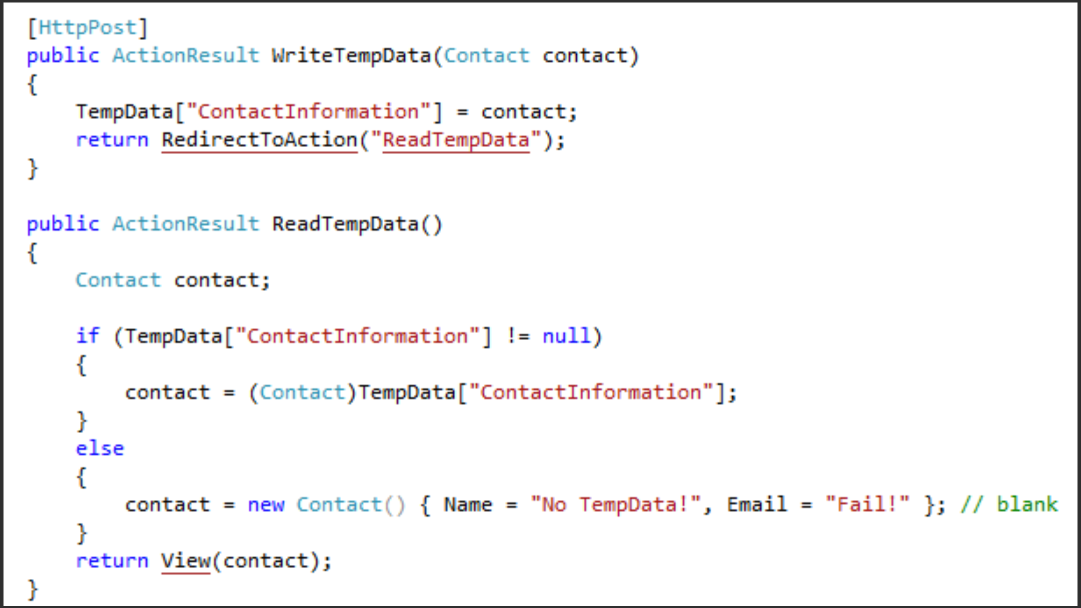
TempData is per user and only lasts for a single request.

Advantages:

1. It does not stay in memory long like session.
2. It makes passing data across controllers easier.
3. It can be useful in messaging scenarios

Disadvantages:

1. You have to be careful to check whether the data is actually there before using it



```
[HttpPost]
public ActionResult WriteTempData(Contact contact)
{
    TempData["ContactInformation"] = contact;
    return RedirectToAction("ReadTempData");
}

public ActionResult ReadTempData()
{
    Contact contact;

    if (TempData["ContactInformation"] != null)
    {
        contact = (Contact)TempData["ContactInformation"];
    }
    else
    {
        contact = new Contact() { Name = "No TempData!", Email = "Fail!" }; // blank
    }
    return View(contact);
}
```

SOFTWARE CRAFTSMANSHIP GUILD

Summary

Storage Type	Use When
Session	You need specific user data to be stored for the duration that the user is active on the website. Generally when you need temporary storage for secure data. Due to memory concerns, session should be avoided if possible.
Cookies	You need specific user data to be stored between website visits. Do not put secure data in cookies!
Cache	You have common data that can be shared by many users. Generally this is to take load off of databases for data that does not change frequently.
TempData	You need some information to persist for one additional request. Typically used to pass object data in a RedirectToAction() call.