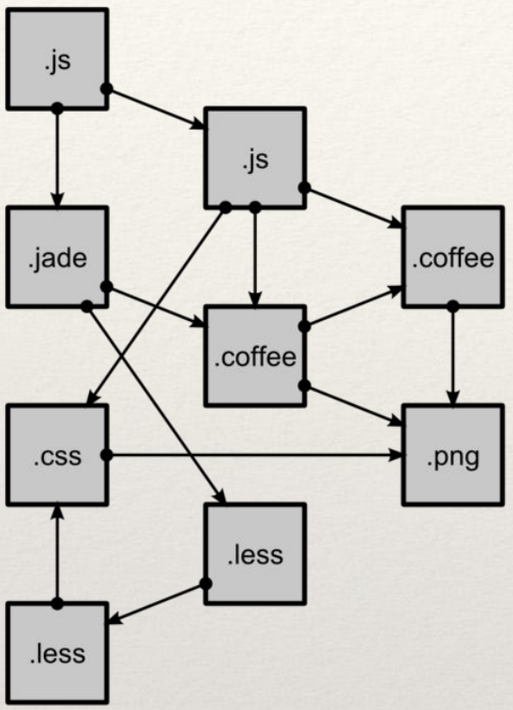
Webpack

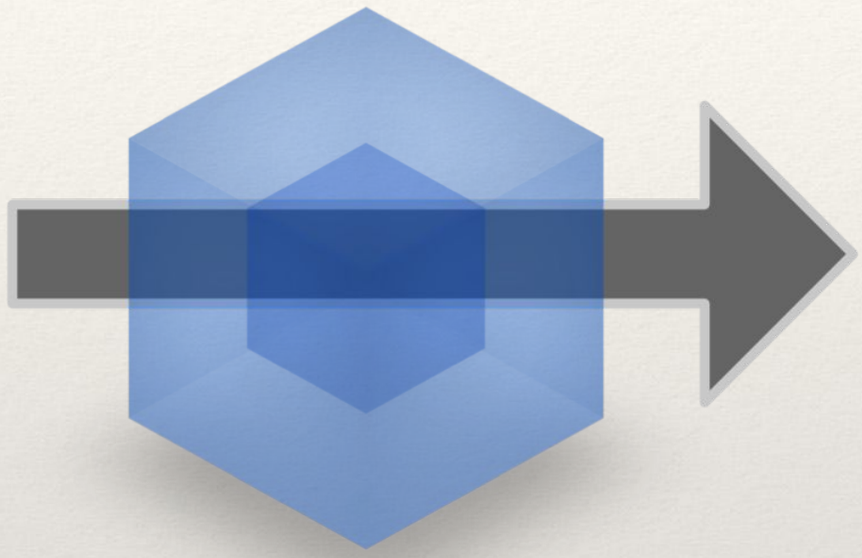
Что такое webpack

webpack - статический модульный сборщик для приложений на JavaScript.

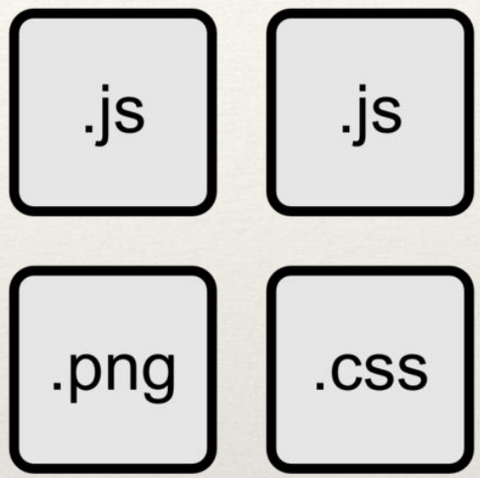
Когда webpack обрабатывает ваше приложение, он строит граф зависимостей, который отображает каждый модуль и генерирует один или несколько бандловю



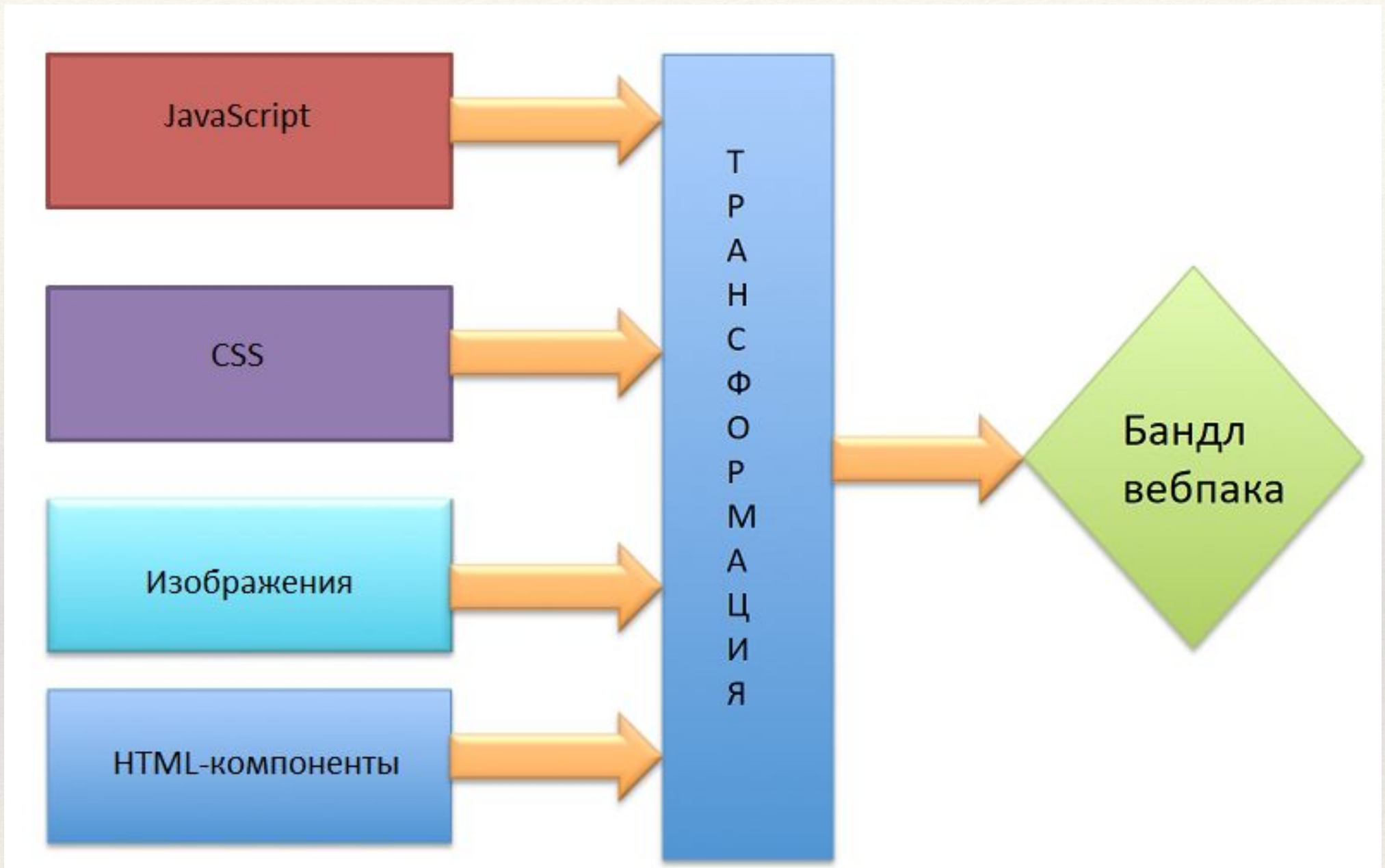
modules
with dependencies



webpack
MODULE BUNDLER



static
assets


JavaScript
CSS
Изображения
HTML-компоненты
ТРАНСФОРМАЦИЯ
Бандл вебпака

webpack выполняет много операций

- помогает связывать ваши ресурсы;
- наблюдает за изменениями и перезапускает задачи;
- может запустить Babel, благодаря этому, вы сможете использовать последние фичи JS без мыслей о поддержке браузеров;
- может перевести TypeScript в JavaScript;
- позволяет использовать require() для CSS файлов, JSON файлов, для изображений и т.д.
- может запустить web-server разработки;
- может разделить выходные файлы на несколько файлов, чтобы избежать загрузки большого файла при входе в приложение.

Установка

- npm i -D webpack
- npm i -D webpack-cli

Если вы используете webpack версии 4 или выше, то вам нужно также установить webpack-cli.

Запуск

- npx webpack

Использование конфигурации

Используя webpack версии 4, файл конфигурации можно и не создавать, но для больших проектов и для лучшего определения настроек нужно создавать файл **webpack.config.js.** Именно в нем мы будем прописывать, как должен вести себя webpack.

```
const path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'main.js',
    path: path.resolve(__dirname, 'dist')
  }
};
```

- npx webpack —config webpack.config.js

В данном примере оператор **—config** не нужен, так как имя конфига - по умолчанию. Этот оператор мы добавляем, когда хотим изменить имя конфига или когда у нас несколько конфигов.

Конфигурационный файл дает больше гибкости, чем простые команды в терминале. В нем можем подключать плагины, настраивать правила загрузок и т.д.

loaders

loaders - это преобразования, которые применяются к исходному коду. С помощью их можно предварительно обрабатывать файлы при импорте.

Использование CSS

Чтобы импортировать файл CSS из JS, вам нужно установить и добавить **style-loader** и **css-loader** в ваш конфиг.

- npm i -D style-loader css-loader

```
const path = require("path");

module.exports = {
  entry: "./src/index.js",
  output: {
    filename: "bundle.js",
    path: path.resolve(__dirname, "dist")
  },
  module: {
    rules: [
      {
        test: /\.css$/,
        use: ["style-loader", "css-loader"]
      }
    ]
  }
};
```

- import “./style.css”;

Теперь, когда модуль будет запущен, тэг **<style>** с CSS кодом в виде строки будет вставлен в **<head>** вашего **html** файла.

Также вы можете использовать разные препроцессоры (sass, less). Нужно лишь установить соответствующий лоадер.

Использование изображений

- npm i -D file-loader

Используя **file-loader** вы также можете включить изображения в ваше приложение

```
const path = require("path");

module.exports = {
  entry: "./src/index.js",
  output: {
    filename: "bundle.js",
    path: path.resolve(__dirname, "dist")
  },
  module: {
    rules: [
      {
        test: /\.(png|svg|jpg|gif)$/,
        use: ["file-loader"]
      }
    ]
  }
};
```

```
1   import Icon from './icon.png';
2
3   var myIcon = new Image();
4   myIcon.src = Icon;
5
6   document.body.appendChild(myIcon);
```

Следующим шагом, является минимизация и оптимизация ваших изображений. Вы можете использовать **image-webpack-loader** и **url-loader** для улучшения процесса загрузки изображений.

Использование шрифтов

```
const path = require("path");

module.exports = {
  entry: "./src/index.js",
  output: {
    filename: "bundle.js",
    path: path.resolve(__dirname, "dist")
  },
  module: {
    rules: [
      {
        test: /\.(woff|woff2|eot|ttf|otf)$/,
        use: ["file-loader"]
      }
    ]
  }
};
```

```
@font-face {
  font-family: "MyFont";
  src: url("./my-font.woff2") format("woff2"),
    url("./my-font.woff") format("woff");
  font-weight: 600;
  font-style: normal;
}
```

Плагины

plugins - набор инструментов, позволяющие дополнять функционал webpack.

HtmlWebpackPlugin

- npm i -D html-webpack-plugin

```
const HtmlWebpackPlugin = require("html-webpack-plugin");

module.exports = {
  plugins: [
    new HtmlWebpackPlugin({
      title: "Output Management"
    })
  ]
};
```

Чистка папки dist

clean-webpack-plugin

- npm i -D clean-webpack-plugin

```
1  const CleanWebpackPlugin = require("clean-webpack-plugin");
2
3  module.exports = {
4    plugins: [new CleanWebpackPlugin()]
5  };
```

Development

```
1   module.exports = {
2     mode: "development"
3
4     // previous code base
5   };
```

Source maps

Есть много случаев, когда вам понадобятся **source maps.** Их нужно добавлять в соответствии с вашими потребностями.

Мы будем использовать **inline-source-map**.

```
module.exports = {
  mode: "development",
  devtool: 'inline-source-map',

  // previous code base
};
```

Постоянно использовать команду для сборки проекта, когда вы хотите скомпилировать код, быстро становится проблемой.

В **webpack** есть ряд решений, которые помогут вам автоматически компилировать код при его изменении.

- watch mode;
- webpack-dev-server.

Чаще всего вам придется использовать **webpack-dev-server.** К нему можно добавить ряд настроек для работы с проектом.

Watch mode

Вы можете включить **watch mode. Webpack** будет наблюдать за вашими зависимостями, если в одном из файлов произошли изменения, то код будет автоматически перекомпелирован.

```
module.exports = {
  mode: "development",
  watch: true,
```

```
{
  "name": "push-notif",
  "version": "1.0.0",
  "description": "",
  "main": "webpack.config.js",
  "scripts": {
    "build": "webpack",
    "watch": "webpack --mode development --watch"
  },
```

webpack-dev-server

- npm i -D webpack-dev-server

```
{
  "name": "push-notif",
  "version": "1.0.0",
  "description": "",
  "main": "webpack.config.js",
  "scripts": {
    "watch": "webpack-dev-server",
    "build": "webpack"
  },
```

Hot Module Replacement

HMR - добавляет, удаляет или изменяет модули во время работы приложения без полной перезагрузки. Это может значительно ускорить разработку несколькими способами:

- сохранять состояние приложения, которое будет потеряно при полной перезагрузке;
- экономьте время разработки, обновляя то, что изменилось;
- изменения вносимые в CSS, JS в коде, приводит к мгновенному обновлению браузера.

```
const webpack = require("webpack");

module.exports = {
  devServer: {
    contentBase: "./dist",
    hot: true
  },
  plugins: [new webpack.HotModuleReplacementPlugin()]
};
```

Production

```
module.exports = {
  mode: "production"
};
```

Цели разработки и сборки в **production** сильно различаются. В процессе разработки нам нужно сильное сопоставление исходного кода(source map) и локальный сервер с быстрой перезагрузкой модулей.

Для **production** наши цели сдвигаются в сторону минимизации кода.
Для этого мы можем разделить наш коняги на несколько частей, вынести общую часть в один коняги и использовать его. А для **dev** и **prod** мы настроим отдельные кончики.

webpack-merge

- npm i -D webpack-merge

```
webpack-demo
   |- package.json
-  |- webpack.config.js
+  |- webpack.common.js
+  |- webpack.dev.js
+  |- webpack.prod.js
   |- /dist
   |- /src
     |- index.js
     |- math.js
   |- /node_modules
```

Минификация кода

- встроенный в **webpack** объект настроек **minimizer**;
- BabelMinifyWebpackPlugin;
- UglifyjsWebpackPlugin;
- TerserWebpackPlugin.