**I. Getting Started**

**1. Tạo player**

const player = videojs("player-id");

\*return: object player

**2. Xoá player**

player.dispose()

\* Lưu ý: Không được dùng cách xoá dom thông thường với videojs element vì nó không xoá bộ nhớ cũng như event của player. Khi muốn xoá, hãy dùng dispose()

**3. Xử lý khi player sẵn sàng play**

player.on("ready", () => {

console.log("Hahaha");

});

hoặc

player.ready(() => {

console.log("hihi");

})

**4. Kiểm tra xem player có tồn tại không?**

player.isDisposed()

\* return: boolean

**5. Những lỗi sẽ xảy ra nếu 1 Player bị xoá bằng cách xoá phần tử dom thông thường:**

5.1. TypeError: this.el\_.vjs\_getProperty is not a function

5.2. TypeError: Cannot read property 'vdata1234567890' of null

**6. Hiển thị và ẩn 1 player**

Không nên ẩn player và giữ nó chạy ẩn trong nền khi ta thêm 1 overlay hay modal vào video.

videojs khuyến nghị rằng ta nên tạo player khi modal mở và dispose nó khi modal đóng. Điều này sẽ làm giảm việc tiêu tốn tài nguyên không cần thiết, giúp các thiết bị yếu chạy mượt hơn. Tuỳ thuộc vào các lib/framework đang được sử dụng, nó sẽ trông giống như thế này:

modal.on('show', function() {

var videoEl = modal.findEl('video');

modal.player = videojs(videoEl);

});

modal.on('hide', function() {

modal.player.dispose();

});

**7. Thay đổi volume của 1 player**

\* Thay đổi volume trong khoảng từ 0-1

player.volume(0.5)

\* Tắt tiếng

player.muted(true)

**8. Làm player fullScreen**

\* Bật fullScreen:

player.isFullscreen(true)

hoặc:

player.requestFullscreen()

\* Tắt fullScreen:

player.isFullscreen(false)

hoặc:

player.exitFullscreen()

**9. Sử dụng các chức năng của playback**

- play: Bắt đầu phát lại

player.play()

- pause: Tạm dừng phát lại

player.pause()

- paused: Kiểm tra xem player có đang bị tạm dừng không

player.paused()

- currentTime

\*get: Lấy currentTime player đang phát

player.currentTime()

\*set: Set currentTime cho player

player.currentTime(120)

- duration: Return tổng thời lượng của video

player.duration()

- ramainingTime: Return thời lượng còn lại (Thời lượng chưa phát) của video

player.remainingTime();

- buffered: Trả về 1 timeRange object thể hiện đoạn đã buffer

player.buffered()

- bufferedPercent: Trả về % video đã buffer

player.bufferedPercent()

**10. Sử dụng source hay poster trên player**

**10.1. Set source**

\* Source đơn:

myPlayer.src('http://www.example.com/path/to/video.mp4');

\* Source Object:

myPlayer.src(

{

type: 'video/mp4',

src: 'http://www.example.com/path/to/video.mp4'

}

);

\* Array Of Source Object:

myPlayer.src([

{type: 'video/mp4', src: 'http://www.example.com/path/to/video.mp4'},

{type: 'video/webm', src: 'http://www.example.com/path/to/video.webm'},

{type: 'video/ogg', src: 'http://www.example.com/path/to/video.ogv'}

]);

**10.2. Set poster**

myPlayer.poster('http://example.com/myImage.jpg');

**11. Thay đổi tech sử dụng cho player**

myPlayer.tech(false)

**II. Video.JS Options**

**1. Những options chung của Video Element HTML5**

- autoplay

- controls

- height

- loop

- muted

- poster: Url hình ảnh hiển thị trước khi video bắt đầu phát

- preload: Có 3 mode: 'auto', 'metadata', 'none' để báo trình duyệt có load trước data không. 'auto': trình duyệt tự chọn tuỳ thuộc vào băng thông khả dụng, 'metadata': Chỉ load trước metadata, không load trước data, 'none': Không load trước cái gì cả.

- src

- width

**2. Những options của riêng Video.js**

- aspectRatio (string): Đặt trình phát ở chế độ linh hoạt (fluid) và chỉ định tỉ lệ màn hình ("16: 9" hoặc "4: 3"). Cũng có thể add các class vjs-16-9, vjs-16-9, vjs-4-3 hay vjs-1-1 cho player để chỉ định aspectRatio

- autoSetup (boolean): Set để player có tự động setup media element với data-setup không? (Mặc định sẽ set thành false đêr videojs source được load)

- breakpoints (object): Khi đã setup responsive option, set breakpoints sẽ config cách các class được chuyển đổi để thay đổi kích cỡ player

Theo mặc định, breakpoints sẽ là:

Class Name Width Range

vjs-layout-tiny 0-210

vjs-layout-x-small 211-320

vjs-layout-small 321-425

vjs-layout-medium 426-768

vjs-layout-large 769-1440

vjs-layout-x-large 1441-2560

vjs-layout-huge 2561+

Ví dụ, để set breakpoints ta sẽ làm như sau:

breakpoints: {

tiny: 300,

xsmall: 400,

small: 500,

medium: 600,

large: 700,

xlarge: 800,

huge: 900

}

\* Ghi chú:

+ key của object breakpoints được lấy từ các classname loại bỏ bớt đi phần prefix vjs-layout- và bỏ đi các ký tự -

+ giá trị của breakpoints định nghĩa max width cho 1 dải

+ Không phải là tất cả các key đều phải được định nghĩa. Nếu như ta không ghi đè breakpoints nào thì cái breakpoint đó sẽ nhận giá trị mặc định.

- children(Array | Object): Kế thừa từ Component options. Dùng để xác định để xác định children nào và thứ tự chúng được tạo trên 1 player. Phần component options sẽ trình bày chi tiết hơn.

- fluid(Boolean): Khi fluid là true, Video.js player sẽ có kích cỡ lính hoạt.Nói cách khác, nó sẽ thay đổi tỷ lệ khung hình để vừa với container chứa video element

- inactivityTimeout (number): Video.js có thể xác định người dùng có đang hoạt động hay không chính là dựa vào option này. inactivityTimeout xác định bao nhiêu mili giây sau khi người dùng không tương tác thì trạng thái người dùng sẽ được chuyển thành "không hoạt động".

- language (String): language code sẽ được sử dụng để xác định ngôn ngữ ban đầu được hiển thị cho player. Nhưng đây chỉ là ngôn ngữ hiển thị ban đầu, người dùng có thể thay đổi nó

- languages (Objects): Customize một ngôn ngữ có sẵn trong 1 player.

keys: language code

values: 1 Object có key là English keys và value là translated value

Ví dụ:

languages = {

'es': {

Play: 'Reproducción',

Pause: 'Pausa',

'Current Time': 'Tiempo reproducido',

'Duration': 'Duración total',

'Remaining Time': 'Tiempo restante',

}

}

Đoán thế :)) Chưa thử, nếu sai đừng ngạc nhiên nhá :3

- liveui(boolean, mặc định: false):

Cho phép người dùng sử dụng giao diện dành cho phát trực tiếp thay cho giao diện truyền thống. Bao gồm:

+ 1 progress bar để tua trng live window

+ 1 button có thể được click để tua tới thời điểm đang phát trực tiếp và 1 chấm tròn cho biết bạn có đang xem ở live edge hay không

- liveTracker.trackingThreshold (number, default: 20): Không hiểu, pass

- liveTracker.liveTolerance (number, default: 15): Bỏ qua, làm thì rõ, giờ cũng chưa hiểu

- nativeControlsForTouch: Cũng chưa hiểu, bỏ qua

- normalizeAutoplay: Chỉ định xem setting autoplay: true và <video autoplay> có được xử lý giống như action 'play' không? Tức là nếu giá trị này được set thành true, thì videojs sẽ xoá attribute autoplay khỏi video element và thực hiện play() thủ công bởi người dùng

- notSupportedMessage (string): Cho phép ghi đè các message mặc định được hiển thị khi video.js không thể phát lại 1 media source

- noUITitleAttributes (boolean, default: true): Kiểm soát xem các phần tử giao diện người dùng có thuộc tính title không? Nếu được đặt thành true sẽ ngăn cho title được thêm vào UI element.

- fullscreen(Object, mặc định: {navigationUI: 'hide'}): Hình như là để setup những thành phần nào có thể được hiển thị khi gọi fullscreen

- playbackRates: Một mảng các số lớn hơn 0 (Giá trị nằm trong khoảng từ 0-2) để chỉ định tốc độ phát lại (1 là tốc độ thông thường).

Ví dụ: playbackRates: [0.5, 1, 1.5, 2]

- plugins (Object): Hỗ trợ các plugin được khởi tạo tự động với các custom option ngay sau khi player được khởi tạo - thay vì yêu cầu ta khởi tạo chúng theo cách thủ công

videojs('my-player', {

plugins: {

foo: {bar: true},

boo: {bz: false},

}

});

Nó gần tương đương với việc chúng ta gọi các plugin một cách thủ công:

var player = videojs('my-player');

player.foo({bar: true});

player.boo({bz: false});

- preferFullWindow(boolean, default: true): Setting = true sẽ thay đổi fullscreen behaviour trên các thiết bị không hỗ trợ HTML5 Fullscreen API nhưng hỗ trợ fullscreen trên video element.

Ví dụ: Iphone.

Thay vì đặt video ở chế độ toàn màn hình, player sẽ được kéo dài để lấp đầy cửa sổ trình duyệt.

- responsive(booelan, default: false): Được set thành true sẽ làm player tự customize lại nó dựa trên các breakpoint responsive đã được set ở breakpoint option.

Khi được set thành false, thì các responsive breakpoint sẽ không được sử dụng.

- sources (Array): Có thể được sử dụng thay thế src hay thẻ source. Options source là 1 mảng chứa 1 list các source được thêm vào để playback trên video element. Ví dụ:

videojs('my-player', {

sources: [{

src: '//path/to/video.mp4',

type: 'video/mp4'

}, {

src: '//path/to/video.webm',

type: 'video/webm'

}]

});

Nó tương đương với việc chúng ta dùng thẻ source trong html:

<video ...>

<source src="//path/to/video.mp4" type="video/mp4">

<source src="//path/to/video.webm" type="video/webm">

</video>

- suppressNotSupportedError(Boolean): Nếu được đặt thành true thì lỗi không có nguồn tương thích ( the no compatible source error) sẽ không được trigger ngay lập tức mà thay vào đó nó sẽ được trigger khi người dùng thực hiện thao tác lần đầu với player

- techCanOverridePoster(Boolean): Cho phép tech được sử dụng có thể ghi đè poster của player và tích hợp vào poster life-cycle. Điều này có thể hữu ích khi nhiều tech được sử dụng và mỗi tech đều phải set poster của họ vào một thời điểm nào đó mà player đang phát.

- techOrder(Array, Default: ["html5"])

Xác định thứ tự mà các tech được ưu tiên đăng ký. Theo mặc định, html5 sẽ được ưu tiên đăng ký đầu tiên, các công nghệ khác sẽ được thêm vào sau công nghệ này theo thứ tự được đăng ký ở đây.

- userActions (Object): Object để setup có nhận các thao tác của người dùng lên player không?. Nó bao gồm các setup ở bên dưới:

+ userActions.click (Boolean | Function): Nếu được đặt thành false, khi người dùng click vào player thì player cũng sẽ không pause hay play. Ngoài true hoặc false thì ta cũng có thể setup 1 function, để khi người dùng click vào player thì function này sẽ được gọi.

Ví dụ:

function myClickHandler(event) = {

if (this.isFullscreen()) {

this.exitFullscreen();

} else {

this.requestFullscreen();

}

};

videojs('my-player', {

userActions: {

click: myClickHandler

}

});

+ userActions.doubleClick (Boolean | Function): Nếu được set thành false, nhấp đúp sẽ được tắt. Nếu được set thành true, khi người dùng nhấp đúp vào player thì nó sẽ chuyển sang chế độ toàn màn hình.

+ userActions.hotkeys (Boolean | Function | Object): Kiểm soát cách hotkey hoạt động. Nếu được set thành false hoặc undefined, hotkeys sẽ được disabled. Nếu được set thành true hay 1 object, các hotkey sẽ được bật.

Ví dụ:

var player = videojs('my-player', {

userActions: {

hotkeys: function(event) {

// `this` is the player in this context

// `x` key = pause

if (event.which === 88) {

this.pause();

}

// `y` key = play

if (event.which === 89) {

this.play();

}

}

}

});

Các hotkey mặc định là:

| **Key** | **Action** | **Enabled by** |
| --- | --- | --- |
| f | toggle fullscreen | only enabled if a Fullscreen button is present in the Control Bar |
| m | toggle mute | always enabled, even if no Control Bar is present |
| k | toggle play/pause | always enabled, even if no Control Bar is present |
| Space | toggle play/pause | always enabled, even if no Control Bar is present |

+ userActions.hotkeys.fullscreenKey (function): Ghi đè định nghĩa phím toàn màn hình. Nếu nó được set, function sẽ nhận sự kiện keydown, nếu hàm trả về true, fullscreen action sẽ được thực hiện

var player = videojs('my-player', {

userActions: {

hotkeys: {

muteKey: function(event) {

// disable mute key

},

fullscreenKey: function(event) {

// override fullscreen to trigger when pressing the v key

return (event.which === 86);

}

}

}

});

+ userActions.hotkeys.muteKey (Function): Ghi đè định nghĩa phím tắt tiếng. Nếu điều này được set, hàm sẽ nhận sự kiện keydown. Nếu hàm trả về true, hành động tắt tiếng sẽ được thực hiện.

+ userActions.hotkeys.playPauseKey (Function): Ghi đè định nghĩa phím pause/play. Nếu được set, hàm sẽ nhận sự kiện keydown. Nếu hàm trả về true, pau/play action sẽ được thực hiện.

- vtt.js(String): Cho phép ghi đè URL mặc định thành vtt.js

**3. Component Options**

Video.js là 1 Component. Giống như tất cả các component khác, ta có thể xác định nó gồm những thành phần con nào, thứ tự xuất hiện của chúng và những option nào được ưu tiên cho chúng.

Điều này sẽ được đề cập kỹ hơn trong component guide.

- children (Array | Object):

Nếu nó là 1 Array (Mặc định là array), nó sẽ được sử dụng để xác định các thành phần con của videojs và thứ tự nó được tạo trong 1 player.

Ví dụ:

videojs('my-player', {

children: [

'bigPlayButton',

'controlBar'

]

});

Nếu nó là 1 Object, ta có thể làm được nhiều thứ hơn với nó, có thể set option cho các thành phần con luôn. Ví dụ:

videojs('my-player', {

children: {

controlBar: {

fullscreenToggle: false

}

}

});

- ${componentName} (Object): Các component có thể được cung cấp các option tuỳ chỉnh cho nó thông qua tên của nó. Ví dụ:

videojs('my-player', {

controlBar: {

fullscreenToggle: false

}

});

Chưa hiểu lắm thì đến phần Component sẽ hiểu hơn

**4. Tech Options**

- ${techName} (Object): Các Video.js playback tech có thể được cung cấp các custom option.

- html5

**III. SKIN**

**1. Skin mặc định**

Khi ta include file Video.js CSS (video-js.css), Video.js skin mặc định sẽ được sử dụng. Điều đó có nghĩa là tuỳ chỉnh giao diện player Video.js là chỉnh sửa css

**2. Thêm các thẻ <style>**

Ngoài tệp Video.js CSS, có một số style được tạo tự động bởi JS và được include vào <head> dươis dạng các thẻ <style>

- Thẻ "vjs-styles-defaults" đặt kích thước mặc định cho tất cả các Video.js player trên trang

- Thẻ "vjs-styles-dimensions" được tạo cho mỗi player trên trang và được sử dụng để điều chỉnh kích thước của nó.

**3. Disable các thẻ style được thêm vào.**

Đối với các web app sử dụng framework, ta không nên sử dụng các thẻ style được thêm vào. Ta có thể disable các thẻ style bằng cách:

window.VIDEOJS\_NO\_DYNAMIC\_STYLE = true

**4. Tạo 1 custom class**

Có 2 cách để thêm 1 class vào 1 video player:

- Thêm trực tiếp vào video element thông qua attribute class:

<video class="vjs-matrix video-js">...</video>

- Thêm bằng JS với function addClass():

var player = videojs('my-player');

player.addClass('vjs-matrix');

\* Lưu ý khi tạo custom class: Tiền tố vjs- được quy ước cho tất cả các class có trong một Video.js player

**5. Custom style**

Bước đầu tiên khi ghi đè các style mặc định bằng custom style của chúng ta là xác định selector và các thuộc tính cần ghi đè. Ví dụ: Ta không thích màu sắc mặc định của các control (Màu trắng) và muốn thay đổi chúng thành màu xanh lục sáng (Giả sử #00ff00).

Để làm điều đó. đầu tiên chúng ta phải dùng developer tool của browser để xác định xem selector chúng ta cần override là gì và properties cần override.

Và nhớ là phải thêm selector .vjs-matrix vào trước để chắc chắn là chúng ta chỉ override class của videojs chứ không ghi nhầm thêm vào đâu đó.

Ví dụ:

/\* Change all text and icon colors in the player. \*/

.vjs-matrix.video-js {

*color*: #00ff00;

  }

  /\* Change the border of the big play button. \*/

  .vjs-matrix .vjs-big-play-button {

*border-color*: #00ff00;

  }

  /\* Change the color of various "bars". \*/

  .vjs-matrix .vjs-volume-level,

  .vjs-matrix .vjs-play-progress,

  .vjs-matrix .vjs-slider-bar {

*background*: #00ff00;

  }

Và cuối cùng, chúng ta có thể lưu nó thành 1 file css và include nó vào sau Video.js CSS:

<link rel="stylesheet" type="text/css" href="path/to/video-js.min.css">

 <link rel="stylesheet" type="text/css" href="path/to/videojs-matrix.css">

**IV. Video.js Plugin**

Một trong những điểm mạnh nhất của Video.js là hệ sinh thái các plugin cho phép các tác giả từ khắp nơi trên thế giới chia sẻ các video player customization của họ.

Plugin bao gồm mọi thứ, từ những chỉnh sửa giao diện người dùng đơn giản tới các công nghệ playback mới và trình xử lý source.

**1. Viết 1 Plugin cơ bản**

// Tạo 1 basic plugin

*function* examplePlugin(){

            console.log("hihihi")

            this.on("playing", () *=>* {

                videojs.log("Hahahaa");

            })

        }

        videojs.registerPlugin("examplePlugin", examplePlugin);

// Muốn gọi hàm examplePlugin thì ta phải register nó trước. Nếu không register thì nó sẽ báo lỗi ngay.

// Register thực chất là khai báo 1 function examplePlugin cho object player video.js

        player.examplePlugin();

**2. Viết 1 plugin nâng cao**

*const* Plugin = videojs.getPlugin('plugin');

*class* ExamplePlugin extends Plugin {

*constructor*(*player*, *options*){

                super(*player*, *options*);

                if(*options* && *options*.customClass) {

                    // Xem xem options trong Plugin có customClass nào không, nếu có thì add vào player

*player*.addClass(*options*.customClass);

                }

*player*.on('playing', () *=>* {

                    videojs.log('playback began!');

                })

            }

        }

        videojs.registerPlugin("examplePlugin", ExamplePlugin);

        player.examplePlugin();

**3. Sự giống và khác giữa plugin cơ bản và plugin nâng cao**

- this:

+ Với plugin cơ bản: this đại diện cho bản thân player

+ Với plugin nâng cao: this đại diện cho bản thân class, player sẽ được kế thừa từ class Plugin

- Cả plugin cơ bản và plugin nâng cao đều được thiết lập bằng cách gọi tên của plugin.

- Plugin nâng cao có nhiều tính năng hơn plugin cơ bản nhiều. Sẽ được đề cập ở phần tới.

**4. Tính năng của plugin nâng cao**

Plugin nâng cao trên thực tế có nhiều chức năng hơn plugin cơ bản nhiều.

**4.1. Events**

Giống như các component, các plugin nâng cao cung cấp cho chúng ta khả năng thực thi các sự kiện. Bao gồm:

4.1.1. "on" và "one": Lắng nghe các event trên instance plugin 1 lần hoặc nhiều lần.

Ví dụ:

        player.examplePlugin().on('example-event', *function*() {

        videojs.log('example plugin received an example-event');

        });

4.1.2. trigger: Thực thi các custom event của chúng ta

player.examplePlugin().trigger('example-event');

4.1.3. off: Huỷ bỏ việc lắng nghe các custom event chúng ta đã tạo bằng on(hay one)

player.examplePlugin().off('example-event');

**4.2. Data event**

Khi 1 event kích hoạt thì sẽ trả về 1 function có 2 đối số:

- Đối số thứ 1 là event được kích hoạt: tên event,…

- Đối số thứ 2 là thông tin về plugin kích hoạt event: Tên plugin, object instance plugin,….

        player.examplePlugin().on('example-event', *function*(*event*, *plugin*) {

            videojs.log(*event*);

            videojs.log(*plugin*);

        });

**4.3. State**

Trong plugin nâng cao cũng có State giống như trong React Component

Mỗi plugin instance đều có chứa 1 thuộc tính state, nó là 1 object Javascript thuần - Có thể chứa bất kỳ cặp key - value nào mà ta muốn.

- 1 default state có thể được khai báo bằng cách thêm vào plugin constructor:

this.setState({customClass: 'custom-class'})

hoặc gọi bên ngoài nó:

        ExamplePlugin.defaultState = {

          customClass: 'default-custom-class'

        };

Khi state được update thông qua method setState, plugin sẽ thực thi event "statechanged" (Chỉ khi có sự thay đổi về dữ liệu, còn nếu giá trị được setState giống với giá trị ban đầu thì "statechanged" cũng sẽ không thực thi đâu.

**4.4. Lifecycle**

// Tạo 1 plugin instance

player.examplePlugin();

// Phá huỷ nó bằng cách sử dụng dispose

player.examplePlugin().dispose()

dispose() sẽ thực thi 1 số công việc sau:

- trigger 1 event "dispose" trên plugin instance

- Dọn dẹp toàn bộ các plugin instance, điều này sẽ giúp chúng ta tránh đc các lỗi do các sự kiện được kích hoạt sau khi instance plugin đã bị clean.

- Xoá các plugin state và tham chiếu tới player để tránh lỗi rò rỉ bộ nhớ (memory leaks)

- Khôi phục tên của player về ban đầu (ví dụ: player.examplePlugin), do đó, plugin có thể được thiết lập lại.

Bổ sung thêm là nếu player bị dispose thì tất cả các plugin nâng cao được khai báo cho player đó cũng sẽ bị dispose

**4.5. Version**

\*Thêm số phiên bản vào plugin (Lưu ý là phải thêm version trước khi nó được register thì nó mới nhận):

        ExamplePlugin.VERSION = '1.0.1';

        videojs.registerPlugin('examplePlugin', ExamplePlugin);

\* Lấy số phiên bản của plugin:

*var* version = videojs.getPluginVersion('examplePlugin');

**4.6. Logging**

Theo mặc định, mỗi instance plugin đều có 1 thuộc tính log riêng cho nó (Giống như đối với instance của player và videojs). Các log messages sẽ có tiền tố là: Player's ID và tên của plugin:

player.examplePlugin().log('hello world!');

Log ở trên sẽ console ra như sau:

VIDEOJS: $PLAYER\_ID: examplePlugin: hello world!

Ngoài log, ta cũng có thể sử dụng những function khác như:

error(), warn(), level(),…

**4.7. Ví dụ:**

import videojs from 'video.js';

*const* Plugin = videojs.getPlugin('plugin');

*class* Advanced extends Plugin {

*constructor*(*player*, *options*) {

            super(*player*, *options*);

            // Whenever the player emits a playing or pause event, we update the

            // state if necessary.

            this.on(*player*, ['playing', 'pause'], this.updateState);

            this.on('statechanged', this.logState);

        }

        dispose() {

            super.dispose();

            videojs.log('the advanced plugin is being disposed');

        }

        updateState() {

            this.setState({playing: !this.player.paused()});

        }

        logState(*changed*) {

            videojs.log(`the player is now ${this.state.playing ? 'playing' : 'paused'}`);

        }

        }

        videojs.registerPlugin('advanced', Advanced);

*const* player = videojs('example-player');

        player.advanced();

        // This will begin playback, which will trigger a "playing" event, which will

        // update the state of the plugin, which will cause a message to be logged.

        player.play();

        // This will pause playback, which will trigger a "paused" event, which will

        // update the state of the plugin, which will cause a message to be logged.

        player.pause();

        player.advanced().dispose();

        // This will begin playback, but the plugin has been disposed, so it will not

        // log any messages.

        player.play();

**4.8. Thiết lập 1 plugin**

Có 2 cách để thiết lập hoặc khởi tạo 1 plugin trên player.

\* Cách 1:

Khi khởi tạo player thì khai báo luôn plugin sẽ sử dụng:

player sẽ tự biết mà register Plugin

        videojs('example-player', {

            plugins: {

                examplePlugin: {

                customClass: 'example-class'

                }

            }

        });

\* Cách 2: Setup thủ công

*var* player = videojs('example-player');

videojs.registerPlugin("examplePlugin", ExamplePlugin);

player.examplePlugin({customClass: 'example-class'});

**4.9. Plugin Setup Events**

Đôi khi, 1 vài trường hợp chúng ta cần phải đợi một plugin được khởi tạo. Từ Video.js 6 trở đi, chúng ta có thể thực hiện điều này bằng cách lắng nghe các sự kiện Plugin Setup.

Đối với bất kỳ khởi tạo plugin nào, có 4 sự kiện cần lưu ý:

- beforepluginsetup: Được kích hoạt ngay lập tức trước khi bất kỳ plugin nào được khởi chạy.

- beforepluginsetup:examplePlugin: Được kích hoạt ngay trước khi examplePlugin được khởi tạo

- pluginsetup: Được kích hoạt sau khi bất kì plugin nào được khởi tạo

- pluginsetup:examplePlugin: Được kích hoạt sau khi examplePlugin được khởi tạo.

**V. Components**

Kiến trúc của Video.js player được xây dựng xung quanh các component. Class Player và tất cả các class đại diện cho player controls và UI Element đều được kế thừa từ class Component. Kiến trúc này giúp dễ dàng xây dựng UI của Video Player theo kiến trúc cây tương tự DOM.

1. Component là gì?

1 component là 1 javascript Object có các tính năng sau:

- 1 phần tử DOM được liên kết

- 1 liên kết tới object Player

- Khả năng quản lý bất kỳ thành phần con nào

- Khả năng lắng nghe và trigger các event

- 1 vòng đời của khởi tạo và dispose

2. Tạo 1 component

Các video.js component có thể được kế thừa và đăng ký với Video.js để thêm các tính năng và UI cho player.

Các method được dùng để tạo component:

videojs.getComponent(String name): Truy xuất 1 component constructor có sẵn trong Video.js

videojs.registerComponent(String name, Function Comp): Đăng ký mới 1 component constructor với Video.js

videojs.extend(Function component, Object properties): Kế thừa và mở rộng component constructor đã có, trả về 1 constructor mới với các thuộc tính đã cho.

Ví dụ:

*var* Button = videojs.getComponent("Button");

// Lấy ra constructor button từ videojs

*var* button = new Button(player, {

    clickHandler: *function*(*event*) {

    videojs.log("Clicked");

    }

});

player.addChild('button');

3. Component con

3.1. Ví dụ đơn giản

Khi child component được thêm vào parent component. Video.js sẽ chèn element của child vào trong element của parent. Ví dụ, thêm 1 component như sau:

player.addChild('BigPlayButton');

Thì DOM nó sẽ như thế này:

        <div class="video-js">

            <div class="vjs-big-play-button"></div>

        </div>

Ngược lại, khi xoá child component, video.js cũng sẽ xoá child component element khỏi DOM:

player.removeChild('BigPlayButton');

Thì DOM sẽ ntn:

        <div class="video-js">

        </div>

3.2. Đưa vào các options cho child constructor và các children của child

*var* player = videojs('some-vid-id');

*var* Component = videojs.getComponent('Component');

*var* myComponent = new Component(player);

*var* myButton = myComponent.addChild('MyButton', {

            text: 'Press Me',

            buttonChildExample: {

                buttonChildOption: true

            }

        });

Thậm chí, các children cũng có thể được thêm vào khi 1 component khởi tạo

*var* myComp = new MyComponent(player, {

  children: ['button', {

    name: 'button',

    someOtherOption: true

  }, {

    name: 'button',

    someOtherOption: false

  }]

});

4. Event Listening

4.1. on

*var* player = videojs('some-player-id');

*var* Component = videojs.getComponent('Component');

*var* myComponent = new Component(player);

*var* myFunc = *function*() {

*var* myComponent = this;

  console.log('myFunc called');

};

myComponent.on('eventType', myFunc);

myComponent.trigger('eventType');

Ngữ cảnh của myFunc sẽ là myComoponent trừ khi nó bị ràng buộc, ta có thể thêm listener vào 1 element hay component khác.

*var* otherComponent = new Component(player);

// myComponent/myFunc is from the above example

myComponent.on(otherComponent.el(), 'eventName', myFunc);

myComponent.on(otherComponent, 'eventName', myFunc);

otherComponent.trigger('eventName');

4.2. off

*var* player = videojs("some-player-id");

*var* Component = videojs.getComponent("Component");

*var* myComponent = new Component(player);

*var* myFunc = *function* () {

*var* myComponent = this;

        console.log("myFunc called");

      };

      myComponent.on("eventType", myFunc);

      myComponent.trigger("eventType");

      // logs 'myFunc called'

      myComponent.off("eventType", myFunc);

      myComponent.trigger("eventType");

      // does nothing

Nếu ta bỏ trống tên event: "eventType" thì tất cả các listener đều sẽ bị xoá khỏi component.

Nếu ta bỏ trống myFunc, thì tất cả các listeners được gán cho event "eventType" sẽ bị xoá khỏi component

Hoặc ta cũng có thể sử dụng off để xoá các listener được thêm vào element hoặc component khác

myComponent.off(otherComponent, 'eventType', myFunc);

4.3. one

Tương tự

4.4. trigger

Tương tự

5. Component tree mặc định

├── MediaLoader (has no DOM element)

├── PosterImage

├── TextTrackDisplay

├── LoadingSpinner

├── BigPlayButton

├── LiveTracker (has no DOM element)

├─┬ ControlBar

│ ├── PlayToggle

│ ├── VolumePanel

│ ├── CurrentTimeDisplay (hidden by default)

│ ├── TimeDivider (hidden by default)

│ ├── DurationDisplay (hidden by default)

│ ├─┬ ProgressControl (hidden during live playback, except when liveui: true)

│ │ └─┬ SeekBar

│ │   ├── LoadProgressBar

│ │   ├── MouseTimeDisplay

│ │   └── PlayProgressBar

│ ├── LiveDisplay (hidden during VOD playback)

│ ├── SeekToLive (hidden during VOD playback)

│ ├── RemainingTimeDisplay

│ ├── CustomControlSpacer (has no UI)

│ ├── PlaybackRateMenuButton (hidden, unless playback tech supports rate changes)

│ ├── ChaptersButton (hidden, unless there are relevant tracks)

│ ├── DescriptionsButton (hidden, unless there are relevant tracks)

│ ├── SubtitlesButton (hidden, unless there are relevant tracks)

│ ├── CaptionsButton (hidden, unless there are relevant tracks)

│ ├── SubsCapsButton (hidden, unless there are relevant tracks)

│ ├── AudioTrackButton (hidden, unless there are relevant tracks)

│ ├── PictureInPictureToggle

│ └── FullscreenToggle

├── ErrorDisplay (hidden, until there is an error)

├── TextTrackSettings

└── ResizeManager (hidden)

6. Chi tiết cụ thể về các component

6.1. Play Toggle

Play Toggle có 1 option tên là "replay" có thể setup hiện hoặc ẩn icon "replay". Nếu muốn ẩn icon replay: {replay: false}. Nếu muốn hiện: {replay: true}

*let* player = videojs('myplayer', {

        controlBar: {

            playToggle: {

            replay: false

            }

        }

    });

6.2. Volume Panel

Volume Panel bao gồm MuteToggle và VolumeControl Components. Chúng ta có 1 option quan trọng cho VolumePanel có thể khiến VolumeControl xuất hiện theo chiều dọc. Điều này có thể được thực hiện bằng cách set VolumePanel {inline: false}. Nếu set inline: true thì VolumeControl sẽ xuất hiện theo chiều ngang

Ví dụ về Volume Control theo chiều dọc:

*var* player = videojs("#my-player", {

        controls: true,

        autoplay: true,

        preload: "auto",

        controlBar: {

            volumePanel: {

            inline: true

            }

        }

      });

6.3. Quản lý resize

Thành phần mới này chịu trách nhiệm kích hoạt 1 playerresize event khi kích cỡ trình phát video thay đổi.

7. Ngôn ngữ

Video.js bao gồm các bản dịch hỗ trợ nhiều ngôn ngữ khác nhau.

7.1. Sử dụng Video.js languages

Video.js cung cấp nhiều bản dịch trong các file Javascript được lưu trữ "dist/lang/". Ta có thể thêm tập lang script cho ngôn ngữ mà ta muốn hỗ trợ. Mỗi file language này có thể thêm vào webpage để hỗ trợ cho tất cả các Video.js player.

<script src="//example.com/path/to/video.min.js"></script>

<script src="//example.com/path/to/lang/es.js"></script>

Ngoài việc chọn ngôn ngữ hiển thị cho Video.js bằng cách thêm script lang thì ta cũng có thể chọn các ngôn ngữ hiển thị cho từng player riêng biệt bằng cách sử dụng option languages:

    videojs('my-player', {

        language: 'es'

    });

7.2. Thay đổi file languages

Ngoài sử dụng bản dịch có sẵn, chúng ta cũng có thể tự tạo các bản dịch của riêng mình bằng cách chỉnh sửa file language hoặc chỉnh sửa thủ công cho từng player

videojs("my-player", {

        languages: {

          es: {

            Play: "Reproducir",

          },

        },

      });

7.3. Xác định ngôn ngữ mà player sẽ hiển thị

Điều gì sẽ xảy ra nếu chúng ta cùng sử dụng nhiều cách khác nhau để set language cho player? Và những language đó là khác nhau?

Thứ tự ưu tiên của nó như sau:

- language được chỉ định bởi options

- language được chỉ định bởi attribute "lang" trong player element

- language được chỉ định attribute "lang" của parent element gần nhất, tối đa đến <html> element

- Tuỳ chọn ngôn ngữ của trình duyệt người dùng (Sẽ chọn ngôn ngữ đầu tiên trong danh sách nếu có nhiều ngôn ngữ)

- Nếu không có option nào thì nó sẽ chọn English theo mặc định.

7.4. Mã ngôn ngữ

- Mã ngôn ngữ không phân biệt chữ hoa và chữ thường (Ví dụ: en-US == en-us)

- Khi khai báo thì ta sẽ thường khai báo 2 mã ngôn ngữ (Mã ngôn ngữ chính và mã ngôn ngữ phụ). Mã ngôn ngữ phụ thường chi tiết hơn và sẽ được kiểm tra trước.

Ví dụ như en-us, en-uk. Cùng tiếng anh nhưng còn anh - mỹ và anh - anh, nó khác nhau.

- Nếu không có mã ngôn ngữ khớp với mã phụ thì mã chính sẽ được sử dụng.

8. Hook

Hook tồn tại giúp chúng ta tham gia vào các khoảnh khắc nhất định trong vòng đời của Video.js

8.1. Các Hook hiện có

\* beforesetup: Xảy ra ngay trước khi 1 player được tạo. Điều này cho phép:

- Sửa đổi các option được chuyển đến Video.js function (Ví dụ: videojs("some-id", options)

- Sửa đổi phần tử video DOM sẽ được sử dụng cho player sẽ được tạo

Các function beforesetup hook khi thiết lập cần phải:

- Nhận 2 đối số:

+ videoEl: Phần tử DOM mà Video.js sẽ sử dụng để tạo trình phát.

+ options:Object options mà Video.js được gọi và sẽ được chuyển đến player trong suốt quá trình tạo

- Trả về 1 object (Object này sẽ được merge với các options ban đầu nhận được).

Ví dụ:

videojs.hook("beforesetup", *function* (*videoEl*, *options*) {

        // videoEl will be the video element with id="some-id" since that

        // gets passed to videojs() below. On subsequent calls, it will be

        // different.

*videoEl*.className += " some-super-class";

        // autoplay will be true here, since we passed it as such.

        if (*options*.autoplay) {

*options*.autoplay = false;

        }

        // Options that are returned here will be merged with old options.

        //

        // In this example options will now be:

        //   {autoplay: false, controls: true}

        //

        // This has the practical effect of always disabling autoplay no matter

        // what options are passed to videojs().

        return *options*;

      });

      // Create a new player.

      videojs("some-id", { autoplay: true, controls: true });

\* setup: Xảy ra ngay sau khi 1 player được tạo. Điều này cho phép:

- Chạy các plugin hoặc các custom function trên player

- Thực hiện thay đổi với object player

Các function setup hook:

- Nhận vào 1 đối số:

+ player: player mà Video.js tạo

- Không cần return về cái gì.

Ví dụ:

videojs.registerPlugin("foo", *function* () {

        // This basic plugin will add the "some-super-class" class to a player.

        this.addClass("some-super-class");

      });

      videojs.hook("setup", *function* (*player*) {

        // Initialize the foo plugin after any player is created.

*player*.foo();

      });

      // Create a new player.

      videojs("some-id", { autoplay: true, controls: true });

\* beforeerror: Xảy ra khi chúng ta gặp lỗi. Điều này cho phép các plugin hoặc custom function ngăn chặn lỗi hoặc sửa đổi nó thành một thứ gì đó khác.

Hay nói cách khác là xoá cái lỗi đáng ra sẽ được hiển thị ra màn hình hoặc làm điều gì đó để khắc phục lỗi thay vì để nó tác oai tác quái.

Các beforerror hook function:

- Nhận vào 2 đối số:

+ player xảy ra lỗi

+ error object đã được truyền vào

- Return: 1 error object thay thế error nhận được

videojs.hook("beforeerror", *function* (*player*, *err*) {

*const* error = *player*.error();

        // prevent current error from being cleared out

        if (*err* === null) {

          return error;

        }

        // but allow changing to a new error

        return *err*;

      });

\* error: Xảy ra sau khi player đã đẩy lỗi ra ngoài (Console.log hoặc cái gì đó ra trình duyệt). Hook này nó cho phép ta nhận được thông tin về error từ tất cả các player (Tất nhiên là nếu ta sử dụng cả beforeerror, thì cái error này sẽ là error đã bị sửa đổi bởi beforeerror

Các error hook function:

- Lấy 2 đối số:

+ player: Player xảy ra lỗi

+ Error: Object Error đã được xử lý trước đó bởi beforeerror hook.

- Return: Không cần return

    videojs.hook('error', *function*(*player*, *err*) {

        console.log(`player ${*player*.id()} has errored out with code ${*err*.code} ${*err*.message}`);

    });

8.2. Cách sử dụng hook

8.2.1. Thêm Hook

Hook có thể được thêm bằng cách gọi hàm videojs.hook(<name>, function> trước khi chạy function videojs()

videojs.hook("beforesetup", *function* (*videoEl*, *options*) {

        // This hook will be called twice. Once for "vid1" and once for "vid2".

        // The options will match what is passed to videojs() for each of them.

      });

      videojs.hook("setup", *function* (*player*) {

        // This hook will be called twice. Once for "vid1" and once for "vid2".

        // The player value will be the player that is created for each element.

      });

      videojs("vid1", { autoplay: false });

      videojs("vid2", { autoplay: true });

Trong trường hợp ta chỉ muốn sử dụng hook 1 lần duy nhất. Ta có thể dùng hookOnce(<name>, function) trước khi gọi hàm videojs()

videojs.hookOnce("beforesetup", *function* (*videoEl*, *options*) {

        // This hook will be called once for "vid1", but not for "vid2".

        // The options will match what is passed to videojs().

      });

      videojs.hookOnce("setup", *function* (*player*) {

        // This hook will be called once for "vid1", but not for "vid2".

        // The player value will be the player that is created for each element.

      });

      videojs("vid1", { autoplay: false });

      videojs("vid2", { autoplay: true });

8.2.2. Lấy danh sách các hook function đang tồn tại

Để lấy mảng các hook function đang tồn tại cho 1 hook nào đó, ta sử dụng hàm videojs.hooks(<hook\_name>)

// Get an array of all the 'beforesetup' hooks.

*var* beforeSetupHooks = videojs.hooks("beforesetup");

      // Get an array of all the 'setup' hooks.

*var* setupHooks = videojs.hooks("setup");

8.2.3. Xoá Hook

Để ngăn 1 hook function được thực thi trong tương lai, chúng ta có thể xoá hook bằng cách sử dụng videojs.removeHook(<hook\_name>, function)

Nếu không chắc về function thì hãy dùng videojs.hooks() ở trên để lấy các function đã khai báo cho hook.

*var* beforeSetup = *function* (*videoEl*, *options*) {};

      // Add the hook.

      videojs.hook("beforesetup", beforeSetup);

      // Remove the same hook.

      videojs.removeHook("beforesetup", beforeSetup);