# 入口 / entry

## 入口配置不同写法

### 单个入口

- 使用entry属性的简写语法
- 将一个相对路径传给entry属性
- 传递一个路径数组
  - 创建一个multi-main entry，
  - 一次性注入多个依赖文件
  - 本且将依赖关系绘制在一个"chunk"中
- 扩展或者调整配置的灵活性不高

### 对象语法，包含一个入口名字以及对应路径的键值对

- 意味着有多个入口
- 可扩展性更强
  - 将关注点从环境、构建目标、运行时中分离出来
  - 使用专门的工具将他们进行合并
- 可以传空对象给他
- 每个单独的入口可以使用一个对象描述，包括如下属性
  - dependOn——依赖的入口，必须先加载
  - filename——输出的文件名称
  - import——启动时需要加载的模块，即入口起点的相对文件路径
  - library——创建一个library
  - runtime——可以设为false，避免创建新的运行时chunk，可以没有
  - publicPath——入口的输出文件在浏览器中被引用时，指定一个公共的地址
  - 注意：runtime和dependOn不能同时使用
  - runtime不能使用已存在的入口名称
  - dependOn不能循环依赖
  - 使用对象语法

```
module.exports = {
 entry: {
      a2: 'dependingfile.js',
      b2: {
         dependOn: 'a2',
         import: './src/app.js'
      }
   }
}
```

- 经典用例，用于分离app(应用程序)和vendor(第三方库)入口
  - vendor.js中一般存入的时一些不会做修改的静态资源或者外部库，如Bootstrap，jQuery，图片等
  - 将他们打包在一起，然后使用一个内容哈希作为版本号，这样浏览器可以独立存储他们，从而减少加载时间，静态资源应该使用独立的缓存策略

```
module.exports = {
    entry:{
        main: './src/app.js',
        vendor: './src/vendor.js',
    }
}

/* webpack.prod.js */
module.exports = {
    output:{
        filename: '[name].[contenthash].bundle.js'
    }
}

/* webpack.dev.js */
module.exports = {
    output:{
        filename: '[name].bundle.js'
    }
}
```

  - 但是在4以及之后的的版本不建议这么做，即将vendor作为一个单独的入口并且编译为单独的文件，而是使用optimization.splitChunks选项将vendor和app模块分开，不应该为一个不是执行起点的模块创建entry
- 多页面应用程序
  - 如：三个一面直接配置为三个入口起点，三个独立的依赖图
  - 多页面程序中server会拉去一个新的HTML文档，重新加载新文档，资源重新下载，这样就可以使用optimization.splitChunks为页面间共享的代码创建共享的bundle，入口起点数量的增加，多页应用能够复用多个入口起点之间的大量代码 / 模块