第六章 Communication API

内容安排

- 6.1 跨文档消息通信
- 6.2 XMLHTTPRequest Level 2
- 6.3 使用XMLHTTPRequest API
- 6.4 创建XMLHttpRequest应用
- 6.5 进阶功能
- 6.6 课后思考
- 6.7 小结

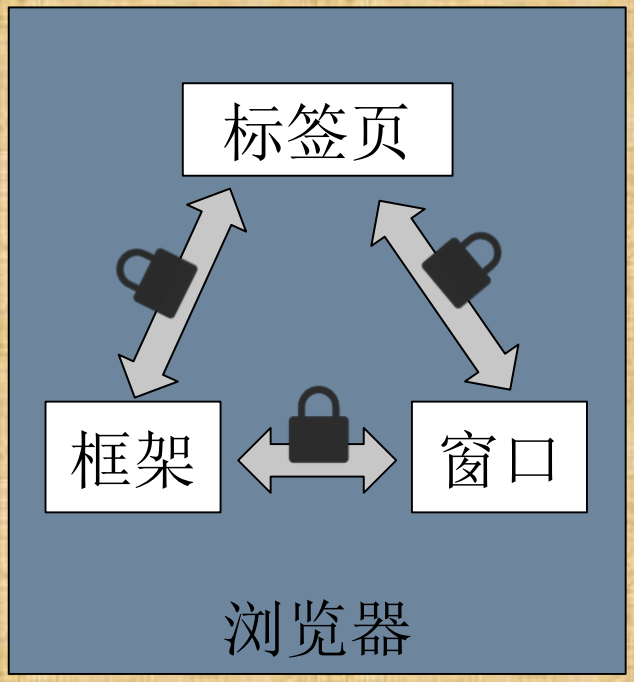
6.1 跨文档消息通信

- 理解源安全
- 跨文档消息通信的浏览器支持情况
- 使用postMessage API
- 使用postMessage API创建应用

6.1 跨文档消息通信

为了防止某些网站利用脚本来**窃取**其他网站的信息，浏览器厂商都合理的限制了框架、标签页和窗口间的通信。

如果是合理的交互需求，则需要使用一种新功能：**跨文档消息通信**。（由浏览器厂商和标准指定机构y一致同意，可以确保这三者之间安全地跨源通信。）



6.1 跨文档消息通信

将postMessage API定义为发送消息的标准方式。其语法为：

要发送到其他窗口的数据

要发送到其他窗口的数据是与消息一起传输的Transferable对象序列。(可选值)

```
targetWindow.postMessage(message,targetOrigin,[ transfer ]);
```

对将接收消息的窗口的引用。

指定要调度的事件的targetWindow的原点

6.1 跨文档消息通信

利用postMessage发送消息代码代码：引用框架将hello,world消息发送到http://www.example.com。

```
chatFrame.contentWindow.postMessage('Hello, world','http://www.example.com/');
```

接收消息时仅需在页面中增加一个事件处理函数。当消息送达时，通过检查消息的来源来决定是否对条消息进行处理。

6.1 跨文档消息通信

将消息传递给messageHandler()函数的事件监听器：

```
window.addEventListener("message",messageHandler,true);
function messageHandler(e){
    switch(e.origin){
        case "friend.example.com":
        //处理消息
        processMessage(e.data);
        break;
        default;
        //消息来源无法识别
        //消息被忽略
    }
}
```

6.1 跨文档消息通信

消息事件包含了两个DOM事件属性：

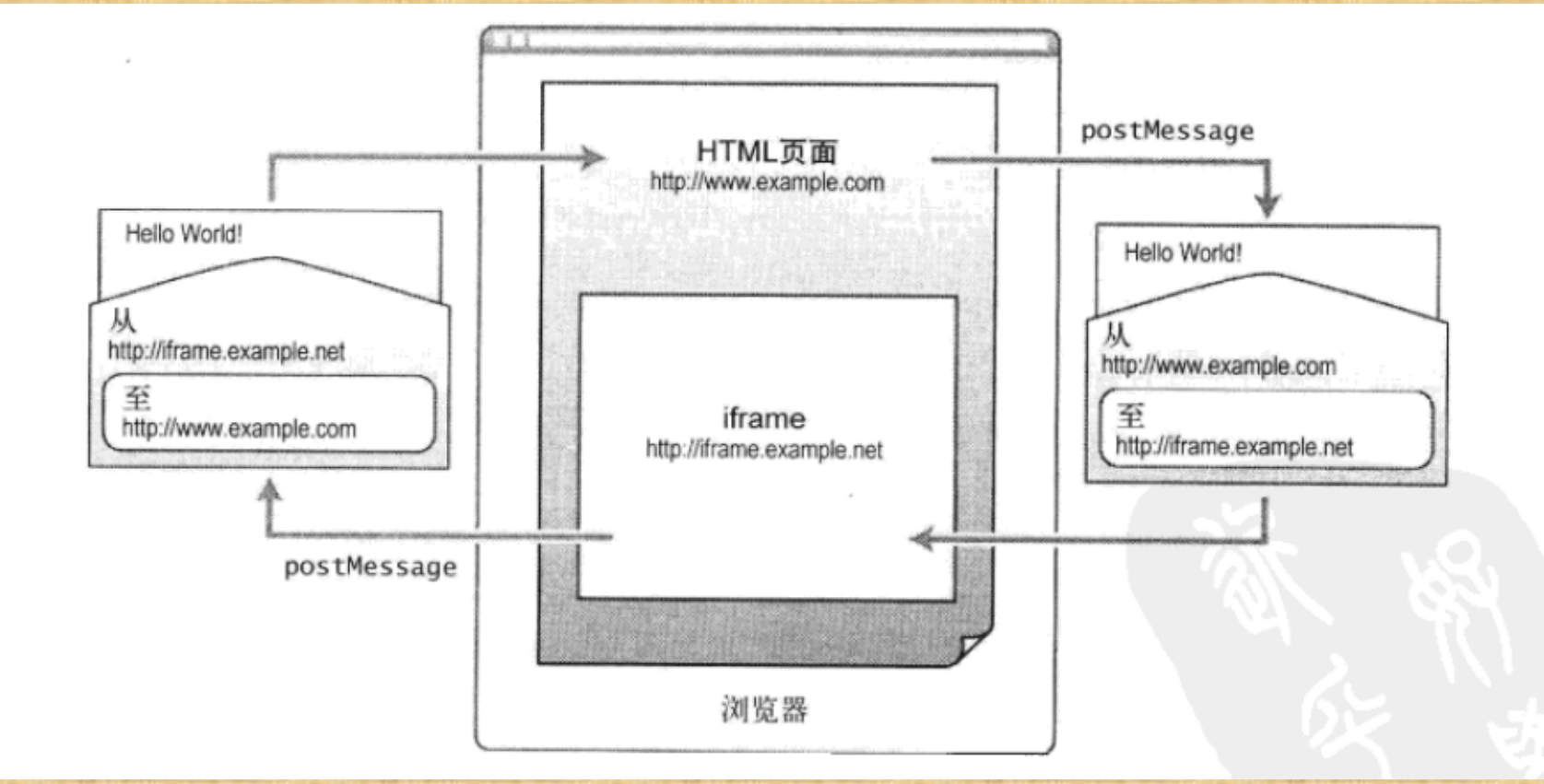
data属性：发送方传递的实际消息；

origin属性：发送来源。接收方利用该属性能轻易的忽略掉来自不可信源的消息。

postMessage API提供了一种交互方式，使得不同源的iframe可以与其父页面进行通信。

6.1 跨文档消息通信

iframe与其父页面间的postMessage通信：



6.1 跨文档消息通信

上述示例使用postMessage让父页面和部件相互之间能发送消息，它们通过把彼此的源加到可信源的白名单中，就都能接收到来自对方的信息。

postMessage API不但可以胜任同源文档间的通信，而且在浏览器不允许非同源通信的情况下也能安全使用（即跨源通信）。

理解源安全

源由规则(scheme)、主机(host)、端口(port)组成。例如：

因此针对上述三部分任一部分不同都表明源是不同的。如https://www.example.com/和http://www.example.com/是不同源。

理解源安全

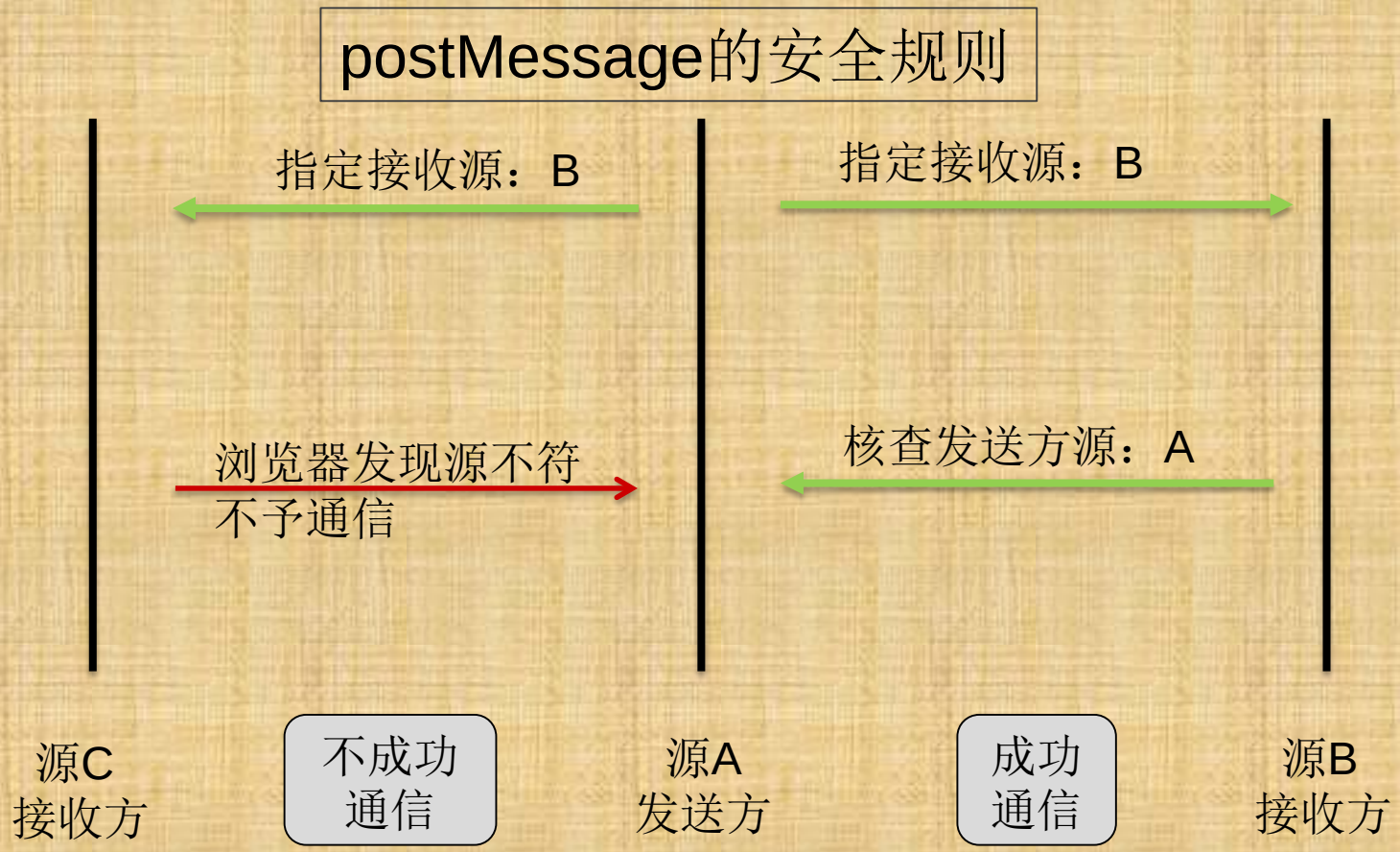
另外源的概念不考虑路径，如：

http://www.example.com/index.html

http://www.example.com/page2.html

跨源通信通过源来确定发送者，这就使得接受方可以忽略来自不可信源的消息。同时，各种应用必须加入事件监听器以接收消息，从而避免被不可信应用程序的信息所干扰。

理解源安全


postMessage的安全规则
指定接收源：B
指定接收源：B
浏览器发现源不符
不予通信
核查发送方源：A
源C
接收方
不成功
通信
源A
发送方
成功
通信
源B
接收方

跨文档消息通信的浏览器支持情况

表 PC端主流浏览器对跨文档消息机制的支持情况

浏览器	对跨文档消息机制的支持
Chrome	2.0及以后的版本支持
Firefox	3.0及以后的版本支持
Internet Explore	8.0及以后的版本支持
Opera	9.6及以后的版本支持
Safari	4.0及以后的版本支持

跨文档消息通信的浏览器支持情况

表 移动端主流浏览器对跨文档消息机制的支持情况

浏览器	对跨文档消息机制的支持
Chrome for Android	75版本支持
Firefox for Android	67版本支持
Android Browser	4.2及以后的版本支持
Opera Mini	all
iOS Safari	9.3及以后的版本支持
QQ Browser	1.2及以后的版本支持
UC Browser	12.12及以后的版本支持

跨文档消息通信的浏览器支持情况

利用代码检测浏览器是否支持跨文档消息机制：

```
<button id="che
<script type="t
    function ch
        if(type
            ale
        }
        else{
            ale
        }
    }
</script>
```

使用postMessage API

windows.postMessage(data,url)

参数说明如下：

data，发送消息中包含的**数据**，通常是一个字符串。

url，指定允许通信的**域名**。注意，不是接受消息的目标域名。使用该参数的主要作用是出于安全的考虑，接受消息的窗口可以根据次参数判断消息是否来自可信任的来源，以避免恶意攻击。如果不对访问的域进行判断，则可以使用‘*’。

使用postMessage API

- 1、浏览器支持情况检测

创建应用程序并在调用postMessage前，首先需要检测浏览器是否支持它。

```
if (typeof window.postMessage === "undefined"){
    // 该浏览器不支持postMessage
}
```

- 2、发送消息

通过调用目标页面window对象中的postMessage()函数可发送消息：

使用postMessae API

```
window.postMessage("hello,world","portal.example.com");
```

同样，要发送消息给iframe, 可以在相应的iframe的contentWindow中调用postMessage, 代码如下：

```
document.getElementsByTagName("iframe")[0].contentWindow.postMessage
("hello,world","chat.example.net");
```

使用postMessage API

- 3、监听消息事件

脚本可以通过监听window对象中的事件来接收信息，在事件监听函数中，接收方可以决定接收或者忽略消息。

```
var originWhiteList = ["portal.example.com",
     "games.example.com","www.example.com"];
function checkWhiteList(origin) {
    for (var i=0;i<originWhiteList.length;i++){
        if(origin === originWhiteList[i]){
            return true;
        }
    }
    return false;
 }
```

使用postMessage API

```
function messageHandler(e) {
    if(checkWhiteList(e.origin)){
        processMessage(e.data);
    } else{
        //忽略来自未知源的消息
    }
}
```

上述代码通过创建一个白名单数组来鉴定源实现监听消息事件，如果能检测到与白名单里的源一致，则予以通信，否则将提示无法通信。

使用postMessage API 创建应用

利用跨文档消息通信来实现门户页面和聊天部件之间的交互。最终效果如图所示：

使用postMessage API 创建应用

这个示例演示了如何在门户页面中以iframe方式嵌入第三方的部件。

使用一个来自chat.example.net的部件。门户页面和部件通过postMessage来通信。iframe中的聊天部件通过父页面**标题内容的闪烁**来通知用户。

- 1、创建门户页面

首先，添加来自不同源的iframe以包含聊天部件：

```
<iframe id="widget" src="http://chat.example.net:9999/postMessageWidget.html"></iframe>
```

使用postMessage API 创建应用

然后，添加事件监听器messageHandler监听来自聊天部件的消息事件。整个函数是确保信息来于为http://chat.example.net:9999。

```
var targetOrigin = "http://chat.example.net:9999";

function messageHandler(e) {
    if (e.origin == targetOrigin) {
        notify(e.data);
    } else {
        // 忽略来自其他源的消息
    }
}
```

闪烁通知用户

使用postMessage API 创建应用

最后，添加一个与聊天部件通信的函数。

```
function sendString(s) {
    document.getElementById("widget").contentWindow.postMessage(s, targetOrigin);
}
```

它用postMessage发送一个状态，更新门户页面中的部件iframe。对于在线的聊天应用中，可以使用这种方式来更新用户状态（在线、离开等）。

使用postMessage API 创建应用

- 2、创建聊天部件页面

首先，添加事件监听器messageHandler监听来自门户页面的消息事件。

```
var targetOrigin = "http://portal.example.com:9999";

function messageHandler(e) {
    if (e.origin === "http://portal.example.com:9999") {
        document.getElementById("status").textContent = e.data;
    } else {
        // 忽略其他源发来的消息
    }
}
```

使用postMessage API 创建应用

其次，编写用于与门户页面通信的函数。

```
function sendString(s) {
    window.top.postMessage(s, targetOrigin);
}
```

在接受到新消息时，部件将请求门户页面通知用户，并用postMessage()函数向门户页面发送消息。

使用postMessage API 创建应用

- 3、完整代码

门户页面的完整代码(postMessagePortal.html)

```
<!DOCTYPE html>
<title>Portal [http://portal.example.com:9999]</title>
<link rel="stylesheet" href="styles.css">
<link rel="icon" href="http://apress.com/favicon.ico">
<script>
var defaultTitle = "Portal [http://portal.example.com:9999]";
var notificationTimer = null;
var targetOrigin = "http://chat.example.net:9999";
```

使用postMessage API 创建应用

```
function messageHandler(e) {
    if (e.origin == targetOrigin) {
        notify(e.data);
    } else {
        // 忽略来自其他源的消息
    }
}

function sendString(s) {
    document.getElementById("widget").contentWindow.postMessage(s, targetOrigin);
}
```

使用postMessage API 创建应用

```
function notify(message) {
    stopBlinking();
    blinkTitle(message, defaultTitle);
}

function stopBlinking() {
    if (notificationTimer !== null) {
        clearTimeout(notificationTimer);
    }
    document.title = defaultTitle;
}

function blinkTitle(m1, m2) {
    document.title = m1;
    notificationTimer = setTimeout(blinkTitle, 1000, m2, m1)
}
```

通知函数

停止闪烁
标题函数

闪烁标题
m1:闪烁的信息
m2:默认的标题信息

使用postMessage API 创建应用

```
/**
 * 发送状态消息。
 **/
function sendStatus() {
    var statusText = document.getElementById("statusText").value;
    sendString(statusText);
}
/**
 * 封装一个想聊天小部件页面发送通知的函数，绑定点击事件。
 **/
function loadDemo() {
    document.getElementById("sendButton").
    addEventListener("click", sendStatus, true);
    document.getElementById("stopButton").
    addEventListener("click", stopBlinking, true);
    sendStatus();  //页面加载立即调用，马上发送
}
```

使用postMessage API 创建应用

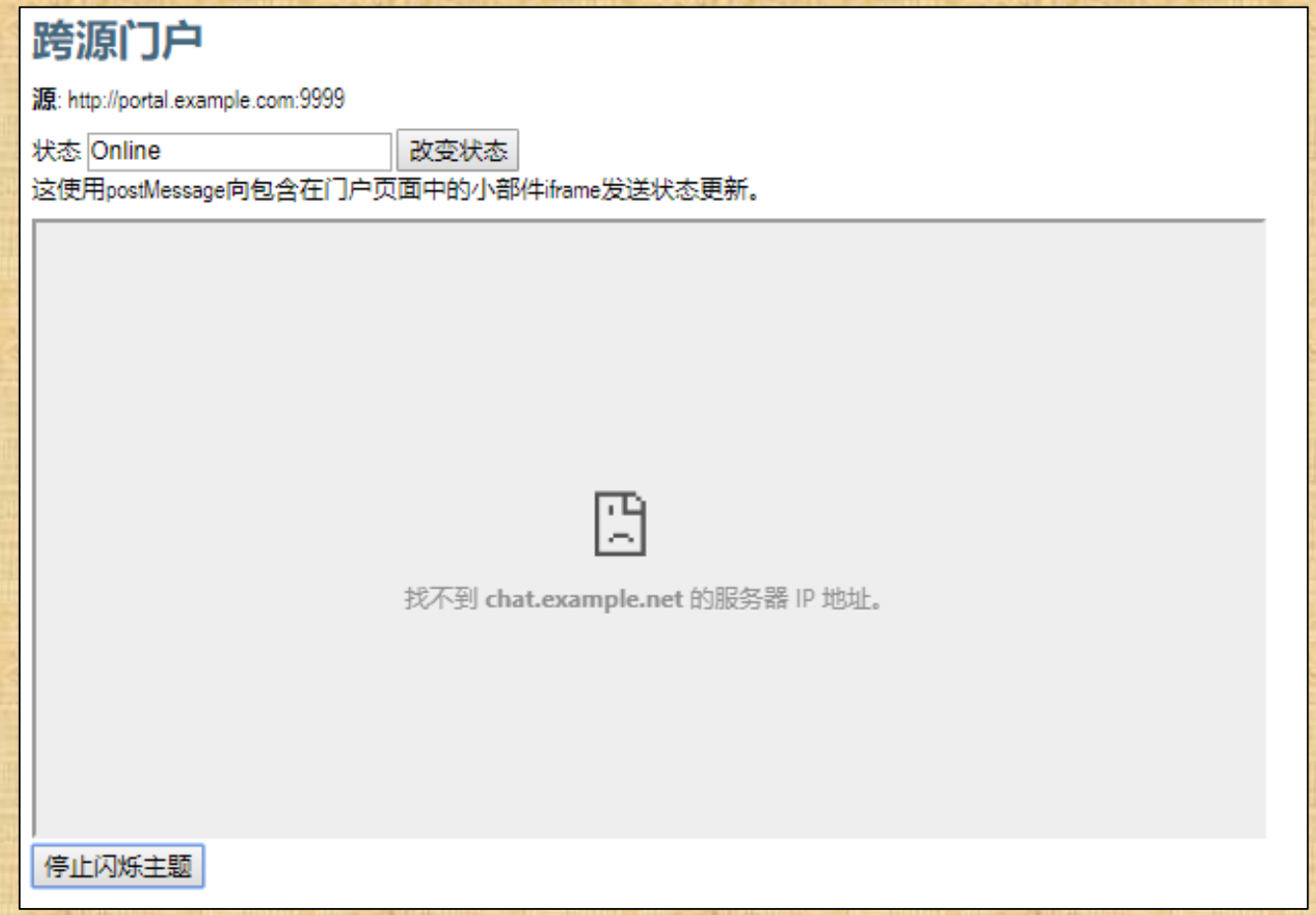
```
/**
 * 为window绑定load加载事件和message消息事件监听器
 **/
window.addEventListener("load", loadDemo, true);
window.addEventListener("message", messageHandler, true);

</script>

<h1>跨源门户</h1>
<p><b>源</b>: http://portal.example.com:9999</p>
状态 <input type="text" id="statusText" value="Online">
<button id="sendButton">改变状态</button>
```

使用postMessage API 创建应用

```
<p>
    这使用postMessage向包含在门户页面中的小部件iframe发送状态更新。
</p>
<iframe id="widget" src="http://chat.example.net:9999/postMessageWidget.html"></iframe>
<p>
    <button id="stopButton">停止闪烁主题</button>
</p>
```

使用postMessage API 创建应用

此时代码的浏览效果如图所示：



ps：此刻还未部署服务器。

使用postMessage API 创建应用

聊天部件的完整代码(postMessageWidget.html)

```
<!DOCTYPE html>
<title>聊天小部件页面</title>
<link rel="stylesheet" href="styles.css">
<script>

var targetOrigin = "http://portal.example.com:9999";
```

使用postMessage API 创建应用

```
/**
 * 监听消息事件函数
 **/
function messageHandler(e) {
    if (e.origin === "http://portal.example.com:9999") {
        document.getElementById("status").textContent = e.data;
    } else {
        // 忽略其他源发来的消息
    }
}
/**
 * 编写用于与门户通信的函数。
 **/
function sendString(s) {
    window.top.postMessage(s, targetOrigin);
}
```

使用postMessage API 创建应用

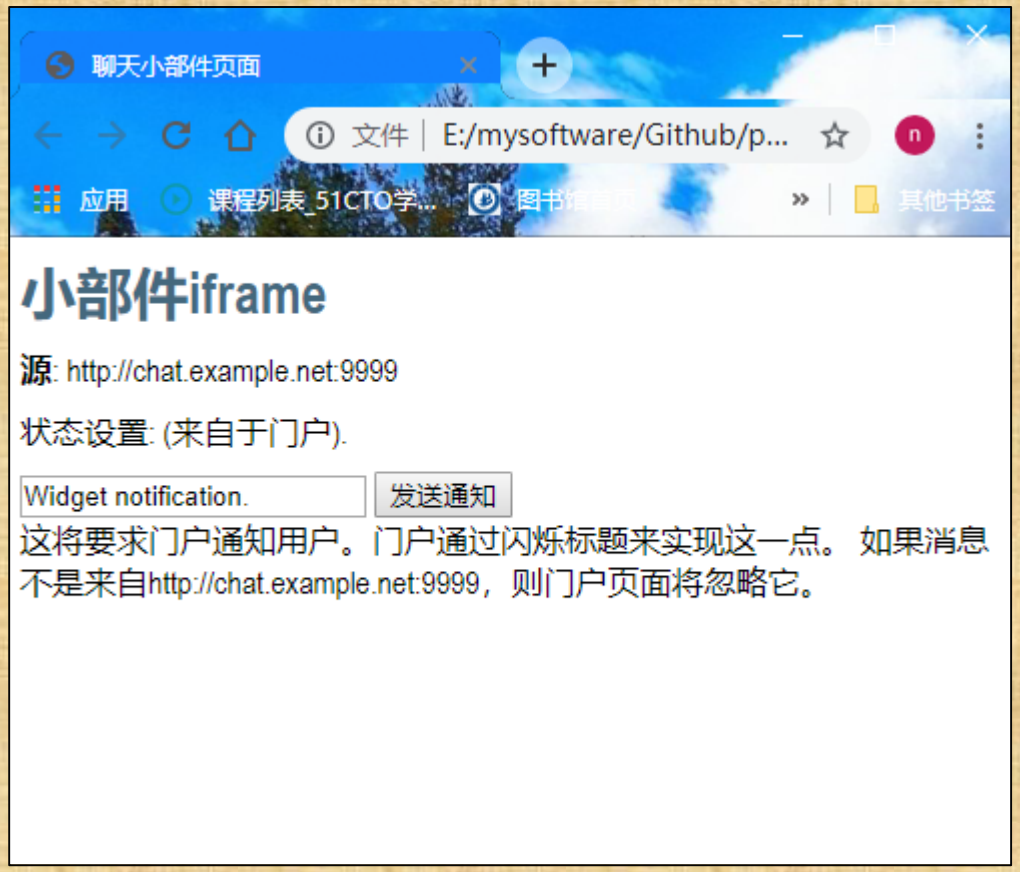
```
/**
 * 封装一个向门户页面发送通知的函数，绑定点击事件
 **/
function loadDemo() {
    document.getElementById("actionButton").addEventListener("click",
        function() {
            var messageText = document.getElementById("messageText").value;
            sendString(messageText);
        }, true);

}
/**
 * 为window绑定load加载事件和message消息事件监听器
 **/
window.addEventListener("load", loadDemo, true);
window.addEventListener("message", messageHandler, true);
```

使用postMessage API 创建应用

```
</script>
<h1>小部件iframe</h1>
<p><b>源</b>: http://chat.example.net:9999</p>
<p>状态设置: <strong id="status"></strong> (来自于门户).<p>

<div>
    <input type="text" id="messageText" value="Widget notification.">
    <button id="actionButton">发送通知</button>
</div>

<p>
    这将要求门户通知用户。门户通过闪烁标题来实现这一点。
    如果消息不是来自http://chat.example.net:9999，则门户页面将忽略它。
</p>
```

使用postMessage API 创建应用

此时代码的浏览效果如图所示：



使用postMessage API 创建应用

- 4、部署应用

示例应用的运行依赖条件：（1）页面需要部署在Web服务器上；（2）两个页面必须来自不同的域。

在本机部署运行示例应用，使用Python的SimpleHTTPServer Web服务器：

（1）下载并安装Python，https://www.python.org/ ；

（2）更新Windows系统下的hosts文件；

文件目录为：C:\Windows\System32\drivers\etc\hosts

使用postMessage API 创建应用

在该文件下增加两条指向localhost的记录：

```
#          127.0.0.1    chat.example.net
#          127.0.0.1    portal.example.com
```

（3）利用DOS控制台进入示例文件的目录下；

（4）启动Web服务器：

python –m http.server 9999

（5）打开浏览器，输入

http://protal.example.com:9999/postMessagePortal.html

6.2 XMLHttpRequest Level 2

- 跨源XMLHttpRequest
- 进度事件
- HTML5 XMLHttpRequestLevel 2的浏览器支持情况

6.2 XMLHttpRequest Level 2

XMLHttpRequest是一个浏览器接口，开发者可以使用它提出HTTP和HTTPS请求，而且不用刷新页面就可以修改页面的内容。其最常见的应用是提交表单和获取额外的内容。

使用XMLHttpRequest对象可以实现下面的功能：

- 在不重新加载页面的情况下跟新页面；
- 在页面已加载后从服务器请求数据；
- 在页面已加载后从服务器接收数据；
- 在后台向服务器发送数据。

6.2 XMLHttpRequest Level 2

XMLHttpRequest API使Ajax技术的实现成为了可能。

AJAX = Asynchronous JavaScript and XML（异步的 JavaScript 和 XML）。

通过在后台与服务器进行少量数据交换，AJAX 可以使网页实现异步更新。这意味着可以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。

传统的网页（不使用 AJAX）如果需要更新内容，必需重载整个网页面。

跨源XMLHttpRequest

过去，XMLHttpRequest仅限于同源通信，即open()函数中使用相对路径，请求方和响应方同源，Level 2通过CORS实现了跨源的XMLHttpRequest 。

跨源的HTTP请求包括一个origin头部，为服务器提供http请求的源信息。

通过跨源的XMLHttpRequest可以整合来自不同源的内容。相对于使用服务器端对来自不同源的内容进行整合，具有一定优势。

```
xmlhttp.open("GET","demo_get2.asp?fname=Bill&lname=Gates",true);
```

```
xmlhttp.open("GET","http://www.example.com/
    demo_get2.asp?fname=Bill&lname=Gates",true);
```

跨源XMLHttpRequest

代码清单6-5　请求的头部示例

```
POST /main HTTP/1.1
Host: www.example.net
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.1.3) Gecko/20090910 Ubuntu/9.04
(jaunty) Shiretoko/3.5.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/
Origin: http://www.example.com
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 0
```

Origin：Http://www.example.com

跨源XMLHttpRequest

服务器端对于CORS的支持，主要就是通过设置Access-Control-Allow-Origin来进行的。如果浏览器检测到相应的设置，就可以允许Ajax进行跨域的访问。

```
HTTP/1.1 201 Created
Transfer-Encoding: chunked
Server: Kaazing Gateway
Date: Mon, 02 Nov 2009 06:55:08 GMT
Content-Type: text/plain
Access-Control-Allow-Origin: http://www.example.com
Access-Control-Allow-Credentials: true
```

Access-Control-Allow-Origin：Http://www.example.com

进度事件

在XMLHTTPRequest之前的版本中，仅有readystatechange一个事件能够被用来响应进度。并且浏览器对事件的实现并不兼容，如在IE浏览器中永远都不会触发readyState 3。此外，readyState的更改事件缺乏与上传进程通信的方法。

```
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState==4 && xmlhttp.status==200){
        document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
    }
}
```

进度事件

表 readystatechang事件响应

属性	描述
onreadystatechange	存储函数，每当readyState属性改变时，就会调用改函数
readyState	存有XMLHttpRequest的状态。从0到4发送变化
	0:请求未初始化
	1:服务器连接已建立
	2:请求已接收
	3:请求处理中
	4:请求已完成，且响应已就绪
status	200:“OK”
	404:未找到页面

进度事件

XMLHttpRequest Level 2使用Progress进度来命名进度事件。通过为事件处理程序属性设置回调函数，可以实现对这些事件的监听。

表 XMLHttpRequest Level 2中的进度事件

事件名称	描述
loadstart	传输已经开始
progress	在传输中
error	传输失败
abort	传出被用户取消
timeout	进程终止，因为预定的时间到期
load	传输成功
loadend	传输结束，但不知成功or失败

HTML5 XMLHttpRequestLevel 2的浏览器支持情况

查看浏览器详细支持情况：https://caniuse.com/。

表 主流浏览器支持情况

浏览器	支持情况
IE	IE9以上版本支持
Edge	支持
Firefox	Firefox 6以后版本支持
Chrome	Chrome 29以后版本支持
Safari	Safari 5.1以后版本支持
Opera	支持

6.3 使用XMLHttpRequest API

- 1、浏览器支持情况检测

在使用XMLHttpRequest Level 2功能——如跨源支持之前，首先要检测浏览器是否支持该功能。常用的做法是检测XMLHttpRequest对象中是否存在withCredentials属性。

```
var xhr = new XMLHttpRequest()
if (typeof xhr.withCredentials === undefined) {
    document.getElementById("support").innerHTML =
    "Your browser <strong>does not</strong> support cross-origin XMLHttpRequest";
} else {
    document.getElementById("support").innerHTML =
    "Your browser <strong>does</strong> support cross-origin XMLHttpRequest";
}
```

6.3 使用XMLHttpRequest API

- 2、构建跨源请求

为了构建跨源XMLHttpRequest, 首先要创建一个新的XMLHttpRequest对象：

```
var crossOriginRequest = new XMLHttpRequest()
```

接下来，通过指定不同源的地址来构造跨源XMLHttpRequest：

```
crossOriginRequest.open("GET","http://www.example.net/stockfeed",true);
```

6.3 使用XMLHttpRequest API

- 3、使用进度事件

在表示请求和响应的不同阶段，XMLHttpRequest不在使用数值型状态表示法，而是提供了命名进度事件。

```
crossOriginRequest.onprogress = function (e) {
    var total = e.total;
    var loaded = e.loaded;
    if(e.lengthComputable){
        // 利用进度信息做些事情
    }
}
crossOriginRequest.upload.onprogress = function (e) {
    var total = e.total;
    var loaded = e.loaded;
    if(e.lengthComputable){
        // 利用进度信息做些事情
    }
}
```

6.3 使用XMLHttpRequest API

- 4、二进制数据

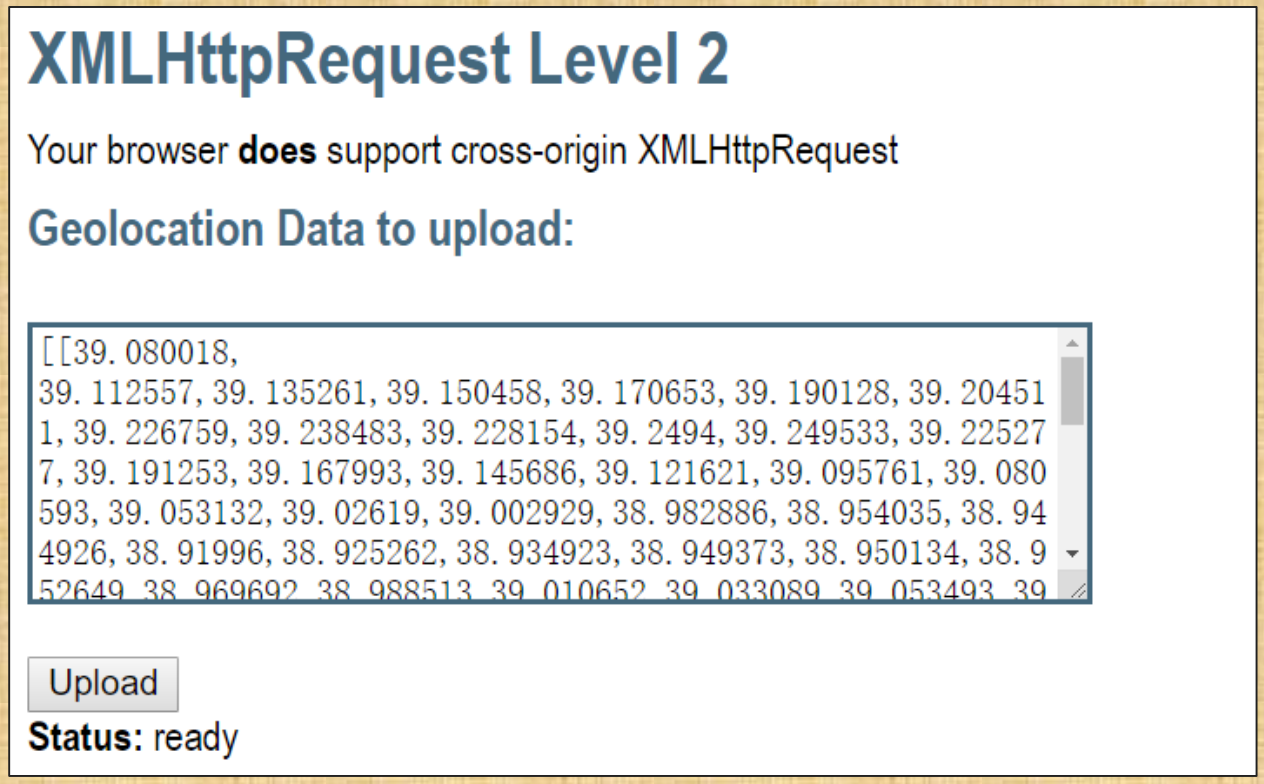
支持新的二进制API的浏览器可能会使用XMLHttpRequest来发送二进制数据。XMLHttpRequest Level 2规范支持调用send()方法发送Blod和ArrayBuffer对象。

```
var a = new Uint8Array([8,6,7,5,3,0,9]);
var xhr = new XMLHttpRequest();
xhr.open("post","/data/",true)
console.log(a)
xhr.send(a.buffer);
```

生成一个二进的HTTP POST请求，内容字节为8、6、7、5、3、0、9。

6.4 创建XMLHttpRequest 应用

在这个示例中，将赛事位置坐标上传到非同源的Web服务器端，并使用新的进度事件监控包括上传进度在内的HTTP请求的状态。示例效果图如下：



6.4 创建XMLHttpRequest 应用

创建HTML文件crossOriginUpload.html。

（1）创建一个新的XMLHttpRequest对象：

```
var xhr = new XMLHttpRequest()
```

（2）检测浏览器是否支持跨源XMLHttpRequest：

```
if (typeof xhr.withCredentials === undefined) {
    document.getElementById("support").innerHTML =
    "Your browser <strong>does not</strong> support cross-origin XMLHttpRequest";
} else {
    document.getElementById("support").innerHTML =
    "Your browser <strong>does</strong> support cross-origin XMLHttpRequest";
}
```

6.4 创建XMLHttpRequest 应用

（3）设置回调函数以处理进度事件，并计算上传和下载的完成率。

```
xhr.upload.onprogress = function(e) {
    var ratio = e.loaded / e.total;
    setProgress(ratio + "% uploaded");
}

xhr.onprogress = function(e) {
    var ratio = e.loaded / e.total;
    setProgress(ratio + "% downloaded");
}
```

```
xhr.onload = function(e) {
    setProgress("finished");
}

xhr.onerror = function(e) {
    setProgress("error");
}
```

6.4 创建XMLHttpRequest 应用

（4）打开请求并发送包含编码后的地理位置数据的字符串。由于目标位置的URL与当前页面不同源，所以这是一个跨源请求。

```
var targetLocation = "http://geodata.example.net:9999/upload";
```

```
xhr.open("POST", targetLocation, true);

geoDataString = dataElement.textContent;
xhr.send(geoDataString);
```

6.4 创建XMLHttpRequest 应用

完整代码：

```
<!DOCTYPE html>
<title>Upload Geolocation Data</title>
<meta charset="utf-8">
<link rel="stylesheet" href="styles.css">
<link rel="icon" href="http://apress.com/favicon.ico">
<script>

function loadDemo() {
    var dataElement = document.getElementById("geodata");
    dataElement.textContent = JSON.stringify(geoData).replace(",", ", ", "g");

    // 浏览器支持情况检测
    var xhr = new XMLHttpRequest()  // 创建一个新的XMLHttpRequest对象
    if (typeof xhr.withCredentials === undefined) {
        document.getElementById("support").innerHTML =
        "Your browser <strong>does not</strong> support cross-origin XMLHttpRequest";
```

6.4 创建XMLHttpRequest 应用

```
} else {
    document.getElementById("support").innerHTML =
    "Your browser <strong>does</strong> support cross-origin XMLHttpRequest";
}

var targetLocation = "http://geodata.example.net:9999/upload";

function setProgress(s) {
    document.getElementById("progress").innerHTML = s;
}

document.getElementById("sendButton").addEventListener("click",
    function() {
        xhr.upload.onprogress = function(e) {
            var ratio = e.loaded / e.total;
            setProgress(ratio + "% uploaded"); // 计算上传的完成率
        }
```

6.4 创建XMLHttpRequest 应用

```
        xhr.onprogress = function(e) {
            var ratio = e.loaded / e.total;
            setProgress(ratio + "% downloaded"); // 计算下载的完成率
        }

        xhr.onload = function(e) {
            setProgress("finished");
        }

        xhr.onerror = function(e) {
            setProgress("error");
        }

        // 通过指定不同源的地址来构造跨源XMLHttpRequest
        xhr.open("POST", targetLocation, true);

        geoDataString = dataElement.textContent;
        xhr.send(geoDataString);
    }, true);
```

6.4 创建XMLHttpRequest 应用

```
}
window.addEventListener("load", loadDemo, true);

</script>

<h1>XMLHttpRequest Level 2</h1>
<p id="support"></p>

<h4>Geolocation Data to upload:</h4>
<textarea id="geodata">
</textarea>

<button id="sendButton">Upload</button>

<p>
    <b>Status: </b> <span id="progress">ready</span>
</p>
<script>
```

6.4 创建XMLHttpRequest 应用

```
geoData = [[39.080018000000003, 39.112557000000002, 39.135261, 39.150458,
            39.170653000000001,39.190128000000001, 39.204510999999997,
            39.226759000000001, 39.238483000000002, 39.228154000000004,
            39.249400000000001, 39.249533, 39.225276999999998,
            39.191253000000003, 39.167993000000003, 39.145685999999998,
            39.121620999999998,39.095761000000003, 39.080593,
            39.053131999999998, 39.02619, 39.002929000000002,
            38.982886000000001, 38.954034999999998, 38.944926000000002,
            38.919960000000003, 38.925261999999996, 38.934922999999998,
            38.949373000000001, 38.950133999999998, 38.952649000000001,
            38.969692000000002,  38.988512999999998, 39.010652, 39.033088999999997,
            39.053493000000003, 39.072752999999999],
            [-120.15724399999999, -120.15818299999999, -120.15600400000001,
            -120.14564599999999,  -120.141285, -120.10889900000001,
            -120.09528500000002, -120.077596, -120.045428, -120.0119,
            -119.98897100000002,-119.95124099999998, -119.93270099999998,
            -119.927131, -119.92685999999999, -119.92636200000001,
            -119.92844600000001, -119.911036, -119.942834, -119.94413000000002,
            -119.94555200000001, -119.95411000000001, -119.941327, -119.94605900000001,
            119.97527599999999, -119.99445, -120.028998, -120.066335,
            -120.07867300000001, -120.089985, -120.112227, -120.09790700000001,
            -120.10881000000001, -120.116692, -120.117847,
            -120.11727899999998, -120.14398199999999]];
```

6.4 创建XMLHttpRequest 应用

（5）部署应用

依赖条件：1)页面不能同域；2)目标页面必须由能够解析CORS头部的Web服务器来提供。

配置步骤：

1)更新主机文件，并添加两个指向localhost的项

C:\Windows\System32\drivers\etc\hosts

```
#       127.0.0.1       portal.example.com
#       127.0.0.1       geodata.example.net
```

6.4 创建XMLHttpRequest 应用

2)安装包括轻量级SimpleHTTPServer Web 服务器的Python环境。

3)打开包含示例文件和CORS系统服务器Python的目录。

4)运行Python脚本：

python CORSServer. py 9999

5)在浏览器中打开网页即可。

6.5 进阶功能

- 结构化数据
- Framebusting

结构化数据

postMessage支持字符串、js对象、canvas imageData、和文件等其他数据类型。因浏览器不同而支持的情况也不同。例如IE8/9仅支持字符串。

Framebusting

Framebusting技术可以用来保证某些内容不被加载到iframe中，即防止网页被别人iframe内嵌。

```
if(window !==window.top){
    Window.top.location  = location;
}
```

应用程序首先检测其所在的窗口是否为最外层的窗口，如果不是，就把自己变为最上层窗口。从而实现网页不被别人iframe内嵌。

Framebusting

有选择的允许某些合作方引用自己的内容，可以通过使用postMessage在互信页面间握手通信。

```
var framebustTimer;
var timeout = 3000;

if(window !== window.top){
  framebustTimer = setTimeout(
      function(){
          window.top.location = location;
      }, timeout);
}

window.addEventListener("message", function(e){
  switch(e.origin){
      case trustedFramer:
          clearTimeout(framebustTimer);
          break;
  }
},true);
```

6.6 课后思考

- http与https的区别是什么？为什么源会不同？
- 跨源请求中分析都有哪些请求不成功的原因？

6.7 小结

- postMessage和源安全
- postMessage API
- XMLHttpRequest Level 2
- XMLHttpRequest应用