链滴

关于微前端实现原理与 ngx-planet(四) 服务端渲染

作者：someone61489

原文链接：https://ld246.com/article/1614224127325

来源网站：链滴

1.为什么要服务端渲染

因为公司后端服务在k8s上，是分布式的微服务，之前端全部打包部署在了物理机器(虚拟机)nginx上，果通过helm做应用商店的话，nginx前端这部分无法处理，包括灰度部署，CI等，全部都只能做到接级别的处理，并不能连带静态资源文件一起处理，所以基于分布式的前端整改迫在眉睫。

偶然发现了ngx-planet，整篇文章基于前几篇文章，可以看之前的几篇文章。

2.如何基于ngx-palnet进行服务端渲染

2.0 有路由前缀的情况下服务端渲染ngx-planet如何改进

```
start(): void {
  this.subscription = this.router.events
    .pipe(
      filter((event) => {
        console.log(event)
        return event instanceof NavigationEnd;
      }),
      map((event: NavigationEnd) => {
        console.log(event)
        return event.urlAfterRedirects || event.url;
      }),
      startWith(location.pathname.split("/star-universe")[1]),
      distinctUntilChanged()
    )
    .subscribe((url: string) => {
      console.log(url)
      this.starApplicationLoader.reroute({ url: url });
    });
}
```

在 start函数中监听路由阶段，其中的 startWith过滤路由要把 location.pathname修改好，要将前去除掉，至于如何动态去除不写死，仁者见仁智者见智。

2.1 首先，准备打包好的静态文件

将 build命令修改，注意 --deploy-url的意思是，打包完成后，静态资源路径是什么

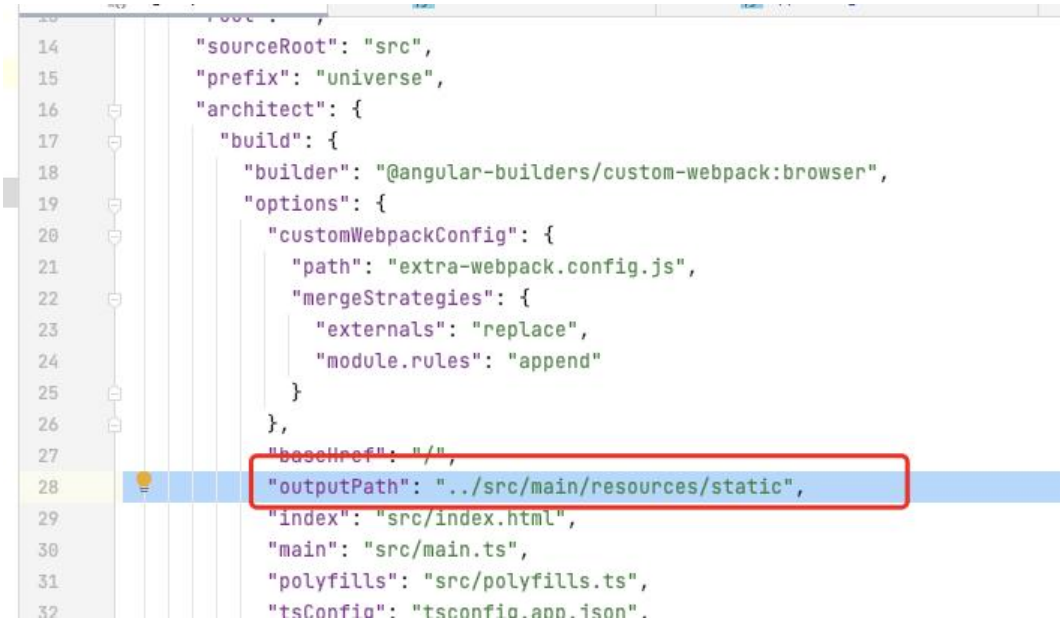
```
"build": "ng build --prod --deploy-url=/static/star-universe/ --base-href=/star-universe",
```

```
                    </div>
                  </div>
                </div>
              </td>
            </tr>
          </table>
        </td>
      </tr>
    </table>
  </div>
  <script src="/static/star-universe/polyfills-es5.b83c781575d38e12d616.js" nomodule defer></script>
</html>
```

打包后的index.html如图，静态资源路径变成了 /static/star-universe前缀，这里和 base-href不能突，否则在渲染时有死循环问题

其次 base-href就是项目访问路径前缀

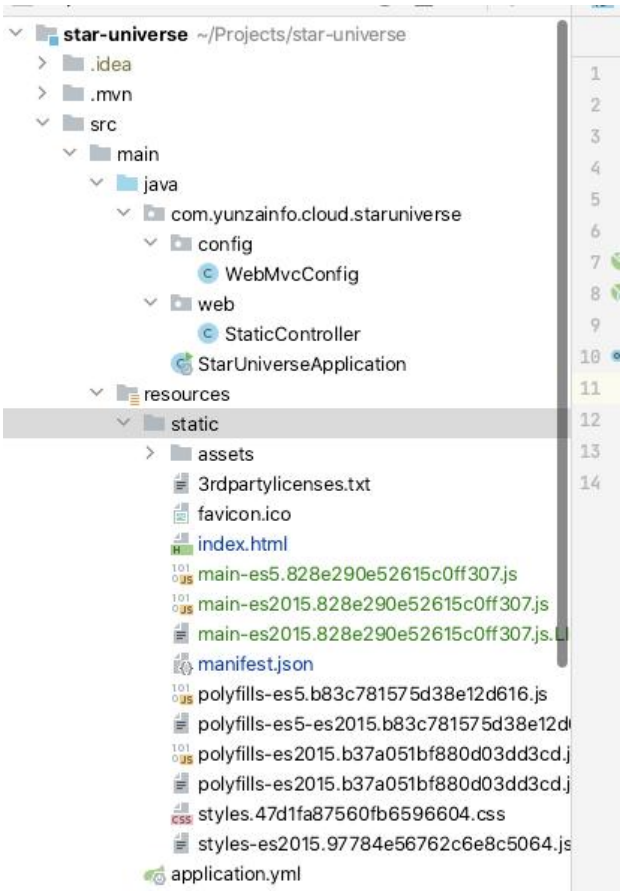
然后修改 angular.json中的 build放到你的静态资源目录，这样在执行 npm run build后静态资源自放入后台项目中的静态目录中



```
"sourceRoot": "src",
"prefix": "universe",
"architect": {
  "build": {
    "builder": "@angular-builders/custom-webpack:browser",
    "options": {
      "customWebpackConfig": {
        "path": "extra-webpack.config.js",
        "mergeStrategies": {
          "externals": "replace",
          "module.rules": "append"
        }
      },
      "baseHref": "/",
      "outputPath": "../src/main/resources/static",
      "index": "src/index.html",
      "main": "src/main.ts",
      "polyfills": "src/polyfills.ts",
      "tsConfig": "tsconfig.app.json",
```

2.2 准备Portal后台项目

我的项目基于SpringBoot自动生成，项目结构如下



以下是用到的maven配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
MLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xs
/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.4.3</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.yunzainfo.cloud</groupId>
    <artifactId>star-universe</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>star-universe</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

一个Controller负责重定向路由到 index.html

```
package com.yunzainfo.cloud.staruniverse.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;

@Controller
public class StaticController {

    @RequestMapping(value = {"/star-universe", "/star-universe/**"})
    public String index(HttpServletRequest request) {
        System.out.println(request.getRequestURI());
        return "index";
    }

}
```

portal的静态资源处理，通过实现 WebMvcConfigurer的 addResourceHandlers函数,注意函数传数的静态资源路径谓 /static/star-universe/**要和 --deploy-url对应起来，保证 index.html中的静资源被正确加载

```
package com.yunzainfo.cloud.staruniverse.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebMvcConfig implements WebMvcConfigurer {
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/static/star-universe/**").addResourceLocations("classpath
/static/");
    }
}
```

然后是portal的applicaiton.yml，给一个端口吧，这里我选择的是3000,和proxy.config中的相同，于静态资源路径为什么是 /static也是为了和注册应用时的路径保持一致

```
server:
  port: 3000

spring:
  thymeleaf:
    cache: false
    prefix: classpath:/static/
```

关于 app.component中的应用注册,resourcePathPrefix和 manifest全都加入应用路径完成远程注册

```
[
  {
    "name": "star-dust",
    "hostParent": "#wormhole",
    "hostClass": "star-dust",
    "routerPathPrefix": "/star-dust",
    "stylePrefix": "star-dust",
    "resourcePathPrefix": "http://localhost:3001/static/star-dust/",
    "loadSerial": true,
    "preload": false,
    "switchMode": "coexist",
    "scripts": [
      "main.js"
    ],
    "styles": [
      "styles.css"
    ],
    "manifest": "http://localhost:3001/static/star-dust/manifest.json"
  }
]
```

2.3 准备子项目前台项目

同样修改 build和 angular.json 然后打包

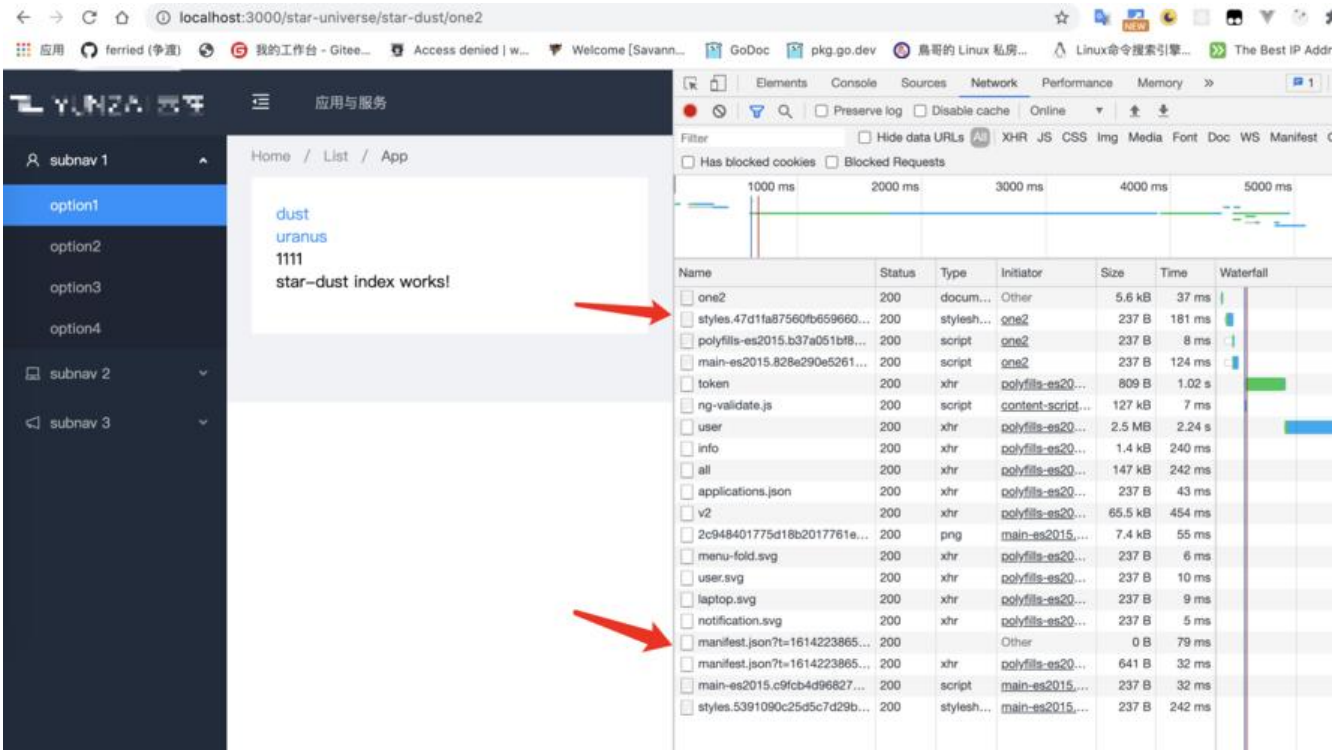
2.4 准备子项目后台项目

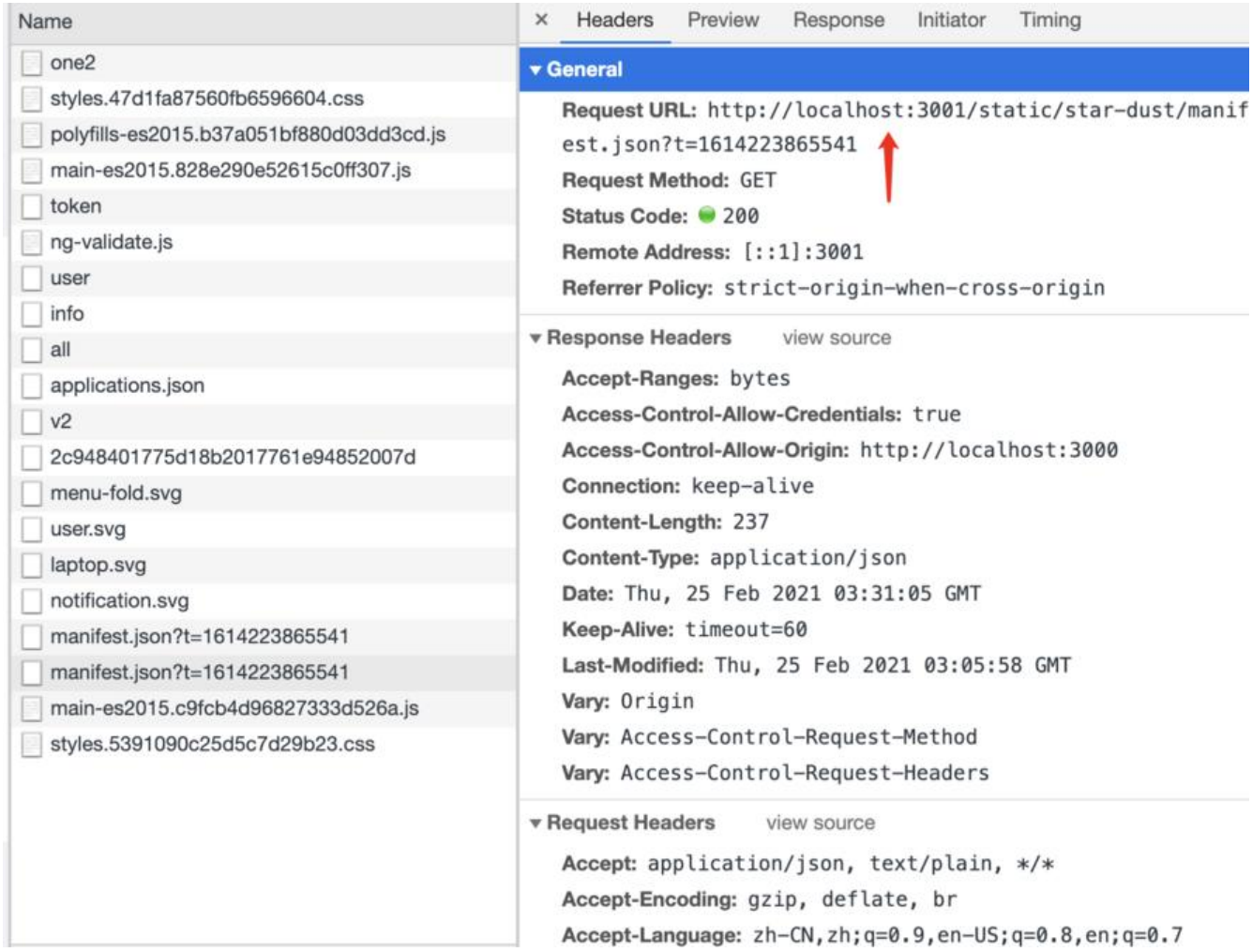
这里唯一和portal项目不同的时，要加入跨域访问允许，因为portal加载js/css的时候是远程加载的。

```
package com.yunzainfo.cloud.stardust.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.*;

@Configuration
public class WebMvcConfig implements WebMvcConfigurer {

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/static/star-dust/**").addResourceLocations("classpath:/sta
ic/");
    }

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowCredentials(true)
                .allowedOrigins("http://localhost:3000", "http://localhost:3001", "http://localhost:30
2", "http://localhost:3003")
                .allowedMethods("POST", "GET", "PUT", "OPTIONS", "DELETE")
                .maxAge(3600)
                .allowCredentials(true);

    }
}
```

其余和portal一致，只是路径不一样

3.看效果





这样，静态资源全都被远程请求过来了

4.这种远程技术可以运用到什么地方

首先组件共享的好处不说了，结合了后端微服务的 真正的前端微服务而不是依托于 nginx静态部署的端微服务

可以优化开发流程，可以做灰度部署了，可以完善CI了等等，好处也有，坏处也有。有利有弊，看各需要吧。

原文链接：关于微前端实现原理与 ngx-planet(四) 服务端渲染