基于 HTML5 WebSocket 的在线群聊

杨令潇*(PB10000316)*　张义飞*(PB10007143)*

介绍

HTML5 中引入了 WebSocket，实现了客户端与服务器端的直接双向通信。WebSocket 的实现遵循 RFC 6455 规定[1]，包括 HTTP 握手、数据打包和状态控制等具体协议。在服务器端的配合下，可以实现浏览器中非常便利的双向数据转发，并进而实现群聊与文件传输的功能。

客户端的表现形式是网页文件（HTML），可以在服务器端静态存储。用户打开这个网页文件后，用户的浏览器与服务器通过一次 HTTP 握手建立 WebSocket 连接，之后则以 TCP 连接的形式发送数据包，不再需要 HTTP 协议。

服务器端处理握手 HTTP 报文，并返回适当的 HTTP 响应，随后则是 TCP 交流，实现由 Node.js 完成。

实验报告与所有源码都由 Github 托管：https://github.com/evojimmy/web-socket-chat

使用方法

0. 必要配置

   本机需要运行 apache 或其他静态资源服务器，以及 Node.js，并安装扩展包 socket.io（`npm install -g socket.io`）。同时客户端需要安装现代浏览器。

1. 启动服务器

   执行 `node ./server/server.js`，即启动服务器，在整个过程中控制台不会再有提示。

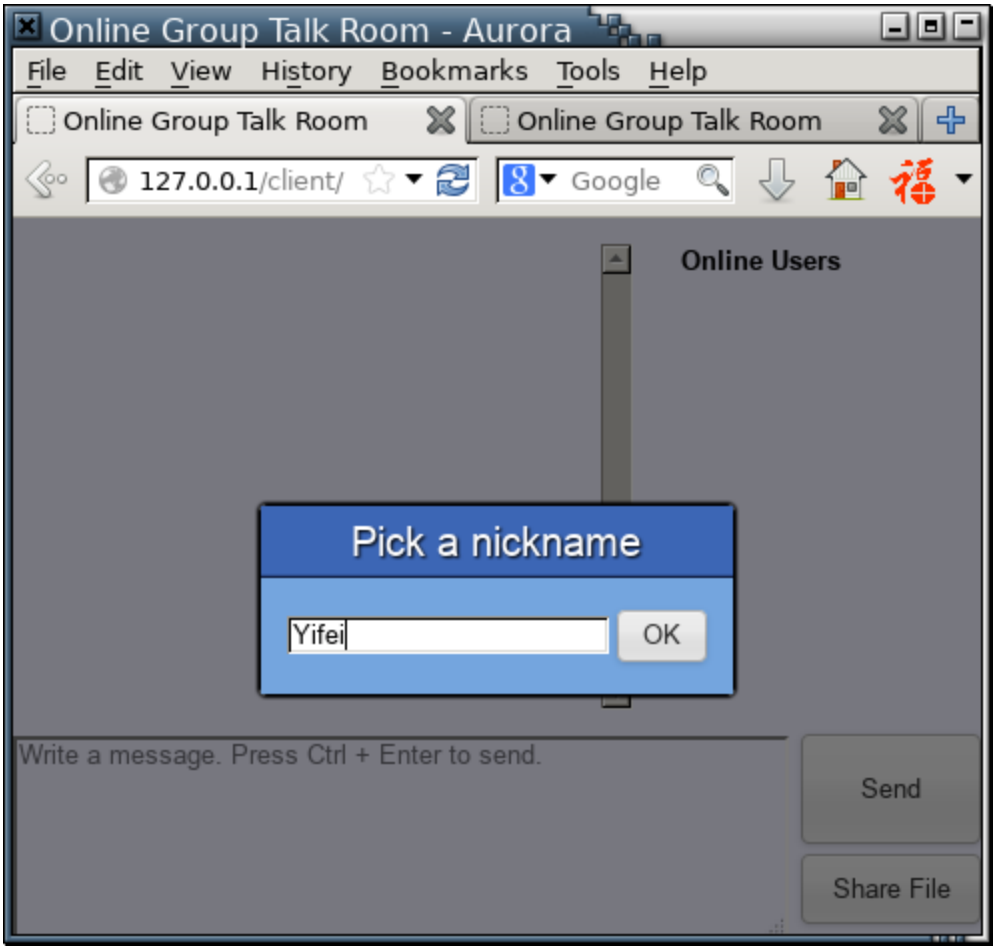
   进入 `server` 文件夹，创建 `upload` 文件夹，此文件夹用来托管用户上传的内容。

1 RFC 6455 - The WebSocket Protocol: http://tools.ietf.org/html/rfc6455

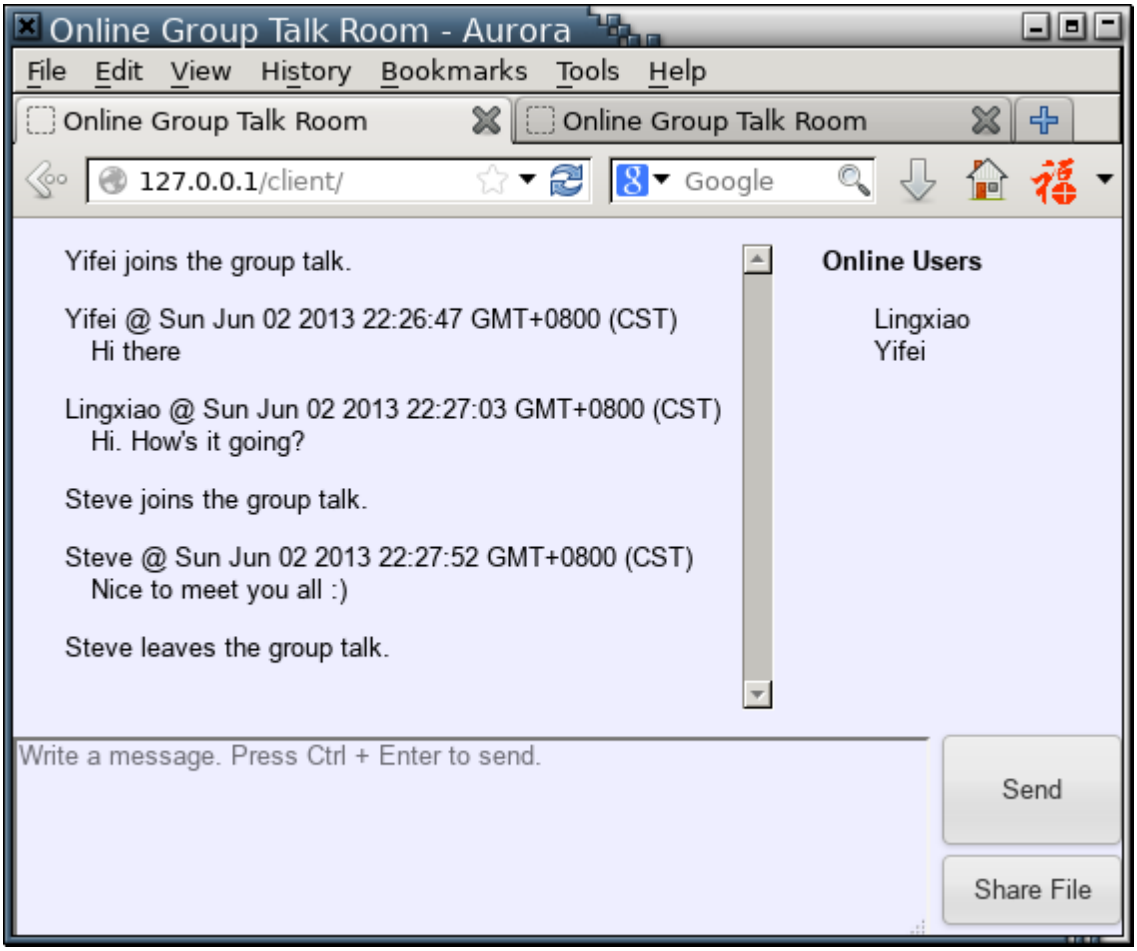
2. 访问网页应用

将 `client` 文件夹拷贝至服务器托管区域，从浏览器访问文件夹中 `index.html`，如 `http://127.0.0.1/client/index.html`。

首先需要输入昵称。



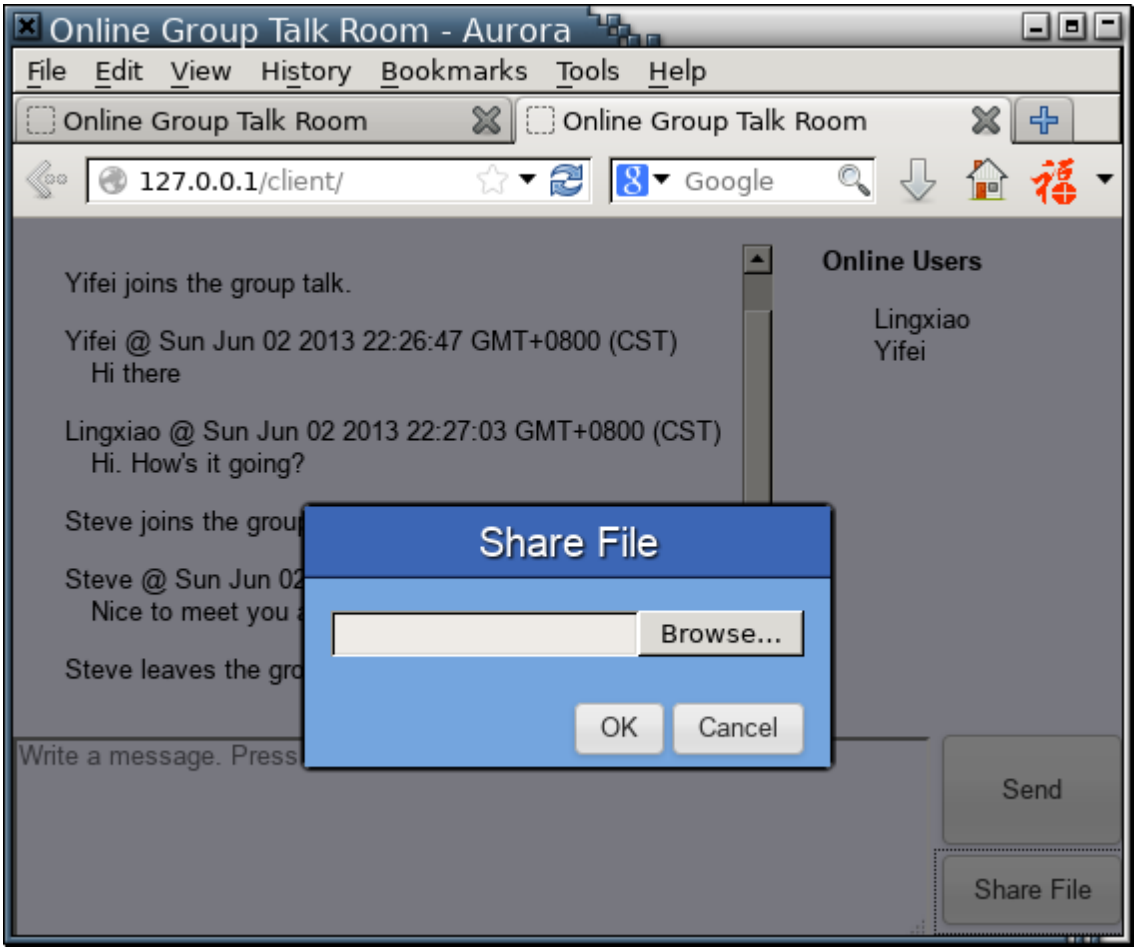
点击 OK 后进入聊天窗口。右上方为当前在线用户，在昵称上悬浮可以看到其临时服务器 ID。左上方为消息框，显示聊天内容及系统通知。左下方为输入框，右下方有发送消息按钮与文件共享按钮。



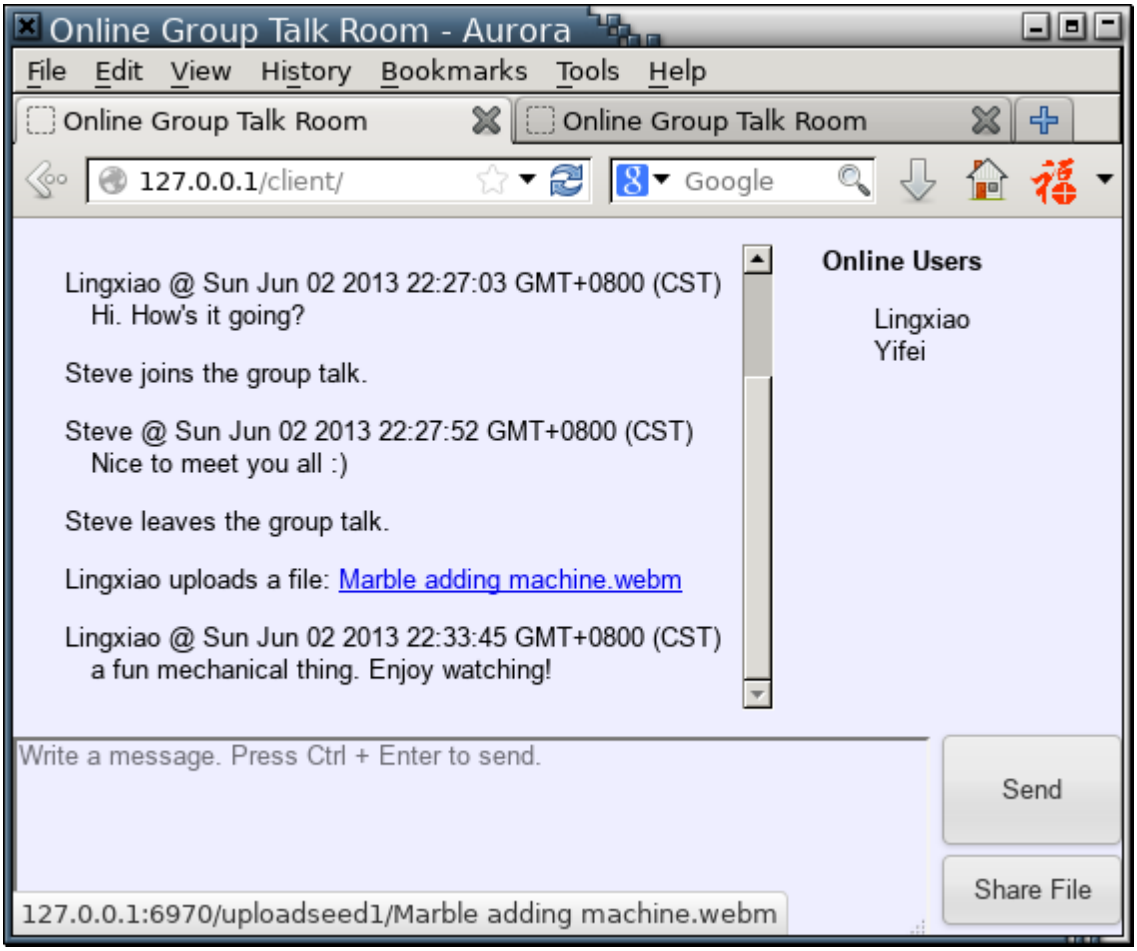
系统通知包括其他用户的加入与退出，以及文件共享。

3. 文件共享

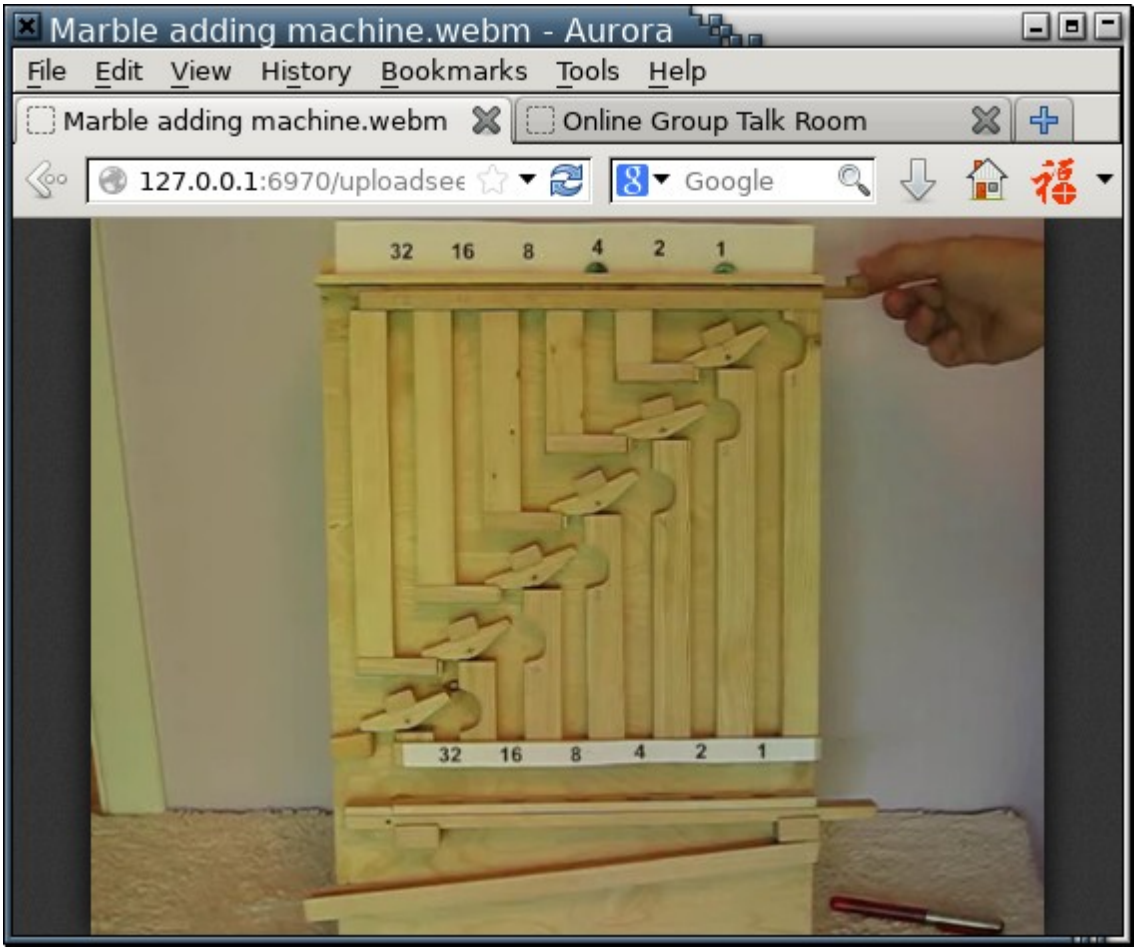
点击右下方 Share File 按钮后，弹出提示框，需要选择本地文件并点击 OK，如下所示。



上传成功后，系统对所有用户广播此文件，并以超链接的形式提供下载。



服务器同时提供文件的 MIME 类型，方便浏览器直接打开。如图中所示为 webm 格式的视频文件，HTTP 请求返回 `Content-Type: video/webm`，使之能够在浏览器中直接播放。



原理

RFC 6455: WebSocket 协议规范

HTTP 握手

首先客户端向服务器发起握手请求，报文如下所示：

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

若要协议成功，报文头部必须注明 `websocket`。更重要的一条是 `Sec-WebSocket-`

Key，这是客户端随机生成的一段 base64，用来检测服务器端是否有能力生成相对应的一段 base64。客户端会检查 HTTP 响应头部中的 Sec-WebSocket-Key，若有误则必须终止连接。

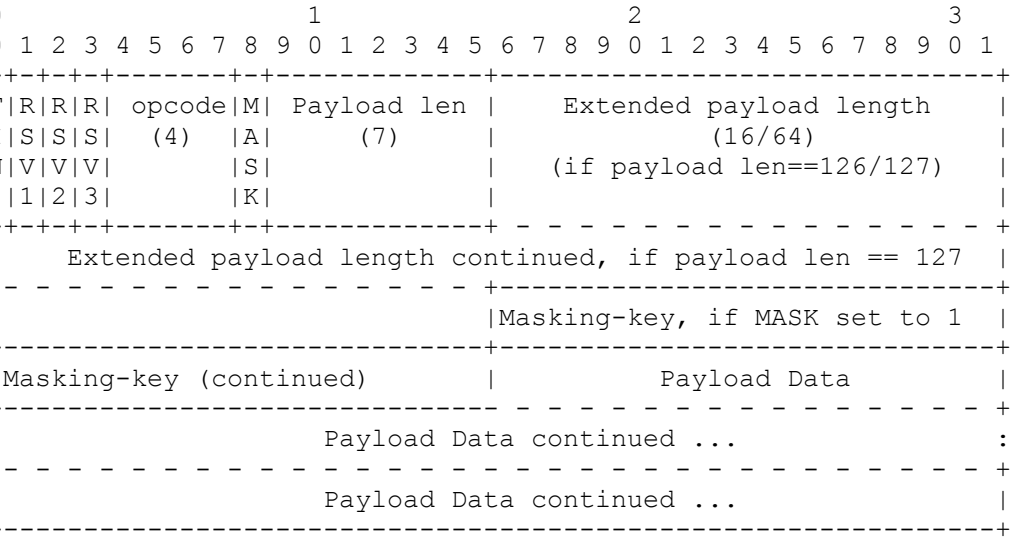
随后服务器端返回 HTTP 响应如下：

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

其状态码必须为 101，同时返回另一段 Sec-WebSocket-Accept，规则为：将请求中的 base64 字符串与字符串 258EAFA5-E914-47DA-95CA-C5AB0DC85B11 组成形成字符串 dGhlIHNhbXBsZSBub25jZQ==258EAFA5-E914-47DA-95CA-C5AB0DC85B11，再对其 base64 编码生成返回值 s3pPLMBiTxaQ9kYGzzhZRbK+xOo=。如果这一步出现错误，则握手不能成功完成。

数据与指令传输

如下所示为 RFC 6455 规定的数据帧（Frame）格式。所有的数据与指令都要封装在这样的数据帧中传输，服务器端接收到的是二进制 Buffer 类型。比如终止 WebSocket 也需要发送一个数据帧，只是 opcode 设为 0x8 而不是指示数据的 0x1。



值得一提的是，RFC 6455 规定所有数据传送都必须经过编码，并将 MASK 位置为 1，否则服务器应当终止 WebSocket 连接。此处编解码规则非常繁琐，所以最终没有继续实现 WebSocket 的底层 TCP 协议，而采用了标准封装库 socket.io。

WebSocket 统一协议：socket.io

开源项目 socket.io 针对不同浏览器的新旧技术，搭建了一套统一的 Socket 协议。现代浏览器通过 WebSocket 协议工作，旧的浏览器能够通过 Ajax 实现同样的功能。如果浏览器支持 Flash，也能够通过此插件与服务器建立连接。客户端与服务器端的 Socket 均由事件驱动，每一方可向对方发送事件（`socket.emit`），同时可监听收到的所有事件（`socket.on`）。发送事件的同时，可以附带 JSON 类型的数据，其大小不受限制。

服务器端的控制顺序为：

1. `io.sockets.listen(PORT)`

   PORT 是 socket.io 占用的端口，任何一个用户端向这个端口发起 socket.io 通信，即可被捕捉到。

2. `io.sockets.on('connection', callback)`

   如前文所述，通过 HTTP 握手协议后，触发 connection 事件，同时新增一对用户端与服务器端的独立 socket，作为回调函数的参数传入 callback。

3. `socket.on('my-event', callback)`与`socket.emit('my-event', message)`

   on 方法是监听事件，emit 方法是向这个 socket 发送事件。同时`io.sockets.emit`方法可以对所有连接到服务器的 socket 发送事件，起到广播的作用。

4. `socket.on('disconnect', callback)`

   每一对 socket 不断发送心跳信号（heartbeat signal），一旦连接断开，如用户关闭浏览器窗口，会引发心跳信号的停止，进而引起 disconnect 事件。

客户端的控制顺序为：

1. `socket = io.connect('http://127.0.0.1:6969')`

   客户端主动建立连接，并指定服务器 IP 与端口号。

2. `socket.on('my-event', callback)`与`socket.emit('my-event', message)`

类似服务器端，on 方法是监听事件，emit 方法是发送事件。只不过客户端仅能与服务器进行通信。

文件上传 API 与 base64 编码

浏览器通常通过<input type="file">输入框来获得对用户文件的访问权限。点击这个输入框，弹出选择文件的窗口。选择后浏览器即可访问这个文件。

HTML 中定义的 File API 提供了几个接口：文件名称、文件大小与文件 MIME 类型等，例子如下。

HTML:

```
<input type="file" id="choose-file">
```

Javascript:

```
var el = document.getElementById('choose-file');
el.addEventListener('change', function (e) {
    var files = e.target.files;  //e.target 与 el 等价
    var f = files[0];
    console.log(f.name);  //真实文件名
    console.log(f.size);  //字节数
    console.log(f.type);  //MimeType
}, false);
```

获得文件的 File 对象之后，还可以用 FileReader 对象获得文件的内容，在此处仅使用 readAsDataURL 方法。

Javascript:

```
var r = new FileReader;
r.onload = function (e) {
    console.log(e.target.result);  //e.target 与 r 等价
};
r.readAsDataURL(f);  //f 是上面的 File 对象
```

Data URL 是以 base64 编码表示文件内容的字符串，如对于一个短文件，其 Data URL 为：

```
data:application/zip;base64,UEsDBBQAAAAAjgAAAG91dHB1dDAxLnR4dFBLBQYAAAAABAAEAOYAAAC/AAAAAAA=
```

base64,之后的字符即为文件内容的 base64 编码。编码后不含特殊字符，可以安全传送到服务器并解码至二进制缓冲区（Binary buffer），最后写入文件。

技术实现

Socket 通信

建立在 socket.io 之上，我们仅需考虑客户端与服务器端事件的安排。下表中列出了两个方向上所有事件的意义，大致按时间排序，其中 ↑ 表示客户端到服务器端，↓ 表示服务器端到客户端。

通信方向	事件名称	功能描述
↑	connection	建立连接（socket.io 内置）
↓	hello	向客户端发送其 ID 号码
↑	register-nickname	在服务器注册昵称，与 ID 相联系
↓	message	广播消息字符串（如上线、下线）
↓	online-update	广播在线用户名单
↑	client-send	向服务器发送文字消息
↓	server-ack	广播最新消息内容与发送者等
↑	send-file	将文件内容经 base64 编码后传送至服务器
↓	msg-file	广播文件上传者、文件名与下载地址
↑	disconnect	断开连接（socket.io 内置）

文件上传与下载

如前所述，文件从客户端上传到服务器可通过 base64 编码实现。服务器收到 base64 编码之后，解码至二进制文件，并写入 `upload` 文件夹下的文件，文件名取为服务器产生的唯一数字编号，如 `upload/1`，`upload/2` 等。

获取文件通过向服务器发送类似 `http://127.0.0.1:6970/uploadseed1/Marble%20adding%20machine.webm` 的 URL 请求。实际上，这不是服务器托管文件的真实路径，而是为了方便用户按原文件名存储产生的虚拟路径。这个虚拟路径中，真实文件编号写在 `uploadseed` 之后的数字中，服务器监听 6970 端口，从 HTTP 请求中抓取 URL，将编号为 1 的文件内容写入 HTTP 响应。

同时在用户上传文件时，服务器维护每一个文件编号的 MIME 类型，以便在用户下载时

将其写入 HTTP 响应头部。

客户端界面控制

网页文件采用 YUI 3 作为开发框架，包括 CSS 重置、DOM 基础操作与事件捕捉。同时也包含模板填充，如：

Javascript:

```
var template = '<p>{user} uploads a file: <a href="{href}">{name}</a></p>';
var message = {
    user: 'Lingxiao',
    href: 'fakepath',
    name: 'robots.txt'
}
Y.Lang.sub(template, message);
// <= '<p>Lingxiao uploads a file: <a href="fakepath">robots.txt</a></p>'
```

源码文件列表

```
---- web-socket-chat
   |
   +--- README.md    Github repo 说明
   |
   +--- REPORT.md    实验报告
   |
   +--- assets    实验报告外链图片
   |
   +--- server
   |   |
   |   +--- server.js    服务器端程序
   |   |
   |   +--- server.js.old    废弃的服务器端程序
   |   |
   |   +--- upload    上传文件托管区域
   |
   ---- client
      |
      +--- index.html    HTML 文件
      |
      +--- yui.css    YUI 3 Library
      |
      +--- yui.js    YUI 3 Library
      |
      +--- style.css    页面布局与样式设置
      |
      +--- main.js    客户端主要功能
      |
      ---- socket.io    socket.io 客户端库
```

注：

- server.js.old: 能够实现 RFC 6455 规定中 WebSocket 协议的 HTTP 握手，但是不能够解析 WebSocket 数据包。
- yui.css: 集合了 CSS Reset，CSS Base，CSS Font 与 YUI Style Button。
- yui.js: 集合了 YUI Core, node, event, button 模块等。