|  |  |
| --- | --- |
| Bundle | Một tập hợp code đã được xử lý (biên dịch, chuyển đổi), được đóng gói gọn lại thành một định dạng độc lập bởi một trình Bundler (ví dụ: Webpack, Rollup, v.v.). Nó bao gồm không chỉ code nguồn đã xử lý mà còn cả tất cả các phụ thuộc cần thiết và thường là các tài nguyên tĩnh. |
| Main bundle | Một dạng đặc biệt của Bộ thư viện (Bundle), có thể tự thực thi (độc lập). Nghĩa là việc chạy Bộ thư viện chính bên trong Máy ảo JavaScript sẽ khởi chạy ứng dụng của bạn. |
| Chunk | Một phiên bản nhẹ hơn của Bộ thư viện (Bundle), được thiết kế để nạp và sử dụng cùng với Bộ thư viện chính. Một khối (chunk) thường bị trì hoãn và được nạp theo nhu cầu bởi Bộ thư viện chính hoặc một khối khác. Các khối thường có thể chia sẻ và tái sử dụng ngầm các phụ thuộc với nhau và với Bộ thư viện chính. |
| Async chunk | Một dạng đặc biệt của Khối (Chunk), được nạp theo phương thức không đồng bộ. Trong ngữ cảnh của React Native, tất cả các Khối đều được nạp theo phương thức không đồng bộ, do đó Khối (Chunk) và Khối không đồng bộ (Async Chunk) có thể được sử dụng thay thế cho nhau. |
| Local chunk | Một Khối được lưu trữ cục bộ trên hệ thống tập tin (của thiết bị di động), đối lập với Khối từ xa (Remote Chunk). |
| Remote chunk | Một Khối được lưu trữ từ xa trên server, CDN hoặc bất kỳ vị trí mạng nào khác, đối lập với Khối cục bộ (Local Chunk). |
| Script | File tùy ý chứa mã có thể thực thi. Nó có thể là một Bộ thư viện (Bundle) được tạo bởi một Trình Bundler (ví dụ: Webpack, Rollup, v.v.) hoặc được tạo thủ công bằng tay. |
| Container | Một dạng đặc biệt của Bộ thư viện (Bundle) được tạo trong thiết lập Module Federation và được sử dụng bởi Bộ thư viện chính (Main Bundle). Các Container (Thùng chứa) cũng có thể sử dụng các Container khác và Khối (Chunk). |

Triển khai microfrontend với Webpack

1. Một số khái niệm (concepts):
   * **Entry**

* Một entry point xác định cái mà module webpack sẽ sử dụng ngay khi bắt đầu. Webpack sẽ là thứ mà sẽ phải xác định, định nghĩa nhưng modules hay thư viện mà cái entry point đó phụ thuộc( trực tiếp haowjc gián tiếp)
* Mặc định thì giá trị của nó sẽ ở ./src/index.js, và có thể được định nghĩa ở một vị trí bất kì nào mà bạn muốn với việc gán giá trị entry property trong webpackconfiguration

A black and white rectangle

Description automatically generated

* + **Output**
* Thuộc tính output cho Webpack biết nơi để xuất ra các bundle mà nó tạo ra và cách đặt tên cho các tệp này. Mặc định, tệp đầu ra chính là ./dist/main.js và thư mục cho các tệp được tạo khác là ./dist.
* A computer screen shot of a computer code

  Description automatically generated
* Trong ví dụ trên, chúng ta sử dụng các thuộc tính output.filename và output.path để cho Webpack biết tên của bundle và nơi chúng ta muốn nó được xuất ra. Trong trường hợp bạn đang thắc mắc về mô-đun path được nhập ở đầu, đó là một mô-đun cốt lõi của Node.js được sử dụng để thao tác với các đường dẫn tệp.
  + **Loaders**

-Mặc định, Webpack chỉ hiểu các tệp JavaScript và JSON. Loaders cho phép Webpack xử lý các loại tệp khác và chuyển đổi chúng thành các module hợp lệ có thể được ứng dụng của bạn sử dụng và thêm vào đồ thị phụ thuộc.

- loaders có hai thuộc tính trong cấu hình webpack của bạn:

Thuộc tính test xác định tệp hoặc các tệp nào nên được chuyển đổi. Thuộc tính use chỉ định loader nào nên được sử dụng để thực hiện quá trình chuyển đổi.

A computer screen shot of a computer code

Description automatically generated

* + **Plugins**
* Trong khi loaders được sử dụng để chuyển đổi một số loại module nhất định, plugins có thể được tận dụng để thực hiện một loạt các tác vụ rộng hơn như tối ưu hóa bundle, quản lý tài sản và tiêm các biến môi trường.
* Để sử dụng một plugin, cần:
  + require() nó và thêm nó vào mảng plugins.
  + Hầu hết các plugin đều có thể tùy chỉnh thông qua các tùy chọn.
  + Vì có thể sử dụng một plugin nhiều lần trong một cấu hình cho các mục đích khác nhau, bạn cần tạo một instance của nó bằng cách gọi nó với toán tử new.

A computer screen shot of a program code

Description automatically generated

* + **Mode**
* Bằng cách thiết lập tham số mode thành development, production hoặc none, bạn có thể kích hoạt các tối ưu hóa tích hợp sẵn của webpack tương ứng với từng môi trường. Giá trị mặc định là production.

A close-up of a blue rectangle

Description automatically generated

* + **Browser Compatitbility**

-Webpack hỗ trợ tất cả các trình duyệt tuân thủ ES5 (IE8 trở xuống không được hỗ trợ). Webpack cần Promise cho import() và require.ensure(). Nếu bạn muốn hỗ trợ các trình duyệt cũ hơn, bạn sẽ cần tải một polyfill trước khi sử dụng các biểu thức này.

Nguồn và nội dung liên quan

**Module Federation**

Mục đích:

* Multiple separate builds should form a single application. These separate builds act like containers and can expose and consume code among themselves, creating a single, unified application.

Concepts:

+

Chúng ta phân biệt giữa các module cục bộ và từ xa. Module cục bộ là các module thông thường là một phần của bản build hiện tại. Module từ xa là các module không phải là một phần của bản build hiện tại nhưng được tải tại thời điểm chạy từ một container từ xa.

Việc tải module từ xa được coi là một hoạt động bất đồng bộ. Khi sử dụng một module từ xa, các hoạt động bất đồng bộ này sẽ được đặt vào các hoạt động tải chunk tiếp theo nằm giữa module từ xa và điểm nhập chính. Không thể sử dụng một module từ xa mà không có một hoạt động tả chunk.

Một hoạt động tải chunk thường là một cuộc gọi import(), nhưng các cấu trúc cũ như require.ensure hoặc require([...]) cũng được hỗ trợ.

Một container được tạo thông qua một entry container, nó cung cấp quyền truy cập bất đồng bộ đến các module cụ thể. Quyền truy cập được chia thành hai bước:

* Tải module (bất đồng bộ)
* Đánh giá module (đồng bộ)

Bước 1 sẽ được thực hiện trong quá trình tải chunk.

Bước 2 sẽ được thực hiện trong quá trình đánh giá module xen kẽ với các module khác (cục bộ và từ xa). Bằng cách này, thứ tự đánh giá không bị ảnh hưởng bởi việc chuyển đổi một module từ cục bộ sang từ xa hoặc ngược lại.

Có thể lồng các container. Các container có thể sử dụng các module từ các container khác. Tương tự, các phụ thuộc tuần hoàn giữa các container cũng có thể xảy ra.

+

Mỗi bản dựng đóng vai trò như một container và cũng sử dụng các bản dựng khác làm container. Bằng cách này, mỗi bản dựng có thể truy cập bất kỳ module được tiếp xúc nào khác bằng cách tải nó từ container của nó.

Các module được chia sẻ là các module có thể được ghi đè và được cung cấp dưới dạng ghi đè cho các container lồng nhau. Chúng thường trỏ đến cùng một module trong mỗi bản dựng, ví dụ: cùng một thư viện

Tùy chọn packageName cho phép đặt tên gói để tìm kiếm một requiredVersion. Nó được suy luận tự động cho các yêu cầu module theo mặc định, đặt requiredVersion thành false khi suy luận tự động nên bị vô hiệu hóa.