15 Simple ASP.NET Performance Tuning Tips

SIMON TIMMSJANUARY 31, 2018DEVELOPER TIPS, TRICKS & RESOURCES

Performance of your ASP.NET web application is important. There is a lot of evidence to suggest that slow loading times and clunky interaction will drive customers elsewhere. Even in the case of internal applications where the users have no option but to use the application, their satisfaction is tightly coupled to speed.

There are a ton of ways to improve the performance of a website, let's look at fifteen of them.

1. Measure everything

The first thing to do is gather a baseline of your application's performance. Sometimes you'll make a change to the site, thinking it will improve performance, but it will actually reduce performance. Although not quite a black art, performance tuning gives you unexpected results. Measuring performance should be a holistic exercise measuring server, JavaScript, and loading performance. Put away your stopwatch: there are some great tools for measuring performance such as Prefix.

Prefix will allow you to highlight slow queries, large JavaScript files, and more. The measurements should give you an idea of which of these optimizations might help you the most. Make yourself a list, and order it from largest impact to smallest. You'll often find that the items at the bottom of the list aren't important enough to worry about until far down the road.

2. Pick the low-hanging fruit first

Once you have your list, then pick the item with the largest impact first. If you can show a large impact on your users right away, then it will give you some great political capital to continue optimizing and you'll feel fantastic. Items which are global (JavaScript loading, CSS loading, and their ilk) will likely have a larger impact than changes to a single page.

The rest of this blog post is arranged in a rough order of things that have a large impact to those with a small impact. Obviously, this will vary slightly from site to site so be sure to take it with a grain of salt.

3. Enable compression

The HTTP protocol is not a particularly efficient protocol and, by default, there is no compression of the content. Some web resources are already compressed, especially images, but HTML, CSS and JavaScript are typically transferred as text. Even the most archaic of browsers support compression of HTTP content using the gzip algorithm. The savings from using gzip compression on an HTML file are around two thirds; that is to say that a 100kb uncompressed file will end up being 33kb over the wire. This is a stunning savings!

For the more adventurous there is an updated algorithm called Brotli, which is showing great promise and is quite well supported on modern browsers.

4. Reduce HTTP requests

Every time the browser needs to open a connection to the server there a tax that must be paid. This tax is in the form of TCP/IP connection overhead. This problem is especially noticeable in scenarios with high latency where it takes a long time to establish these new connections. Add to this the fact that browsers limit the number of requests they will make to a single server at once, and it becomes apparent that reducing the number of HTTP requests is a great optimization.

Aside: Latency vs. bandwidth

When optimizing web page loading, it is important to understand the difference between latency and bandwidth. Let's imagine that you have twenty donkeys you need to move from Banff to the Grand Canyon (two popular donkey hotspots). To get the donkeys moved as quickly as possible, you need to optimize two things: how many donkeys you move at once, and how long it takes to move a donkey.

Bandwidth is how many donkeys you can move at once – in high bandwidth scenarios, you can move a lot of donkeys at once in your cattle hauler. In low bandwidth scenarios, you can only fit one donkey in the passenger seat of your 2001 Honda Civic, and that one donkey insists on listening to Destiny's Child the whole way.

Latency is how quickly you drive between Banff and the Grand Canyon. High latency means there are lots of delays along the way, slowing down the transit time. The low-latency donkey move means that you drive straight through the night, and don't stop at all at any of the donkeys' favourite tourist sites. Ideally, you want to move as many donkeys at a one time and avoid any stops along the way.

Tooling

Depending on the type of the resource being requested from the server, there are a few different approaches to reducing the number of requests. For JavaScript, concatenating the scripts together into a single file using a tool like webpack, gulp or grunt can bundle together all the JavaScript into a single file. Equally, we can combine CSS files into a single file using tasks in the same build tools we use for JavaScript.

For images, the problem is slightly more difficult. If the site uses a number of small images, then a technique called CSS Spriting can be used. In this, where we combine all the images into a single one and then use CSS offsets to shift the image around and show just the single sprite we want. There are some tools to make this process easier, but it tends to be quite manual. An alternative is to use an icon font.

This brings us to HTTP 2.

5. HTTP/2 over SSL

The new version of HTTP, HTTP/2, introduces a number of very useful optimizations. First, the compression we spoke of in #3 has been extended to also cover the protocol headers. More interestingly, the connection with the server can transfer more than one file at a go using a mechanism known as “pipelining”. This means that the reduction of HTTP requests by combining files is largely unnecessary. The difference is quite spectacular.

Most every browser has support for some version of HTTP/2 but ironically, the limitation tends to be more on the server side. For instance, at the time of writing, Azure Web Apps do not have support for HTTP/2.

The server can now make intelligent decisions about the page content and push resources down before they are even

requested. So if the index page contains a JavaScript file that won't be discovered until the browser has parsed the entire page, the server can now be instructed to start the transfer of the file before the browser has realized it needs it.

SSL is part of this tip because all browsers that support HTTP2 require that it be served over HTTPS.

6. Minify your files

Compression is a great tool for reducing the amount of data sent over the wire, but all the compression algorithms used to send HTML, CSS and JavaScript are lossless compression algorithms. This means that the result of doing compress(x) => decompress(x) always equals x. With some understanding of what it is that is being compressed, we can eek out some additional gains in size reduction. For instance, the JavaScript

```
function doSomething(){

    var size_of_something_to_do = 55;

    for (var counter_of_stuff = 0;

        counter_of_stuff < size_of_something_to_do;

        counter_of_stuff++) {

        size_of_something_to_do--;

    }

}
```

is functionally equivalent to

```
function doSomething(){var a=55;for(var b=0;b<a;b++){a--;}}
```

This is because the scope of the variables is entirely private and the whitespace largely unnecessary. This process is

called minification. Similar compression techniques can be applied to CSS and even to HTML.

7. Load CSS first

Load the CSS content of your site first, preferably in the head section of the page.

To understand the reasoning here, you need to understand a little bit about how browsers achieve their incredible speed. When downloading a page, the browser will attempt to start rendering the application as soon as it has any content. Often what it renders is something of a guessing game because the browser doesn't know what content on the page may invalidate the guesses it has made. When the browser realizes that it has made an incorrect guess about how the page should be rendered, then all the work that was done needs to be thrown out and started over again. One of the things which causes one of these reflows is the addition of a new stylesheet. Load style sheets first to avoid having a style that alters an already-rendered element.

8. Load JavaScript last

JavaScript is a complete about-face from CSS, and should be loaded last. This is because we want the page to render as quickly as possible, and JavaScript is not typically necessary for the initial render. Users will typically take a moment to read the page and decide what to do next. This window is be used to load scripts in the background and light up the page's interaction.

Let's attach a caveat to this rule: if your site is a heavy user of JavaScript, for instance, such as an Angular or React application, then you may find that loading JavaScript last is actually detrimental. You may wish to investigate loading only the JavaScript necessary to bootstrap the application, and loading more in the background. If speed is really important, you can even

investigate what are called isomorphic or universal applications. The pages in these applications are rendered on the server side, and then the JavaScript application attaches to the already-rendered HTML and takes over from there. Thes applications have the advantage of being fast to load without giving up the seamless nature of single page applications.

9. Shrink images

In an ideal world, your site would contain no images at all. It is typically a lot more efficient to use inline-SVG or CSS tricks to create vector art for your pages because they are far smaller than raster images. However, it is unlikely that you'll be able to do this, so instead work on shrinking images. Figuring out the right encoding settings can be difficult, but there are some really impressive services to do it for you. I quite like this tinypng service, and only just a little bit because it has a cool panda for a logo.

A WAVING PANDA EXAMPLE FROM TINYPNG

There are also plugins for your JavaScript build tool that perform much of the same optimization, but sadly without the panda.

10. Check your queries

ORMs (object-relational mappers) have been highly beneficial in increasing developer productivity, however they provide a layer of abstraction that can introduce sub-optimal queries. Prefix will highlight times when you might have n + 1 select errors, or are retrieving too much data from the server. It is surprising how easy it is to fix these problems by using eager loading over lazy loading, and examining projections. Microsoft has some more in-

depth recommendations available to optimize how EF (Entity Framework) calls SQL.

11. Cache your pages

Very frequently, the data on your pages changes at a slow pace. As an example, the hot questions page on Stack Overflow could be updated in real time, but the data changes are not significant enough to bother re-querying the database. Instead of taking a hit going to the database and re-rendering a complex looking page, we can shove the page into a cache and serve subsequent requests using that data.

If you happen to be using ASP.NET MVC caching, the response from an action is as simple as adding a single attribute to the action.

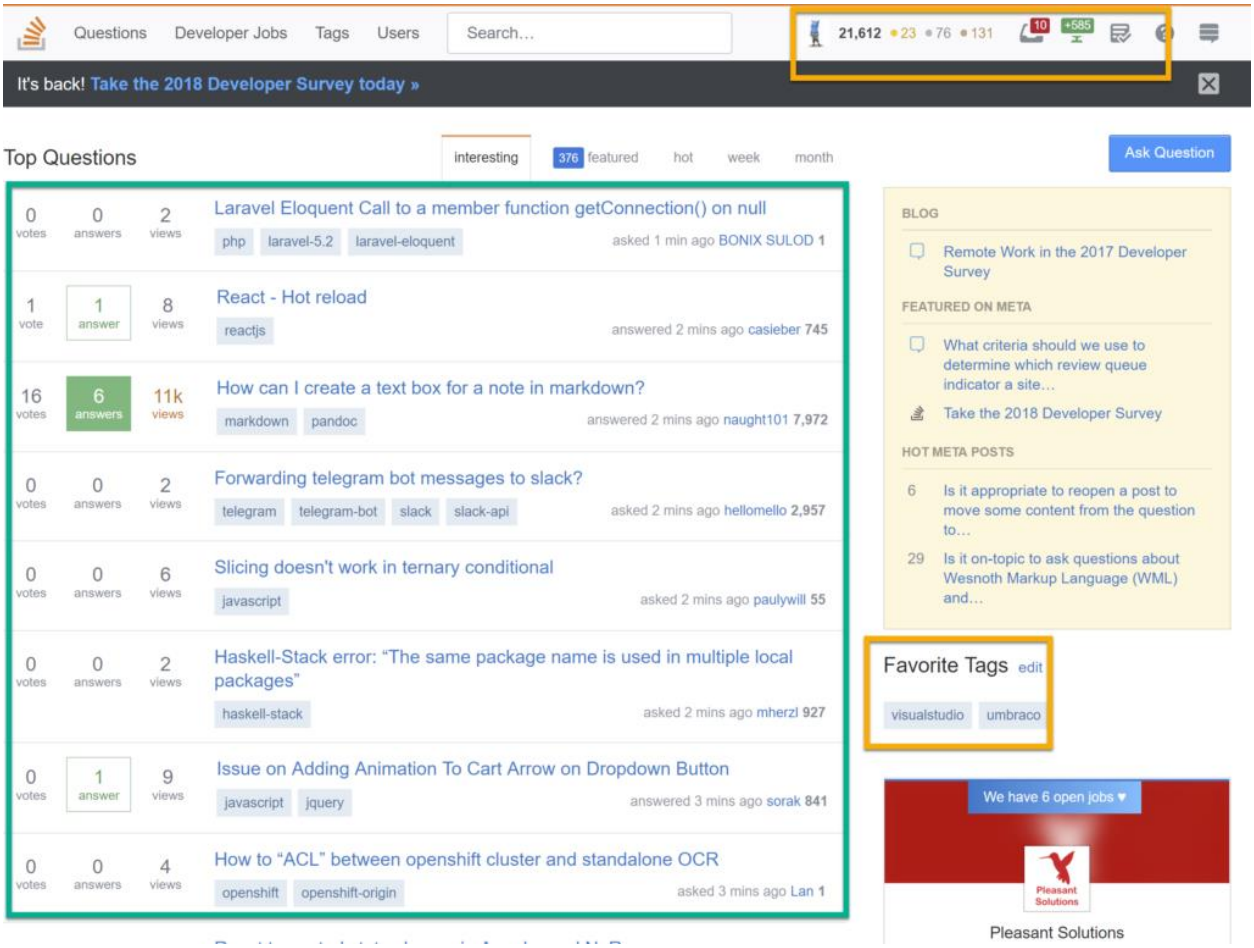
```
[HandleError] public class HomeController : Controller {

    [OutputCache(Duration=10, VaryByParam="none")]

    public ActionResult Index() {

        return View();

    }

}
```

Caching the entirety of a page may not be exactly what you want, in which case tip 12 is for you.

12. Cache parts of your pages

You may want to cache only part of your page; this is colloquially known as donut hole caching. It is a useful approach when you have user-specific data mixed with general data on the same page. The user data varies by the user, while the rest of the page

is the same for all users. In MVC 5 applications, this is done using partial views, and in MVC Core using caching tag helpers.



THE SECTIONS HIGHLIGHTED IN GREEN ARE CACHED PER PAGE AND ORANGE ARE CACHED PER USER

13. Content delivery network (CDN)

The speed of light is 300000 km/s, which is pretty jolly quick; but despite this high speed, it actually helps to keep your data close to your consumers. There are a ton of content delivery networks, which have edge nodes very close to wherever your users might be.

14. Shrink your libraries

If you're making use of libraries like jQuery, consider that you may not be using all the functions and can use a smaller, more focused library. Zeptojs is a library which supports many of the features of jQuery, but is smaller. Other libraries like jQuery UI provide for constructing personalized packages with features removed. If you're on Angular, then the production build performs tree shaking to remove entire chunks of the library that aren't in use for your project. This reduces the payload sent over the wire, while preserving all the same functionality.

15. Avoid client-side redirects

The final tip is to avoid redirecting users through the use of client-side redirects. Redirects add an extra server trip that, on high-latency networks like cellular networks, is undesirable. Instead, leverage server-side redirects – these don't add the extra server trip. One place where this won't work is redirecting users to an SSL version of your page. For that scenario, HTTP Strict Transport Security and a preload list is your ticket. Adding your site to this preload list will automatically direct traffic to the SSL version of your site.

These tips should give you a real leg up on improving the performance of your website and, hopefully, make your users very happy. If you have more tips you think we've missed, then please leave a comment below. You should also check out Web Performance Optimization: Top 3 Server and Client-Side Performance Tips by Matt Watson.

Schedule A Demo