Lab – Despliegue de aplicación Angular

Cuando este listo para desplegar tu aplicación Angular en un servidor remoto, se tendra varias opciones de despliegue.

Opciones de despliegue simples

Antes de desplegar completamente tu aplicación, puedes probar el proceso, la configuración de la compilación y el comportamiento del despliegue utilizando una de estas técnicas provisionales.

Construir y servir desde el disco

Durante el desarrollo, normalmente utilizas el comando ng serve para construir, observar y servir la aplicación desde la memoria local, utilizando webpack-dev-server. Sin embargo, cuando estés listo para desplegar, debes usar el comando ng build para construir la aplicación y desplegar los artefactos de construcción en otro lugar.

Tanto ng build como ng serve limpian la carpeta de salida antes de construir el proyecto, pero sólo el comando ng build escribe los artefactos de construcción generados en la carpeta de salida.

La carpeta de salida es dist/nombre-del-proyecto por defecto. Para que la salida sea una carpeta diferente, cambia el outputPath en angular.json.

A medida que se acerca al final del proceso de desarrollo, servir el contenido de tu carpeta de salida desde un servidor web local puede darte una mejor idea de cómo se comportará tu aplicación cuando se despliegue en un servidor remoto. Usted necesitará dos terminales para obtener la experiencia de recarga en vivo.

- En la primera terminal, ejecuta el comando ng build en modo watch para compilar la aplicación en la carpeta dist.

```
ng build --watch
```

Al igual que el comando ng serve, esto regenera los archivos de salida cuando los archivos fuente cambian.

- En la segunda terminal, instala un servidor web (como lite-server), y ejecútalo contra la carpeta de salida. Por ejemplo

```
lite-server --baseDir="dist/nombredelproyecto"
```

El servidor recargará automáticamente su navegador cuando salgan nuevos archivos.

Este método es sólo para el desarrollo y las pruebas, y no es una forma soportada o segura de desplegar una aplicación.

Deploying una aplicacion

La implementación de su aplicación es el proceso de compilar o construir su código y alojar JavaScript, CSS y HTML en un servidor web.

Esta sección se basa en los pasos anteriores del proyecto Angular que se ha estado desarrollando y se mostrara cómo implementar su aplicación.

Prerequisitos

Una práctica recomendada es ejecutar su proyecto localmente antes de implementarlo. Para ejecutar su proyecto localmente, necesita lo siguiente instalado en su computadora:

- Node.js.
- Angular CLI. Desde la terminal, instale Angular CLI globalmente con:

```
npm install -g @angular/cli
```

Con la CLI de Angular, puede usar el comando ng para crear nuevos espacios de trabajo, nuevos proyectos, servir su aplicación durante el desarrollo o producir compilaciones para compartir o distribuir.

Ejecutando su aplicación localmente

1. Ir a la carpeta del proyecto de proyectos en VS Code y ubicar la carpeta donde esta el código fuente y entra a dicha carpeta, por ejemplo:

   ```
   cd angular-ynqttp
   ```

2. Utilice el siguiente comando CLI para ejecutar su aplicación localmente:

   ```
   ng serve
   ```

3. Para ver su aplicación en el navegador, vaya a http: // localhost: 4200 /. Si el puerto predeterminado 4200 no está disponible, puede especificar otro puerto con el indicador de puerto como en el siguiente ejemplo:

   ```
   ng serve --port 4201
   ```

   Mientras sirve su aplicación, puede editar su código y ver que los cambios se actualizan automáticamente en el navegador. Para detener el comando ng serve, presione Ctrl + c..

Construyendo y hospedando su aplicación

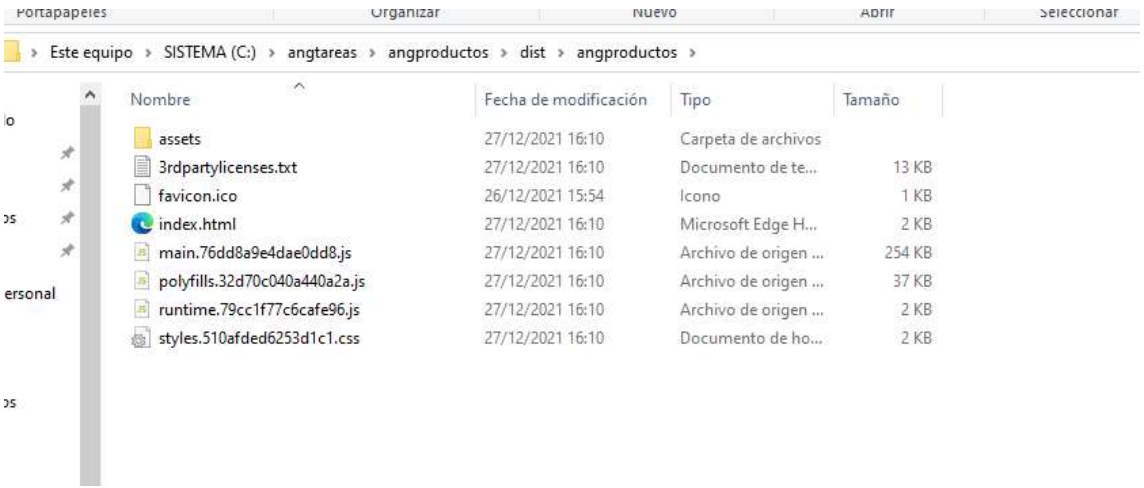
1. Para construir su aplicación para producción, use el comando build. De forma predeterminada, este comando usa la configuración de compilación de producción.

   ```
   ng build
   ```

Este comando crea una carpeta dist en el directorio raíz de la aplicación con todos los archivos que un servicio de alojamiento necesita para atender su aplicación.

Si el comando ng build anterior arroja un error sobre paquetes faltantes, agregue las dependencias faltantes en el archivo package.json de su proyecto local para que coincida con el proyecto completo.

2. Copie el contenido de la carpeta dist/my-project-name en su servidor web. Debido a que estos archivos son estáticos, puede alojarlos en cualquier servidor web capaz de servir archivos; como Node.js, Java, .NET o cualquier backend como Firebase, Google Cloud o App Engine.



Casos de despliegue

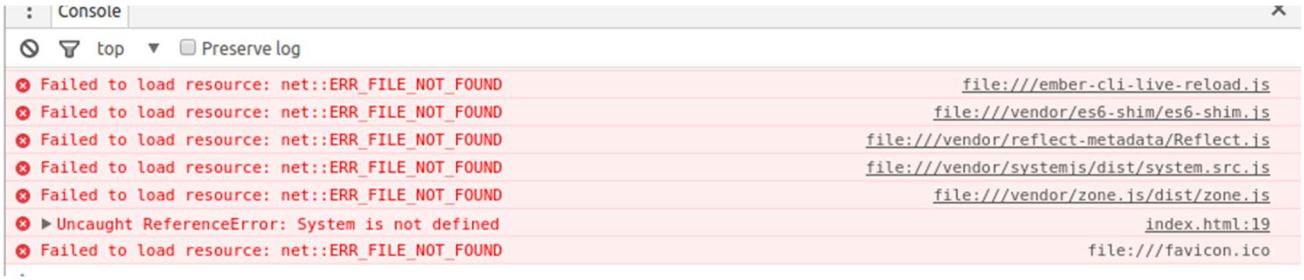
Apache Tomcat

Introducción

Este tema cubriría cómo el proyecto angular-cli está listo para la compilación de producción, cuáles son todos los pasos necesarios tomados antes del despliegue, cómo crear un archivo de guerra para el despliegue del proyecto y, finalmente, cómo configurar el apache tomcat para el despliegue del proyecto angular-cli.

Se tomaron pasos necesarios antes de implementar el proyecto angular-cli para la compilación de producción.

Antes de desplegar el proyecto angular en el servidor, necesitamos construir un proyecto angular para la producción. También debemos cambiar la ruta de enrutamiento en el archivo index.html de <base href="/"> to <base href="./"> si no se realiza, su proyecto no se cargará correctamente, habrá algunos error de enrutamiento que dice el archivo 404 no encontrado.



Comando de compilación Angular-Cli para construir un paquete de proyecto para la implementación de producción

```
ng build –prod
```

El comando anterior con una opción adicional como **–prod** generaría el paquete de proyecto de compilación de producción. Una vez que el comando anterior se ejecute en el directorio raíz de su proyecto, aparecerá un directorio llamado **dist** . En el que aparece todo el conjunto de compilación de producción de su proyecto.

```
C:\angtareas\angproductos>ng build --prod
Option "--prod" is deprecated: No need to use this option as this builder defaults to config
√ Browser application bundle generation complete.
√ Copying assets complete.
√ Index html generation complete.

Initial Chunk Files           | Names         |  Raw Size | Estimated Transfer Size
main.76dd8a9e4dae0dd8.js      | main          | 253.94 kB |                66.02 kB
polyfills.32d70c040a440a2a.js | polyfills     |  36.20 kB |                11.45 kB
styles.510afded6253d1c1.css   | styles        |   1.33 kB |               440 bytes
runtime.79cc1f77c6cafe96.js   | runtime       |   1.05 kB |               600 bytes

                              | Initial Total | 292.52 kB |                78.49 kB

Build at:                          - Hash:                 - Time:     ms

C:\angtareas\angproductos>
```

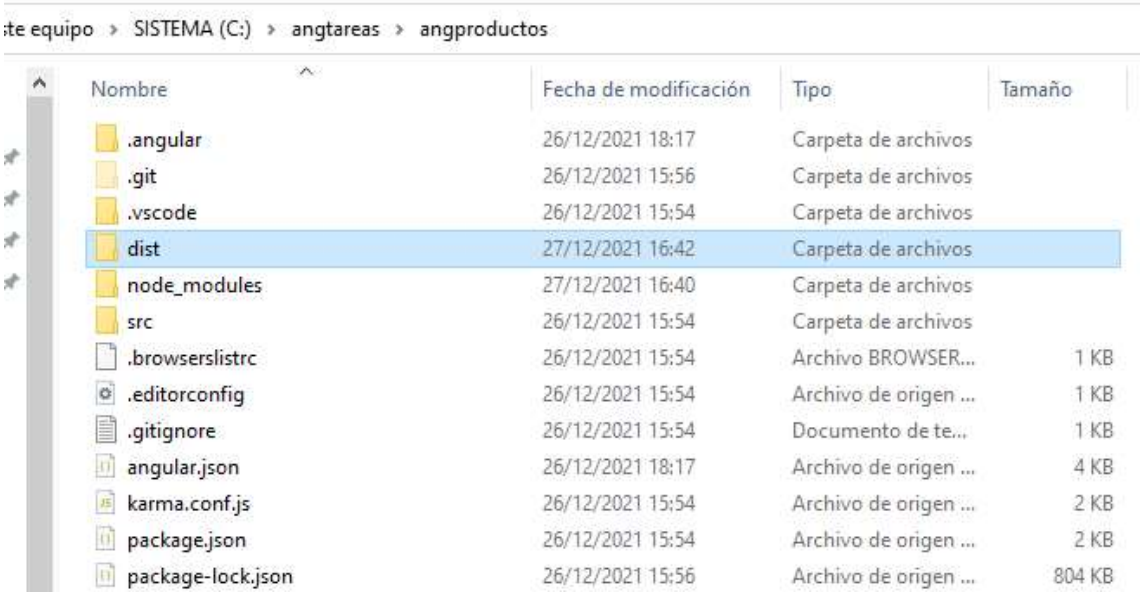
Creación del archivo war para el despliegue de producción del proyecto angular-cli en el servidor tomache apache

Una vez que el directorio **dist** esté listo con sus paquetes construidos de producción. Simplemente abra el directorio **dist** y abra el símbolo del sistema escriba el siguiente comando para crear el archivo **war** para implementar su proyecto en el servidor apache tomcat.

jar cvf dist.war

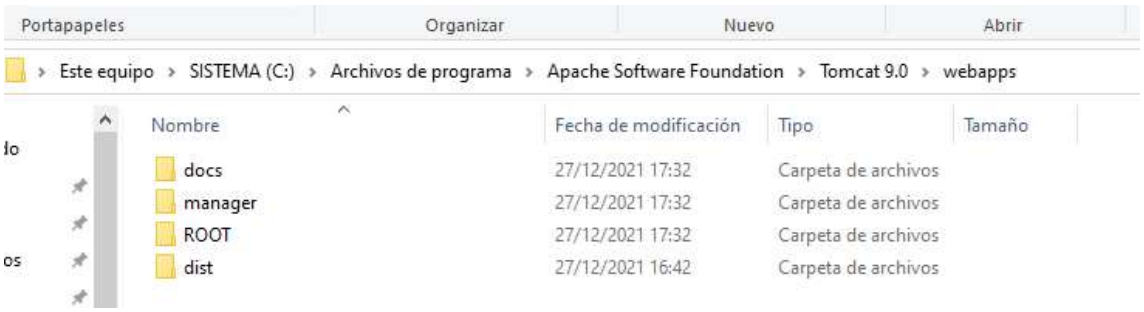
Una vez que los comandos jar anteriores se ejecutan.

Generaría un archivo **dist.war** dentro del directorio **dist**.

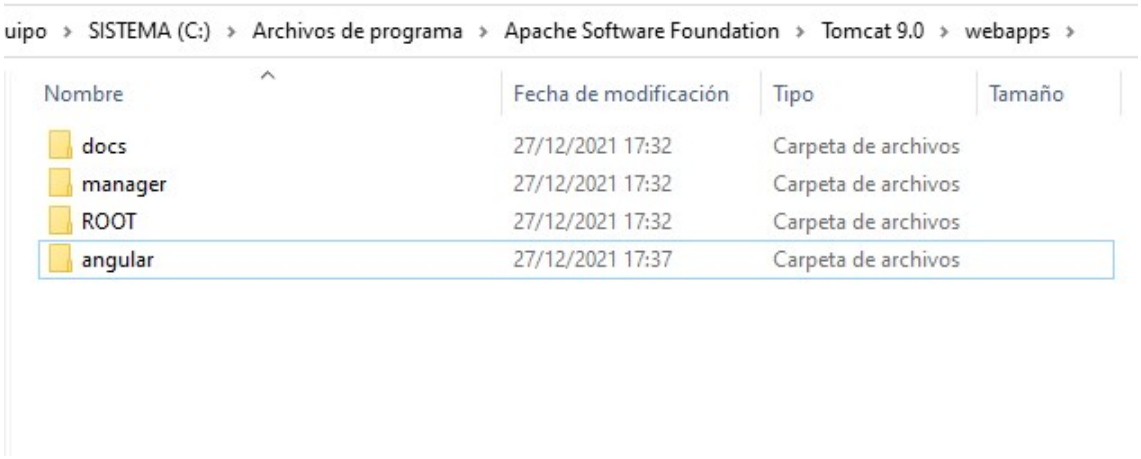


Una vez hecho esto, esta carpeta se copiara dentro de Apache Tomcat.

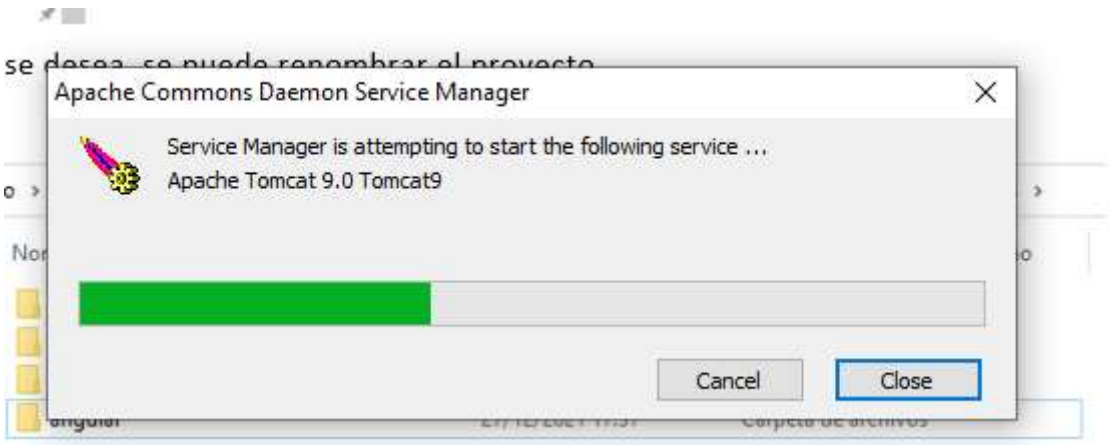
La ruta donde se coloca es en C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps



Si se desea, se puede renombrar el proyecto..



Luego de ello se reinicia el servicio, con lo que el aplicativo se instalara y desplegara en este servidor.



Configuración del apache tomcat para el despliegue del proyecto angular-cli.

1. Corte/Copie el archivo **dist.war** del directorio **dist** y colóquelo en el directorio **apapp** tomcat **webapp** .
2. Vaya a la carpeta **bin de** apache tomcat y haga doble clic en el archivo **startup.bat** .

3. Ahora el servidor tomcat ejecutará el archivo **dist.war** y arrancará el servidor **catalina** tomcat.
4. Una vez que el servidor **Catalina** Tomcat se inicie, abra el navegador web y escriba **localhost: 8080 / dist** y toque la tecla enter para que su proyecto se ejecute en la ventana del navegador web.

Angular Universal

Angular proporciona una solución isomorfa de front-end y back-end para la representación del lado del servidor. Angular Universal (Plataforma unificada), una tecnología que ejecuta aplicaciones angulares en el lado del servidor.

Una aplicación angular estándar se ejecuta en el navegador y muestra la página en el DOM en respuesta a las acciones del usuario.

Angular Universal genera una página de aplicación estática en el servidor a través de un proceso llamado representación del lado del servidor (SSR).

Puede generar estas páginas y usarlas para responder directamente cuando el navegador lo solicite. También puede generar páginas como archivos HTML por adelantado, y luego usarlas como archivos estáticos para que el servidor las use.

Principio de funcionamiento

- Para crear una aplicación universal, debe instalar el paquete **platform-server** . El paquete de platform-server proporciona implementación DOM del lado del servidor, XMLHttpRequest y otras características de bajo nivel, pero ya no depende del navegador.
- Para usar **platform-server** en el lugar de **platform-browser** para compilar la aplicación cliente y ejecutar la aplicación Universal en un servidor web.
- El servidor (el servidor Node Express utilizado en el ejemplo a continuación) pasará la solicitud del cliente a la página de la aplicación **renderModuleFactory** .
- La función renderModuleFactory acepta como entrada una página HTML de plantilla (generalmente index.html), un módulo angular que contiene componentes y una ruta que determina qué componentes mostrar.
- La ruta se pasa de la solicitud del cliente al servidor. Cada solicitud dará una vista adecuada de la ruta solicitada.
- renderModuleFactory en la plantilla **<app>** Marque qué vista se representa y cree una página HTML completa para el cliente.
- Finalmente, el servidor devolverá la página renderizada al cliente.

¿Por qué la representación del servidor?

Tres razones principales:

1. Ayuda Web Crawler (SEO)
2. Mejore el rendimiento en teléfonos móviles y dispositivos de baja potencia.
3. Muestra rápidamente la primera página

Ayuda Web Crawler (SEO)

Google, Bing, Baidu, Facebook, Twitter y otros motores de búsqueda o sitios de redes sociales confían en los rastreadores web para indexar el contenido de su aplicación y hacer que su contenido se pueda buscar a través de la web.

Es posible que estos rastreadores web no naveguen e indexen sus aplicaciones angulares altamente interactivas como lo hacen los humanos.

Angular Universal puede generar una versión estática de la aplicación para usted. Es fácil de buscar y vincular, y no necesita usar JavaScript al navegar. También permite obtener una vista previa del sitio porque cada URL devuelve una página totalmente renderizada.

La habilitación de rastreadores web a menudo se denomina **Posicionamiento en buscadores (SEO)**。

Mejore el rendimiento en teléfonos móviles y dispositivos de baja potencia.

Algunos dispositivos no admiten JavaScript o JavaScript funciona mal, lo que resulta en una experiencia de usuario inaceptable. Para estas situaciones, es posible que necesite una versión de la aplicación renderizada por el servidor y sin JavaScript. Aunque existen algunas restricciones, esta versión puede ser la única opción para aquellos que no tienen forma de usar la aplicación.

Página de inicio de visualización rápida

Mostrar rápidamente la página de inicio es esencial para atraer usuarios.

Si la página se carga durante más de tres segundos, se abandonará el 53% de los sitios móviles. Su aplicación debe comenzar un poco más rápido para atraer la atención de los usuarios antes de que decidan hacer otra cosa.

Con Angular Universal, puede generar "páginas de destino" para sus aplicaciones, y se parecen a la aplicación completa. Estas páginas de destino son HTML puro y se pueden mostrar incluso si JavaScript está deshabilitado. Estas páginas no manejan eventos del navegador, pero pueden usar routerLink para navegar a través de este sitio.

En la práctica, es posible que desee utilizar una versión estática de la página de destino para mantener la atención del usuario. Al mismo tiempo, también cargará la aplicación Angular completa detrás de escena. Los usuarios pensarán que la página de destino aparece casi de inmediato, y cuando se carga la aplicación completa, puede obtener una experiencia interactiva completa.

Ejemplo a usar

Este proyecto es el mismo que el primer proyecto de ejemplo, que se basa en la CLI angular para el desarrollo y la construcción, por lo que la diferencia entre ellos es solo aquellas configuraciones necesarias para la representación del lado del servidor.

Herramienta de instalación

Antes de comenzar, se deben instalar los siguientes paquetes (los proyectos de muestra están configurados, solo**npm install** Sólo):

- **@angular/platform-server** -Componente de servidor universal.

- **@nguniversal/module-map-ngfactory-loader** -Utilizado para manejar la carga diferida en el entorno de representación del lado del servidor.
- **@nguniversal/express-engine** -Aplicación universalMotor expreso。
- **ts-loader** -Utilizado para traducir la aplicación del servidor.
- **express** -Nodo Express server

Use el siguiente comando para instalarlos:

```
npm install --save @angular/platform-server @nguniversal/module-map-ngfactory-loader ts-loader @nguniversal/express-engine express
```

Configuración del proyecto

El trabajo de configuración incluye:

1. Cree un módulo de aplicación de servidor:**src/app/app.server.module.ts**
2. Modifique el módulo de aplicación del cliente:**src/app/app.module.ts**
3. Cree el archivo del gestor de arranque de la aplicación del servidor:**src/main.server.ts**
4. Modifique el archivo del gestor de arranque de la aplicación cliente:**src/main.ts**
5. Cree la configuración del servidor TypeScript:**src/tsconfig.server.json**
6. Modifique el archivo de configuración de @ angular / cli:**.angular-cli.json**
7. Cree un programa de servicio para Node Express:**server.ts**
8. Cree un programa de representación previa del lado del servidor:**prerender.ts**
9. Crear la configuración del servidor Webpack:**webpack.server.config.js**

1. Cree un módulo de aplicación de servidor: src/app/app.server.module.ts

```
1.  import { NgModule } from '@angular/core';
2.  import { ServerModule, ServerTransferStateModule } from '@angular/platform-server';
3.  import { ModuleMapLoaderModule } from '@nguniversal/module-map-ngfactory-loader';
4.
5.  import { AppBrowserModule } from './app.module';
6.  import { AppComponent } from './app.component';
7.
8.   // Puede registrar proveedores de servicios que sean únicos cuando ejecute aplicaciones en el entorno Universal
```

```
9.    @NgModule({
10.       imports: [
11.                   AppBrowserModule,   //   AppModule   de   la
aplicación cliente
12.                   ServerModule, // Módulo angular en el lado
del servidor
13.                   ModuleMapLoaderModule, // utilizado para la
carga diferida del enrutamiento del lado del servidor
14.                   ServerTransferStateModule, // Importado en
el lado del servidor, utilizado para transferir el estado del
servidor al cliente
15.       ],
16.       bootstrap: [AppComponent],
17.   })
18.   export class AppServerModule {
19.   }
```

El módulo de aplicación del lado del servidor (generalmente llamado AppServerModule) es un módulo angular que envuelve el módulo raíz AppModule de la aplicación para que Universal pueda coordinar entre su aplicación y el servidor. AppServerModule también le dirá a Angular cómo iniciar su aplicación cuando se ejecuta en modo Universal.

2. Modifique el módulo de aplicación del cliente: src/app/app.module.ts

```
1.    import   {   BrowserModule,   BrowserTransferStateModule   }   from
'@angular/platform-browser';
2.    import { HttpClientModule } from '@angular/common/http';
3.    import { APP_ID, Inject, NgModule, PLATFORM_ID } from '@angular/core';
4.    import { AppComponent } from './app.component';
5.    import { HomeComponent } from './home/home.component';
6.    import { TransferHttpCacheModule } from '@nguniversal/common';
7.    import { isPlatformBrowser } from '@angular/common';
8.    import { AppRoutingModule } from './app.routes';
9.
10.   @NgModule({
11.     imports: [
12.       AppRoutingModule,
```

```
13.           BrowserModule.withServerTransition({appId: 'my-app'}),
14.               TransferHttpCacheModule, // utilizado para realizar la solicitud de caché de transmisión del servidor al cliente, para evitar que el cliente solicite repetidamente la solicitud completa del servidor
15.               BrowserTransferStateModule, // Importado en el cliente, utilizado para transferir el estado del servidor al cliente
16.           HttpClientModule
17.       ],
18.       declarations: [
19.           AppComponent,
20.           HomeComponent
21.       ],
22.       providers: [],
23.       bootstrap: [AppComponent]
24.   })
25.   export class AppBrowserModule {
26.       constructor(@Inject(PLATFORM_ID) private platformId: Object,
27.               @Inject(APP_ID) private appId: string) {
28.
29.               // Determinar si el entorno operativo es el cliente o el servidor
30.           const platform = isPlatformBrowser(platformId) ? 'in the browser' : 'on the server';
31.           console.log(`Running ${platform} with appId=${appId}`);
32.       }
33.   }
```

Será **NgModule** la importación de BrowserModule en los metadatos se cambia a BrowserModule.withServerTransition ({appId: 'my-app'}), Angular agregará el valor de appId (puede ser cualquier cadena) al nombre de estilo de la página de representación del servidor, de modo que Se puede encontrar y eliminar cuando se inicia la aplicación cliente.

En este momento, podemos usar la inyección de dependencia (**@Inject(PLATFORM_ID)** y **@Inject(APP_ID)**) Obtenga información de tiempo de ejecución sobre la plataforma y la aplicación actual:

```
    constructor(@Inject(PLATFORM_ID)    private    platformId: Object,
                @Inject(APP_ID) private appId: string) {
```

```
            // Determinar si el entorno operativo es el cliente
o el servidor
        const platform = isPlatformBrowser(platformId) ? 'in the
browser' : 'on the server';
        console.log(`Running ${platform} with appId=${appId}`);
    }
```

3. Cree el archivo de arranque de la aplicación del servidor:src/main.server.ts

El archivo se exporta al módulo del servidor:

```
export { AppServerModule } from './app/app.server.module';
```

4. Modifique el archivo del programa de arranque de la aplicación cliente:src/main.ts

Escuche el evento DOMContentLoaded y ejecute nuestro código cuando ocurra el evento DOMContentLoaded para que TransferState funcione correctamente

```
1.  import { enableProdMode } from '@angular/core';
2.  import { platformBrowserDynamic } from '@angular/platform-browser-
dynamic';
3.
4.  import { AppBrowserModule } from './app/app.module';
5.  import { environment } from './environments/environment';
6.
7.  if (environment.production) {
8.    enableProdMode();
9.  }
10.
11.  // Ejecute nuestro código en DOMContentLoaded para hacer que TransferState
funcione
12.  document.addEventListener('DOMContentLoaded', () => {
13.    platformBrowserDynamic().bootstrapModule(AppBrowserModule);
14.  });
```

5. Cree la configuración del servidor TypeScript:src/tsconfig.server.json

```
1.  {
```

```
2.      "extends": "../tsconfig.json",
3.      "compilerOptions": {
4.        "outDir": "../out-tsc/app",
5.        "baseUrl": "./",
6.        "module": "commonjs",
7.        "types": [
8.          "node"
9.        ]
10.     },
11.     "exclude": [
12.       "test.ts",
13.       "**/*.spec.ts"
14.     ],
15.     "angularCompilerOptions": {
16.       "entryModule": "app/app.server.module#AppServerModule"
17.     }
18.   }
```

versus **tsconfig.app.json** La diferencia es:

- El atributo del módulo debe ser common.js para que pueda importarse a la aplicación del servidor mediante el método require ().
- La sección angularCompilerOptions tiene algunas opciones para el compilador AOT:
  - entryModule: el módulo raíz de la aplicación del servidor, el formato es path / to / file # ClassName.

6. Modifique el archivo de configuración de @ angular / cli:.angular-cli.json

En **apps** agregar lo siguiente:

```
1.    {
2.      "platform": "server",
3.      "root": "src",
4.      "outDir": "dist/server",
5.      "assets": [
6.        "assets",
7.        "favicon.ico"
```

```
      ],
      "index": "index.html",
      "main": "main.server.ts",
      "test": "test.ts",
      "tsconfig": "tsconfig.server.json",
      "testTsconfig": "tsconfig.spec.json",
      "prefix": "",
      "styles": [
        "styles.scss"
      ],
      "scripts": [],
      "environmentSource": "environments/environment.ts",
      "environments": {
        "dev": "environments/environment.ts",
        "prod": "environments/environment.prod.ts"
      }
    }
```

7. Cree un programa de servicio Node Express:server.ts

```
import 'zone.js/dist/zone-node';
import 'reflect-metadata';
import { enableProdMode } from '@angular/core';

import * as express from 'express';
import { join } from 'path';
import { readFileSync } from 'fs';

// Faster server renders w/ Prod mode (dev mode never needed)
enableProdMode();

// Express server
const app = express();

const PORT = process.env.PORT || 4000;
const DIST_FOLDER = join(process.cwd(), 'dist');
```

```
17.
18.     // Our index.html we'll use as our template
19.     const template = readFileSync(join(DIST_FOLDER, 'browser',
'index.html')).toString();
20.
21.     // * NOTE :: leave this as require() since this file is built Dynamically from
webpack
22.     const {AppServerModuleNgFactory, LAZY_MODULE_MAP} =
require('./dist/server/main.bundle');
23.
24.     // Express Engine
25.     import { ngExpressEngine } from '@nguniversal/express-engine';
26.     // Import module map for lazy loading
27.     import { provideModuleMap } from '@nguniversal/module-map-ngfactory-
loader';
28.
29.     // Our Universal express-engine (found @
https://github.com/angular/universal/tree/master/modules/express-engine)
30.     app.engine('html', ngExpressEngine({
31.       bootstrap: AppServerModuleNgFactory,
32.       providers: [
33.         provideModuleMap(LAZY_MODULE_MAP)
34.       ]
35.     }));
36.
37.     app.set('view engine', 'html');
38.     app.set('views', join(DIST_FOLDER, 'browser'));
39.
40.     /* - Example Express Rest API endpoints -
41.       app.get('/api/**', (req, res) => { });
42.     */
43.
44.     // Server static files from /browser
45.     app.get('*.*', express.static(join(DIST_FOLDER, 'browser'), {
46.       maxAge: '1y'
```

```
47.     }));
48.
49.     // ALl regular routes use the Universal engine
50.     app.get('*', (req, res) => {
51.       res.render('index', {req});
52.     });
53.
54.     // Start up the Node server
55.     app.listen(PORT, () => {
56.       console.log(`Node Express server listening on http://localhost:${PORT}`);
57.     });
```

Motor de plantilla universal

La parte más importante de este archivo es la función ngExpressEngine:

```
app.engine('html', ngExpressEngine({
  bootstrap: AppServerModuleNgFactory,
  providers: [
    provideModuleMap(LAZY_MODULE_MAP)
  ]
}));
```

ngExpressEngine es una encapsulación de la función renderModuleFactory de Universal. Convierte las solicitudes de los clientes en páginas HTML representadas por el servidor. **Si utiliza una tecnología de servidor diferente de Node, debe llamar a esta función en el motor de plantillas del servidor.**

- El primer parámetro es el AppServerModule que escribió antes. Es el puente entre el renderizador del servidor Universal y su aplicación.
- El segundo parámetro es extraProviders. Es un proveedor opcional de inyección de dependencia angular que solo se necesita cuando se ejecuta en este servidor. Cuando su aplicación necesita información que solo se necesita cuando se ejecuta en una instancia de servidor, proporcione el parámetro extraProviders.

La función ngExpressEngine devuelve una promesa que se analizará en una página representada.

Luego, su motor tiene que decidir qué hacer con esta página. Ahora en la función de devolución de llamada de este motor, la página representada se devuelve al servidor web y luego el servidor la reenvía al cliente a través de una respuesta HTTP.

1. Cree un programa de representación previa del servidor:prerender.ts

```
1.  // Load zone.js for the server.
2.  import 'zone.js/dist/zone-node';
3.  import 'reflect-metadata';
4.  import { readFileSync, writeFileSync, existsSync, mkdirSync } from 'fs';
5.  import { join } from 'path';
6.
7.  import { enableProdMode } from '@angular/core';
8.  // Faster server renders w/ Prod mode (dev mode never needed)
9.  enableProdMode();
10.
11. // Import module map for lazy loading
12. import { provideModuleMap } from '@nguniversal/module-map-ngfactory-
loader';
13. import { renderModuleFactory } from '@angular/platform-server';
14. import { ROUTES } from './static.paths';
15.
16. // * NOTE :: leave this as require() since this file is built Dynamically from
webpack
17. const {AppServerModuleNgFactory, LAZY_MODULE_MAP} =
require('./dist/server/main.bundle');
18.
19. const BROWSER_FOLDER = join(process.cwd(), 'browser');
20.
21. // Load the index.html file containing referances to your application bundle.
22. const index = readFileSync(join('browser', 'index.html'), 'utf8');
23.
24. let previousRender = Promise.resolve();
25.
26. // Iterate each route path
27. ROUTES.forEach(route => {
28.     const fullPath = join(BROWSER_FOLDER, route);
29.
30.     // Make sure the directory structure is there
31.     if (!existsSync(fullPath)) {
32.         mkdirSync(fullPath);
```

```
33.         }
34.
35.         // Writes rendered HTML to index.html, replacing the file if it already exists.
36.         previousRender = previousRender.then(_ =>
renderModuleFactory(AppServerModuleNgFactory, {
37.           document: index,
38.           url: route,
39.           extraProviders: [
40.             provideModuleMap(LAZY_MODULE_MAP)
41.           ]
42.         })).then(html => writeFileSync(join(fullPath, 'index.html'), html));
43.       });
```

9. Cree la configuración del servidor Webpack:webpack.server.config.js

Las aplicaciones universales no requieren ninguna configuración adicional de Webpack, Angular CLI nos ayudará a lidiar con ellas. Sin embargo, dado que el programa de servicio de Node Express en este ejemplo es una aplicación TypeScript (server.ts y prerender.ts), es necesario usar Webpack para traducirlo.

```
// Work around for https://github.com/angular/angular-cli/issues/7200

    const path = require('path');
    const webpack = require('webpack');

    module.exports = {
      entry: {
        server: './server.ts', // This is our Express server for Dynamic universal
        prerender: './prerender.ts' // This is an example of Static prerendering
(generative)
      },
      target: 'node',
      resolve: {extensions: ['.ts', '.js']},
      externals: [/(node_modules|main\..*\.js)/,], // Make sure we include all
node_modules etc
      output: {
```

```
        path: path.join(__dirname, 'dist'), // Puts the output at the root of the dist folder
        filename: '[name].js'
      },
      module: {
        rules: [
          {test: /\.ts$/, loader: 'ts-loader'}
        ]
      },
      plugins: [
        new webpack.ContextReplacementPlugin(
          /(.+)?angular(\\|\/)core(.+)?/, // fixes WARNING Critical dependency: the request of a dependency is an expression
          path.join(__dirname, 'src'), // location of your src
          {} // a map of your routes
        ),
        new webpack.ContextReplacementPlugin(
          /(.+)?express(\\|\/)(.+)?/, // fixes WARNING Critical dependency: the request of a dependency is an expression
          path.join(__dirname, 'src'),
          {}
        )
      ]
    };
```

Configuración de prueba

Con la configuración anterior, hemos creado una aplicación Angular Universal que se puede representar en el servidor.

Configure los comandos relacionados con la compilación y el servicio en el área de scripts de package.json:

```
{
  "scripts": {
    "ng": "ng",
    "start": "ng serve -o",
```

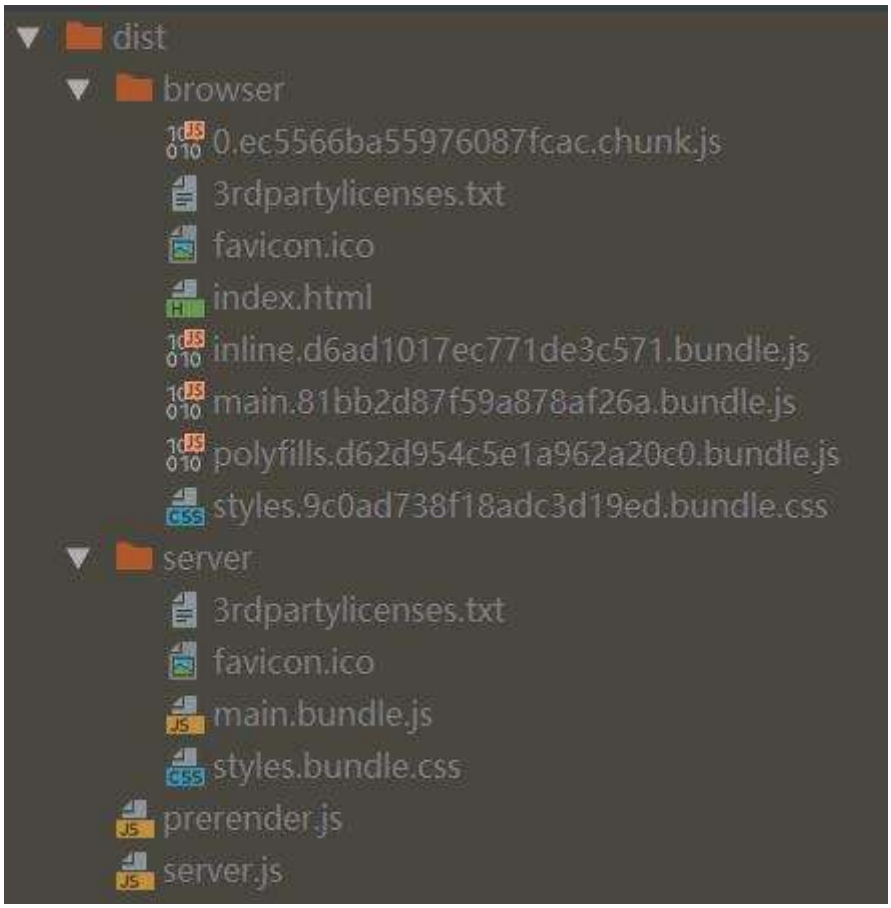
```
    "ssr": "npm run build:ssr && npm run serve:ssr",
    "prerender": "npm run build:prerender && npm run serve:prerender",
    "build": "ng build",
    "build:client-and-server-bundles": "ng build --prod && ng build --prod --app 1 --output-hashing=false",
    "build:prerender": "npm run build:client-and-server-bundles && npm run webpack:server && npm run generate:prerender",
    "build:ssr": "npm run build:client-and-server-bundles && npm run webpack:server",
    "generate:prerender": "cd dist && node prerender",
    "webpack:server": "webpack --config webpack.server.config.js --progress --colors",
    "serve:prerender": "cd dist/browser && http-server",
    "serve:ssr": "node dist/server"
  }
```

El desarrollo solo necesita ejecutarse

```
npm run start
```

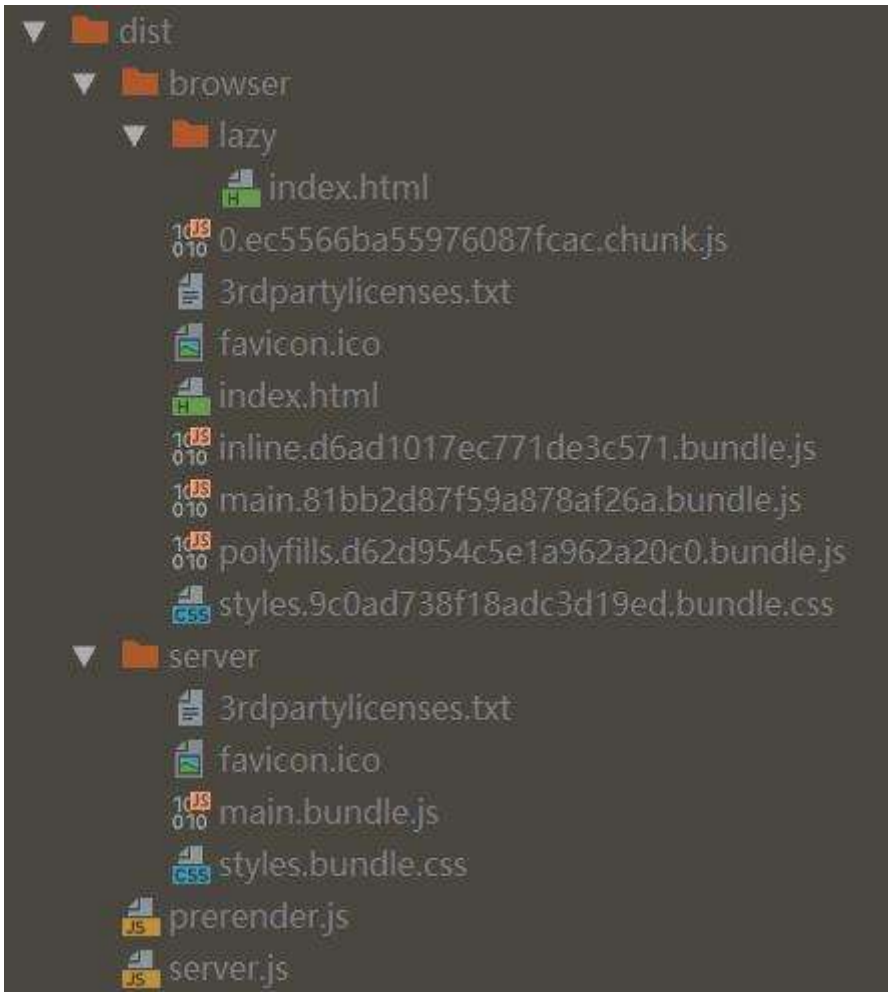
llevado a cabo npm run server. Compile la aplicación e inicie un Node Express para servir la aplicación en http://localhost:4000

directorio dist:



Ejecute npm run prerender-compile la aplicación y pre-renderice el archivo de la aplicación, inicie un servidor http de demostración para que pueda verlo en http://localhost:8080

Nota: Para implementar un sitio web estático en una plataforma de alojamiento estático, debe implementar la carpeta dist / browser, no la carpeta dist



Según la información de enrutamiento real del proyecto y en el directorio raíz**static.paths.ts** Configuración, proporcionada para el análisis prerender.ts.

```
export const ROUTES = [
  '/',
  '/lazy'
];
```

Por lo tanto, como puede ver en el directorio dist, la representación previa del lado del servidor generará el index.html estático correspondiente en el navegador de acuerdo con la ruta configurada. **/index.html**, **/lazy** **/lazy/index.html**。

Módulo de servidor de carga diferida

En la introducción anterior, nosotros **app.server.module.ts** Importado en ModuleMapLoaderModule en **app.module.ts**。

ModuleMapLoaderModule Los módulos pueden hacer que los módulos con carga lenta también se procesen en el lado del servidor, y todo lo que tiene que hacer es**app.server.module.ts**Importar.

Transferencia de estado de servidor a cliente

En la introducción anterior, se importaron sapp.server.module.ts, en ServerTransferStateModule en app.module.ts y BrowserTransferStateModule con TransferHttpCacheModule。

Estos tres módulos están relacionados con la transmisión de estado del servidor al cliente:

- **ServerTransferStateModule**: Importado en el servidor, utilizado para transferir el estado del servidor al cliente
- **BrowserTransferStateModule**: Importado en el cliente, utilizado para transferir el estado del servidor al cliente
- **TransferHttpCacheModule**: Se utiliza para realizar la caché de transmisión de solicitud del servidor al cliente, para evitar que el cliente solicite repetidamente la solicitud completada

Usa estos módulos para resolver**la solicitud http se solicita una vez en el servidor y el cliente** El problema.

Por ejemplo en **home.component.ts** El siguiente código:

Después de que se ejecute el código,

```
    import { Component, OnDestroy, OnInit } from '@angular/core';
    import { HttpClient } from '@angular/common/http';
    import { Observable } from 'rxjs/Observable';

    @Component({
        selector: 'app-home',
        templateUrl: './home.component.html',
        styleUrls: ['./home.component.scss']
    })
    export class HomeComponent implements OnInit, OnDestroy {
        constructor(public http: HttpClient) {
        }

        ngOnInit() {
                    this.poiSearch (this.keyword, 'Beijing').subscribe ((data: any) => {
                console.log(data);
            });
        }
```

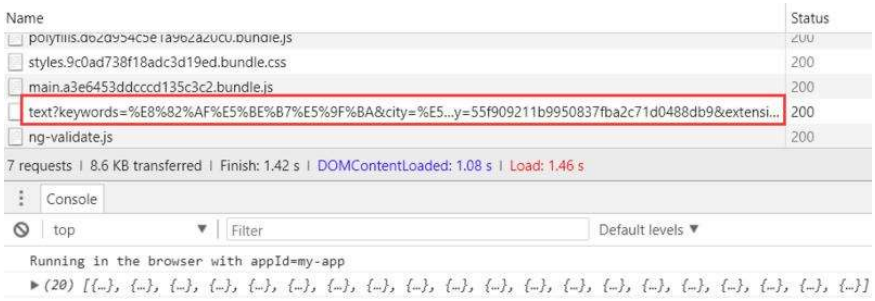
```
        ngOnDestroy() {
        }

        poiSearch(text: string, city?: string): Observable<any>
{
            return
this.http.get(encodeURI(`http://restapi.amap.com/v3/place/text?k
eywords=${text}&city=${city}&offset=20&key=55f909211b9950837fba2
c71d0488db9&extensions=all`));
        }
    }
```

El servidor solicita e imprime:

```
Running on the server with appId=my-app
[ { id: 'B0FFIOY46A',
    name: '肯德基',
    tag: [],
    type: '餐饮服务;快餐厅;肯德基',
    typecode: '050301',
    biz_type: 'diner',
    address: '西单北大街堂子胡同9号新一代商城地下一层',
    location: '116.375098,39.911168',
    tel: [],
    postcode: [],
    website: [],
    email: [],
    pcode: '110000',
```

El cliente solicita e imprime nuevamente:



Método 1: usar TransferHttpCacheModule

Usar TransferHttpCacheModule es simple, no se requieren cambios de código. Después de importar en app.module.ts, Angular almacenará automáticamente en caché la solicitud del servidor al cliente, en otras palabras, los datos de la solicitud del servidor se transmitirán automáticamente al cliente, y el cliente no enviará la solicitud después de recibir los datos. También.

Método 2: usar BrowserTransferStateModule

Este método es un poco más complicado y requiere algunos cambios en el código.

Ajuste el código home.component.ts de la siguiente manera:

```
import { Component, OnDestroy, OnInit } from '@angular/core';
import { makeStateKey, TransferState } from '@angular/platform-browser';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
const KFCLIST_KEY = makeStateKey('kfcList');
@Component({
selector: 'app-home',
templateUrl: './home.component.html',
styleUrls: ['./home.component.scss']
})
export class HomeComponent implements OnInit, OnDestroy {
constructor(public http: HttpClient,
private state: TransferState) {
}
     ngOnInit() {

              // Use una etiqueta para distinguir si el servidor ha recibido los datos. Si no obtiene los datos, se los solicitará al cliente.
         const kfcList : any[] = this.state.get(KFCLIST_KEY, null as any);

         if (!this.kfcList) {
                      this.poiSearch (this.keyword, 'Beijing'). subscribe ((data: any) => {
```

```
                console.log(data);
                            this.state.set (KFCLIST_KEY, data
as any); // almacenar datos
            });
        }
    }

    ngOnDestroy() {
        if (typeof window === 'object') {
                    this.state.set (KFCLIST_KEY, nulo como
cualquiera); // eliminar datos
        }
    }

    poiSearch(text: string, city?: string): Observable<any> {
        return
this.http.get(encodeURI(`http://restapi.amap.com/v3/place/text?k
eywords=${text}&city=${city}&offset=20&key=55f909211b9950837fba2
c71d0488db9&extensions=all`));
    }
}
```

Use const KFCLIST_KEY = makeStateKey ('kfcList') para crear una StateKey que almacene los datos transmitidos, Inject TransferState en el constructor de HomeComponent.
De acuerdo con this.state.get (KFCLIST_KEY, nulo como cualquiera) en ngOnInit, determine si los datos existen (si es un servidor o un cliente) Datos y almacenar y transferir datos a través de this.state.set (KFCLIST_KEY, datos como cualquiera)
En ngOnDestroy, decida si desea eliminar los datos almacenados de acuerdo con el cliente actual o no

Comparación de representación de cliente y servidor

Finalmente, comparamos las tres razones:

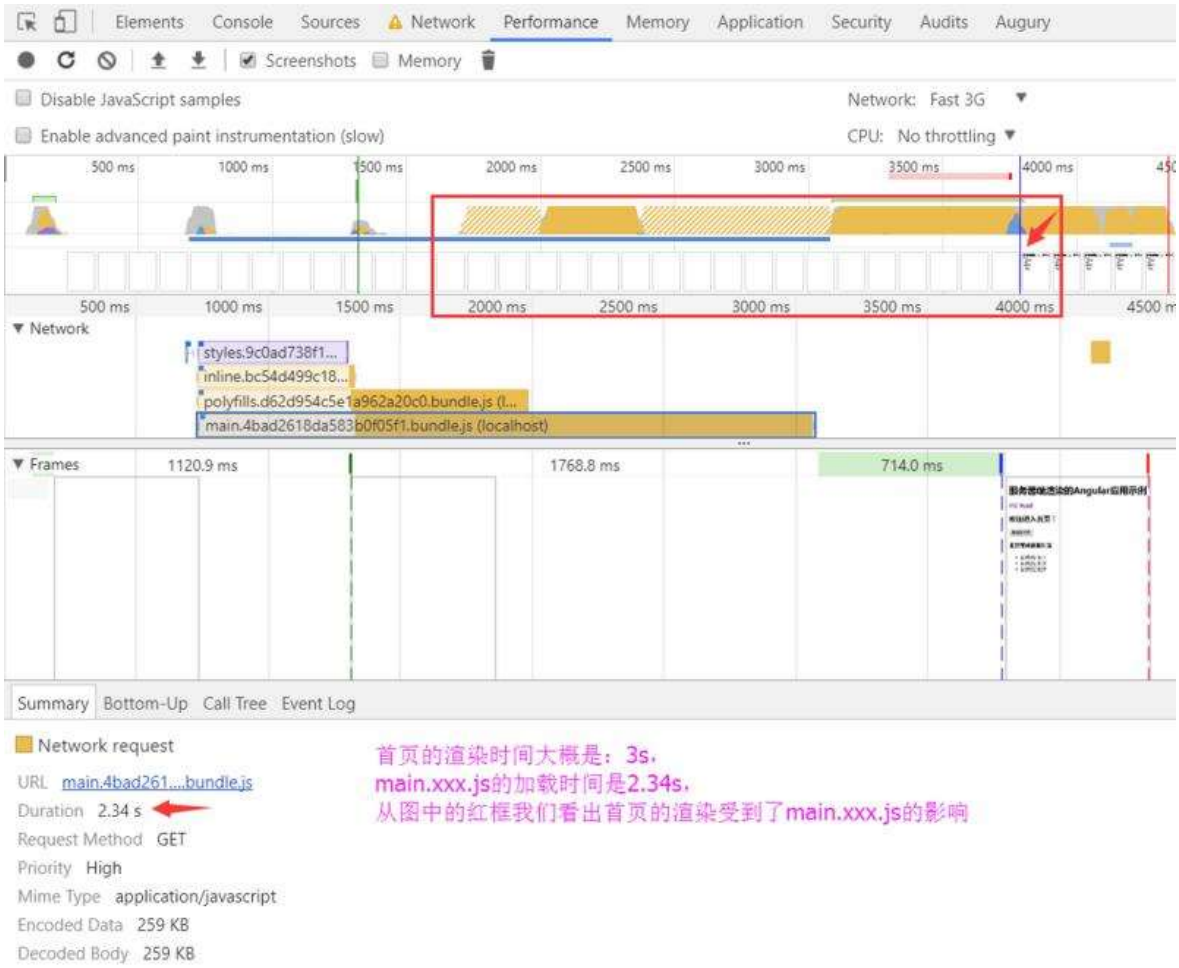
Ayuda Web Crawler (SEO)
Mejore el rendimiento en teléfonos móviles y dispositivos de baja potencia muestra rápidamente la página de inicio
Ayuda del rastreador web (SEO)
Representación del cliente:

Representación del servidor:

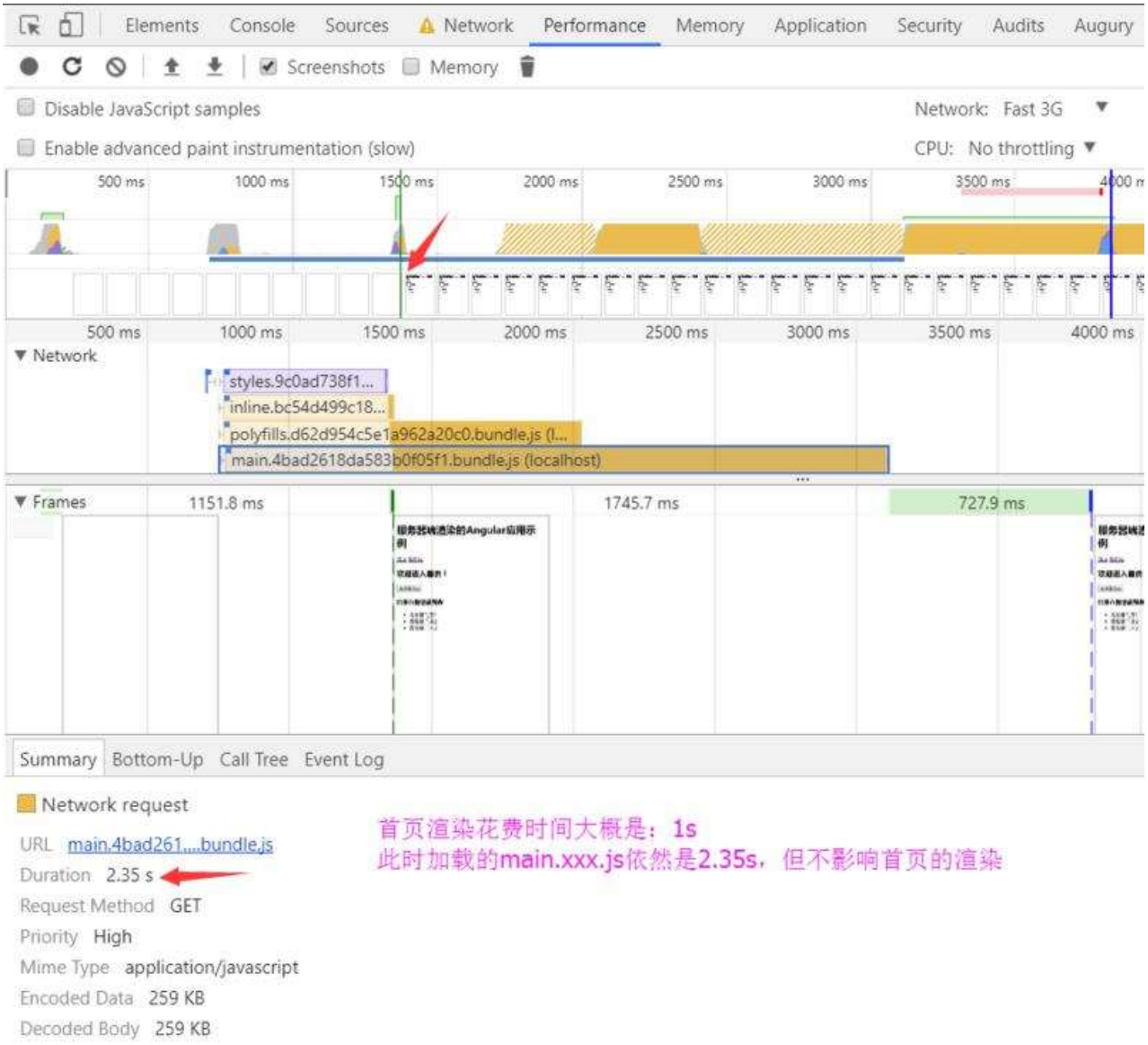
Como puede ver en el parrafo anterior, el servidor presenta la información a la página devuelta por adelantado, de modo que el rastreador web pueda obtener directamente la información (el rastreador web básicamente no analiza JavaScript).

Mejore el rendimiento en teléfonos móviles y dispositivos de baja potencia. Esta razón se puede ver en lo anterior. Para algunos dispositivos de gama baja, mostrar la página directamente siempre es mucho mejor que analizar JavaScript.

Representación del cliente:



Representación del servidor:



Tenga en cuenta algunas cosas

- Para los paquetes de software del servidor, es posible que deba incluir módulos de terceros para **nodeExternals**
- **window**, **document**, **navigator** Y otros tipos de navegador no *existen en el servidor* -Si lo usa directamente, no funcionará correctamente en el lado del servidor. Los siguientes métodos pueden hacer que su código funcione correctamente:
  - Debe de usar **PLATFORM_ID p**ara verificar si la plataforma actual es un navegador o un servidor, y luego usar el tipo específico del navegador

```
import { PLATFORM_ID } from '@angular/core';
import { isPlatformBrowser, isPlatformServer } from '@angular/common';
```

```
    constructor(@Inject(PLATFORM_ID) private platformId:
Object) { ... }


    ngOnInit() {
      if (isPlatformBrowser(this.platformId)) {
               // Solo ejecuta el código en el navegador
        ...
      }
      if (isPlatformServer(this.platformId)) {
             // Código que se ejecuta solo en el servidor
        ...
      }
    }
```

- -Trata de ** limitar ** o ** evitar ** usando setTimeout. Se ralentizará el proceso de representación del lado del servidor. Asegúrese de eliminarlos en el ngOnDestroy del componente
- -Para los tiempos de espera de RxJ, asegúrese de _cancelar_ sus secuencias en caso de éxito, ya que también reducirán la velocidad de representación.

No manipule nativeElement directamente,utilizar Renderer2, Para que la vista de la aplicación se pueda cambiar en todas las plataformas.

```
  constructor(element: ElementRef, renderer: Renderer2) {
  this.renderer.setStyle(element.nativeElement, 'font-size', 'x-
large'
```

- Resuelva el problema de que la aplicación ejecuta la solicitud XHR en el servidor y se ejecuta nuevamente en el cliente
  - Use el caché del servidor al cliente (TransferState)
- Comprender claramente los atributos relacionados con DOM y las diferencias entre los atributos
- Intenta hacer que las instrucciones sean apátridas. Para obtener instrucciones con estado, es posible que deba proporcionar un atributo para reflejar el valor de cadena inicial del atributo correspondiente, como la url en la etiqueta img. Para nuestro elemento nativo, el atributo src se refleja como el atributo src del tipo de elemento HTMLImageElement