Дополнительные пакеты

Rollup в базовой установке справляется со сборкой JS-файлов, однако в нём не хватает нескольких важных функций:

- Поддержка старых браузеров
- Включение в сборку CSS-файлов
- Включение в сборку картинок
- Автоматическая пересборка проекта при наличии изменений в коде

Добавить эту функциональность можно с помощью дополнительных пакетов — плагинов к Rollup.

Babel

Babel — это транспайлер, который в основном используется для конвертации нового синтаксиса JavaScript в старый, чтобы можно было использовать новые возможности языка в старых браузерах. К примеру, стрелочные функции не поддерживаются в браузере Internet Explorer, однако, благодаря Babel, код с ними не придется переписывать вручную для поддержки в IE.

Помимо этого, Babel имеет много других полезных применений. Например, он может преобразовывать JSX-код в JavaScript. С этим мы подробнее познакомимся в модуле по React.

Найдем плагин @rollup/plugin-babel на сайте npm и произведём установку, выполнив команду в терминале:

```
npm install -D @rollup/plugin-babel @babel/preset-env
```

С помощью этой команды мы также установили @babel/preset-env — умный пресет, который как раз и позволяет использовать последнюю версию JavaScript, подключая только нужные плагины, основываясь на браузерах, которые поддерживает конкретный проект.

После установки откроем файл `package.json` и убедимся, что в него добавилась секция `devDependencies` со списком установленных пакетов:

```
{
  "name": "code",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {},
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@rollup/plugin-babel": "^5.3.1",
    "@babel/preset-env": "^7.18.10"
  }
}
```

Перечисленные в секции `devDependencies` пакеты используются только для разработки и не включаются в итоговую сборку. Babel попал сюда, так как мы указали флаг `--save-dev` при установке пакета.

Если при установке не использовать флаг `--save-dev` , то пакет будет включен в сборку проекта. Такие пакеты добавляются в секцию `dependencies` файла `package.json` (пока у нас нет этой секции).

Сокращенным аналогом флагу `--save-dev` при установке пакетов является флаг `-D` .

Чтобы Rollup использовал Babel при сборке, нужно также внести изменения в конфиг `rollup.config.js` :

```
import { babel } from '@rollup/plugin-babel';

export default {
  input: './index.js',
  output: {
    file: './build/bundle.js',
    format: 'cjs'
  },
  plugins: [
    babel({
      babelHelpers: "bundled",
      presets: ["@babel/env"]
    }),
  ]
};
```

В начало конфига добавилась инструкция `import` . О ней мы поговорим в следующем уроке. Также была добавлена секция `plugins` со списком плагинов. Аналогичным образом мы будем в дальнейшем добавлять другие плагины.

Для проверки работы плагина добавим в наш `index.js` файл какой-нибудь код, которого не было в более ранних версиях языка. Например, используем стрелочную функцию в методе `map()` :

```
const array = [1, 2, 3].map(n => n + 1);
console.log(array);
```

После этого запустим сборку с помощью команды `rollup -c` и найдем наш код в выходном файле `build/bundle.js` :

```
var array = [1, 2, 3].map(function (n) {
  return n + 1;
});
console.log(array);
```

Как видно, Babel сделал два изменения в коде:

- заменил ключевое слово `const` на `var`
- заменил стрелочную функцию на обычную

Плагин для работы со стилями

Чтобы Rollup включал в сборку помимо JS-файлов также и CSS-файлы, нужно установить ещё один плагин. Но перед этим добавим в наш проект файл со стилями. Например, такой:

```
body {
  background-color: black;
  color: white;
  margin: 0;
  padding: 0;
}
```

Сохраним его под именем `index.css` .

Далее установим плагин rollup-plugin-styles с помощью команды:

```
npm install -D rollup-plugin-styles
```

Затем добавим информацию о плагине в конфиг Rollup `rollup.config.js` . Делаем это похожим образом, как мы делали это в случае с Babel. В начало файла добавим инструкцию `import` , а в секцию `plugins` — вызов функции `styles()` , как сказано в документации к плагину:

```
import { babel } from '@rollup/plugin-babel';
import styles from "rollup-plugin-styles";

export default {
  input: './index.js',
  output: {
    file: './build/bundle.js',
    format: 'cjs'
  },
  plugins: [
    babel({
      babelHelpers: "bundled",
      presets: ["@babel/env"]
    }),
    styles()
  ]
};
```

После добавления плагина мы можем импортировать CSS-файлы в коде. То есть вместо добавления ссылки в HTML-документе, как мы делали ранее, в начале скрипта `index.js` добавим инструкцию `import` и название CSS-файла:

```
import "./index.css";
```

```
function hello() {
  console.log("Hello world!");
}

hello();
```

Запустим сборку с помощью команды `rollup -c` в терминале, откроем страницу в браузере и убедимся, что стили применяются.

Плагин для работы с картинками

Добавим следующий плагин, который будет включать в сборку картинки.

Плагин называется @rollup/plugin-image и устанавливается аналогичным образом, как и предыдущие плагины — с помощью команды:

```
npm install @rollup/plugin-image --save-dev
```

Далее обновим конфиг `rollup.config.js` в соответствии с документацией плагина, добавим в него:

- `import image from @rollup/plugin-image;` в список импортируемых модулей
- `image()` в секцию `plugins`

Для тестирования возьмём любую картинку и положим её в папку `assets` нашего проекта. Далее в `index.js` напишем код добавления картинки на страницу. Он включает инструкцию `import` с путём до нашей картинки, а также добавление элемента `img` на страницу:

```
import "./index.css";
import MY_IMAGE from './assets/image.png';

// Остальной код

const img = document.createElement("img");
img.src = MY_IMAGE;
document.body.append(img);
```

Не забудь поменять путь к картинке на свой.

Попробуем пересобрать проект с помощью `rollup -c` . Если сборка завершилась успешно, можно перезагрузить страницу и проверить, что картинка появилась на странице.

Если после сборки открыть выходной файл `build/bundle.js` , то можно обратить внимание, что наша картинка представлена в коде в виде base64-строки. Такой подход хорошо работает для маленьких картинок, но для больших лучше использовать обычный подход с указанием ссылки на картинку (без импорта).

Локальный сервер

Добавим ещё один плагин, который позволит при каждой сборке открывать нашу веб-страницу на локальном веб-сервере.

Плагин называется rollup-plugin-serve, установим его:

```
npm install -D rollup-plugin-serve
```

Обновим конфиг `rollup.config.js` , добавив в него:

- `import serve from 'rollup-plugin-serve'` в списке импортов
- `serve({ open: true })` в секцию `plugins`
  - Функция `serve()` принимает объект с параметрами, которые описаны в документации. Пока оставим все параметры по умолчанию, изменим только параметр `open` на `true` , чтобы при сборке наша страницу сразу открывалась в браузере.

Запустим сборку проекта с помощью команды `rollup -c` и убедимся, что наша страница открывается в браузере по адресу `http://localhost:10001/` (если мы не указали другой порт в настройках плагина).

Автоматическое применение изменений

Сейчас при любом изменении нам нужно каждый раз вызывать команду `rollup -c` для пересборки проекта, что не совсем удобно. Добавим плагин, который будет автоматически пересобирать проект и обновлять страницу в браузере.

Установим плагин rollup-plugin-livereload с помощью команды:

```
npm install -D rollup-plugin-livereload
```

Обновим конфиг `rollup.config.js` , добавив в него:

- `import livereload from 'rollup-plugin-livereload'` в списке импортов
- `livereload()` в секцию `plugins`

Для отслеживания изменений в исходном коде запустим сборку с дополнительным флагом `-w` (watch). При необходимости перед этим отключим локальный сервер с помощью сочетания клавиш `Ctrl+C` , чтобы иметь возможность ввести команду в терминале:

```
rollup -c -w
```

Внесём в любой из файлов проекта изменения и сохраним их. Затем проверим, что изменения появляются в браузере без перезагрузки страницы.