Streaming videos with PHP

20 de June de 2014 por Felipe Weckx

Videos for the Web

With the recent support for HTML5 in most modern browsers, nearly all of them can understand the *<video>* tag which allows to place a video in a webpage. However, not any video format can be used for there are limitations between browsers and their versions, so before serving a video on the web it has to be converted to a supported format.

Supported Formats

MP4 is the most widely supported video format on the web today, however it is possible to specify several sources in the *<video>* tag which allows the browser to choose format it supports best. According to W3Schools the supported formats in each of the most common browsers are:

Browser	MP4	WebM	Ogg
Internet Explorer	SIM	NÃO	NÃO
Chrome	SIM	SIM	SIM
Firefox	NÃO	SIM	SIM
Safari	SIM	NÃO	NÃO
Opera	NÃO	SIM	SIM

For this example this video in the Quicktime (.mov) format will be used. The FFmpeg, available for Linux, Windows and MAC will be used.

Converting to MP4

To convert the video into the MP4 format use the following command line:

```
ffmpeg -i Rain_Fire.mov -c:v libx264 -pix_fmt yuv420p -profile:v baseline -preset slower -crf 23 -vf "scale=trunc(in_w/2)*2:trunc(in_h/2)*2" -movflags +faststart rain.mp4
```

Some important parameters of the command above:

- *-c:v libx264* – Use the x264 codec, very importante since its the most widely supported
- *-pix_fmt yuv420p* – Pixel format to use
- *-profile:v baseline* – Video profile. The baseline profile makes it easy for Androi devices to support
- – *preset slower* – Determines the size/encoding time ratio. Can be (faster, fast, medium, slow
- *-crf 23* – Quality encoding level. 0 means lossless and 51 the worst quality possible. The default is 23 which is a good value.
- *-vf "scale=trunc(in_w/2)*2:trunc(in_h/2)*2"* – For some reasons MP4 movies using this encoding need even width and height. This parameter adjusts these values.
- *-movflags +faststart* – Places the video information headers on the beggining of the file, this allows the browser to download the video information first. **This parameter is essential to allow the streaming of the video.**

For more information see FFmpeg documentation on H.264 video encoding.

Converting to WebM

Another possibile video format is WebM which uses the VP8 codec , a royalty free video codec purchased by Google. To convert to this format use the following command:

```
ffmpeg -i Rain_Fire.mov -c:v libvpx -c:a libvorbis -pix_fmt yuv420p -b:v 2M -crf 5 rain.webm
```

The parameters are

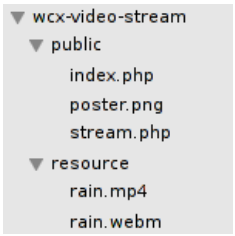
- *-c:v libvpx* – The video codec to use (VP8)

- *-c:a libvorbis* – The audio codec
- *-pix_fmt yuv420p* – pixel format
- *-b:v 2M* – desired bitrate. It is important to set this parameter as the default creates poor quality videos
- *-crf 5* – Video quality. Ranges from 4 – 63. The smaller the better.

More info on the FFmpeg VP8 documentation.

Serving the file

With the correct format files let's create a simple site to serve it. The following directory structure will be used:



Estrutura de diretórios

In the *public* directory are the files that are visible by the webserver and in the *resource* directory the video files which we will serve. The *index.php* file has the HTML and *stream.php* is where we will read the video files using PHP to serve them.

Streaming

Streaming is a data distribution method in which the content is served to the client as needed, in a way that it is not necessary to have the full content before acessing it. That is the best way to serve video files because it allows quick visualization by the user and also lowers bandwidth usage. The web browsers use the HTTP *Range* header to indicate which bytes of the video it wants.

Handling the *Range* header

According to the RFC2616 which specifies the HTTP protocol version 1.1 the Range header specifies the portion of the content to be served on the request and format of the header is one of the following:

- The first 500 bytes: *bytes=0-499*
- The second 500 bytes: *bytes=500-999*
- The last 500 bytes: *bytes=-500*
- All bytes since 9500: *bytes=9500-*

The header value is available on the $_SERVER['HTTP_RANGE'] variable, the code below for the file *stream.php* uses the PHP filesystem functions to read a video file and serve it.

stream.php - Streaming de arquivo de vídeo

```php
//Determine file path according to extension
if (!isset($_GET['ext']) || $_GET['ext'] == 'mp4') {
 $path = dirname(__FILE__) . '/../resource/rain.mp4';
} else if ($_GET['ext'] == 'webm') {
 $path = dirname(__FILE__) . '/../resource/rain.webm';
} else {
 header('HTTP/1.1 400 Bad Request');
 return;
}

// Determine file mimetype
$finfo = new finfo(FILEINFO_MIME);
$mime = $finfo->file($path);

// Set response content-type
header('Content-type: ' . $mime);

// File size
$size = filesize($path);

// Check if we have a Range header
if (isset($_SERVER['HTTP_RANGE'])) {
    // Parse field value
    list($specifier, $value) = explode('=', $_SERVER['HTTP_RANGE']);

    // Can only handle bytes range specifier
    if ($specifier != 'bytes') {
        header('HTTP/1.1 400 Bad Request');
        return;
    }

    // Set start/finish bytes
    list($from, $to) = explode('-', $value);
```

```
    if (!$to) {
        $to = $size - 1;
    }

    // Response header
    header('HTTP/1.1 206 Partial Content');
    header('Accept-Ranges: bytes');

    // Response size
    header('Content-Length: ' . ($to - $from));

    // Range being sent in the response
    header("Content-Range: bytes {$from}-{$to}/{$size}");

    // Open file in binary mode
    $fp = fopen($path, 'rb');
    $chunkSize = 8192; // Read in 8kb blocks

    // Advance to start byte
    fseek($fp, $from);

    // Send the data
    while(true){
        // Check if all bytes have been sent
        if(ftell($fp) >= $to){
            break;
        }

        // Send data
        echo fread($fp, $chunkSize);

        // Flush buffer
        ob_flush();
        flush();
    }
} else {
    // If no Range header specified, send everything
    header('Content-Length: ' . $size);

    // Send file to client
    readfile($path);
}
```

It is **essential** to handle the *Range* header when streaming video files otherwise the forward, reverse and seek player options will not work. Also, without that the video would have to be completely downloaded before watching, which gives a bad user experience.

The web page

Moving on to the web page the file *index.php* will have the HTML with the *<video>* tag and will be the one acessed by the user:

index.php - Página acessada pelo usuário

```php
<?php
// Handle php -S to serve files
if (php_sapi_name() === 'cli-server' && is_file(__DIR__ . parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH))) {
    return false;
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>HTML5 Video + PHP</title>
</head>
<body>
    <video id="video" controls preload="auto" width="640" height="360"
        poster="poster.png">
      <source src="stream.php?ext=webm" type='video/webm' />
      <source src="stream.php?ext=mp4" type='video/mp4' />
</video>
</body>
</html>
```

The file can be tested using a webserver o the PHP built-in webserver executing:

```
php -S localhost:8000 index.php
```

And on the browser access http://localhost:8000

The <video> tag

The *<video>* tag is used to add the video element to the page. Inside the tag there may be any number of *<source>* elements, each one with a video or audio URL. So it is possible, for example, to specify a video and an audio file that will be combined by the browser on playback. In our case we specify two video sources, on for the MP4 file and the other for the WebM file and the browser will decide which one to use. The attributes for the tag are:

- *controls* – Wheter the video controls should be displayed
- *preload* – If the video will be loaded with the webpage or not. The value can be *auto* to load, , *metadata* to load only the metadata or *none* to not load the video util it is clicked (played) by the user.**Internet Explorer does not respect this parameter.**
- *width* e *height* – The video *width* and *height* if it is not the same as the file the video will be resized by the browser
- *poster* – URL for an image that will be displayed in the video container util the video is loaded or played. It is possible to create it using *FFmpeg* as we will see below.
- *autoplay* – if specified the video will start playing as soon as possible

There are several other parameters available.

Creating the poster

The poster image can be created using FFmpeg with the following command:

```
ffmpeg -i Rain_Fire.mov -r 1 -vframes 1 -ss 0:05 poster.png
```

The most important parameter is *-ss 0:05* which specifies the time where the image will be extracted.

Using *video.js*

The page works on any modern browser that supports HTML5. But what about older, but still used, browsers such as Internet Explorer 7 or 8? For those we can use the *video.js* javascript library. It automatically detects if the <video> tag is supported and if it isn't it is replaced with a flash video player automatically. The update *index.php* is shown below:

index.php - Utilizando video.js

```php
<?php
// To handle php -S
if (php_sapi_name() === 'cli-server' && is_file(__DIR__ . parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH))) {
    return false;
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>HTML5 Video + PHP</title>
    <link href="//vjs.zencdn.net/4.6/video-js.css" rel="stylesheet">
    <script src="//vjs.zencdn.net/4.6/video.js"></script>
</head>
<body>
    <video id="video" class="video-js vjs-default-skin" controls preload="auto"
        width="640" height="360"
        poster="poster.png">
      <source src="stream.php?ext=webm" type='video/webm' />
      <source src="stream.php?ext=mp4" type='video/mp4' />
      <p class="vjs-no-js">To view this video please enable JavaScript, and consider upgrading to a web browser that <a hr
</video>
</body>
</html>
```

Conclusion

HTML5's *<video>* tag allows for easy video embedding in web pages, but it is necessary to take care with the video format and make the necessary convertions. It is essential that the files be served via *streaming* to reduce network usage and enhance user experience. PHP allows easy handling of streaming by the use of its file system functions and the Range HTTP header. The code developed here is very basic and can be improved in many ways.

The example code is available at Github: https://github.com/weckx/wcx-video-stream

ffmpeg html5 mp4 php video webm

Deixe um comentário

Translating ZF2 validation messages

21 de April de 2014 por Felipe Weckx

Often we use the InputFilter to validate the elements of a form create with Zend Framework 2. However, when creating a system/website which has to support multiple languages we also need to translate the validation messages. Fortunately ZF2 comes with a series of translations for the default validation classes and we can set up the translation using the \Zend\Validator\AbstractValidator::setDefaultTranslator method:

Bootstrap configuration for translating validation messages

```php
public function onBootstrap(MvcEvent $e)
{
    //Create the translator
```

```
    $translator = new \Zend\Mvc\I18n\Translator(new \Zend\I18n\Translator\Translator());

    //Add the translation file. Here we are using the Portuguese-Brazilian translation
    $translator->addTranslationFile(
        'phpArray',
        __DIR__ . '/../../vendor/zendframework/zendframework/resources/languages/pt_BR/Zend_Validate.php',
        'default',
        'pt_BR'
    );

    //Set the default translator for validators
    \Zend\Validator\AbstractValidator::setDefaultTranslator($translator);
}
```

The code above has to placed inside the *Module* class in the modules you need to translate the messages, or in your application module to enable in all of them. Note that we added it to the *onBootstrap* method, which means it will be executed in **all** actions. That can cause a performance issue, so it is best to separate it and call only on the controllers that actually use validations.

i18n php translation validator zend zf2

1 comentário

PHP: in_array and arrays with a true item

26 de February de 2014 por Felipe Weckx

A very important parameter of the in_array PHP function is the *$strict* parameter. It has the default value of FALSE. But what exactly does it do? Let's look at the following code:

```
$array = array('rice', 'seaweed', 'fish');
var_dump(in_array('slamon', $array));
```

The output will be as expected:

```
bool(false)
```

However, if we add to the array *$array* a new item with the value *true:*

```
$array = array('rice', 'seaweed', 'fish', true);
var_dump(in_array('salmon', $array));
```

Then the output will be:

```
bool(true)
```

Which means that the *in_array* returned **true** even though there's no *salmon* item in $array! That is because by default the *in_array* functio uses PHP's "loose comparison", that is, it uses the "==" operator to check if the value is in the array (more information on http://php.net/manual/en/types.comparisons.php) so any non-empty value that is compared to *true* returns *true*. That can cause a nasty bug that is very hard to find and understand. To solve it we can pass the *$strict* parameter as *true:*

```
$array = array('arroz', 'alga', 'peixe', true);
var_dump(in_array('salmão', $array, true));
```

Now the comparison will behave as expected. That kind of mistake can be easily prevented if the software is developed using TDD with a good test coverage and test cases, but otherwise it can be very tricky to be found.

in_array php

Deixe um comentário
