**通讯机制使用手册**

# 概述

随着富媒体应用的不断发展成熟，AJAX的应用得到越来越多的普及，而AJAX应用的一个核心功能-异步数据载入功能也就显得越为重要，包括HTML5的发展也为此提供了强大的支持，NEJ作为跨平台的WEB前端开发框架当然也为上层应用做了很好的支持。本文档主要介绍NEJ中涉及通讯相关的技术的原理及使用，大家在使用NEJ时如果涉及到通讯相关的内容可先阅读此文档。

本文档主要分以下几部分介绍NEJ中封装的通讯机制

* 消息通讯：跨文档的消息通讯机制，主要是本地的不同框架之间的数据通讯
* 异步请求：即所谓的AJAX请求，主要与服务器端进行通讯

# 消息通讯

由于同源策略的限制，非同域的文档之间不能直接进行数据通讯或共享，因此我们引入这部分内容来分析解决非同域及同域的文档之间的数据通讯或共享问题。

这里的消息通讯是指应用在客户端跨文档之间的消息交互，可以保证在不会被恶意攻击的前提下进行跨文档的数据共享方式，这里不会涉及到客户端与服务器端的信息交互。

本部分内容对应的W3C的规范见<http://www.w3.org/TR/webmessaging/>，关于这部分规范的详细介绍可参阅专题分享《[HTML5 Web Message](file:///D:\Workspace\wd\w3c新技术规范调研\Javascript%20APIs\HTML5%20Web%20Messaging\message.pptx)》

## 原理

消息通讯主要包含两个过程：

* 收消息：文档通过监听当前窗体的onmessage事件来接收来自其他窗体的消息
* 发消息：文档可以通过给定的接口向指定的窗体发送消息，包括跨域的窗体

原理示意图如下所示



这里我们可以简单理解为各个窗体之间有一个消息通道，对于消息通道的关系我们定义为以下两种类型：

* 直接父子关系：消息通道所在的窗体为父子关系
* 兄弟及祖孙关系：消息通道所在的窗体为兄弟关系，或者父窗体和子窗体的若干层子窗体关系

在NEJ中我们主要封装的是直接父子关系的窗体之间的消息交互，其他关系的窗体均可通过此方式来解决。

## 实现

NEJ对于这部分的实现主要分为两部分，一部分针对高版本浏览器，即已经支持HTML5 Web Message规范的浏览器，采用规范定义的API；另一部分针对Trident引擎的低版本浏览器，主要是IE6-7的版本，采用的是window.name的方式传递数据。

NEJ对这部分的封装主要在 {lib}util/ajax/message.js 中，项目使用时直接依赖该文件即可，然后直接使用提供的API进行操作。

### 高版本浏览器

#### 发送消息

发送消息主要通过规范定义的window.postMessage接口来实现

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | Variable | 发送的数据，可以是基本类型，也可以是File、Blob等对象，注IE8-9支持传递字符串格式的数据 |
| String | 目标可接收消息的源信息，如果目标的源不是该参数指定的源则忽略此消息，如果所有源均可接收消息则可以传“\*”，如果只允许同域的窗体接收消息则可以传“/” |
| Array | 消息通道对象，主要用来代理非直接父子关系的窗体之间的消息通讯 |
| **输出** | Void | 无 |
| **描述** | 发送消息到指定窗体 | |

#### 接收消息

窗体通过监听window上的onmessage事件来接收其他窗体发过来的消息，接收的消息对象如下所示



这里我们主要关注的是data、origin、source属性，实际应用中data和origin的使用会比较常见。

### 低版本浏览器（IE）

这里的低版本浏览器主要针对Trident引擎的浏览器，如IE6-7版本，对于这类浏览器本身对Web Message规范的实现不完善，因此NEJ中采用window.name作为代理来实现消息的传递。

#### 原理

这里我们主要利用Trident引擎下跨域窗体可设置window.name的特性来实现消息的传递，具体实现原理示意图如下所示



假设上图Window1需要传递消息至Window2中，则消息的传递步骤如下：

1. Window1的消息发送器按照指定协议拼装消息
2. Window1的消息发送器将拼装好的消息字符串设置到Window2的window.name属性上
3. Window2起定时器轮询window.name的变化情况
4. Window2发现window.name中Window1设置的消息串，按照指定协议解码
5. Window2触发window上的onmessage事件通知上层应用收到消息

反之亦然

#### 协议

这里的协议主要指window.name上设置的消息串的格式

* 必须以字符串 MSG| 作为起始，且必须大写字符
* 参数以键值对方式传入，键与值之间用 = 连接，所有键值均做encodeURIComponent编码，键值对之间以 | 字符分隔，如 a=b|b=a%26b
* 将以上结果做escape后设置到目标window的name属性上，如MSG%7Cdata%3D%257B%2522url%2522%253A%2522http%253A%252F%252Fa.b.com

传递的参数信息主要包括

|  |  |
| --- | --- |
| 参数名称 | 参数描述 |
| origin | 目标接收消息的源信息 |
| data | 传递的消息数据，序列化为JSON字符串 |
| ref | 当前窗体的访问地址 |
| self | 当前窗体名称，默认为\_parent |

#### 缺陷

由于浏览器的限制本解决方案会存在以下一些问题

1. **可能会有消息丢失**

由于使用的是定时器轮询window.name的变化，而window.name为所有其他窗体共享的资源，但是其他窗体仅有设置权限而没有读取权限，因此无法保证在一次读取消息后到下一次读取消息的这段时间内没有两个及两个以上的到达消息，如果出现这种情况则最后设置的消息会覆盖之前的消息，因此之前的消息无法到达当前窗体中。

1. **没能真正意义上保证origin的限制条件**

由于无法读取一个窗体的location信息，因此对origin的验证是在接收到消息后由NEJ框架额外做的封装，因此从真正意义上来说这个消息还是已经到达了当前窗体中，因此可能会存在信息的泄漏情况

综上所述，对于低版本浏览器建议尽量避免传递一些敏感信息，同时系统也尽量避免强依赖消息传递机制来实现系统的重要功能。

## 使用

对于消息的通讯部分NEJ主要提供了一个nej.j.\_$postMessage接口用来发送消息，提供通过nej.e.\_$addEvent接口添加window上的onmessage事件监听接收到的消息。

使用此接口前先依赖 {lib}util/ajax/message.js 文件

#### nej.j.\_$postMessage

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | String|Window | 目标window对象或者Frame的name，或者字符串如\_top、\_parent、\_self |
| Object | 消息配置信息  data [Variable] 消息内容  origin [String] 目标Origin，只有指定的页面可以收到消息，默认为所有源可接收，如http://a.b.com  source [String] 当前窗体标识，除非你非常确定当前窗体的标识是什么，否则请采用自动识别 |
| **输出** | Void | 无 |
| **描述** | 发送消息到指定窗体 | |

#### window.onmessage

注这里的onmessage事件必须通过nej.v.\_$addEvent接口添加，接收到的消息格式遵循W3C中对于Web Message的规范，见[高版本浏览器接收消息](#_接收消息)部分对消息对象的介绍

#### 应用举例

假设http://c.d.com的页面中有一个frame引入了http://a.b.com的页面



http://c.d.com和http://a.b.com都可以通过以下方式添加事件监听接收的消息



假设http://c.d.com页面通过以下方式发送了一个消息



以上代码虽然接收消息的目标窗体是targetFrame，但是targetFrame的源信息为http://a.b.com，而消息发送的目标源是http://e.f.com，所以在http://a.b.com这个页面添加的onmessage事件不会被触发。

如果实际应用中需要用以下方式用event.source来反馈消息的话，请一定确保在Frame上设置了name属性



# 异步请求

对于富媒体WEB应用来说异步请求为其必不可缺的组成部分，对于同域的异步请求基本无需做太多额外配置，对于跨域的请求W3C也提供了相应的规范的支持，但是由于各浏览器对规范支持的程度的差异导致最终使用时会有一些差别，而NEJ框架针对这些差异及低版本的跨域支持做了封装，对上层应用而言可以使用统一的API做同域或非同域的异步请求。NEJ中封装的异步请求的整个体系结构如下图所示



## 代理模式

NEJ中主要通过客户端中间代理的方式实现异步请求，根据代理实现方式的差别主要提供以下几种模式的支持，具体应用可以根据实际情况选择使用的代理模式

* W3C的[Cross-Origin Resource Sharing](http://www.w3.org/TR/cors/)规范，后面简称CORS规范
* Frame代理
* Flash代理
* 文件上传代理

NEJ在所有平台均提供了以上三种模式的支持，由于各模式均有自己的一些缺陷，因此实际应用时请按照具体情况选择不同的模式

## CORS规范

对于这部分的详细信息可以参阅专题分享《[Cross-Origin Resource Sharing](file:///D:\Workspace\wd\w3c新技术规范调研\Javascript%20APIs\Cross-Origin%20Resource%20Sharing\cors.pptx)》，以下选取一些主要内容做简单介绍

#### 原理



以上各流程简要说明

1. 浏览器捕获到a.b.com应用往x.y.com的服务器发起的请求
2. 浏览器检查请求情况确定是否需要先做一次预请求来验证x.y.com的服务器是否允许发当前请求过去，如果需要发预请求则浏览器发起一个OPTIONS的请求到x.y.com的服务器验证继续第3步，否则直接发送请求到x.y.com服务器继续第5步
3. 服务器根据浏览器发过来哦header信息，然后根据服务器端对资源的配置返回资源的实际控制权限配置
4. 浏览器验证预请求返回的信息，判断是否可以将请求发送到x.y.com的服务器，如果不行则异常退出，否则继续第5步
5. 浏览器发送实际请求至x.y.com服务器
6. 服务器返回请求数据及资源控制配置信息至浏览器
7. 浏览器验证资源控制信息是否允许当前实际请求的取到数据，如果不允许则异常退出，否则继续第8步
8. 浏览器返回x.y.com返回的数据至a.b.com的应用

#### 缺陷

需要高版本浏览器支持，Trident引擎的浏览器需要IE10以上才支持，NEJ中对于IE9-版本自动适配到Frame代理或者Flash代理模式

## Frame代理

此模式通过在目标服务器放置一个代理文件，然后通过该代理文件来与服务器端进行数据交互，返回数据通过消息通讯返回给上层应用来实现跨域的数据交互。

此方式也支持通过代理文件配置资源可访问的来源

#### 原理



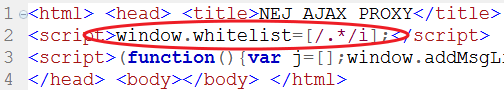
以上各流程简要说明

1. 当a.b.com的应用要往x.y.com的服务器取数据时，首先会用iframe载入预先放置在x.y.com服务器上的代理文件
2. 服务器端返回做了配置的代理文件
3. 代理文件载入完成后a.b.com的应用将要发送的请求指令通过消息通信方式传递给代理文件
4. 代理文件验证a.b.com是否在预先配置的白名单中，如果不在则异常返回，否则直接发送请求至x.y.com服务器
5. 服务器返回数据至代理文件
6. 代理文件通过消息通讯机制将请求结果返回给a.b.com的应用

#### 配置

使用此模式需要注意以下几点：

* 目标服务器必须放置代理文件（该文件由NEJ提供文件模板，具体产品配置白名单）
* 代理文件必须配置白名单列表
* 白名单列表使用js的正则表达式配置，如下图所示



#### 缺陷

该模式主要有以下几个问题：

* 需要在目标服务器放置代理文件
* 由于首次发起请求时需要载入代理文件，在载入代理文件之前的所有请求都会存在一定的延时
* 对于低版本浏览器受限于[消息通讯机制的缺陷](#_问题)，因此此模式在低版本浏览器中可能会存在请求丢失的情况

## Flash代理

此模式与Frame代理模式类似，主要差别在于请求通过Flash来发送，因此可以利用Flash的策略文件crossdomain.xml来控制资源的共享权限

#### 原理



以上各流程简要说明

1. a.b.com的应用从自己的服务器上载入用来做请求的flash代理文件
2. a.b.com服务器返回代理flash文件
3. flash准备完毕后a.b.com应用将请求指令发送给flash
4. flash从目标服务器x.y.com载入策略文件crossdomain.xml
5. flash验证策略文件是否允许a.b.com访问x.y.com的资源，如果不允许则异常返回，否则发送请求至目标服务器
6. x.y.com服务器返回数据至flash代理中
7. flash代理将请求返回的数据提交到a.b.com的应用中

#### 缺陷

该模式主要有以下几个问题：

* 需要浏览器支持flash
* 需要在目标服务器配置flash策略文件crossdomain.xml
* 由于首次发起请求时需要载入flash代理文件，在载入代理文件之前的所有请求都会存在一定的延时

## 文件上传

对于无刷新的系统来说，如果需要上传文件则必须也需要不刷新当前页面，对于高版本浏览器使用W3C中定义的XMLHttpRequest规范来实现，低版本则采用Frame代理的方式来实现

关于对XMLHttpRequest的详细信息可参阅专题分享《[XMLHttpRequest](file:///D:\Workspace\wd\w3c新技术规范调研\Javascript%20APIs\XMLHttpRequest\xhr.pptx)》

#### 原理

##### 高版本浏览器

对于高版本浏览器使用W3C规范中对XMLHttpRequest的定义来实现



##### 低版本浏览器

对于无法支持XMLHttpRequest直接进行文件上传的浏览器，采用Form表单+Frame代理的方式来实现



以上各流程简要说明

1. a.b.com应用中包含要上传的文件的Form表单POST至目标服务器x.y.com，如果需要轮询进度的话此时开始起定时器轮询上传进度
2. x.y.com将返回结果按照nej\_proxy\_upload.html模板文件给定的格式返回结果至Frame代理中
3. Frame代理通过消息通讯机制将上传结果返回到a.b.com的应用中

#### 缺陷

低版本返回的数据格式同高版本的不一致，因此服务器端需额外判断当前请求类别，然后根据类别返回数据内容

## 使用

NEJ框架对这部分内容主要提供两个API供上层应用使用，nej.j.\_$request、nej.j.\_$upload

#### nej.j.\_$request

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | String | 请求地址 |
| Object | sync [Boolean] 是否同步请求  type [String] 返回数据格式,text/json/xml  data [Variable] 要发送的数据  query [Variable] 查询参数,字符串格式a=b&c=d,对象格式{a:'b',c:'d'}  method [String] 请求方式,GET/POST  timeout [Number] 超时时间,0 禁止超时监测  headers [Object] 头信息表  mode [Number] 请求模式,针对跨域请求采用的请求方式 |
| **输出** | Void | 无 |
| **描述** | 发起一个请求，请求模式说明  0 - 自动模式，高版本使用HTML5的CORS协议，低版本采用Frame代理方式  1 - 高版本使用HTML5的CORS协议，低版本采用Flash代理方式  2 - 全部使用Frame代理方式  3 - 全部使用Flash代理方式 | |

代码举例



#### nej.j.\_$upload

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | HTMLFormElement | 待上传表单对象 |
| Object | 配置信息  mode [Number] 上传模式  type [String] 返回数据格式  query [Variable] 查询参数  headers [Object] 头信息  onload [Function] 上传完成回调  onerror [Function] 上传失败回调  onuploading [Function] 上传进度回调  onbeforerequest [Function] 请求之前对数据处理回调 |
| **输出** | Void | 无 |
| **描述** | 文件上传，模式说明  0 - 自动模式，高版本使用XMLHttpRequest，低版本采用Frame代理方式  2 - 全部使用Frame代理方式 | |

代码举例



# 资源载入

这部分主要用来载入样式、脚本、页面结构等资源，对于载入的样式和脚本会直接生效，页面结构的载入后主要提取结构中的body节点，因此需要载入的页面结构在当前页面的同域下。

项目中使用此模块时依赖 {lib}util/ajax/tag.js 文件

## 样式

动态载入外联样式资源，这部分主要以API的形式提供支持

#### nej.j.\_$loadStyle

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | String | 样式资源地址 |
| Object | 可选配置参数，已处理参数列表如下  timeout [Number] 超时时间，0表示禁止超时  version [String] 版本信息  onerror [Function] 出错回调  onloaded [Function] 载入回调  onloading [Function] 加载中回调 |
| **输出** | Void | 无 |
| **描述** | 载入样式资源文件 | |

#### nej.j.\_$queueStyle

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | Array | 样式资源地址列表 |
| Object | 可选配置参数，已处理参数列表如下  timeout [Number] 超时时间，0表示禁止超时  version [String] 版本信息  onerror [Function] 出错回调  onloading [Function] 加载中回调 |
| **输出** | Void | 无 |
| **描述** | 按序载入样式资源文件 | |

## 脚本

动态载入外联脚本资源，这部分主要以API的形式提供支持

#### nej.j.\_$loadScript

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | String | 脚本资源地址 |
| Object | 可选配置参数，已处理参数列表如下  async [Boolean] 异步载入并立刻执行  timeout [Number] 超时时间，0表示禁止超时  version [String] 版本信息  charset [String] 脚本编码  onerror [Function] 出错回调  onloaded [Function] 载入回调  onloading [Function] 加载中回调 |
| **输出** | Void | 无 |
| **描述** | 载入脚本资源，此接口回调信息中的错误代码及含义如下  -10000 无法加载指定脚本文件  -10001 请求超时 | |

#### nej.j.\_$queueScript

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | Array | 脚本资源地址列表 |
| Object | 可选配置参数，已处理参数列表如下  version [String] 版本信息  timeout [Number] 超时时间，0表示禁止超时  charset [String] 脚本编码  onerror [Function] 出错回调  onloaded [Function] 载入回调  onloading [Function] 加载中回调 |
| **输出** | Void | 无 |
| **描述** | 按序载入脚本资源，后一个脚本的执行必须等前一个脚本执行完毕 | |

## 结构

动态载入外联页面结构，这部分主要以API的形式提供支持

#### nej.j.\_$loadHtml

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | String | 资源地址 |
| Object | 可选配置参数，已处理参数列表如下  version [String] 版本信息  onloaded [Function] 加载完成回调 |
| **输出** | Void | 无 |
| **描述** | 载入资源文件 | |

# 协议封装

## DWR

针对[DWR框架](http://directwebremoting.org/dwr/index.html)提供的前端适配接口

### nej.j.\_$setFilter

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | Function | 异常过滤接口 |
| **输出** | Void | 无 |
| **描述** | 设置全局异常过滤接口，过滤接口返回true表示不继续后续错误处理，否则继续执行后续的异常处理 | |

### nej.j.\_$setBatchId

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | String | 请求ID |
| **输出** | Void | 无 |
| **描述** | 设置请求ID，对于需要使用浏览器缓存的DWR请求可以通过此接口设置请求ID并以GET方式发送，建议使用非数字的ID  注：此ID仅对后续第一个请求有效 | |

### nej.j.\_$beginBatch

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | Void | 无 |
| **输出** | Void | 无 |
| **描述** | 批请求开始标记，后续[\_$requestByDWR](#__$requestByDWR)接口调用不直接发起请求，直到[\_$endBatch](#__$endBatch)接口被调用才发送请求 | |

### nej.j.\_$endBatch

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | Void | 无 |
| **输出** | Void | 无 |
| **描述** | 发起批处理请求，与[\_$beginBatch](#__$beginBatch)配合使用 | |

### nej.j.\_$requestByDWR

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | String | 请求地址，class.method格式 |
| Object | 可选配置参数，已处理参数列表如下  path [String] 请求路径，  默认为/dwr/call/plaincall/  query [String] 请求地址附带的查询参数  格式a=aa&b=bb&c=cc  param [Array] 参数列表，不传或空数组表示无参数  script [Boolean] 是否使用脚本方式载入  sync [Boolean] 是否使用同步请求  method [String] 请求方式，GET/POST  timeout [Number] 请求超时时间  onload [Function] 请求完成回调  onerror [Function] 请求出错回调  如果使用脚本载入方式则忽略sync，method参数 |
| **输出** | Void | 无 |
| **描述** | 使用DWR协议发起请求，此接口回调信息中的错误代码及含义如下  -10001 请求超时  -10003 DWR请求返回数据不合法  -10004 DWR返回脚本执行异常  -10005 DWR回调执行异常 | |

## REST

提供REST规范支持的接口

### nej.j.\_$requestByREST

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | String | 请求地址 |
| Object | 可选配置参数，已处理参数列表如下  sync [Boolean] 是否同步请求  data [Object] 发送数据  param [Object] 请求参数，包括模板参数  method [String] 请求方式，GET/POST  timeout [Number] 超时时间，0表示禁止超时  headers [String] 脚本编码  onerror [Function] 出错回调  onload [Function] 载入回调 |
| **输出** | Void | 无 |
| **描述** | REST交互接口，此接口回调信息中的错误代码同[\_$request](#__$request)接口 | |