

Яндекс

Яндекс

Про сборку для начинающих



Подключение js курильщика



```
<body>
```

```
  <div id="app"></div>
```

```
  <script src="/node_modules/jquery/dist/jquery.min.js"></script>
```

```
  <script src="/node_modules/material/dist/material.min.js"></script>
```

```
  <script src="/static/js/components/uploader.js"></script>
```

```
  <script src="/static/js/index.js"></script>
```

```
</body>
```

<script> <script> <script> <script>
<script> <script> <script>
<script>

Почему плохо все подключать
тегами script?

Это самый худший по скорости способ подключения (а какие способы лучше и в каких случаях?)

Обычно, такие «подключения» скриптов бездумно копируются со страницы на страницу, хотя где-то может быть не нужным какой-то компонент или даже целая библиотека

Вдобавок, мы не контролируем совместимость - какой-то код/библиотека может быть написана на es6, что-то - на es5, в итоге какие браузеры мы поддерживаем нам неизвестно

Нужно самому контролировать порядок подключения скриптов на странице



При этом, писать код в отдельных файлах довольно удобно - вы можем разделять компоненты в отдельные классы, писать рядом стили и все вот это

Import

Это «новый» es6 синтаксис модулей, который не поддерживается в node.js нативно, но современными браузерами.

Он в общем весьма несложный, и ничего нового по сравнению с CommonJS модулями там нет

Require меняется на import, а module.exports на export

Import

Пример кода

index.html

```
<script type="module" src="index.js"></script>
```

index.js

```
import generateHash from './generateHash.js';
```

```
console.log(generateHash());
```

generateHash.js

```
export default function generateHash(length = 8) {  
  //...  
}
```



Вот если бы кто-то автоматически собирал только нужное для страницы...

Построение страницы со сборщиком



У нас есть основная логика страницы (часто мы ее пишем инлайново, непосредственно в тегах `<script></script>`)

В ней, если нам что-то нужно, типа jQuery или какого-то компонента, мы просто добавляем его выше в теге `<script src="">`

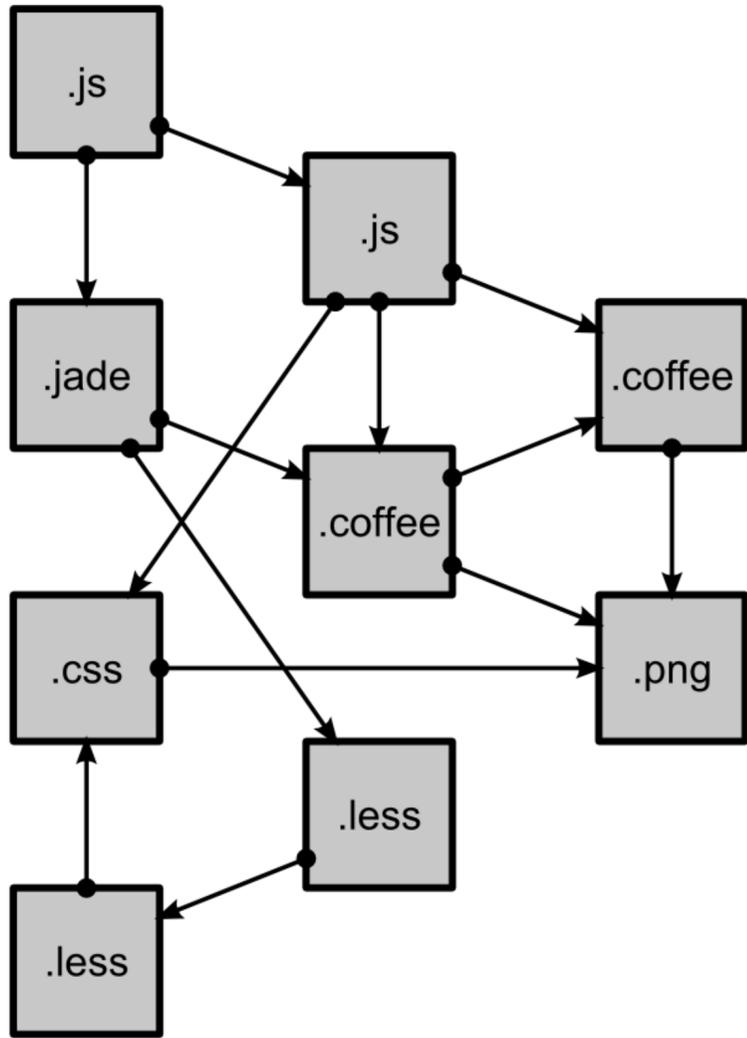
Теперь вместо основной логики будет «точка входа» - исходный файл, а все подключения новых файлов мы будем делать через `import`

| Пример

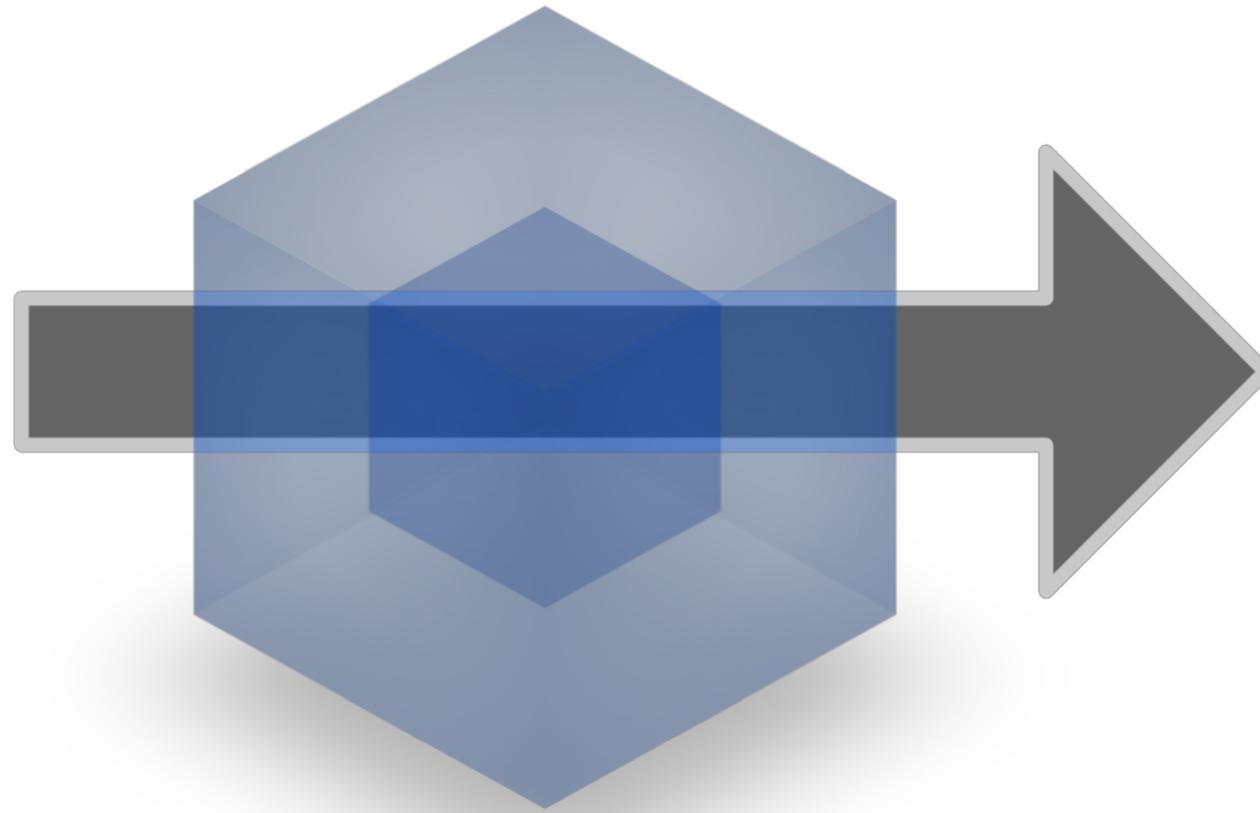
Итого

Мы соединили все файлы в один, начав с «точки входа»

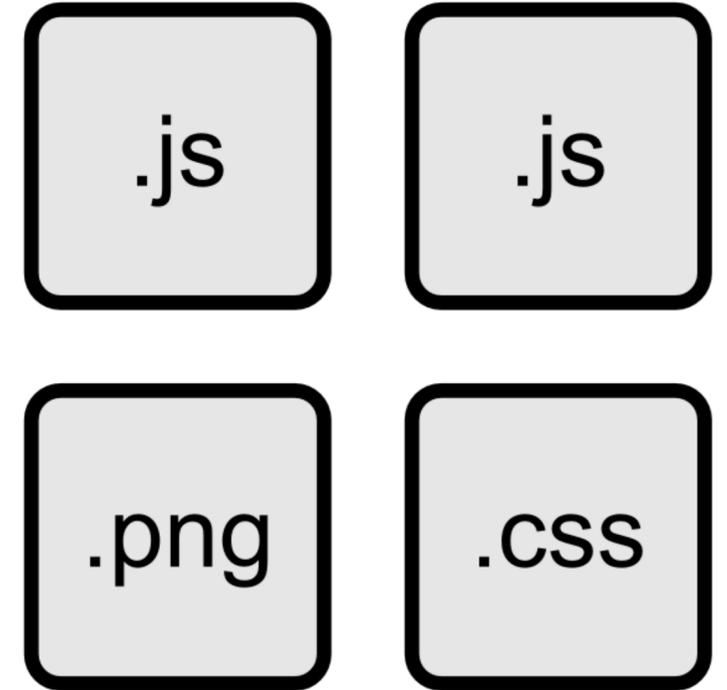
Вебпак встречая `import` идет и приклеивает этот файл. Конечно, предварительно считывая, и, если в нем находятся еще зависимости/импорты, то они приклеиваются аналогично.



modules
with dependencies



webpack
MODULE BUNDLER



static
assets

webpack.config.js



Базовый конфиг

Берем с сайта документации, там указана точка входа, подстраиваем под себя.

Устанавливаем необходимые плагины, если он просит

Видим упоминание некоей сущности под названием babel, давайте разбираться, что это такое.

Babel



Babel

Это инструмент, способный переводить код из одного стандарта языка в другой.

В том числе поддерживаются стандарты, имеющие статусы черновиков.

Есть песочница - <https://babeljs.io/repl>



Отладка и Source Maps

Source Maps

Возможно, увидев преобразованный код, вам подумалось, что случись в нем ошибка, и понять, что произошло будет нереально

Слава богу, об этом уже подумали. Вебпак при любых преобразованиях кода строит соурс мапы - соответствия кода склеенного к исходному. Браузеры эту штуку поддерживают, и благодаря этому мы видим где в исходном коде проблема, и можем установить брейкпойнт



Подключаем сборку в приложение



Собираем CSS



CSS Препроцессоры

CSS препроцессоры

Теперь, когда мы обрабатываем так или иначе наши CSS файлы, мы можем проводить с ними всякие преобразования по ходу сборки.

На этом принципе основаны CSS-препроцессоры - новая штука, которую вы пока не знаете.

На самом деле, у препроцессоров есть много фиш, но на 99% используется одна

Яндекс

Спасибо

Шлейко Александр

Разработчик интерфейсов



dusty@yandex-team.ru



@dustyo_O