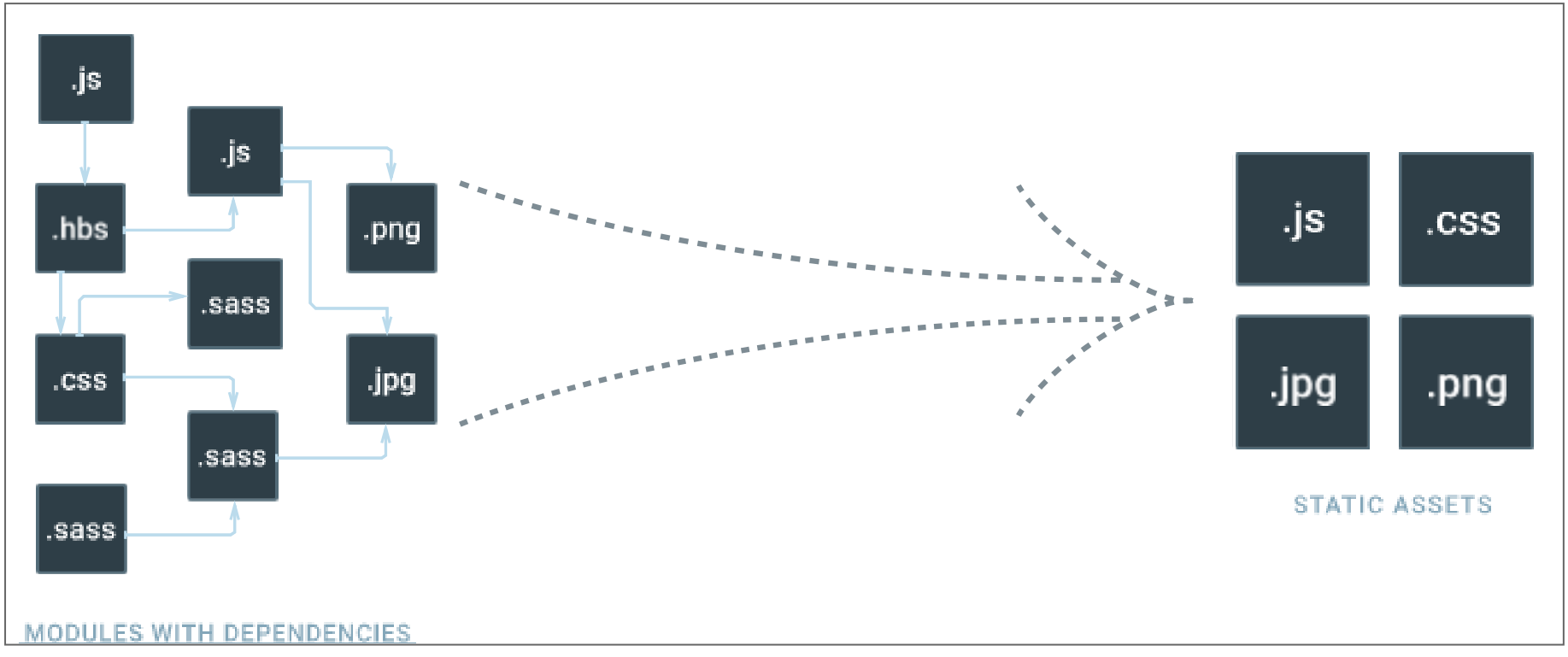
使用*Webpack*构建前端项目

小米移动前端-杨坤



webpack treats every file as a module, but, webpack itself only understands javascript

万物皆模块

构造模块化的开发体系

```
import img from "file.png";
import css from "file.css";
import sass from "file.sass";
```

- 环境搭建
- *开发*
- *部署*

1. 环境搭建

- 本地mock数据
- hot reload
- 本地静态文件服务器
- 本地接线上api

本地Mock数据

依据后端的wiki

```
const router = require('express').Router();
router.get('/wheel/turntableInfo', (req, res) =&gt; {
  res.json({
    rtnCode: 0,
    rtnMsg: 'ok',
    data: {
      remainingTimes: 7,
    },
  });
});
```

热加载

简单版

```
webpack-dev-server --hot
```

```
//css
{
  loader: 'style-loader',
  options: {
    hmr: true,
  }
}
```

热加载

升级版

```
var webpackConfig = require('./webpack.config');
var compiler = webpack(webpackConfig);
app.use(require("webpack-dev-middleware")(compiler, {
    noInfo: true, publicPath: webpackConfig.output.publicPath
}));
app.use(require("webpack-hot-middleware")(compiler, options));
```

```
entry: {
  app: ['webpack-hot-middleware/client?reload=true', 'main.js']
}
plugins: {
  new webpack.HotModuleReplacementPlugin(),
}
```

本地静态文件服务器

```
localhost:8080\page1
localhost:8080\page2
localhost:8080\page3
```

```
page1.html  page2.html  page3.html
```

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>小米移动</title>
</head>
<body>
</body>
</html>
```

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>小米移动</title>
  <link href="/app.css" rel="stylesheet">
</head>
<body>
  <script type="text/javascript" src="/app1.js"></script>
  <script type="text/javascript" src="/app2.js"></script>
</body>
</html>
```

```
plugins: [
  new HtmlWebpackPlugin({
    template: './src/page1.html',
  }),
],
```

```
app.get('/act/:name', (req, res) =&gt; {
  const path = webpackConfig.output.path + '/' + req.params.name + '.html';
  const htmlBuffer = devMiddlewareInstance.fileSystem.readFileSync(path);
  res.cookie('userId', 123456);
  res.send(htmlBuffer.toString());
});
```

本地接线上Api

```
anyProxy  代理
switchOmega  chrome插件  切换规则
```

2. 开发

- *组件化开发*

按照功能划分，分为三个层次的组件：
页面级(page)、组件级(component)、JS模块(utils)

- *做好资源管理*

开发过程中的资源管理：
1. js、css、jsx、vue的编译
2. js、css、图片、字体等路径的处理
3. 框架和库的使用

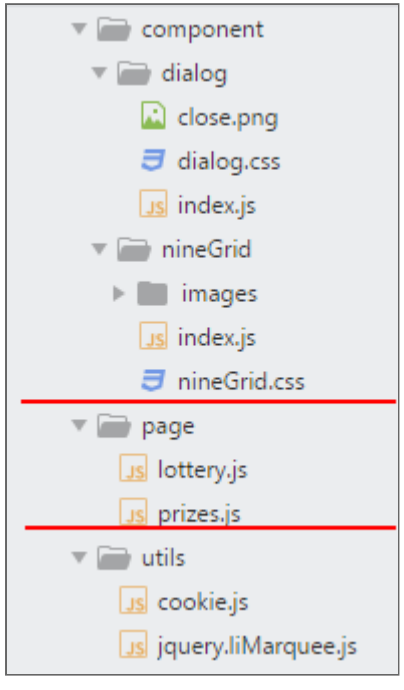
性能优化：
减少文件体积、减少http请求次数、代码分块、按需加载、cdn资源处理、css提取、打包文件

组件化开发

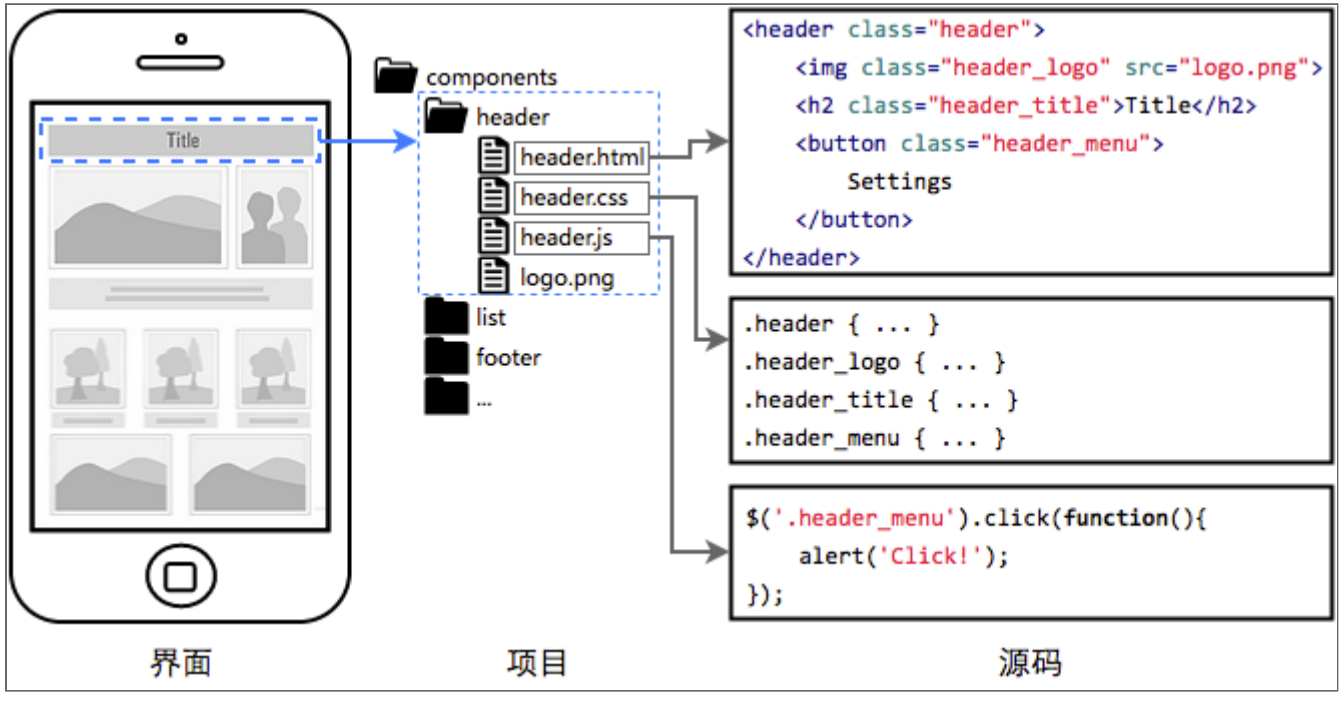
之前的页面结构

- fonts
- skins
- animate.css
- book.css
- bootstrap.min.css
- custom.css
- font-awesome.css
- jquery.fancybox-1.3.4.css
- main.css
- normalize.css
- print.css

模块化后的页面结构


component
dialog
close.png
dialog.css
index.js
nineGrid
images
index.js
nineGrid.css
page
lottery.js
prizes.js
utils
cookie.js
jquery.liMarquee.js

组件示意图


Title
components
header
header.html
header.css
header.js
logo.png
list
footer
...
<header class="header">
<img class="header_logo" src="logo.png">
<h2 class="header_title">Title</h2>
<button class="header_menu">
Settings
</button>
</header>
.header { ... }
.header_logo { ... }
.header_title { ... }
.header_menu { ... }
$('.header_menu').click(function(){
alert('Click!');
});
界面
项目
源码

编写一个Dialog

```
import { getInfo } from '../utils/api';
import getCookie from '../utils/cookie';
import './dialog.css';
import icon from './success-icon.png';
```

用到的webpack loader:

```
1. file-loader
2. url-loader
3. babel-loader
4. css-loader
5. style-loader
```

资源管理

路径处理

```
import img from 'icon.png';
var myImage = new Image(100, 200);
myImage.src = img;
document.body.appendChild(myImage);
```

```
{
  test: /\.(png|jpg|jpeg|gif|eot|ttf|woff|woff2|svg|svgz)(\?.+)?$/,
  use: [{
    loader: 'url-loader',
    options: {
      limit: 8192, //Byte limit to inline files as Data URL
    }
  }]
},
```

小图片

```
<img src="...">
```

大图片

```
<img src="/img/icon.png">
```

框架的引入

React: *.jsx*解析

```
babel-loader  babel-preset-react
```

Vue: *.vue*模板文件

```
{
  test: /\.vue$/,
  loader: 'vue-loader',
}
```

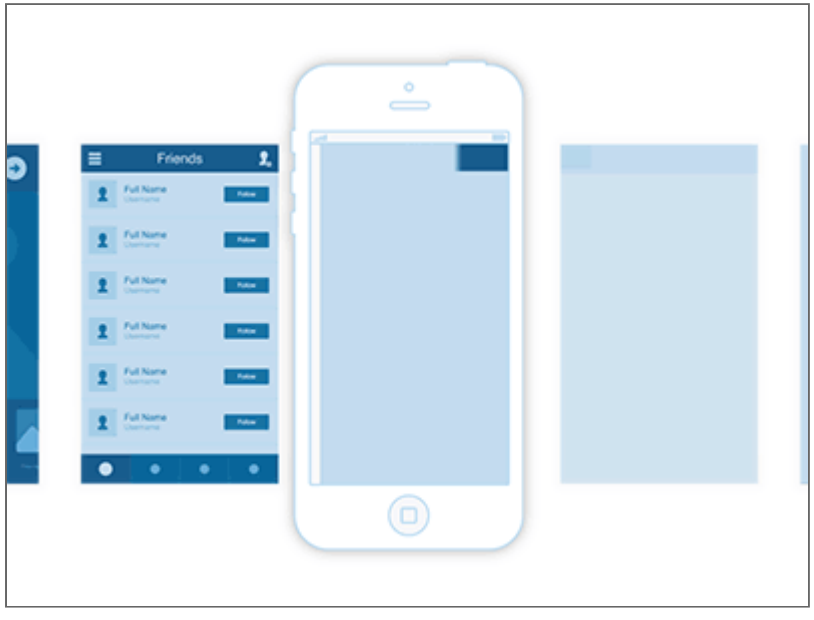
jQuery

以及其扩展库，如bootstrap.modal.js、jquery.datatable.js

```
使用expose-loader暴露$和JQuery
```

性能优化

前端是一种远程部署，运行时增量下载的GUI软件



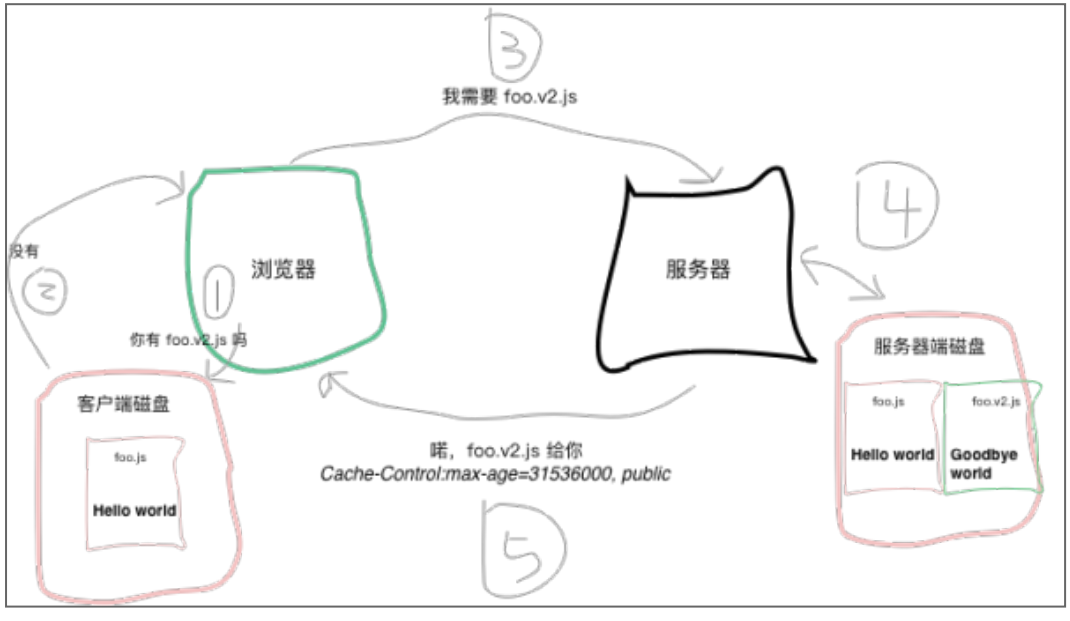
代码压缩

```
new webpack.optimize.UglifyJsPlugin({
  compress: { warnings: false },
  output: { comments: false },
}),
```

代码分块

1. *提取commonsChunk*
2. *动态引入*

持久性缓存


3
我需要 foo.v2.js
4
没有
2
1
浏览器
服务器
你有 foo.v2.js 吗
服务器端磁盘
foo.js
foo.v2.js
Hello world
Goodbye world
客户端磁盘
foo.js
Hello world
喏，foo.v2.js 给你
Cache-Control:max-age=31536000, public
5

1. 添加文件指纹

```
foo.28d663ec.js
bar.d3ea8991.js
```

2. 提取manifest
3. 引入HashedModuleIdsPlugin

提取CSS

```
new ExtractTextPlugin({
  filename: '[name].[hash].css',
  allChunks: true,
  ignoreOrder: true,
}),
```

Cdn

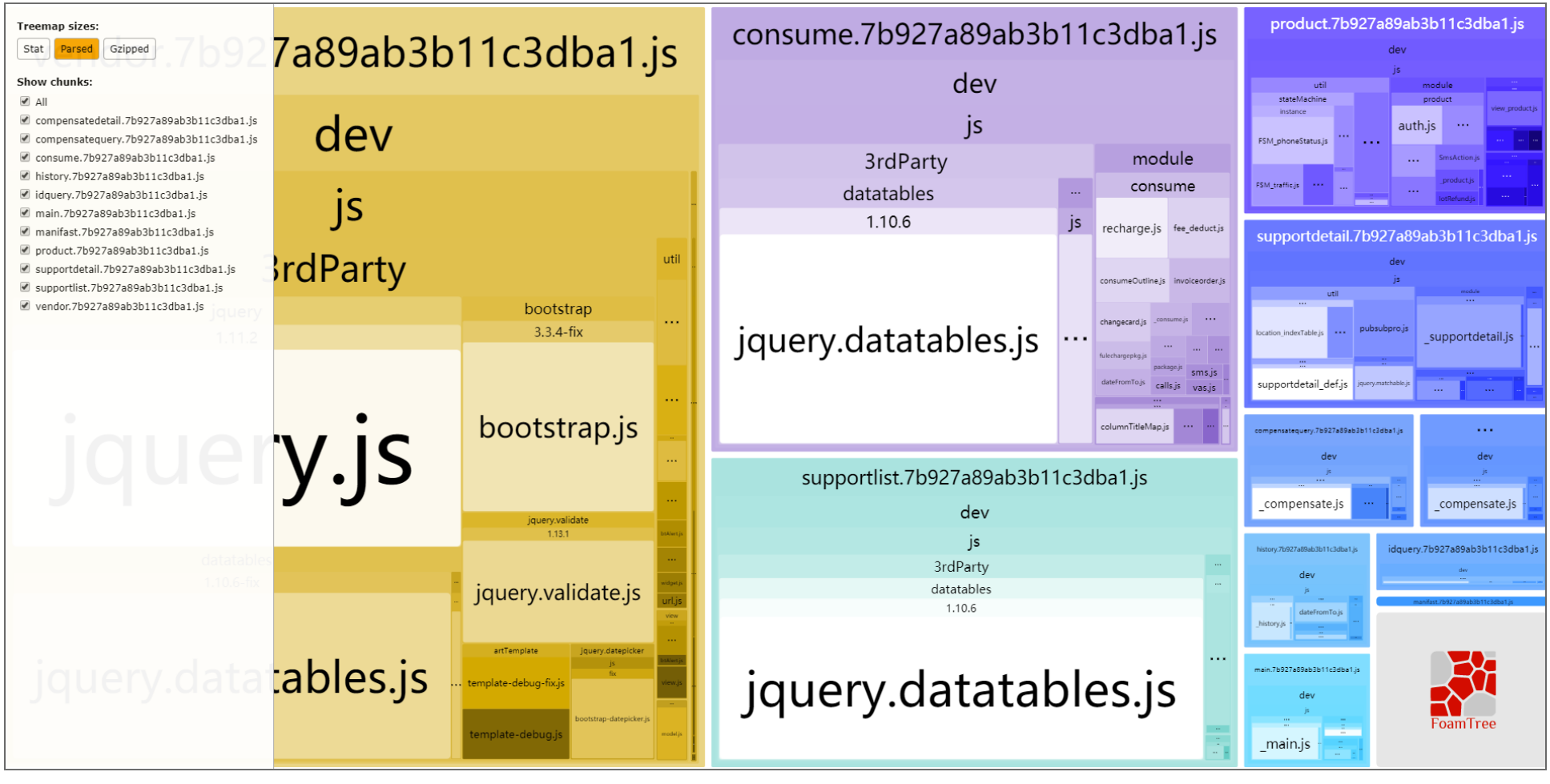
```
output: {
  publicePath: 'https://sec-boss.static.xiaomi.net/'
}
new HtmlWebpackPlugin(options)
```

Source-Map

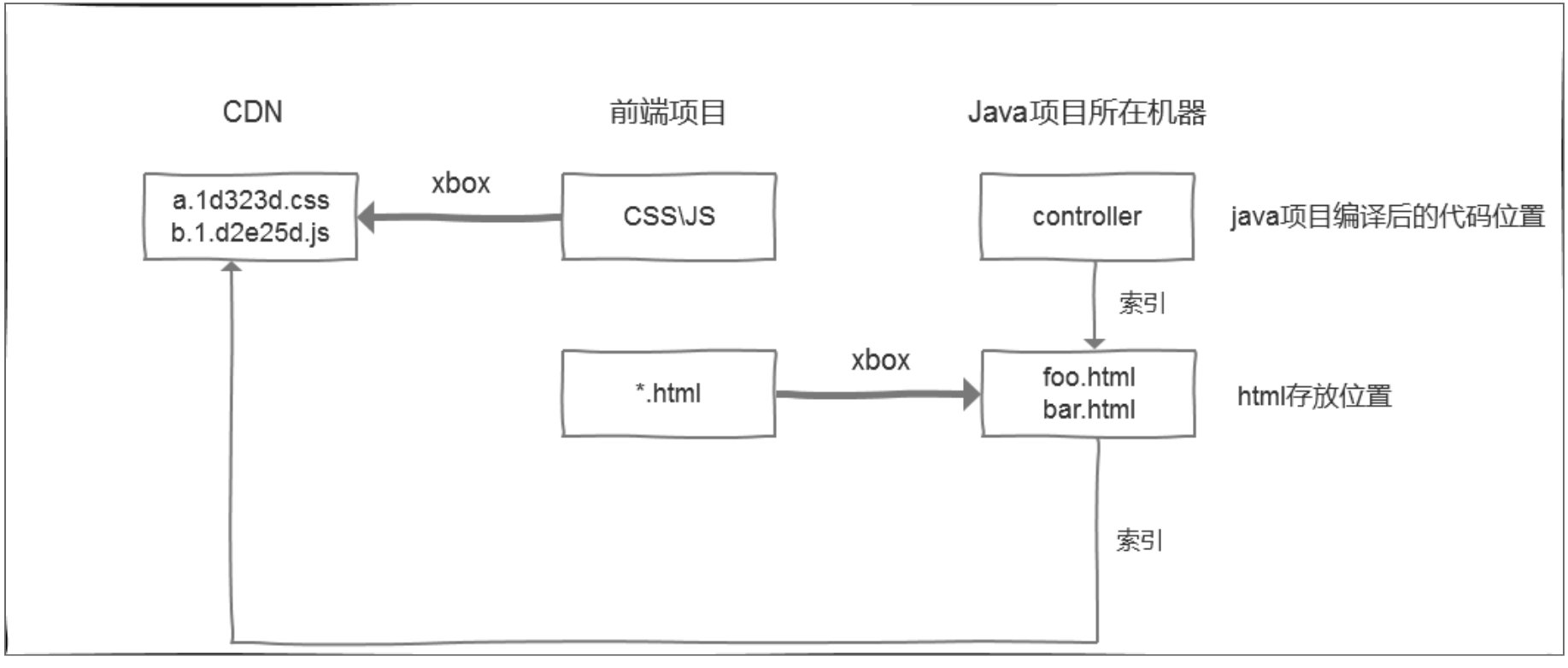
devtool	build	rebuild	production	quality
(none)	+++	+++	no	bundled code
eval	+++	+++	no	generated code
cheap-eval-source-map	+	++	no	transformed code (lines only)
cheap-module-eval-source-map	o	++	no	original source (lines only)
eval-source-map	--	+	no	original source
cheap-source-map	+	o	no	transformed code (lines only)
cheap-module-source-map	o	-	no	original source (lines only)
inline-cheap-source-map	+	o	no	transformed code (lines only)
inline-cheap-module-source-map	o	-	no	original source (lines only)
source-map	--	--	yes	original source
inline-source-map	--	--	no	original source
hidden-source-map	--	--	yes	original source
nosources-source-map	--	--	yes	without source content

分析打包文件

BundleAnalyzerPlugin



3. 部署


CDN
前端项目
Java项目所在机器
a.1d323d.css
b.1.d2e25d.js
xbox
CSS\JS
controller
java项目编译后的代码位置
索引
*.html
xbox
foo.html
bar.html
html存放位置
索引

Webpack原理

Webpack构建过程

1. 读取并初始化option
2. 编译
3. 递归分析依赖，按照依赖build
4. 构建，构建过程中会用相应的loader
5. 构建完毕后编译，生成AST抽象语法树
6. 遍历AST，在有依赖时，收集依赖
7. 打包前合并、压缩等
8. 输出文件

细说 webpsmallck 之流程篇