## 官网

<https://github.com/dbfannin/ngx-logger>

## 安装

yarn add ngx-logger

## 文章一

<https://www.codemag.com/Article/1711021/Logging-in-Angular-Applications>

程序员经常使用console.log在其Angular应用程序中记录错误或其他参考消息。尽管这在调试应用程序时很好，但是对于生产应用程序来说，这不是最佳实践。由于Angular与服务有关，因此最好创建一个日志服务，您可以从其他服务和组件中调用它。在此日志记录服务中，您仍然可以调用console.log，但是您以后也可以修改该服务以记录消息，以通过Web API将消息存储在本地存储或数据库表中。

在本文中，您将通过一系列步骤来构建日志记录服务。首先，创建一个简单的日志服务类以使用来记录消息console.log()。接下来，添加一些日志记录级别，以便可以报告调试，警告，错误和其他类型的日志消息。然后，您创建一个通用的日志记录服务类，以调用其他类以登录到控制台，本地存储和Web API。最后，您创建一个日志发布服务，该服务读取JSON文件以选择要使用的日志服务类。

简单的日志服务

首先，创建一个非常简单的日志记录服务，该服务仅记录到控制台。这里的重点是用对服务的调用替换Angular应用程序中的所有console.log语句。调出现有的Angular应用程序或创建一个新的应用程序。如果还没有，请在该文件夹shared下添加一个名为的\src\app文件夹。创建一个名为的新TypeScript文件log.service.ts。添加以下代码段中显示的代码。

import { Injectable } from '@angular/core';

@Injectable()

export class LogService {

log(msg: any) {

console.log(new Date() + ": " + JSON.stringify(msg));

}

}

这段代码创建了一个可注入服务，可以由Angular创建并注入到您的任何Angular类中。该log()方法接受可以是任何类型的消息。将创建一个新日期，以便可以将每条消息及其附带的日期和时间记录到控制台中。仅登录到控制台时，日期/时间并不那么重要，但是一旦您开始登录到本地存储或数据库，就希望附加日期/时间，以便知道何时创建日志消息。注意参数JSON.stringify周围的使用msg。这使您可以传递对象，并且可以将其记录为字符串。

为了跟随本文的目的，创建一个名为的新文件夹\log-test并添加一个log-test.component.html页面。添加一个按钮以测试日志记录服务。

<button (click)="testLog()">Log Test</button>

创建一个log-test.component.tsTypeScript文件，并添加清单1中所示的代码以响应按钮单击事件。

清单1：用于测试LogService类的LogTestComponent

import { Component } from "@angular/core";

import { LogService } from '../shared/log.service';

@Component({

selector: "log-test",

templateUrl: "./log-test.component.html"

})

export class LogTestComponent {

constructor(private logger: LogService) {

}

testLog(): void {

this.logger.log("Test the `log()` Method");

}

}

在构造函数中添加一个logger变量，以便Angular可以将此服务注入到此组件中。请注意，testLog()您现在在方法中调用this.logger.log()而不是console.log()。结果是相同的（请参见图1），因为消息出现在控制台窗口中。但是，您现在可以灵活地将此消息记录到本地存储，数据库表，控制台或所有这三个消息中。而且，最好的部分是，除了LogService类中的代码外，您无需更改应用程序中的任何代码。要在Angular应用程序中使用此服务，需要将其导入app.module.ts文件中。还要导入LogTestComponent您创建的。

不同类型的日志记录

有时，您可能希望在运行应用程序时仅打开某些类型的日志记录。许多其他语言的日志记录系统允许您记录调试消息，参考消息，警告消息等。通过添加枚举和可以设置为控制显示消息的属性，可以向LogService类添加此功能。首先，在log.service.ts文件中添加一个LogLevel枚举，以跟踪执行哪种日志记录。log.service.ts在文件中的import语句之后添加此枚举。不要将其添加到LogService类中。

export enum LogLevel {

All = 0,

Debug = 1,

Info = 2,

Warn = 3,

Error = 4,

Fatal = 5,

Off = 6

}

将一个属性添加到名为levelLogLevel类型的名为LogService的类中。默认值为“所有”枚举。在添加属性时，请添加一个Boolean名为的属性，logWithDate以指定是否要将日期/时间添加到邮件的开头。

level: LogLevel = LogLevel.All;

logWithDate: boolean = true;

无需level在调用记录器之前设置属性。log()方法，将新方法debug，info，warn，error和致命添加到LogService类（清单2）。这些方法中的每一个都调用ToLog()传递消息的write方法，适当的枚举值和可选的参数数组。删除log()您先前编写的方法，并将其替换为清单2中的所有方法。

清单2：向您的LogService类添加方法以编写不同种类的消息

debug(msg: string, ...optionalParams: any[]) {

this.writeToLog(msg, LogLevel.Debug, optionalParams);

}

info(msg: string, ...optionalParams: any[]) {

this.writeToLog(msg, LogLevel.Info, optionalParams);

}

warn(msg: string, ...optionalParams: any[]) {

this.writeToLog(msg, LogLevel.Warn, optionalParams);

}

error(msg: string, ...optionalParams: any[]) {

this.writeToLog(msg, LogLevel.Error, optionalParams);

}

fatal(msg: string, ...optionalParams: any[]) {

this.writeToLog(msg, LogLevel.Fatal, optionalParams);

}

log(msg: string, ...optionalParams: any[]) {

this.writeToLog(msg, LogLevel.All, optionalParams);

}

可选参数数组意味着您可以传递要记录的任何参数。例如，以下任何调用均有效。

this.logger.log("Test 2 Parameters", "Paul", "Smith");

this.logger.debug("Test Mixed Parameters", true, false, "Paul", "Smith");

let values = ["1", "Paul", "Smith"];

this.logger.warn("Test String and Array", "Some log entry", values);

的writeLog()方法，清单3，检查由针对在值集的方法之一传递的水平level特性。level在shouldLog()方法中检查此属性。这两个方法现在都应该添加到您的LogService类中。

清单3：writeToLog（）方法创建消息以写入日志

private writeToLog(msg: string, level: LogLevel, params: any[]) {

if (this.**shouldLog**(level)) {

let value: string = "";

// Build log string

if (this.logWithDate) {

value = new Date() + " - ";

}

value += "Type: " + LogLevel[this.level];

value += " - Message: " + msg;

if (params.length) {

value += " - Extra Info: " + this.formatParams(params);

}

// Log the value

console.log(value);

}

}

该*shouldLog*()方法根据levelLogService类中设置的属性确定是否应该进行日志记录。该服务由Angular创建为单例，因此一旦level设置了此属性，它将保留该值，直到您在应用程序中对其进行更改为止。该shouldLog()检查的参数传递对level在属性集LogService类。如果传入的级别大于或等于该level属性，并且未关闭日志记录，则true此方法返回一个值。一个true返回值告诉writeToLog()记录的消息的方法。

private shouldLog(level: LogLevel): boolean {

let ret: boolean = false;

if ((level >= this.level && level !== LogLevel.Off) || this.level === LogLevel.All) {

ret = true;

}

return ret;

}

有一个在一个多方法调用writeToLog()方法调用的formatParams()。此方法用于创建参数数组的逗号分隔列表。如果数组中的所有参数都是简单数据类型而不是对象，则在使用ret该join()方法从数组创建逗号分隔列表之后，将返回名为local的变量。如果有一个对象，则循环遍历params数组中的每个项目，并ret使用该JSON.stringify()方法构建变量以将每个参数转换为字符串，然后在每个变量后面添加逗号。

private formatParams(params: any[]): string {

let ret: string = params.join(",");

// Is there at least one object in the array?

if (params.some(p => typeof p == "object")) {

ret = "";

// Build comma-delimited string

for (let item of params) {

ret += JSON.stringify(item) + ",";

}

}

return ret;

}

创建日志条目类

无需构建日志信息的字符串并格式化writeToLog()方法中的参数，而是创建一个名为的类LogEntry来为您完成所有这些工作。将此新类放置在log.service.ts文件中。清单4中所示的LogEntry类具有以下属性：日志条目的日期，要记录的消息，日志级别，要记录的额外信息数组，以及您设置为指定在日志中包括日期的布尔值信息。

清单4：创建一个LogEntry类，使创建日志消息更加容易

export class LogEntry {

// Public Properties

entryDate: Date = new Date();

message: string = "";

level: LogLevel = LogLevel.Debug;

extraInfo: any[] = [];

logWithDate: boolean = true;

buildLogString(): string {

let ret: string = "";

if (this.logWithDate) {

ret = new Date() + " - ";

}

ret += "Type: " + LogLevel[this.level];

ret += " - Message: " + this.message;

if (this.extraInfo.length) {

ret += " - Extra Info: " + this.formatParams(this.extraInfo);

}

return ret;

}

private formatParams(params: any[]): string {

let ret: string = params.join(",");

// Is there at least one object in the array?

if (params.some(p => typeof p == "object")) {

ret = "";

// Build comma-delimited string

for (let item of params) {

ret += JSON.stringify(item) + ",";

}

}

return ret;

}

}

该buildLogString()方法类似于您writeToLog()先前在方法中编写的内容。此方法从此类的属性中收集值，并以一个长字符串返回它们，该字符串可用于输出到控制台窗口。建立类后，formatParams()从LogService类中删除方法LogEntry。您将LogEntry在本文其余部分中使用的每个不同的日志记录类中使用此类。

现在您已经建立了这个LogEntry类，并且已经formatParams()从LogService该类中删除了该类，然后重写该writeToLog()方法，使其看起来像下面的代码。

private writeToLog(msg: string, level: LogLevel, params: any[]) {

if (this.shouldLog(level)) {

let entry: LogEntry = new LogEntry();

entry.message = msg;

entry.level = level;

entry.extraInfo = params;

entry.logWithDate = this.logWithDate;

console.log(entry.buildLogString());

}

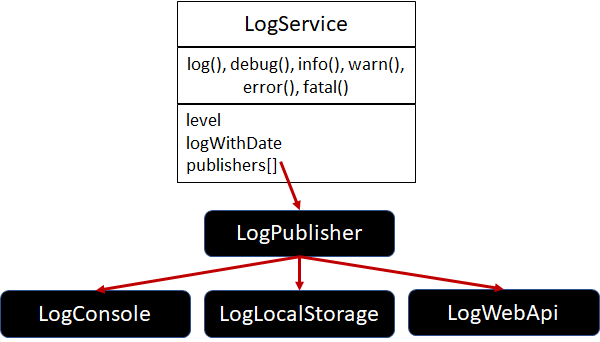
}

修改writeToLog()方法后，重新运行应用程序，您仍然应该在控制台窗口中看到日志消息。

日志发布系统

在记录异常或任何类型的消息时，最好将这些日志条目写入不同的位置，以防其中之一无法访问。这样，您就更有可能不会丢失任何消息。为此，您需要创建三种不同的日志记录类。第一个发布者是一个LogConsole登录到控制台窗口的类。第二个发布者是LogLocalStorage将消息记录到Web本地存储。第三发布者LogWebApi用于调用Web API以将消息记录到数据库的后端表中。

您将创建一个publishers属性（图2），而不是在LogService类中对每个这些类进行硬编码，该属性是一个名为的抽象类的数组LogPublisher。您创建的每个日志记录类都将扩展此抽象类。



本LogPublisher类包含一个属性命名位置。此属性用于设置本地存储的密钥和Web API的URL。此类还需要两种方法：log()和clear()。该log()方法在扩展的每个类中均被覆盖，LogPublisher并负责执行日志记录。该clear()方法从数据存储中删除所有日志条目。

将一个名为log-publishers.ts的新TypeScript文件添加到\shared项目中的文件夹中。您将需要在该文件顶部添加一些import语句。事实上，你会更晚加，但现在，只需添加Observable的Observable of，和LogEntry类。编写下一个代码片段中显示的代码，以创建您的抽象LogPublisher类。

import { Observable } from 'rxjs/Observable';

import 'rxjs/add/observable/of';

import { LogEntry } from './log.service';

export abstract class LogPublisher {

location: string;

abstract log(record: LogEntry):

Observable<boolean>

abstract clear(): Observable<boolean>;

}

现在您已经有了用于创建每个日志发布类的模板，让我们开始构建它们。

登录控制台

您创建的用于扩展LogPublisher类的第一个类将消息写入控制台。您最终将要从先前创建console.log()的LogService类中删除对的调用。LogConsole这是一个非常简单的类，它使用来将日志数据显示到控制台窗口console.log()。LogPublisher在log-publisher.ts文件中的类下方添加以下代码。

export class LogConsole extends LogPublisher {

(entry: LogEntry): Observable<boolean> {

// Log to console

console.log(entry.buildLogString());

return Observable.of(true);

}

clear(): Observable<boolean> {

console.clear();

return Observable.of(true);

}

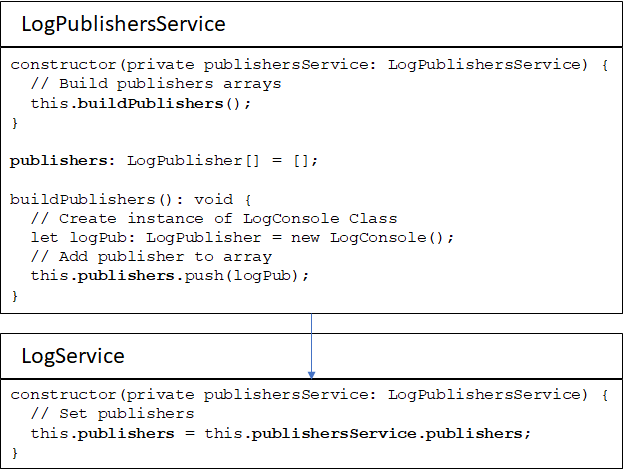
}

请注意，此类中的log()方法接受该类的实例LogEntry。名为的此参数传入entry，调用该buildLogString()方法以创建要显示在控制台窗口中的完整日志条目数据的字符串。每个log()方法都需要将布尔类型的observable返回给调用方。因为登录到控制台不会出错，所以只需对True返回值进行硬编码。

所述clear()`` method must also be overridden in any class that extends the LogPublisherclass. For the console window, call the清晰（）`方法来清除发布到控制台窗口的所有消息。

日志发布者服务

如您在图2中看到的，publishers在LogService类中有一个array属性。您需要使用LogPublisher类的实例填充此数组。到目前为止，您唯一已构建的类是LogConsole，但很快您还将构建LogLocalStorage并进行LogWebApi类化。无需构建LogService该类中的发布者列表，而是创建另一个服务类来构建日志发布者列表。这个名为的服务类LogPublishersService负责构建日志发布类的数组。将该服务传递到LogService类中，因此publishers可以从中分配数组LogPublishersService（图3）。首先，您将仅对每个日志类进行硬编码，但是在本文的后面，您将从JSON文件中读取列表发布者。



需要将LogPublishersService类（清单5）定义为可注入服务，以便Angular可以将其注入到LogService类中。在此类的构造函数中，您调用名为的方法buildPublishers()。此方法创建a的每个实例，LogPublisher并将每个实例添加到publishers数组中。现在，只需添加代码即可创建LogConsole该类的新实例，并将其推入publishers数组。

清单5：LogPublishersService负责创建发布对象列表

import { Injectable } from '@angular/core';

import { LogPublisher, LogConsole } from "./log-publishers";

@Injectable()

export class LogPublishersService {

constructor() {

// Build publishers arrays

this.buildPublishers();

}

// Public properties

publishers: LogPublisher[] = [];

// Build publishers array

buildPublishers(): void {

// Create instance of LogConsole Class

this.publishers.push(new LogConsole());

}

}

更新AppModule类

像任何Angular服务一样，一旦创建它，​​就必须app.module.ts通过导入它并将其添加到@NgModule的providers属性中来在文件中注册它。打开app.module.ts并在文件顶部附近添加导入。

import { LogPublishersService }

from "./shared/log-publishers.service";

接下来，将服务添加到@NgModuledecorator函数中的provider属性。

@NgModule({

imports: [BrowserModule, FormsModule, HttpModule],

declarations: [AppComponent, LogTestComponent],

bootstrap: [AppComponent],

providers: [LogService, LogPublishersService]

})

修改LogService类

现在是时候修改LogService该类以使用LogPublishersService该类了。打开log.service.tsTypeScript文件，并在文件顶部附近添加两个import语句。

import { LogPublisher } from "./log-publishers";

import { LogPublishersService }

from "./shared/log-publishers.service";

添加名为属性publishers的LogPublisher类型数组。

publishers: LogPublisher[];

在LogService类中添加一个构造函数，以便Angular注入LogPublishersService。在此构造函数中，从中获取publishers属性LogPublishersService并将内容分配给类中的publishers属性LogService。

constructor(private publishersService: LogPublishersService) {

// Set publishers

this.publishers = this.publishersService.publishers;

}

找到该writeToLog()方法，然后从该方法中删除以下代码行。

console.log(entry.buildLogString());

在删除上述代码行的位置，添加一个for循环以遍历发布者列表。每次循环时，调用log()记录器的方法，并传入LogEntry对象。因为该log()方法返回一个可观察的值，所以您应该订阅结果并将布尔返回值写入控制台窗口。

for (let logger of this.publishers) {

logger.log(entry).subscribe(response => console.log(response));

}

再次运行该应用程序，然后单击“测试日志”按钮以查看写入控制台窗口的日志条目。您添加了一些类和服务，仅用于发布到控制台窗口。您应该能够看到这种方法的优点。现在LogPublisher，您可以添加新的类，将它们添加到LogPublishersService该类的数组中，然后将其发布到其他位置。

登录到本地存储

下一个要添加的发布者是将一系列LogEntry对象存储到Web浏览器的本地存储中的发布者。打开log-publishers.ts文件，并将清单6中所示的代码添加到该文件。本LogLocalStorage类需要设置的项目到本地存储设置键值。使用该location属性设置要使用的键值。在这种情况下，location在构造函数中设置。当派生类中有构造函数时，始终需要调用该super()方法才能调用基类的构造函数。

清单6：创建一个LogLocalStorageService类以将消息存储到本地存储中

export class `LogLocalStorage` extends LogPublisher {

constructor() {

// Must call `super()`from derived classes

super();

// Set location

this.location = "logging";

}

// Append log entry to local storage

log(entry: LogEntry): Observable<boolean> {

let ret: boolean = false;

let values: LogEntry[];

try {

// Get previous values from local storage

values = JSON.parse(localStorage.getItem(this.location)) || [];

// Add new log entry to array

values.push(entry);

// Store array into local storage

localStorage.setItem(this.location, JSON.stringify(values));

// Set return value

ret = true;

} catch (ex) {

// Display error in console

console.log(ex);

}

return Observable.of(ret);

}

// Clear all log entries from local storage

clear(): Observable<boolean> {

localStorage.removeItem(this.location);

return Observable.of(true);

}

}

本地存储使您可以存储大量数据，因此让我们在每次log()调用该方法时添加每个日志条目。该setItem()方法用于将值设置到本地存储中。如果调用setItem()并传递一个值，则键位置中的任何旧值都将被新值替换。首先使用该getItem()方法从本地存储中读取先前的值。将其解析为LogEntry对象数组，或者如果该键位置中没有存储值，则返回一个空数组。将新LogEntry对象推到数组上，对新数组进行字符串化，然后将字符串化的数组放入本地存储中。

关于本地存储的注意事项之一；每个浏览器都设置了一个限制，可以存储多少数据。限制因浏览器而异，在撰写本文时，限制从2MB到10MB不等。您可能需要考虑在方法的catch块中编写一些其他代码，以log()在存储新的日志条目之前从数组中删除最旧的值。

该clear()方法用于清除指定键位置上的本地存储。调用对象的removeItem()方法localStorage以清除该位置内的所有值。

现在您有了新的类来将日志条目存储到本地存储中，您需要将其添加到Publishers数组中。打开log-publishers.service.ts文件并修改import语句以包含您的新LogLocalStorage类。

import { LogPublisher, LogConsole, `LogLocalStorage` } from "./log-publishers";

修改类的buildPublishers()方法LogPublishersService以创建该类的实例，LogLocalStorage并将其推送到Publishers数组，如以下代码所示。

buildPublishers(): void {

// Create instance of LogConsole Class

this.publishers.push(new LogConsole());

// Create instance of `LogLocalStorage` Class

this.publishers.push(new `LogLocalStorage`());

}

现在，您可以重新运行该应用程序，并且应该同时登录到控制台窗口和本地存储。为了测试这一点，请log()在LogLocalStorage类的方法中设置一个断点，然后查看它是否检索您登录到本地存储中的先前值。

登录到Web API

您要创建的最后一个日志记录类是将LogEntry类的实例发送到Web API方法的类。然后，您可以从该Web API编写代码以将日志条目存储到数据库表中。我不会为您提供桌子-我会留给您。我正在使用Visual Studio和C＃创建我的Web API调用，因此我将向我的项目中添加一个与我在Angular中创建的LogEntry类具有相同名称和相同属性的C＃类。

public class LogEntry

{

public DateTime EntryDate { get; set; }

public string Message { get; set; }

public LogLevel Level { get; set; }

public object[] ExtraInfo { get; set; }

}

我还在项目中添加了C＃枚举，以映射到TypeScriptLoggingLevel枚举。

public enum LogLevel

{

All = 0,

Debug = 1,

Info = 2,

Warn = 3,

Error = 4,

Fatal = 5,

Off = 6

}

最后，我将向我的项目中添加一个Web API控制器类（清单7）。此类此时具有一个方法，以允许Angular将LogEntry记录发布到此方法。通过这种Post()方法，您可以编写代码以将日志数据存储到数据库表中。为了本文的目的，我将把OK（true）的结果返回给调用者。

清单7：LogController允许您通过Web API调用存储日志条目

public class LogController : ApiController

{

// POST api/<controller>

[HttpPost]

public IHttpActionResult Post([FromBody]LogEntry value)

{

IHttpActionResult ret;

// TODO: Write code to store logging data in a database table

// Return OK for now

ret = Ok(true);

return ret;

}

}

现在，您已经创建了Web API类，您可以返回该log-publishers.ts文件并添加一些导入语句以调用Web API。在此文件顶部附近添加以下导入语句。

import { Http, Response, Headers, RequestOptions } from '@angular/http';

import 'rxjs/add/operator/map';

import 'rxjs/add/operator/catch';

import 'rxjs/add/observable/throw';

在文件底部添加清单8中LogWebApi所示的类。该类的构造函数与您为该类编写的构造函数非常相似。您确实需要包括HTTP服务，因为您将需要此服务来调用Web API。您还必须调用以执行基类的构造函数。最后，将属性设置为Web API调用的URL。log-publishers.tsLogLocalStoragesuper()location

清单8：创建一个LogWebApiService类以调用用于跟踪日志消息的Web API

export class `LogWebApi` extends LogPublisher {

constructor(private http: Http) {

// Must call `super()`from derived classes

super();

// Set location

this.location = "/api/log";

}

// Add log entry to back end data store

log(entry: LogEntry): Observable<boolean> {

let headers = new Headers({ 'Content-Type': 'application/json' });

let options = new RequestOptions({ headers: headers });

return this.http.post(this.location, entry, options).map(response => response.json()).catch(this.handleErrors);

}

// Clear all log entries from local storage

clear(): Observable<boolean> {

// TODO: Call Web API to clear all values

return Observable.of(true);

}

private handleErrors(error: any): Observable<any> {

let errors: string[] = [];

let msg: string = "";

msg = "Status: " + error.status;

msg += " - Status Text: " + error.statusText;

if (error.json()) {

msg += " - Exception Message: " + error.json().exceptionMessage;

}

errors.push(msg);

console.error('An error occurred', errors);

return Observable.throw(errors);

}

}

该log()方法接受LogEntry发送到Web API方法的对象。由于您要POST对Web API执行操作，因此需要创建适当的标头以指定要发送的内容类型application/json。在post()对角HTTP服务方法被调用到传递LogEntry的对象为您创建的Web API类。

clear()在此类中，您还需要一个方法来覆盖基类中的抽象方法。为了使本文的长度更短，我没有展示如何清除日志条目，但是它与log()方法类似。您调用Web API方法，该方法编写适当的SQL来从数据库中的日志表中删除所有行。

打开log-publishers.service.ts文件并修改import语句以包含您的新LogWebApi类。

import { LogPublisher, LogConsole, `LogLocalStorage`, LogWebApi } from "./log-publishers";

打开log-publishers.service.ts文件，并在文件顶部附近为HTTP服务添加导入。

import { Http } from '@angular/http';

接下来，将HTTP服务添加到此类的构造函数中。

constructor(private http: Http) {

// Build publishers arrays

this.buildPublishers();

}

最后，在该buildPublishers()方法中，创建LogWebApi该类的新实例并传递HTTP服务，如下面的代码所示。

buildPublishers(): void {

// Create instance of LogConsole Class

this.publishers.push(new LogConsole());

// Create instance of `LogLocalStorage` Class

this.publishers.push(new `LogLocalStorage`());

// Create instance of `LogWebApi` Class

this.publishers.push(new LogWebApi(this.http));

}

您需要使用来注册HTTP服务AppModule。打开app.module.ts并在文件顶部附近添加以下导入。

import { HttpModule } from '@angular/http';

将添加HttpModule到@NgModule()函数装饰器中的imports属性。

imports: [BrowserModule, HttpModule],

现在您已经将此新发布者添加到数组中，您应该可以运行日志记录应用程序，并且当您单击Log Test按钮时，您应该看到它正在调用Web API方法。

从JSON文件读取发布者

打开log-publishers.ts文件并添加一个名为的新类LogPublisherConfig。该类将保存从清单9中看到的JSON文件中读取的各个对象。

class LogPublisherConfig {

loggerName: string;

loggerLocation: string;

isActive: boolean;

}

通过\assets在下方添加一个文件夹来构建JSON文件\src\app folder。log-publishers.json在该\assets文件夹中添加一个名为的JSON文件，并添加清单9中的代码。JSON数组中的每个对象文字都与您创建的用于登录到控制台，本地存储和Web API的类之一有关。

清单9：创建一个LogLocalStorage Service类以将消息存储到本地存储中

[

{

"loggerName": "console",

"loggerLocation": "",

"isActive": true

},

{

"loggerName": "localstorage",

"loggerLocation": "logging",

"isActive": true

},

{

"loggerName": "webapi",

"loggerLocation": "/api/log",

"isActive": true

}

]

在您的log-publishers.service.ts文件中再添加一些导入语句。

import { Observable } from 'rxjs/Observable';

import 'rxjs/add/operator/map';

import 'rxjs/add/operator/catch';

import 'rxjs/add/observable/throw';

在这些导入之后添加一个常量，以指向此文件。

const PUBLISHERS\_FILE = "/src/app/assets/log-publishers.json";

在LogPublishersService该类中，添加一个名为handleErrors的新方法（清单10）来处理任何HTTP服务调用期间可能发生的任何错误。另外，在LogPublishersService该类中，创建一个新方法以从此JSON文件读取数据。让我们将此方法称为getLoggers()。因为您已经将HTTP服务注入到此类中，所以可以使用此服务从JSON文件读取。

getLoggers(): Observable<LogPublisherConfig[]> {

return this.http.get(PUBLISHERS\_FILE).map(response => response.json()).catch(this.handleErrors);

}

清单10：使用HTTP服务时，您始终应该有一种方法来处理错误

private handleErrors(error: any):

Observable<any> {

let errors: string[] = [];

let msg: string = "";

msg = "Status: " + error.status;

msg += " - Status Text: " + error.statusText;

if (error.json()) {

msg += " - Exception Message: " + error.json().exceptionMessage;

}

errors.push(msg);

console.error('An error occurred', errors);

return Observable.throw(errors);

}

现在您已经有了从该文件返回数组的buildPublishers()方法，请修改该方法以订阅此ObservableLogPublisherConfig对象数组。清单11中buildPublishers()显示了该方法的新代码。

清单11：通过读取JSON文件中的值来创建发布者数组

buildPublishers(): void {

let logPub: LogPublisher;

this.getLoggers().subscribe(response => {

for (let pub of response.filter(p => p.isActive)) {

switch (pub.loggerName.toLowerCase()) {

case "console":

logPub = new LogConsole();

break;

case "localstorage":

logPub = new `LogLocalStorage`();

break;

case "webapi":

logPub = new LogWebApi(this.http);

break;

}

// Set location of logging

logPub.location = pub.loggerLocation;

// Add publisher to array

this.publishers.push(logPub);

}

});

}

该buildPublishers()方法调用getLoggers()方法并预订该方法的输出。输出是LogPublisherConfig对象数组。对配置对象数组进行过滤以仅遍历那些将其isActive属性设置为True值的对象。对于循环中的每次迭代，请检查loggerName属性并将该值与每个case语句中列出的值进行比较。如果找到匹配项，则创建对应的LogPublisher类的新实例。该loggerLocation属性设置为每个LogPublisher类的location属性。然后将新实例化的发布者对象添加到publishers数组属性。因为所有这些都发生在该服务类的构造函数中；当Publishers数组被注入到LogService类中时，已经将其设置为要使用的发布者列表。您应该能够运行Angular应用程序，然后单击Log Test按钮，并查看发布到JSON文件中标记为isActive的所有发布者的日志消息。