**基于HTML5的新特性WebSocket和PHP的在线五子棋游戏**

倪云建 数字媒体技术0801 200826470116

---------------------------------------------------

9号开始查资料，并着手编写程序，有些仓促，因此还未经充分测试，程序的稳定性、性能以及安全性等各方面可能尚未考虑周全，但是通过这个游戏的编写，对Socket编程有了进一步的理解。

程序中的socket信息传输层主要借鉴了Nico Kaiser（<https://github.com/nicokaiser/php-websocket>）的phpwebsocket 类，其他参考或者借鉴的在本文末尾给出链接。

服务器端的代码均添加了（蹩脚的，英语不是很好）的注释，浏览器端由于写的比较杂乱（一开始没有想到会这么繁琐），也没怎么添加注释，可能阅读起来会有点困难。但是具体的通信指令都已经在wesocket.js文件中写明。

诸多纰漏，欢迎给予意见，谢谢！

---------------------------------------------------

# 摘要

当前，很多网站为了实现即时通讯，所用的技术都是轮询(polling)。轮询是指在特定的时间间隔（如每1秒），由浏览器对服务器发出HTTP请求，然后由服务器返回最新的数据给客服端的浏览器。这种传统的HTTP request 的模式具有很明显的缺点：

浏览器需要不断的向服务器发出请求(request)。然而HTTP 请求的header是很长，而真正要求传输的数据可能只是一个很小的值，这将带来很大的带宽压力。

相比而言，目前比较新的轮询技术是Comet – 用了AJAX。这种技术虽然可达到全双工通信，但依然需要发出请求(reuqest)。

为了解决上述问题，W3C组织在最新的HTML5草案中提出了WebSocket这一解决方案。WebSocket protocol 是HTML5新增的一种新的协议。它实现了浏览器与服务器全双工通信(full-duplex)。利用WebSocket API，浏览器和服务器只需要经过一次握手（handshake），双方之间就形成了一条快速通道。两者之间就直接可以数据互相传送。

本游戏在正是在WebSocket这一优势的基础上，完成了在线多人五子棋对战的实现。服务器（Apach/2.2.14）端主由PHP（5.3.1）编写，浏览器端主要使用jQuery这一JavaScript框架，并适当使用了少量的其他HTML5中的新特性，如Canvas, input 元素中的 placeholder属性等。

游戏在Chrome:10.0.628.0, Opera:11.00 internal, Safari:5.03 下测试通过（IE9 和 FireFox 4 beta8 尚未支持websocket)。

# 目录

[摘要 1](#_Toc282642844)

[1 游戏介绍 3](#_Toc282642845)

[1.1 游戏截图 3](#_Toc282642846)

[1.2 功能介绍 6](#_Toc282642847)

[1.2.1 实时更新在线玩家信息 6](#_Toc282642848)

[1.2.2 自动匹配对手 6](#_Toc282642849)

[1.2.3 实时通讯 7](#_Toc282642850)

[1.2.4 五子棋的互弈 7](#_Toc282642851)

[1.3 游戏流程 7](#_Toc282642852)

[2游戏实现 8](#_Toc282642853)

[1.4 关键技术 8](#_Toc282642854)

[1.4.1 WebSocket 8](#_Toc282642855)

[1.4.2 五子棋的胜负判断 11](#_Toc282642856)

[1.5 游戏架构 15](#_Toc282642857)

[1.5.1 服务器端 15](#_Toc282642858)

[1.5.2 浏览器端 19](#_Toc282642859)

[参考资料 21](#_Toc282642860)

# 游戏介绍

## 游戏截图



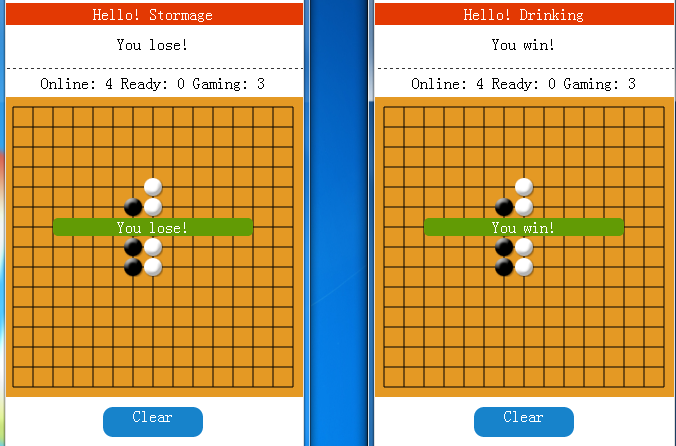
四个玩家在线，从左到右，状态分别为：游戏中，准备好，游戏中，尚未准备好



自动匹配对手后，通过roll number 决定下次次序（白子先下）



Drinking 为9，Stormage 为 8，故Drinking为白子



游戏结果



向对手发送信息



若在游戏中有一方突然断开（比如关闭网页），另一方将得到提示，并且初始化状态和棋盘

## 功能介绍

由于时间关系，悔棋，投降等功能尚未实现（虽然具体数据结构已经写好）。主要完成的功能如下。

### 实时更新在线玩家信息

一旦用户在线状态发生变化（尚未准备，准备，游戏中，断开），所有用户的页面显示均会实时更新。

### 自动匹配对手

玩家一旦处于准备状态，服务器便会自动匹配同样处于准备状态的玩家，组成一个Game，并对双方进行通知，准备开始游戏。

### 实时通讯

一旦两个玩家处于互弈状态，游戏的一方便能实时地给对方发送信息，方便玩家之间的交流。

### 五子棋的互弈

实现了五子棋的期盼状态实时更新。一旦一方下子，服务器将判断期盼状态（是否有人胜出），并将结果发送给双方。

## 游戏流程

是

是

否

否

有人胜出

准备阶段

游戏阶段

若此时任一玩家与服务器断开链接

**显示结果**

**清除棋盘**

**下子**

**是否为下子局**

**决定下子顺序**

找到对手

**准备完毕**

**尚未准备**

输入昵称

与服务器建立连接

# 2游戏实现

## 关键技术

### WebSocket

HTML5 WebSocket定义在HTML5规范的通信章节，它代表Web通信的下一个演变：通过一个单一的Socket实现一个全双工，双向通信的信道。HTML5 WebSocket提供了一个真正的标准，你可以使用它构建可扩展的实时Web应用程序。此外，由于它提供了一个浏览器自带的套接字，消除了Comet解决方案的许多问题，Web Socket显著降低了系统开销和复杂性。

#### 初始化WebSocket

[[Constructor](http://www.w3.org/TR/websockets/#dom-websocket)(in DOMString url, in optional DOMString protocol)]

这是官方给出的WebSocket对象的构造Interface，其中URL为请求的地址，如：ws://localhost:8080

其中ws为WebSocket协议，第二个参数可选，如果指定，则通知服务器使用这个子协议进行通信（若服务器不支持该协议，则链接无法建立）。具体说明可见原文：

The **WebSocket(url, protocol)** constructor takes one or two arguments. The first argument, url, specifies the URL to which to connect. The second, protocol, if present, specifies a sub-protocol that the server must support for the connection to be successful. The sub-protocol name must be a non-empty ASCII string with no control characters in it (i.e. only characters in the range U+0020 to U+007E).

因此，建立一个WebSocket链接非常简单：

var socket = new WebSocket('ws://localhost:8080');  这样我们便得到了一个WebSocket的链接套接字。以后便可以利用该套接字进行信息的传输。

#### 握手协议

当WebSocket与服务器之间的连接建立以后，还需要进行一次握手，只有成功握手后，WebSocket和服务器之间才能自由的进行通信。

WebSocket标准已经更新到[draft-ietf-hybi-thewebsocketprotocol](http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol" \o "draft-ietf-hybi-thewebsocketprotocol replaces this draft)。

从<http://tools.ietf.org/html/draft-hixie-thewebsocketprotocol-76>这一版本之后，协议降低了对握手消息的要求，以便更容易地与HTTP库进行整合。同时还引入了基于随机数的口令-应答机制来预防跨协议的攻击。这些变化使得新版本不再向下兼容，使**用**不同版本的客户端与服务器之间无法进行通讯。

下面简单介绍下新增的“基于随机数的口令-应答机制”（针对协议：<http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-00 ，最新的是4>）：

在握手阶段，浏览器向服务器发送的header如下：

GET /demo HTTP/1.1

Host: example.com

Connection: Upgrade

Sec-WebSocket-Key2: 12998 5 Y3 1 .P00

Sec-WebSocket-Protocol: sample

Upgrade: WebSocket

Sec-WebSocket-Key1: 4 @1 46546xW%0l 1 5

Origin: http://example.com

^n:ds[4U

其中从域Sec-WebSocket-Key2和域Sec-WebSocket-Key1中分别提取出数字密码：

Key1 = 4146546015

Key2 = 1299853100

并且两个域中的空格数分别为5和5，和两个数字密码相处得到

Key1 = Key1 / 5 = 829309203

Key2 = Key2 / 5 = 259970620

其中header的末尾有一串随机的字符串作为Key3：^n:ds[4U

得到的Key正好为32位的整数，转化为16进制后再转化为字符串，再加上Key3，可以计算128位的MD5值：

386a4b5327793a472a436f2c5778612d

将得到的MD5值转化为字符串，便是对服务器端发给浏览器的应答。我们来看一下服务器端的回复header：

HTTP/1.1 101 WebSocket Protocol Handshake

Upgrade: WebSocket

Connection: Upgrade

Sec-WebSocket-Origin: http://example.com

Sec-WebSocket-Location: ws://example.com/demo

Sec-WebSocket-Protocol: sample

8jKS'y:G\*Co,Wxa-

#### 握手协议应答码函数实现

游戏的具体实现中使用了Nico Kaiser已经实现好的应答码计算的部分，下面是我自己写的一个函数，思路是一样的。

function get\_digest($key1, $key2, $key\_str)

{

**// 获取key1\_num**

$key1\_num = 0;

$key1\_space = array();

$key1\_str = array();

if(preg\_match\_all('#[0-9]#', $key1, $key1\_str) && preg\_match\_all('# #', $key1, $key1\_space))

{

$key1\_str = implode('', $key1\_str[0]);

$key1\_num = $key1\_str / count($key1\_space[0]);

}

else $key1\_str = "";

**// 获取key2\_num**

$key2\_num = 0;

$key2\_space = array();

$key2\_str = array();

if(preg\_match\_all('#[0-9]#', $key2, $key2\_str) && preg\_match\_all('# #', $key2, $key2\_space))

{

$key2\_str = implode('', $key2\_str[0]);

$key2\_num = $key2\_str / count($key2\_space[0]);

}

else $key2\_num = "";

**// 合并三个16进制并转化为MD5**

if($key2\_num == "" || $key1\_num == "" || $key\_str == "")

return false;

$key\_md5 = md5(pack("N", $key1\_num) . pack("N", $key2\_num) . $key\_str);

**// 将MD5转化为字符串**

$md5\_arr = str\_split($key\_md5,2);

$digest = "";

foreach($md5\_arr as $md5\_hex)

{

$digest .= chr(hexdec($md5\_hex));

}

return $digest == "" ? false : $digest;

}

#### 其他

WebSocket复杂的部分应该就是上面的握手协议，其他的部分，如信息的收发等，熟悉JS的应该都会觉得比较简单，下面直接给出W3C草案的API部分说明：

interface **WebSocket** {

readonly attribute DOMString [URL](http://www.w3.org/TR/websockets/#dom-websocket-url);

// ready state

const unsigned short [CONNECTING](http://www.w3.org/TR/websockets/#dom-websocket-connecting) = 0;

const unsigned short [OPEN](http://www.w3.org/TR/websockets/#dom-websocket-open) = 1;

const unsigned short [CLOSED](http://www.w3.org/TR/websockets/#dom-websocket-closed) = 2;

readonly attribute unsigned short [readyState](http://www.w3.org/TR/websockets/#dom-websocket-readystate);

readonly attribute unsigned long [bufferedAmount](http://www.w3.org/TR/websockets/#dom-websocket-bufferedamount);

// networking

attribute Function [onopen](http://www.w3.org/TR/websockets/#handler-websocket-onopen);

attribute Function [onmessage](http://www.w3.org/TR/websockets/#handler-websocket-onmessage);

attribute Function [onclose](http://www.w3.org/TR/websockets/#handler-websocket-onclose);

boolean [send](http://www.w3.org/TR/websockets/#dom-websocket-send)(in DOMString data);

void [close](http://www.w3.org/TR/websockets/#dom-websocket-close)();

};

### 五子棋的胜负判断

本游戏的五子棋判断算法如下：

在一个新的子落下后，以该字为起点，沿水平，垂直，两个斜角四个方向进行检查，若出现连续五个子，则获胜。

具体实现如下：

function check\_if\_win($positon, $role) **// $position 为当前子的位置，$role为下子方**

{

$pos = explode("#", $positon);

$x = $pos[0];

$y = $pos[1];

$hor = $ver = $slo = $len = 1;

$temp\_x = $x;

$temp\_y = $y;

**// 水平方向检查**

while (true)

{

$temp\_x++;

if($temp\_x < 15)

{

if($this->chessboard[$temp\_x][$y] === $role)

{

$hor++;

if($hor >= 5)

{

return true;

}

}

else break;

}

else break;

}

$temp\_x = $x;

while (true)

{

$temp\_x--;

if($temp\_x >= 0)

{

if($this->chessboard[$temp\_x][$y] === $role)

{

$hor++;

if($hor >= 5) return true;

}

else break;

}

else break;

}

**// 垂直方向检查**

while (true)

{

$temp\_y++;

if($temp\_y < 15)

{

if($this->chessboard[$x][$temp\_y] === $role)

{

$ver++;

if($ver >= 5) return true;

}

else break;

}

else break;

}

$temp\_y = $y;

while (true)

{

$temp\_y--;

if($temp\_y >= 0)

{

if($this->chessboard[$x][$temp\_y] === $role)

{

$ver++;

if($ver >= 5) return true;

}

else break;

}

else break;

}

**// 斜角1方向检查**

$temp\_x = $x;

$temp\_y = $y;

while (true)

{

$temp\_y++;

$temp\_x++;

if($temp\_y < 15 && $temp\_x < 15)

{

if($this->chessboard[$temp\_x][$temp\_y] === $role)

{

$slo++;

if($slo >= 5) return true;

}

else break;

}

else break;

}

$temp\_x = $x;

$temp\_y = $y;

while (true)

{

$temp\_y--;

$temp\_x--;

if($temp\_y >= 0 && $temp\_x >= 0)

{

if($this->chessboard[$temp\_x][$temp\_y] === $role)

{

$slo++;

if($slo >= 5) return true;

}

else break;

}

else break;

}

**// 斜角2方向检查**

$temp\_x = $x;

$temp\_y = $y;

while (true)

{

$temp\_y--;

$temp\_x++;

if($temp\_y >= 0 && $temp\_x < 15)

{

if($this->chessboard[$temp\_x][$temp\_y] === $role)

{

$len++;

if($len >= 5) return true;

}

else break;

}

else break;

}

$temp\_x = $x;

$temp\_y = $y;

while (true)

{

$temp\_y++;

$temp\_x--;

if($temp\_y < 15 && $temp\_x >= 0)

{

if($this->chessboard[$temp\_x][$temp\_y] === $role)

{

$len++;

if($len >= 5) return true;

}

else break;

}

else break;

}

return false;

}

## 游戏架构

### 服务器端

#### 层次结构

服务器端主要可以分为三层：

主要负责ShakeHand的完成，若成功，则Connection变成一个Player，开始进入Application层的逻辑。

同时负责信息的封装，链接的断开处理等。

主要负责Socket链接的建立以及将信息发送给Connection层或者从Connetction层接受信息，发送给客户端。

Socket链接建立后，该链接变成为一个Connection

Connetcion层

Socket层

Application层

主要考虑游戏层面的业务逻辑。

该层主要有三个类gobang（负责整体的控制），game（两个玩家之间的游戏控制），player（玩家）。

#### 类详解

##### Socket层

1. **Socket**

Socket的初始化，包括绑定，监听等。

底层Socket级的信息发送。

这个类存储了服务器监听的套接字，并且具有一个所有套接字（包括服务器监听套接字和用户的链接套接字）。

1. **Server**

该类继承自Socket类，主要用于对监听接口的轮训，并和Connection层进行信息的交互和调度。

（为了使得更清楚说明函数的功能，具体实现不完全给出，具体代码请参考源代码）主要函数为：

Public function run()

{

**// 进行轮询**

while (true)

{

$changed\_sockets = $this->allsockets;

@socket\_select($changed\_sockets, $write = NULL, $exceptions = NULL, NULL);

foreach ($changed\_sockets as $socket)

{

**// 如果是服务器的监听套接字有数据读入**

**// 即有新链接到达**

if ($socket === $this->master)

{

**// 添加新链接到socket数组中**

}

// Client sockets

else

{

**// 将数据交由Connection层处理**

}

}

}

}

##### Connection层

1. **Connection**

该类主要完成和客户端之间的握手（若握手成功，则实例化一个player，并添加到Application层中去），格式化Socket层传递的数据，并发送给Application层（或者封装好Application层发来的信息，传递给Socket层），断开连接的处理等。

主要函数：

handshake($data); **// 其中$data为client发来的header**

public function onData($data)

{

**// 若已经完成握手，则进行数据的处理**

if ($this->handshaked) {

$this->handle($data);

} else {

**// 若该链接尚未完成握手**

if($this->handshake($data))

{

**// 握手成功**

$this->player = new player($this, $this->server->gobang);

$this->server->gobang->add\_player($this->player);

}

else $this->handshaked = false;

}

}

##### Application层

1. **Player**

主要存储了玩家的昵称，当前状态（未准备，准备，游戏中，游戏结束）等。若在游戏中，还将存储对手的信息。

Player还提供了聊天（发送信息给对手），改变状态等函数。具体实现就不详述了

1. **Game**

主要控制两个玩家之间的游戏逻辑。

该类存储了两个玩家的信息，游戏当前的进程状况，当前的下子局，棋盘以及历史记录。

主要的成员函数：

roll\_number($player, $num)  **// 当一个玩家投掷了随机数并发送给服务器，这个函数会// 被调用，它将决定游戏的出手先后顺序**

chess($player, $position) **// 该函数子用户下子的时候被调用，它将检查检查是否合// 法，是否有人胜出等。**

check\_if\_win($positon, $role) **// 执行五子棋胜负判断算法，返回结果。**

1. **Gobang**

主要控制整个游戏平台的业务逻辑。

主要存储了三个玩家数组，players（所有玩家），ready（处于准备状态的玩家），gaming（处于游戏中的玩家），一个games数组（所有正在进行中的游戏）。

同时负责解析下层传递的信息，并调度player或者game去执行相应的事件。

这三个类同处于Application层，其信息的传递的层次结构大体如图：

Connection层

Player

Game

Gobang

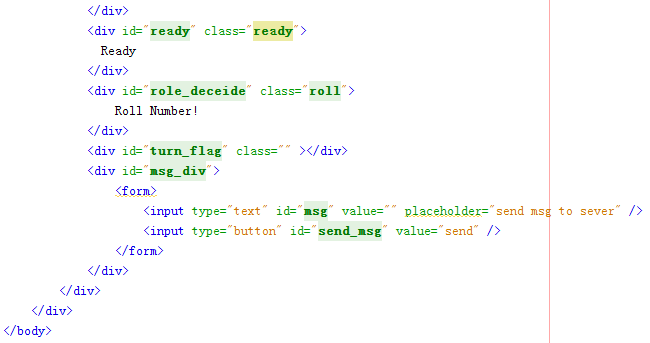
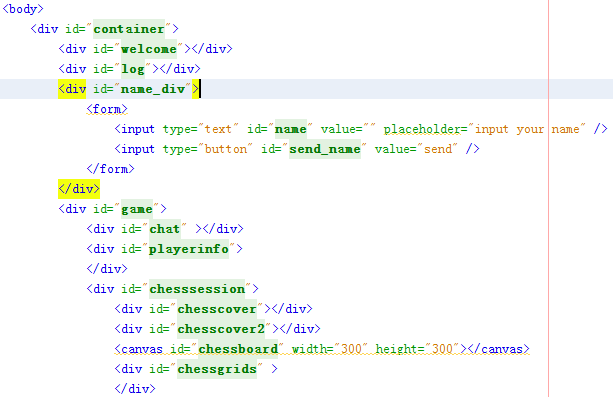
### 浏览器端

#### 结构

HTML布局部分一开始没有很好的规划，很多部分都是后来加上去的，因此整体显得比较凌乱。

整个游戏主要用了两张图片，也就是黑棋子和白棋子。棋盘使用了HTML5的新元素Canvas结合JS进行了绘制。

游戏中的“格子”均是一个个div，每个div的id代表了它的位置，如“6#7”。（本来想用Canvas来实现，但是时间比较紧，然后对Canvas也不是很熟悉，特别是设计到事件的方面，所以还是用来了div，不过以后的趋势肯定是Canvas来实现游戏和动画。）



#### 表现

CSS 部分也写的比较乱，也没写注释。

需要提一点的是，本游戏中的很多状态效果的切换都是通过添加和删除类名来实现，比如玩家状态按钮，棋盘的格子效果等。

#### 行为

游戏的交互，动画实现以及信息的传递均是由JQuery来实现。如文章的开篇所说，js部分也的甚为凌乱……

Js主要为client.js和websocket.js。前者负责整个游戏的运行，后者负责websocket实例的创建，并封装了信息的发送和接受函数。

##### WebSocket的创建以及信息的发送和接收

function creat\_socket(host, onopen, onmessage, onclose)

{

try

{

socket = new WebSocket(host); **// 创建WebSocket实例**

log('WebSocket初始化状态 '+socket.readyState);

socket.onopen = onopen; **// 链接建立回调函数**

socket.onmessage = onmessage;  **// 信息到达回调函数**

socket.onclose = onclose; **// 链接被关闭回调函数**

}

catch(ex){log(ex);}

##### 其他

在此不详述了，具体请查看源代码。

# 参考资料

[1] The WebSocket protocol - draft-ietf-hybi-thewebsocketprotocol-00

<http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-00>

[2] The Web Sockets API

<http://www.w3.org/TR/websockets/>

[3] [nicokaiser](https://github.com/nicokaiser) / [php-websocket](https://github.com/nicokaiser/php-websocket)

<https://github.com/nicokaiser/php-websocket>

[4] websocket新版协议分析＋python实现

<http://zhuowater.appspot.com/2010/08/30/webscoket-python.html>

[5] Mark Pilgrim <HTML5:Up and Running> Published by O’Reilly Media,Inc. August 2010