Webはどんどん進化している

クロスドメインアクセス
node.js
リアルタイム双方向通信
WebRTC

2014年2月1日

Webの進化

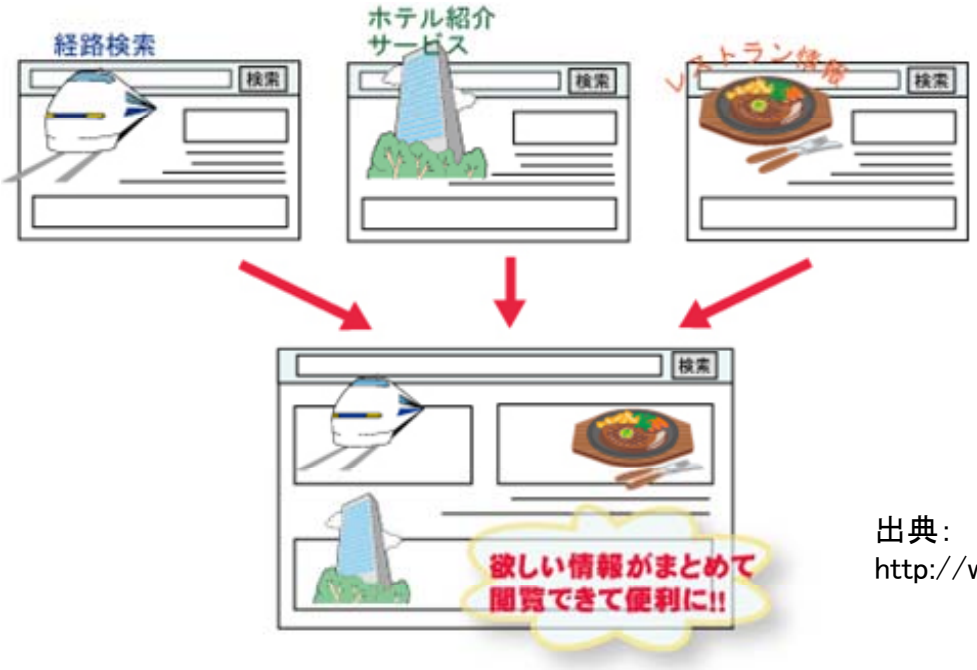
Webの進化は、いかにサーバーとやり取りするかにある。

第1世代：GET&POSTを用いたHTTP/1.1通信（はじまり）
第2世代：JavaScriptのXMLHttpRequestによる非同期な通信（Web2.0）
　WebAPI
　ページ・リロード
　クロスドメイン
第3世代：HTML5による新しいWeb（これから）
　WebStorage、IndexedDB
　WebSocket　Socket.ioによるリアルタイムな双方向通信
　SPDY　HTTP/2.0

環境の変化

マッシュアップ再考

マッシュアップは、複数の異なるサービスを組み合わせて新しいサービスを創り出す技術。GoogleMapをはじめ、幅広い分野で、手軽に利用できるサービスが提供されている。



出典:
http://www.kobelcosys.co.jp/column/itwords/45/

マッシュアップが流行ったころ、マッシュアップを利用すると、企業内情報を集約したダッシュボードを作ることも可能
と言われたが、その後・・・

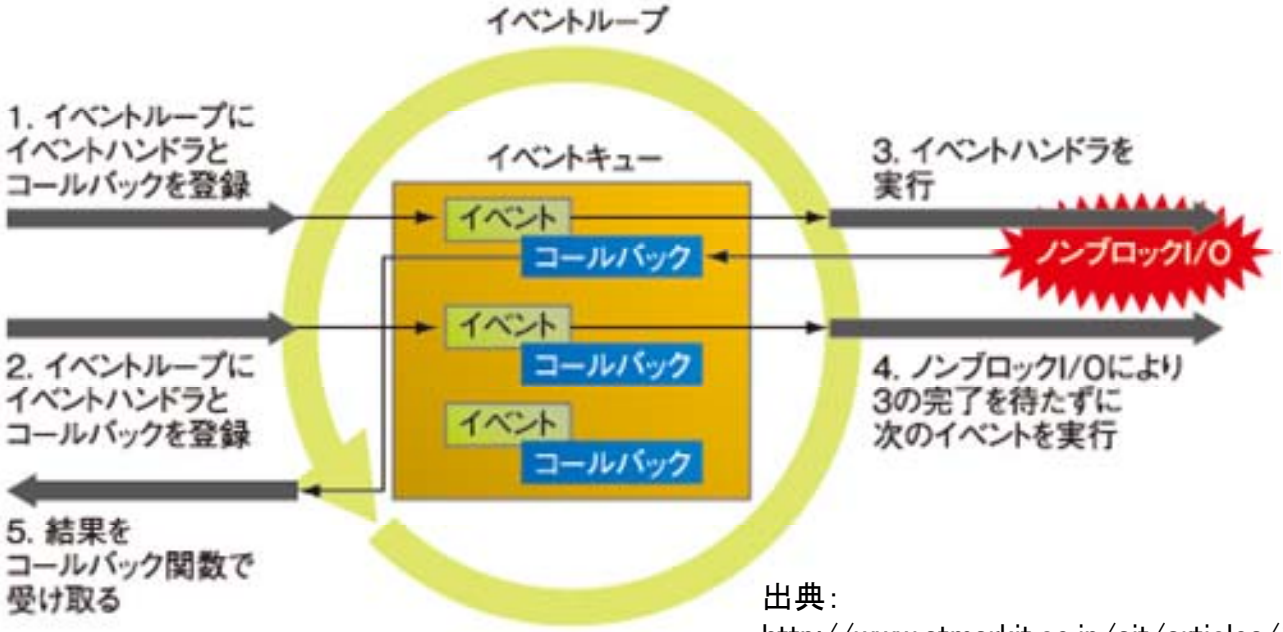
2009年生まれ
「V8」Javascriptエンジンの上で動くJavascript言語
ネットワーク通信ライブラリをベースに、
　　各種ライブラリが充実している
C10K問題でクローズアップされた

シングルスレッド、イベント駆動型、非同期

特徴
　　(1)使いやすい
　　(2)高速
　　(3)拡張性(スケーラビリティ)が高い

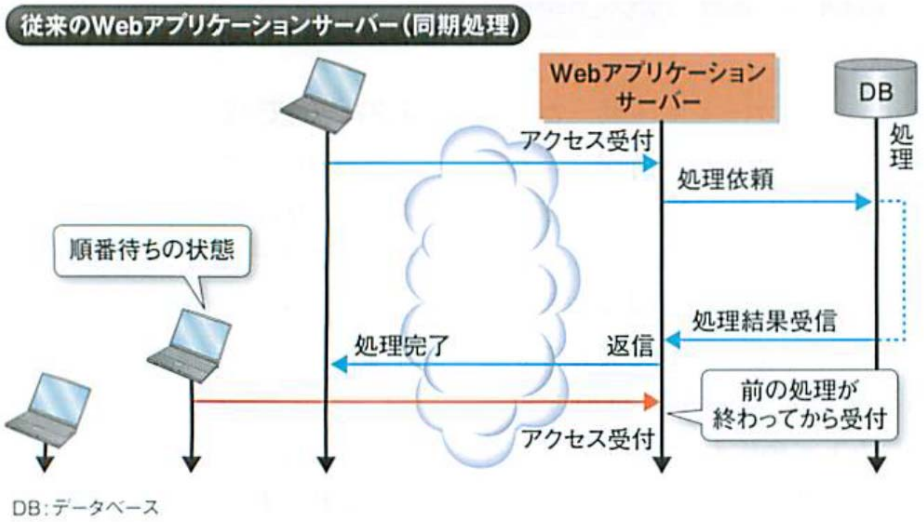
node.jsの仕組み

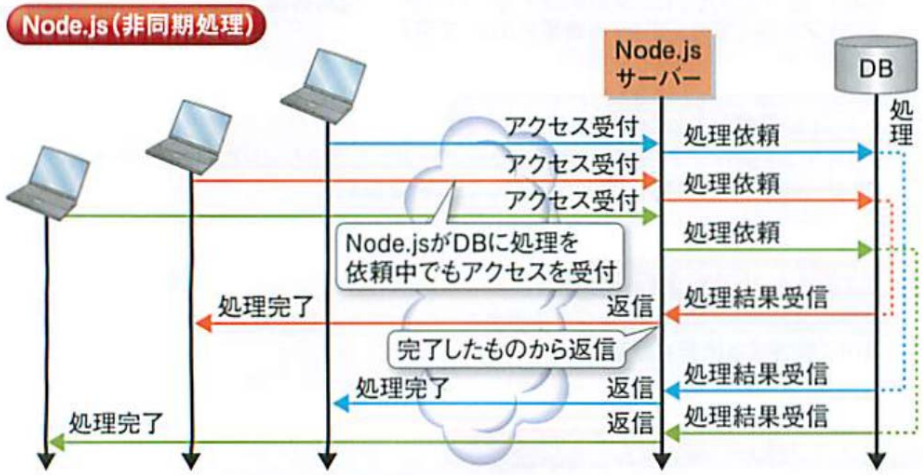
node.jsが早い理由
イベントループは、実行する処理をキューにいったん格納し、順にイベントハンドラを実行。ノンブロッキングI/Oを使い、イベントの完了を待たずに次イベントハンドラを実行させる。イベントハンドラ完了結果は、登録したコールバック関数により受け取る。



出典:
http://www.atmarkit.co.jp/ait/articles/1103/23/news101_2.html

node.jsの動き





出典:日経コンピュータ2011.12.22号

レンタルビデオ屋で例えると

スレッド・モデル

・お客様が持ってきた空パッケージを受け付け、バーコードにより、在庫を確認し、あれば、倉庫からビデオパッケージを持ってきて、・レンタル料を受け取り、貸し出しをする。
混んで来たら、レジと店員を増やす。

イベントループ・モデル

・お客様は空パッケージを受付窓口に置き、引換券を受け取る。倉庫から対応するビデオパッケージを持ってきて、
・支払窓口で、番号を呼び、レンタル料を受け取り、貸し出しをする。
混んで来たら、一生懸命走り回る。

node.jsのコードサンプル　ファイル読込

非同期コード

```
var fs = require('fs');
fs.readFile('./filetest.js', 'utf8', function(err, text) {//イベントハンドラの実行とコールバック処理
    console.log("非同期¥n"+text);
});
```

同期っぽいコード

```
var fs = require('fs');
var text = fs.readFileSync('./filetest.js', 'utf8');
console.log("同期¥n"+text);
```

node.jsのコードサンプル　HTTPサーバー


```
var express = require('express');
var http = require('http');
var path = require('path');
var app = express();

app.configure(function(){
        app.set('port', process.env.PORT || 3000);
        app.use(express.favicon());
        app.use(express.logger('dev'));
        app.use(express.bodyParser());
        app.use(express.methodOverride());
        app.use(express.static(path.join(__dirname, 'public')));
});

http.createServer(app).listen(app.get('port'), function(){
        console.log("Express server listening on port " + app.get('port'));
});
```

これだけのコードで、/public以下に置いたHTMLコンテンツが表示できるHTTPサーバーとなる。

node.jsのコードサンプル　HTTPサーバー


```
var express = require('express');
var http = require('http');
var path = require('path');
var app = express();

app.configure(function(){
        app.set('port', process.env.PORT || 3000);
        app.use(express.favicon());
        app.use(express.logger('dev'));
        app.use(express.bodyParser());
        app.use(express.methodOverride());
        app.use(express.static(path.join(__dirname, 'public')));
});
app.post('/httpserv', function(req, res) {
        if(req.param('mode')=="testJson1"){ //  /httpserv?mode=testJson1
            res.writeHead(200, {'Content-Type': 'text/javascript; charset="UTF-8"'});
                var sdata="var data="+'{"name":"山田太郎","sex":"男","age":"20"}';
            res.end(sdata);
        }
});
http.createServer(app).listen(app.get('port'), function(){
        console.log("Express server listening on port " + app.get('port'));
});
```

このようにGET、POSTメソッドに対応できる

JSON　配列


```
var person={
"users":[
        {
                "name":"山田",
                "age":24,
                "language":["XHTML","CSS"]
        },
        {
                "name":"田中",
                "age":25,
                "language":["PHP","Java"]
        },
        {
                "name":"中山",
                "age":26,
                "language":["JS","jQuery"]
        }
]};
```

```
var s="";
for(var i=0;i<person.users.length;i++){
  if(person.users[i].name=="山田")
        s+=person.users[i].name+"
"+person.users[i].age
        +" "+person.users[i].language+"¥n";
}

alert(s);
```

山田 24 XHTML,CSS

```
alert(JSON.stringify(person.users[0]));
```

{"name":"山田
","age":24,"language":["XHTML","CSS"]}

JSON　連想配列


```
var seiza={
        "牡羊座":{"date":"3/21-4/19","image":"ohitsuji.gif","yomi":"おひつじざ"},
        "牡牛座":{"date":"4/20-5/20","image":"oushi.gif","yomi":"おうしざ"},
        "双子座":{"date":"5/21-6/21","image":"futago.gif","yomi":"ふたござ"},
        "蟹座":{"date":"6/22-7/22","image":"kani.gif","yomi":"かにざ"},
        "獅子座":{"date":"7/23-8/22","image":"shishi.gif","yomi":"ししざ"},
        "乙女座":{"date":"8/23-9/22","image":"otome.gif","yomi":"おとめざ"},
        "天秤座":{"date":"9/23-10/23","image":"tenbin.gif","yomi":"てんびんざ"},
        "蠍座":{"date":"10/24-11/21","image":"sasori.gif","yomi":"さそりざ"},
        "射手座":{"date":"11/22-12/21","image":"ite.gif","yomi":"いてざ"},
        "山羊座":{"date":"12/22-1/19","image":"yagi.gif","yomi":"やぎざ"},
        "水瓶座":{"date":"1/20-2/18","image":"mizugame.gif","yomi":"みずがめざ"},
        "魚座":{"date":"2/19-3/20","image":"uo.gif","yomi":"うおざ"}
};

var a="牡羊座";
alert(a+" "+seiza[a].date+" "+seiza[a].image);
```

牡羊座　3/21-4/19 ohitsuji.gif

JSON　JSONオブジェクトの生成

JSONオブジェクトの生成

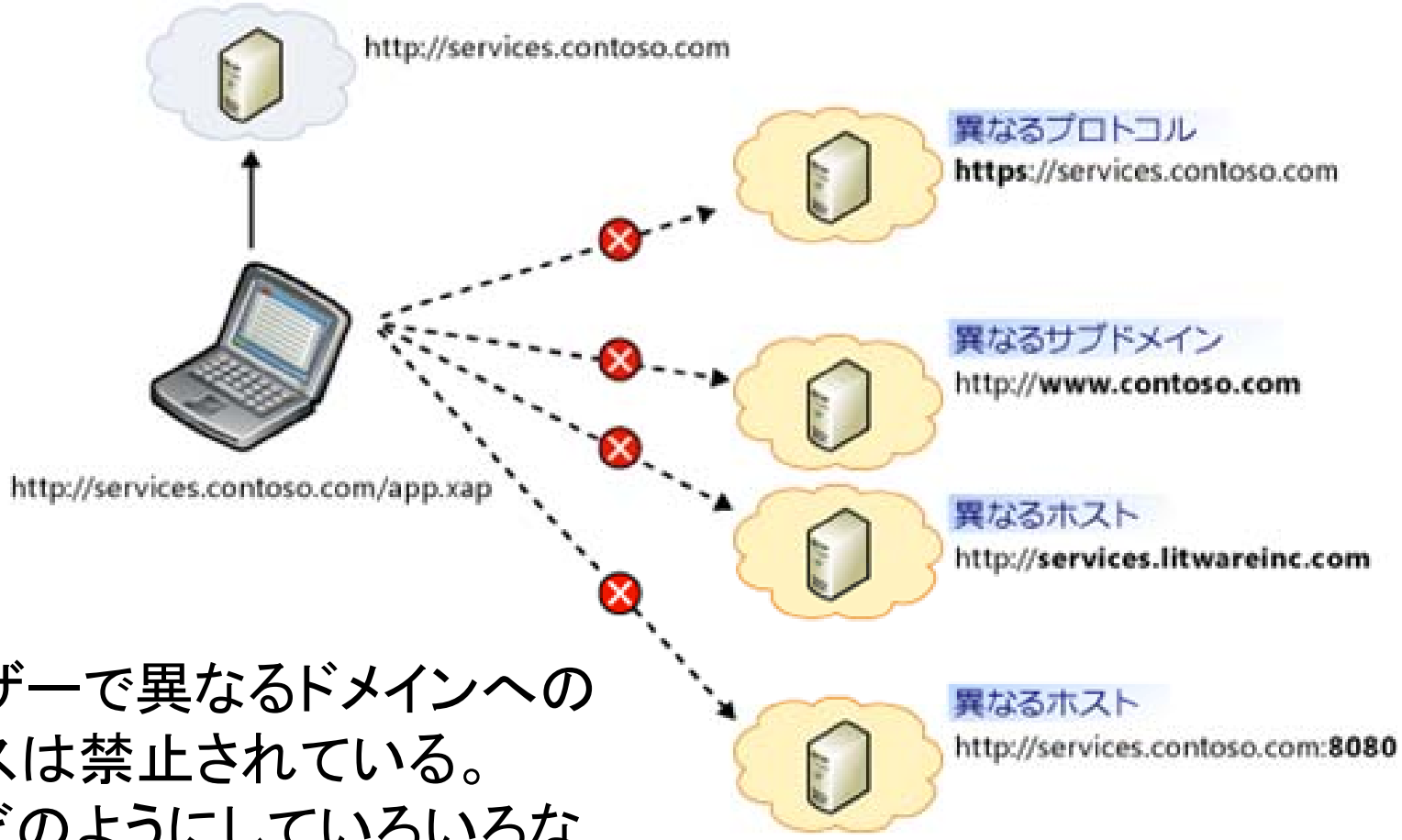
```
var comment=new Object();
    comment.message = "メッセージ";
    comment.date = new Date();
alert(JSON.stringify(comment));
        {"message":"メッセージ","date":"2014-02-01T09:00:00.000Z"}
```

文字列からJSONオブジェクトの生成

```
var json = '{"message":"メッセージ","date":"2014年2月1日","age":27}';
var comment = JSON.parse(json);
alert(JSON.stringify(comment));
        {"message":"メッセージ","date":"2014年2月1日","age":27}
```

JSONは、noSQL DBの必須要件

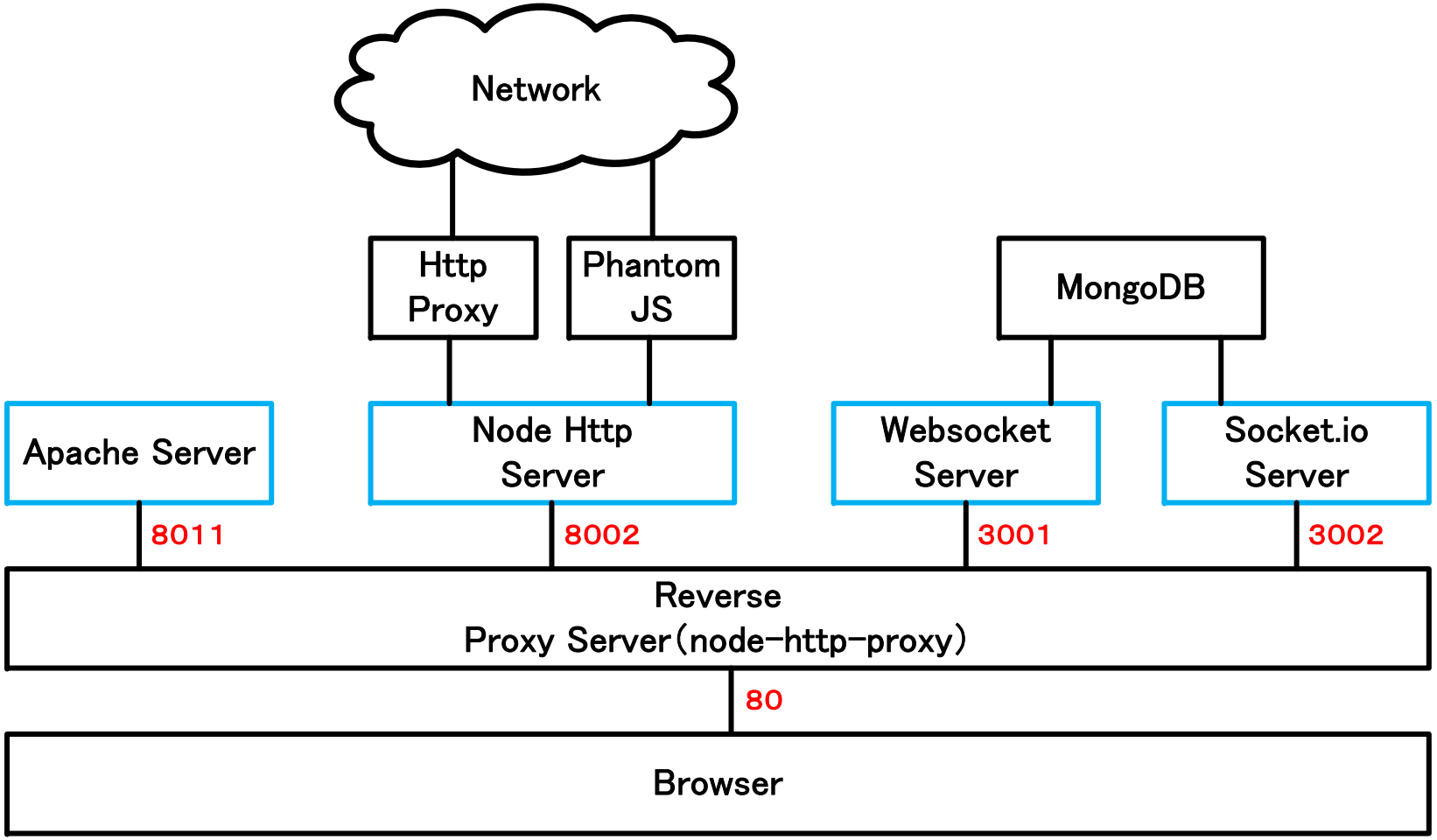
クロスドメインへのアクセス



ブラウザーで異なるドメインへのアクセスは禁止されている。
では、どのようにしていろいろなクロスサイトへのアクセスが実現できるのか？その必要性は？

出典：
http://msdn.microsoft.com/ja-jp/library/cc972657(v=vs.95).aspx

node.jsを中心したデモ環境



リアルタイム双方向通信

これからのWebでは、リアルタイム双方向通信が熱い。

Websocketは、HTML5の規格で、HTML5対応ブラウザー上で、
socket.ioは、ライブラリを組み込み、各種デバイス、各種ブラウザー上で、
それぞれリアルタイム双方向通信ができる。

WebSocket ブラウザー側


```
<script type="text/javascript">
var ws = new WebSocket("ws://localhost:3001"); // socketのopen

ws.onopen = function(v) { // コネクション接続
        console.log("connection is opened.");
}

ws.onclose = function() { // コネクション切断時
        console.log("connection is closed.");
};

ws.onmessage = function (data) { // メッセージ受信
        getMessage(data.data);
}

function sendMessage(msg) { // メッセージ送信
        ws.send("{message:"+msg+"}");
}
</script>
```

WebSocket　サーバー側


```
var WebSocketServer = require('ws').Server;
var server = http.createServer(function(req, res) { ・・・・ }
var wss = new WebSocketServer({server:server});
var connections = [];
//websocketの接続
wss.on('connection', function (ws) {
        connections.push(ws);//WebSocket接続情報の保存
//受信メッセージのブロードキャスト
        ws.on('message', function (data) {
                data=JSON.parse(data);
                broadcast(JSON.stringify(data.message));
        });
//切断イベント
        ws.on('close', function () {
                connections = connections.filter(function (conn, i) {
                        return (conn === ws) ? false : true;
                });
        });
});
//メッセージのブロードキャスト
function broadcast(message) {
        connections.forEach(function (con, i) {
                con.send(message);
        });
};
```

Socket.io　ブラウザー側


```
<script type="text/javascript" src="/socket.io/socket.io.js"></script>
<script type="text/javascript">
var socket = io.connect("http://localhost:3000"); // socketのopen

//サーバーから受け取るイベント
socket.on("connect", function () { // コネクション接続
        console.log("connection is opened.");
})

socket.on("disconnect", function (client) { // コネクション切断
        console.log("connection is closed.");
});

socket.on("message", function (data) { // メッセージ受信
        getMessage(data.message);
});

function sendMessage(msg) {
        socket.emit("message", {message:msg});
}
</script>
```

Socket.io　サーバー側

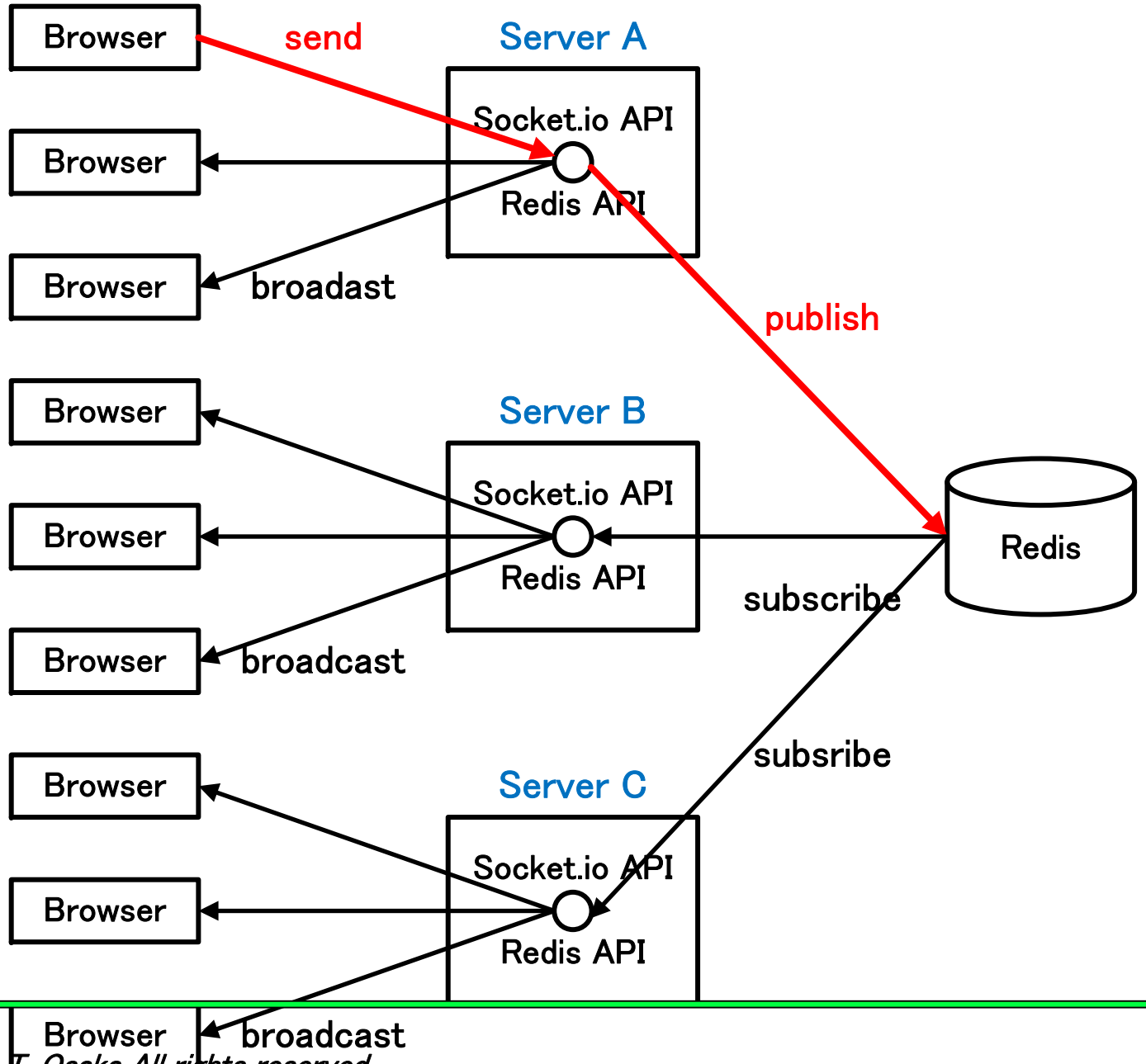

```
var socketio = require("socket.io");
var server = http.createServer(function(req, res) { ・・・・ }
var io = socketio.listen(server);
var member={};

//socket.ioの接続
io.sockets.on("connection", function (socket) {
        if(!member[socket.id])member[socket.id]=socket.id;//socket.id情報の保存

//受信メッセージのブロードキャスト
        socket.on("message", function (data) {
                socket.broadcast.emit("message", {message:data.message});
        });

//切断イベント
        socket.on("disconnect", function () {
console.log("user disconnected "+socket.id);
                delete member[socket.id];
        });
});
```

node.jsによるメッセージング　Socket.io　Redis 21

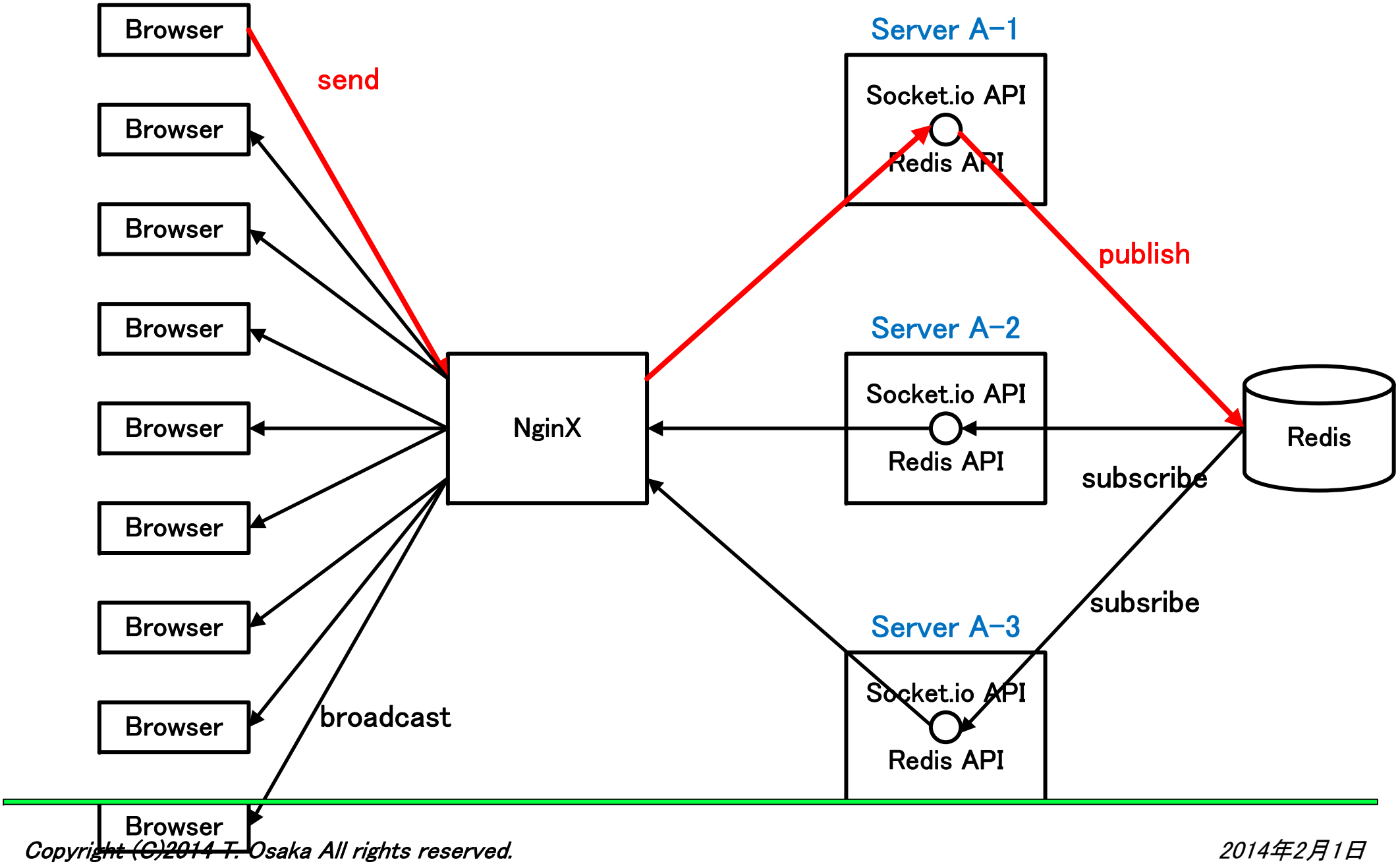
```
var server = require('http').createServer(function(req, res){
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('server connected');
});
server.listen(3010);
var io = require('socket.io').listen(server);
//RedisStoreを読み込みます
var RedisStore = require('socket.io/lib/stores/redis');
//redisサーバーの接続先情報を定義します
opts = {host:'10.0.0.200', port:6379};
//storeをRedisStoreにし、redisPub, redisSub, redisClientをredisサーバーに向けます
io.set('store', new RedisStore({redisPub:opts, redisSub:opts, redisClient:opts}));
io.sockets.on('connection', function (socket) {
  socket.emit('info', { msg: 'welcome' });
  socket.on('msg', function (msg) {
    io.sockets.emit('msg', {msg: msg});
  });
  socket.on('disconnect', function(){
    socket.emit('info', {msg: 'bye'});
  });
});
```

Socket.IOにはRedisStoreというredisに接続情報を保存するオプションがあり、分散されたnodeサーバーが同じredisサーバーを参照して、各nodeの接続情報を共有することができる。
出典：Node.jsってなんじゃ？（redisでSocket.IOをスケール）
http://memocra.blogspot.jp/2012/11/nodejsredissocketio.html

redisによるメッセージング

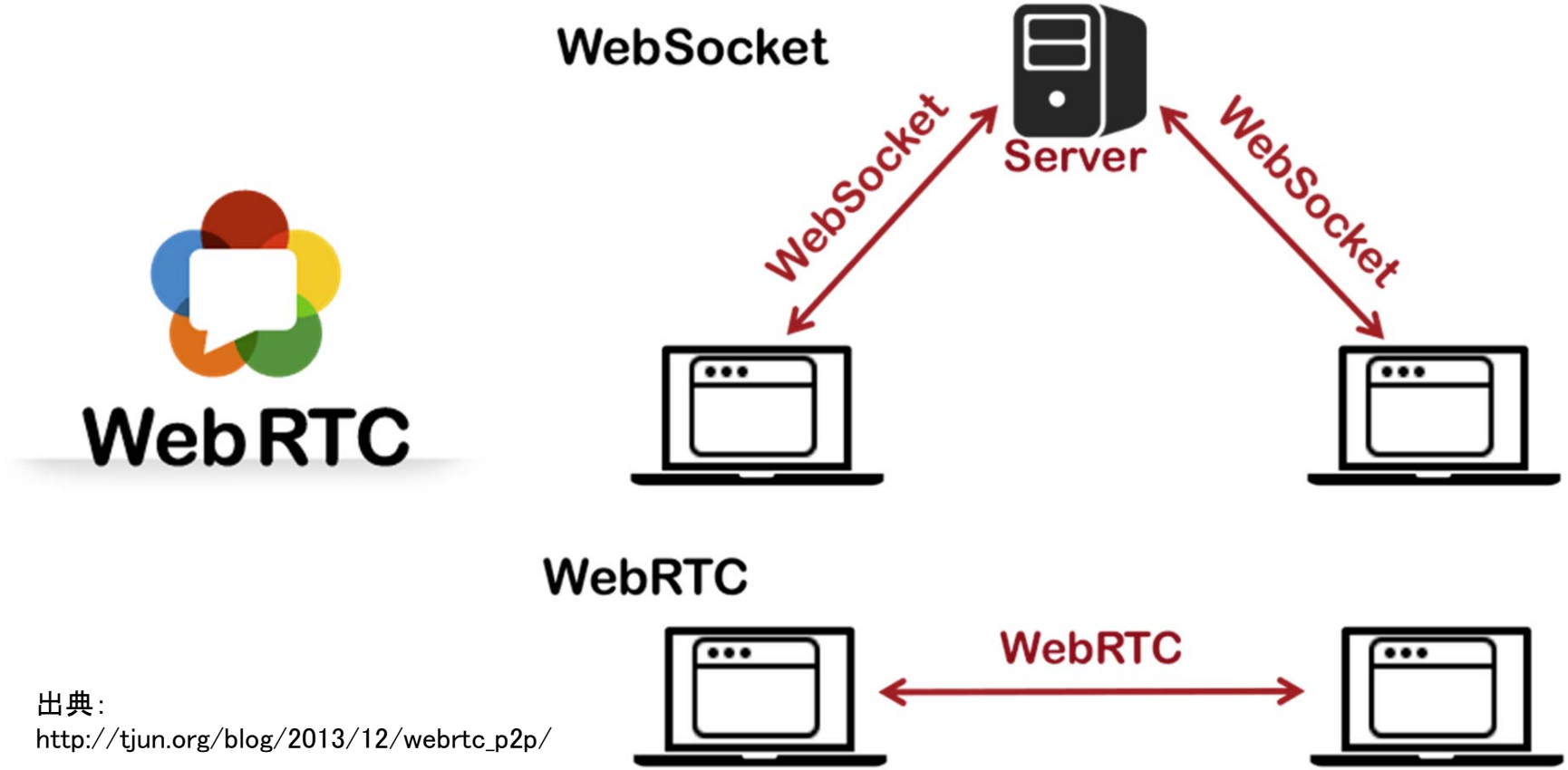


redisによるメッセージング



WebRTC (Web Real-Time Communication)とは、W3Cが提唱するリアルタイムコミュニケーション用のAPIの定義で、プラグイン無しでウェブブラウザ間のボイスチャット、ビデオチャット、ファイル共有ができる。

WebRTC: Wikipediaより引用



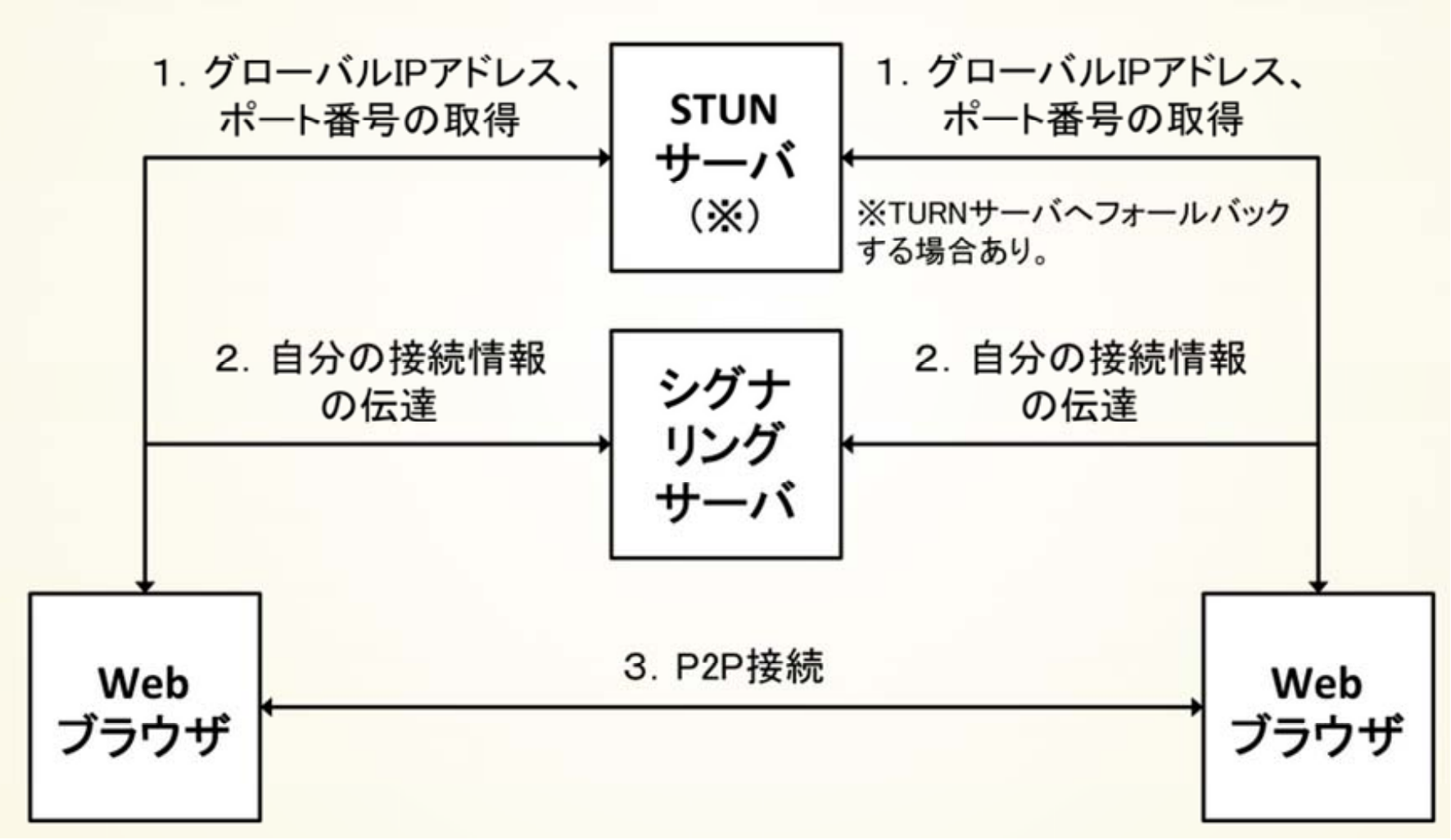
出典:
http://tjun.org/blog/2013/12/webrtc_p2p/

WebRTCの主なAPIとできること

getUserMedia
マイク・カメラから入力された音声・映像データをWebブラウザーへ取り込む
RTCPeerConnection
ブラウザー間で音声・映像データを送受信する
RTCDataChannel
ブラウザー間でバイナリーデータを送受信する

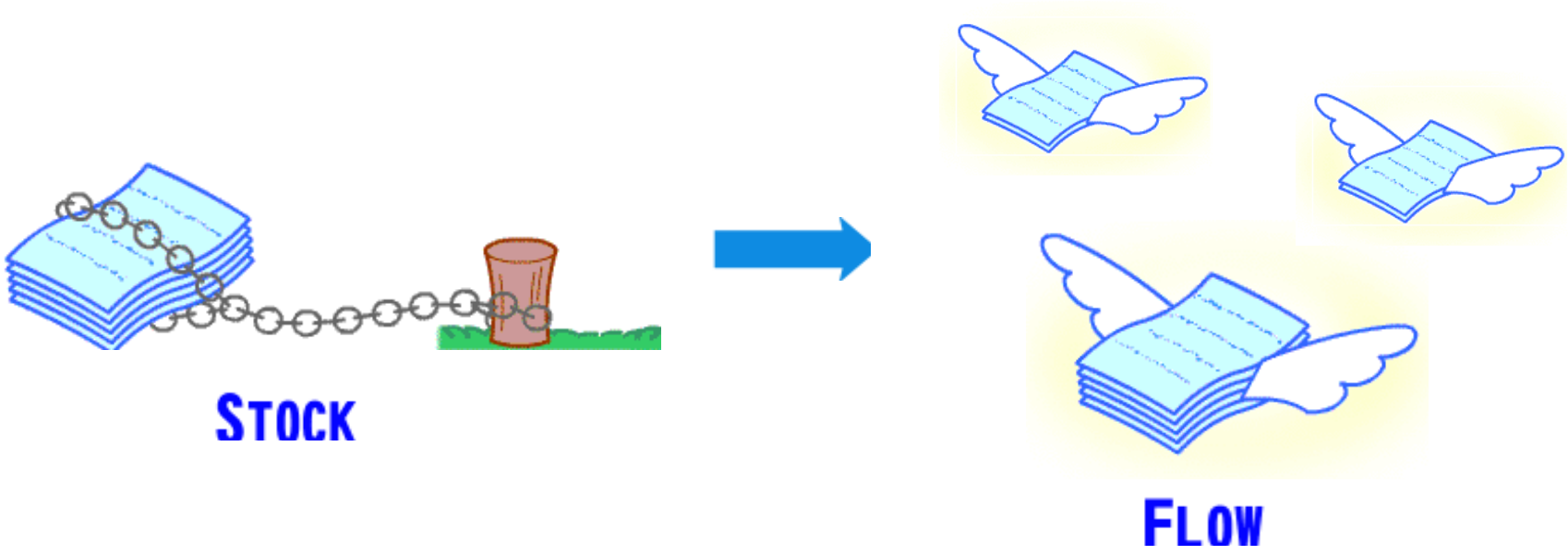
テキストチャット、ファイル共有、スクリーン共有、ゲーム、センサーデータフィード、音声・ビデオチャットなど
サーバー・ストレージの排除によって、より効率的で高速、安全なWebブラウザー間データ通信

WebRTCのP2P接続の仕組み



出典：http://yosssi.github.io/jscafe15/#/

結論　Webはどんどん進化している



大坂　哲司　osaka@fujimic.com
http://www.osaka3t.com

2014年2月1日