高级node第十五单元

2020年6月18日 9:17

再谈webpack

1. webpack是node的一个包 可以 npm I webpack 下载
2. webpack的功能
   a. 让浏览器端运行的js代码支持commonjs规范（把commonjs规范的代码转换成浏览器端可以识别的代码）
   b. 在打包的过程中提供了自定义插件的接口，理论上可以随意的在打包过程中添加功能；
   c. 提供了loader能力可以把非js资源转换成js模块

通过nodejs使用webpack

1. 安装webpack npm I webpack -D

```
λ npm i webpack -D
```

2. 准备配置文件

```
const path = require('path');
module.exports = {
  entry: path.join(__dirname, '../src/index.js'),
  output: {
    path: path.join(__dirname, '../dist'),
    filename: 'index.boundle.js'
  },
  mode: 'development'
}
```

3. 引入webpack模块执行打包

```
const webpack = require('webpack'); // 引入webpack模块

const webpackConfig = require('./config.js'); // 引入webpack相关的配置

// 根据配置开始执行编译

webpack(webpackConfig, (err, stats) => {
  if (err || stats.hasErrors()) {
    console.log(err);
    return;
  }
  console.log(stats);
})
```

4. 使用compiler的打包方式

```
// 2.生出compiler对象的使用方式

const compiler = webpack(webpackConfig);
compiler.run((err, stats) => {
  if (err || stats.hasErrors()) {
    console.log(err);
    return;
  }
  let str = stats.toString({
    // 增加控制台颜色开关
    colors: true
  });
  console.log(str);
})
```

5. 可以实时监听的打包方式

```
// 3. 可以实时监听
const compiler = webpack(webpackConfig); // 生成编译实例
const watching = compiler.watch({}, (err, stats) => { // 生成监听实例
  if (err || stats.hasErrors()) {
    console.log(err);
    return;
  }
  let str = stats.toString({
    // 增加控制台颜色开关
    colors: true
  });
  console.log(str);
})
```

通过nodejs使用webpack-dev-server

```
const WebpackDevServer = require('webpack-dev-server');
const webpack = require('webpack')
const config = require('./config')
const compiler = webpack(config);
const devServerConfig = require('./devServerConfig');
const devServer = new WebpackDevServer(compiler, Object.assign({}, devServerConfig));
devServer.listen(8080, '0.0.0.0', () => {
  console.log('成功')
});
```

devServer的相关配置

打包后的资源

1. webpack打包出来资源默认会存储在磁盘中，路径就是output中配置的path和filename
2. 我们可以通过修改 compiler的`outputFileSystem`值，来控制打包后资源的存储位置

```
const MemoryFS = require("memory-fs"); //引入内存系统模块

// 2.生成compiler对象的使用方式
Const fs = new MemoryFS();
const compiler = webpack(webpackConfig);
//* 更改文件系统
compiler.outputFileSystem = fs;
compiler.run((err, stats) => {
  if (err || stats.hasErrors()) {
    console.log(err);
    return;
  }
  let str = stats.toString({
    // 增加控制台颜色开关
    colors: true
  });
  console.log(str);
  // 验证写打包位置
  const { path: pathname, filename } = webpackConfig.output;
  const resourse = fs.readFileSync(path.join(pathname, filename));
  console.log(resourse);
  console.log(resourse.toString());
})
```

资源不输出在磁盘，而是输出在内存中

3. Webpack-dev-server中就更换了文件系统

提取webpack-dev-server打包后的资源

1. 更改文件系统

```
const MemoryFS = require("memory-fs"); //引入内存系统模块
const fs = new MemoryFS();
module.exports = fs;

const config = require('./config')
const compiler = webpack(config);
const devServerConfig = require('./devServerConfig');
const fs = require('./memFs');
compiler.outputFileSystem = fs;
const devServer = new WebpackDevServer(compiler, Obje
devServer.listen(8080, '0.0.0.0', () => {
  console.log('成功')
```

2. 页面请求资源

```
  <div id="root"></div>
  <!-- 页面去请求打包后的资源 -->
  <script src="/mm"></script>
</body>
```

3. 给webpack-dev服务器添加接口返回打包后的资源

```
module.exports = {
  contentBase: path.join(__dirname, '../public'),
  quiet: true,
  before(app) {
    //在服务器启动的时候执行一次
    // 接收应用实例
    // 可以添加中间件和接口
    app.get('/mm', (req, res) => {
      const { path: pathname, filename } = config.output;
      // 返回打包后的资源
      const buf = fs.readFileSync(path.join(pathname, filename));
      res.set('Content-Type', 'application/javascript')
      res.send(buf);
    })
  }
}
```

Webapck-dev-middleware的使用

1. 作用实现webpack的功能
2. 可以作为express服务器的中间件

```
const express = require('express')
const webpackDevMiddleware = require('webpack-dev-middleware');
const webpack = require('webpack')
const webpackConfig = require('./build/config');
const compiler = webpack(webpackConfig);
const app = express();
const options = {
  writeToDisk: true
}
app.use(webpackDevMiddleware(compiler, options))

app.listen(3000, () => {
  console.log('localhost:3000')
})
```