背景

原文链接：Chrome 自动播放限制策略

Web浏览器正在朝着更严格的自动播放策略发展，以便改善用户体验，最大限度地降低安装广告拦截器的积极性并减少昂贵和/或受限网络上的数据消耗。这些更改旨在为用户提供更大的播放控制权，并使开发商获得合法用例。

新的特性

Chrome的自动播放政策很简单：

- 静音自动播放总是允许的。
- 在下列情况下允许使用声音自动播放：
  - 用户已经与域进行了交互（点击，tap等）。
  - 在桌面上，用户的媒**体参与指数阈值**(MEI)已被越过，这意味着用户以前播放带有声音的视频。
  - 在移动设备上，用户已将该网站添加到主屏幕。
  - 顶部框架可以将自动播放权限授予其iframe以允许自动播放声音。

媒体参与指数（Media Engagement Index)(MEI)

MEI衡量个人在网站上消费媒体的倾向。Chrome 目前的方法是访问每个来源的重要媒体播放事件的比率：

- 媒体消耗（音频/视频）必须大于7秒。
- 音频必须存在并取消静音。
- 视频选项卡处于活动状态。
- 视频大小（以像素为单位）必须大于200x140。

因此，Chrome会计算媒体参与度分数，该分数在定期播放媒体的网站上最高。足够高时，媒体播放只允许在桌面上自动播放。MEI是谷歌自动播放策略的一部分。它是一个算法，参考了媒体内容的持续时间、浏览器标签页是否活动、活动标签页视频的大小这一系列元素。不过也正因此，开发者难以在所有的网页上都测试这一算法的效果。

用户的MEI位于chrome://media-engagement/内部页面

开发者开关

作为开发者，您可能需要在本地更改Chrome浏览器自动播放政策行为，以根据用户的参与情况测试您的网站。

- 您可以决定通过将Chrome标志“自动播放策略”设置为“无需用户手势”来完全禁用自动播放策略chrome://flags/#autoplay-policy。这样您就可以测试您的网站，就好像用户与您的网站保持紧密联系一样，并且始终允许播放自动播放。
- 您也可以决定禁止使用MEI以及默认情况下全新MEI获得播放自动播放的网站是否允许新用户使用，从而决定禁止播放自动播放。这可以用两个来完成 内部开关用chrome.exe --disable-features=PreloadMediaEngagementData, AutoplayIgnoreWebAudio, MediaEngagementBypassAutoplayPolicies

iframe委托授权

一个功能政策使开发人员可以选择性地启用和禁用的各种浏览器的功能和API。一旦来源获得了自动播放权限，它就可以将该权限委托给具有自动播放功能的跨源iframe 。默认情况下，同源iframe可以使用自动播放。

```
<!  - 允许自动播放。- >
<iframe src = "https://cross-origin.com/myvideo.html" allow = "autoplay" />
<!  - 允许自动播放和全屏播放。- >
<iframe src = "https://cross-origin.com/myvideo.html" allow = "autoplay; fullscreen" />
```

当禁用自动播放的功能策略时，play()不带用户手势的调用将拒绝带有NotAllowedErrorDOMException 的promise。自动播放属性也将被忽略。

示例场景:

示例1：每次用户在他们的笔记本电脑上访问 iqiyi.com 时，他们都会观看电视节目或电影。由于其媒体参与度较高，因此可以自动播放。

示例2：iqiyi.com 同时具有文字和视频内容。大多数用户偶尔会去该网站获取文字内容并观看视频。用户的媒体参与度较低，因此如果用户直接从社交媒体页面或搜索导航，则不允许自动播放。

示例3：news.iqiyi.com同时具有文字和视频内容。大多数人通过主页进入网站，然后点击新闻报道。由于用户与域名互动，新闻文章页面上的自动播放将被允许。但是，应该注意确保用户不会对自动播放内容感到意外。

示例4： 在爱奇艺泡泡页面将iframe与电影预告片一起嵌入其评论中。用户与域进行交互以访问特定的网站，因此允许自动播放。但是，泡泡需要将该特权显式委托给iframe以便内容自动播放。

Chrome 企业政策

Chrome企业策略可以改变这种新的自动播放行为，以用于例如信息亭或无人值守系统。查看 配置策略和设置帮助页面，了解如何设置这些新的与自动播放相关的企业策略：

- 该“AutoplayAllowed”策略控制自动播放是否允许。
- 该“AutoplayWhitelist”政策，允许您指定的URL模式的白名单，其中自动播放将始终启用。

开发人员最佳实践

视频元素

- 永远不要假设视频会播放，并且在视频不是真正播放时不要显示暂停按钮。
- 关注播放函数返回的Promise

```
var promise = document.querySelector('video').play();
if (promise !== undefined) {
  promise.then(_ => {
    // Autoplay started!
  }).catch(error => {
    // Autoplay was prevented.
    // Show a "Play" button so that user can start playback.
  });
}
```

- 使用静音自动播放

```
<video id="video" muted autoplay>
<button id="unmuteButton"></button>

<script>
  unmuteButton.addEventListener('click', function() {
    video.muted = false;
  });
</script>
```

音频元素

原生播放音频除了使用audio标签之外，还有另外一个API叫AudioContext，AudioContext接口表示由音频模块连接而成的音频处理图，每个模块对应一个AudioNode。AudioContext可以控制它所包含的节点的创建，以及音频处理、解码操作的执行。做任何事情之前都要先创建AudioContext对象，因为一切都发生在这个环境之中。

AudioContext播放声音

1. 先请求音频文件，放到ArrayBuffer里面，然后用AudioContext的API进行decode解码，解码完了再让它去play。

```
function request (url) {
    return new Promise (resolve => {
        let xhr = new XMLHttpRequest();
        xhr.open('GET', url);
        // set response Type arraybuffer
        xhr.responseType = 'arraybuffer';
        xhr.onreadystatechange = function () {
            if (xhr.readyState === 4 && xhr.status === 200) {
                resolve(xhr.response);
            }
        };
        xhr.send();
    });
}
```

2. 实例化AudioContext// Safari是使用webkit前缀

```
let context = new (window.AudioContext || window.webkitAudioContext)();
```

3. 解码播放

```
function play (context, decodeBuffer) {
    let source = context.createBufferSource();
    source.buffer = decodeBuffer;
    source.connect(context.destination);
    // 从0s开始播放
    source.start(0);
}
// 请求音频数据
let audioMedia = await request(url);
// 进行decode和play
context.decodeAudioData(audioMedia, decode => play(context, decode));
```

AudioContext创建时机

- 页面加载时创建

  那么resume()在用户与页面进行交互之后（例如，用户单击按钮），您必须在某个时间进行调用。

```
// Existing code unchanged.
window.onload = function() {
  var context = new AudioContext();
  // Setup all nodes
  ...
}

// One-liner to resume playback when user interacted with the page.
document.querySelector('button').addEventListener('click', function() {
  context.resume().then(() => {
    console.log('Playback resumed successfully');
  });
});document.querySelector('button').addEventListener('click', function() {
  var context = new AudioContext();
  // Setup all nodes
  ...
});
```

- 在用户与该页面进行交互时创建。