Bundling And Minifying In ASP.NET Core Applications

Rion Williams Updated date Jan 24 2017 53.5k 3 9

No one likes an application or site that loads slowly.

Thankfully, there are things that we can do, as developers, to help mitigate load times and often turn seconds into milliseconds. Web performance is such a large subject, but this post is going to focus on bundling and minifying resources, particularly CSS and JavaScript files within ASP.NET Core applications.

What's changed in Bundling in .NET Core?

If you have been working in the ASP.NET ecosystem for the past few years, you know that you have been able to rely on the built-in bundling and minification that it offers. While it may not have been the friendliest thing in the world, it generally worked as follows within a BundleConfig.cs file.

```
public static void RegisterBundles(BundleCollection bundles)
{
     // Bundling jQuery
     bundles.Add(new ScriptBundle("~/bundles/jquery")
            .Include("~/Scripts/jquery-{version}.js"));
     bundles.Add(new ScriptBundle("~/bundles/jqueryval")
     bundles.Add(new StyleBundle("~/Content/css")
            .Include("~/Content/site.css"));
}
```

This would handle smashing all of your files together as a single, cachable resource that your users could download once, as opposed to making a request for every single file and dependency in your application; which saves them time.

However, in this new ASP.NET Core world, long gone is the BundleConfig.cs file.

Enter Grunt and Gulp

As ASP.NET Core became more lean and modular, many of the built-in features, like bundling, were removed and those tasks were delegated to other tools and utilities, like Grunt and Gulp.

Grunt and Gulp are both JavaScript based tools for running tasks and building respectively. These tools are easily integrated into ASP.NET Core projects, and are incredibly useful for performing any type of file manipulation, automation, and other tasks like bundling and minifying files as mentioned in the documentation.

The problem here is that it often isn't pretty or easy to look at, especially for folks not familiar with JavaScript.

```
// Defining dependencies
var gulp = require("gulp"),
    rimraf = require("rimraf"),
    concat = require("gulp-concat"),
    cssmin = require("gulp-cssmin"),
    uglify = require("gulp-uglify");
// Defining paths
var paths = {
    js: webroot + "js/**/*.js",
    minJs: webroot + "js/**/*.min.js",
    css: webroot + "css/**/*.css",
    minCss: webroot + "css/**/*.min.css",
    concatJsDest: webroot + "js/site.min.js"
```

```
// Bundling (via concat()) and minifying (via uglify()) Javascript
gulp.task("min:js", function () {
    return gulp.src([paths.js, "!" + paths.minJs], { base: "." })
        .pipe(concat(paths.concatJsDest))
        .pipe(uglify())
        .pipe(gulp.dest("."));
});
// Bundling (via concat()) and minifying (via cssmin()) Javascript
gulp.task("min:css", function () {
    return gulp.src([paths.css, "!" + paths.minCss])
        .pipe(concat(paths.concatCssDest))
        .pipe(cssmin())
        .pipe(gulp.dest("."));
});
```

You can, then, schedule tasks like these to run on certain events (i.e. Build, Publish, etc.) to ensure that your files stay in sync. But, what if you aren't a big fan of JavaScript, or this all just looks too complicated? Surely, there must be an easier way!

An Easier Way - The Bundler and Minifier Extension

Mads Kristensen, the mad scientist behind Visual Studio extensibility, decided to automate this process with the release of the Bundler and Minifier Extension, which integrates into Visual Studio and allows you to easily select and bundle the files you need, without writing a line of code.

Some of the current features are as follows.

- Bundles CSS, JavaScript or HTML files into a single output file
- Saving a source file triggers re-bundling automatically
- Support for globbing patterns
- MSBuild support for CI scenarios supported
- Minify individual or bundled CSS, JavaScript and HTML files

- Task Runner Explorer integration
- Command line support
- Shortcut to update all bundles in solution
- Suppress output file generation
- Convert to Gulp

After installing the extension, you select all of the specific files that you want to include within a bundle and use the "Bundle and Minify Files" option from the extension.

This will prompt you to name your bundle and choose a location to save it at. You'll then notice a new file within your project called bundleconfig.json which looks like the following.

```
[
  {
    "outputFileName": "wwwroot/app/bundle.js",
    "inputFiles": [
      "wwwroot/lib/jquery/dist/jquery.js",
      "wwwroot/lib/bootstrap/dist/js/bootstrap.js",
      "wwwroot/lib/jquery-validation/dist/jquery.validate.js",
      "wwwroot/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"
    ]
  }
]
```

Note

The order in which the files are selected, will determine the order that they appear in, within the bundle. So, if you have any dependencies, ensure that you take that into account.

the bundleconfig.json file. Simply add a minify block like the following to your existing bundle to indicate that you want it minified:

```
[
  {
    "outputFileName": "wwwroot/app/bundle.js",
    "inputFiles": [
      "wwwroot/lib/jquery/dist/jquery.js",
      "wwwroot/lib/bootstrap/dist/js/bootstrap.js",
      "wwwroot/lib/jquery-validation/dist/jquery.validate.js",
      "wwwroot/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"
    ],
    "minify": {
      "enabled": true
    }
  }
]
```

Automating Bundles (via the extension)

Finally, if you want to automate this process, you can schedule a task to run whenever your application is built, to ensure that your bundles reflect any changes within your application.

To do this, you'll need to do the following.

- Open the Task Runner Explorer (via Tools > Task Runner Explorer).
- Right-click on the "Update All Files" option below JSON file name.
- Select your preferred binding from the "Bindings" context menu.

Using dotnet bundle

Since there are quite a few developers out there that are big command-line fans, I couldn't leave you folks out.

existing ASP.NET Core applications and doesn't require any external extensions or script files.

Using BundlerMinifer.Core and Configuring Your Bundles

To use this tool, simply add a reference to BundlerMinifier.Core within the tools section of your existing project.json file, as seen below.

```
"tools": {
  "BundlerMinifier.Core": "2.0.238",
  "Microsoft.AspNetCore.Razor.Tools": "1.0.0-preview2-final",
  "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.0.0-preview2-final"
}
```

After adding the tool, you'll need to add a bundleconfig.json file in your project that will be used to configure the files that you wish to include within your bundles. A minimal configuration can be seen below.

```
[
  {
    "outputFileName": "wwwroot/css/site.min.css",
    "inputFiles": [
      "wwwroot/css/site.css"
    ]
  },
  {
    "outputFileName": "wwwroot/js/site.min.js",
    "inputFiles": [
      "wwwroot/js/site.js"
    ],
    "minify": {
      "enabled": true,
      "renameLocals": true
    },
    "sourceMap": false
  },
  {
    "outputFileName": "wwwroot/js/semantic.validation.min.js"
    ],
    "minify": {
      "enabled": true,
      "renameLocals": true
    }
  }
]
```

Building Your Bundles

After your bundles have been configured, you can bundle and minify your existing files, using the following command.

dotnet bundle

Automating Your Bundles

The Bundling and Minification process can be automated as part of the build process by adding the "*dotnet bundle*" command in the precompile section of your existing project.json file.

```
"scripts": {
  "precompile": [
    "dotnet bundle"
  ]
}
```

Available Commands

You can see a list of all the available commands and their descriptions below.

- *dotnet bundle* - Executes the bundle command using the bundleconfig.json file to bundle and minify your specified files.
- *dotnet bundle clean* - Clears all of the existing output files from disk.

- *dotnet bundle help* - Displays all available help options and instructions for using the Command-Line Interface.

That's all there is to it.

While bundling and minification may be a bit different that you may have been accustomed to, it's going to be one of the smaller things that you have to worry about transitioning to this new modular ASP.NET Core world.

Regardless of the approach that you choose, bundling and minification can translate into smaller requests and quicker load times for your users; and as demonstrated here, it hardly takes any work at all.

Next Recommended Article
Send Email Using Templates In ASP.NET Core

ASP.NET Core Applications | Bundling In ASP.NET Core Applications | Minifying In ASP.NET Core Applications

Rion Williams TOP 500 134 935.2k 4 3
I'm a software developer, designer and Microsoft MVP and I enjoy building all sorts of things.
http://rion.io

9 3

Type your comment here and press Enter Key (Minimum 10 characters)

•203 •10.1k •1.6m 0 0 Reply

Thank you sir. Can I do bundling of ES6 in this manner. I tried, but not able to succeed, the bundle generates, but the import statements still refer to the original files not the bundle.

Srivathsan S Sep 02, 2018

•1888 •4 •0 0 0 Reply

Very nice article ,well explained.

Bikesh Srivastava Jan 25, 2017

•75 •22.5k •2.2m 0 0 Reply