web\_socket.js加载后

* swfobject.addDomLoadEvent(function(){WebSocket.\_\_initialize();});
  + 创建swf的container, holder等，swfObject.embed()，将holder替换为swf
    - [swf]在swf内嵌as脚本中，实例化WebSocketMain，添加各种接口供js调用，同时调用js的WebSocket.\_\_onFlashInitialized
      * 设置自定义WebSocket的\_\_flash, \_\_tasks属性，其中\_\_flash引用页面上的swf元素// setTimeout: We need to set a timeout here to avoid round-trip calls to flash during the initialization process.

new WebSocket()

* 若浏览器不支持WebSocket API，则实例化自定义的WebSocket。
  + 延迟调用WebSocket.\_\_flash.create。setTimeout(function(){}, 0)，非堵塞。
    - [WebSocketMain.as]调用as接口create，实例化WebSocket
      * [WebSocket.as]实例化Socket/XMLSocket，给Socket实例添加事件监听器，调用connect方法。
    - [WebSocketMain.as]给WebSocket实例添加事件监听器
      * [WebSocket.as]成功调用Socket的connect方法后，调度Event.CONNECT事件，监听器响应并调用回调方法onSocketConnect，发出握手请求。
        + [WebSocket.as]服务器发出握手响应，触发PorgressEvent.SOCKET\_DATA事件，监听器响应并调用回调方法onSocketData，若成功握手，设置WebSocket实例的readyState属性为OPEN，并调度new WebSocketEvent("open")事件。

[WebSocketMain.as]WebSocket实例的监听器响应并调用回调方法onSocketEvent

[WebSocketMain.as]解析事件对象并将解析结果加入到事件队列中，调用processEvent

[WebSocketMain.as]轮调直至js响应并调用js接口WebSocket.\_\_onFlashEvent

[js]获取后清空as中的事件队列，按顺序处理事件队列的事件，\_\_handleEvent。

。。。接上

* [js]获取flashEvent中的readyState, protocols等属性并赋给jsEvent，然后判断flashEvent的type属性，若为”open”，调用\_\_createSimpleEvent创建简单事件open，其中\_\_createSimpleEvent方法完美处理浏览器兼容性问题；最后调度该事件dispatchEvent。
  + [js]若通过WebSocket实例.addEventListener(“onopen”, callback)或WebSocket实例.onopen = callback给WebSocket实例添加对该事件的处理器，则调用处理器方法。

传输数据  
[js]WebSocket实例.send

* 判断readyState，若为CONNECTING，则抛出错误。
  + 对data进行encode，调用as接口send
    - [WebSocketMain.as]WebSocket实例.send(data)
      * [WebSocket.as]Socket实例.send
        + [WebSocket.as]若抛出IOError，Socket实例上的监听器响应并调用onSocketIoError

[WebSocket.as]onConnectionError

[WebSocket.as]close(STATUS\_CONNECTION\_ERROR)

* + - * + [WebSocket.as]对将要传输的数据encode，调用Socket的send方法将数据传输给服务器。