前端框架选择

为了更加高效的开发，我们选择了 `jQuery` 作为前端框架。让开发者更方便的处理 HTML、events、实现动画效果。

```
...
var video = $('#video');
$('#play').bind('click', function () {
    video.get(0).play();
})
...
```

前端和服务器端的交互

我们采用了 `AJAX` （Asynchronous Javascript and XML）技术。前端与服务器进行少量数据交换， `AJAX` 可以使网页实现异步更新。在不重新加载整个网页的情况下，对网页的某部分进行更新，提升用户体验。

```
// 请求历史视屏列表
function getHistoryVideoList(data) {
   ...
    $.ajax({
        type: 'GET',
        url: url
    })
    .done(function (res) {
        // 处理服务器返回的视频列表
        loadHistoryVideo(res);
    });
}
```

视频播放解决方案

Web 端的视频播放，我们采用了 HTML5 技术中的 `video` 标签，支持多种编码格式，以及提供了大量的 API ，为 Web 端的视频播放提供了解决方案。

```
<video id="myVideo" class="video">
  <source src="/i/movie.ogg" type="video/ogg">
  <source src="/i/movie.mp4" type="video/mp4">
</video>
```

实时监控解决方案

实时直播采用了最新的 HTML5 技术。
服务器端与前端通过 WebSocket 协议，实时转播数据流。

```
...
var client = new WebSocket( 'ws://<host>:<port>/' );
...
```

前端获取到视频流后，用 JS 对视频解码，在用 `canvas` 逐帧画出图像。本系统中我们采用的是 `jsmpeg.js` 库对视频进行解码。

```
var canvas = document.getElementById('videoCanvas');
var player = new jsmpeg(client, {canvas:canvas});

// 以下部分为 jsmpeg.js 部分源码
var jsmpeg = window.jsmpeg = function( url, opts ) {
    ...
    this.canvasContext = this.canvas.getContext('2d');
    this.renderFrame = this.renderFrame2D;
    ...
}
...
jsmpeg.prototype.renderFrame2D = function() {
    this.YCbCrToRGBA();
    this.canvasContext.putImageData(this.currentRGBA, 0, 0);
};
...
```