Pomelo

Node.js를 위한 게임 서버 프레임워크

설치

당연히 node.js와 npm이 설치되어 있어야 한다. Node.js 페이지 참고. 그 후 npm을 이용하여 설치한다.

```
$ npm install pomelo -g
```

git을 이용하여 다운로드 받을 수 있다. 공식 홈페이지 참고.

Hello World

Hello World 프로젝트를 시작해보자.

```
$ pomelo init ./HelloWorld
```

다음과 같이 할 수도 있다.

```
$ mkdir HelloWorld
$ cd HelloWorld
$ pomelo init
```

그 후 HelloWorld 디렉토리로 이동하여 스크립트 실행.

```
$ sh npm-install.sh
```

프로젝트 실행은 다음과 같이 한다.

```
$ cd game-server
$ pomelo start --daemon
```

–daemon 옵션이 없으면 실행 후 종료 전까지 command 입력 불가. 다른 쉘로 이동해야 할 수 밖에…

```
$ cd web-server
$ node app
```

node가 아닌 forever로 실행할 수도 있다. 데몬으로 시작되니깐 편리.

서비스를 시작하면 웹브라우저를 열고 http://localhost:3001로 접속한다. 가운데 Test Game Server 버튼을 클릭하였을 때 'game server is ok' 팝업이 떠야 정상.

서버 구성

Gate 서버

Frontend load balancing을 위한 서버로 일정한 규칙에 의해 connector 서버와 연결해준다. 즉, 클라이언트가 gate 서버로 요청을 보내면 gate 서버는 이를 특정 connector 서버에 할당한다. 보통은 이를 gate.gateHandler에서 처리하며 기본적인 load balancing 방법은 다음과 같다.

```
// gateHandler.js
var connectors = this.app.getServersByType('connector');

// select connector, because more than one connector existed.
var dispatcher = require('/util/dispatcher');
var res = dispatcher.dispatch(uid, connectors);
// ...

// dispatcher.js
var crc = require('crc');
module.exports.dispatch = function(key, list) {
    var index = Math.abs(crc.crc32(key)) % list.length;
    return list[index];
};
```

관련된 설명은 다음 페이지를 참고한다.

- Server scalability

Gate 서버는 단지 클라이언트와 connector 서버를 연결해주기 때문에 servers.json에 정의할 때 port 속성을 정의하지 않는다. clientPort 속성만 정의해 주면 된다.

Connector 서버

Client의 connection 요청을 받는 역할을 한다. 연결이 성립되면 (`session.bind()` 이후) 세션 정보가 유지되고 정해진 routing 정책에 의해 client의 요청을 특정 backend 서버로 전달한다. 반대로 backend 서버에서 요청을 처리하거나 메시지를 push할 때 connector 서버를 통해 client로 전달한다.

Connector 서버는 client의 연결을 listen하기 위해 clientPort 속성, backend server와의 통신을 위해 port 속성을 정의한다.

Gate 서버, Connector 서버 연결에 대해서는 다음 페이지를 참고한다.

- Getting source code & installation

Application 서버

Application 서버는 logic을 가지고 있어서 client에게 service를 제공하는 서버이다. 앞서 설명한 gate, connector 서버는 frontend 서버로 client와 직접 통신을 한다. 반면 application 서버는 backend 서버로

client와 직접 통신을 하지 않는다. 이 서버는 frontend 서버를 통해 client에게 service를 제공한다. Backend 서버간 통신은 뒤에서 설명할 rpc 호출을 통해 이루어진다. 그리고 client에 직접 통신하지 않으므로 clientPort 속성을 갖고 있지 않다. Port 속성만 지정하면 된다. 이 서버를 디버깅하기 위해서는 args 속성에 다음과 같이 디버깅 포트를 지정한다.

```
"game": [
    {
        "id": "game-server-1",
        "host": "127.0.0.1",
        "port": 3250,
        "args": " --debug=32315 "
    }
]
```

이렇게 지정한 경우 32315번 포트로 디버깅할 수 있다.

Master 서버

설정 파일들을 로딩하고 여기에 정의되어 있는 서버 클러스터를 시작한다. 그리고 서버들을 관리하는 역할을 한다.

RPC invocation

Interprocess communication을 위해 RPC 요청을 사용한다. Interprocess communication이란 frontend 서버와 backend 서버, 또는 backend 서버간의 통신을 의미하며 사용 형식은 다음과 같다.

```
app.rpc.game.gameRemote.request( routeParam, args, cb );
```

여기에서 game은 서버, gameRemote는 remote 파일 이름, request는 함수 이름이다. pomelo 서버 폴더의 app/servers 폴더에 game 폴더를 만들고(이름이 다르면 안된다) game/remote 폴더 밑에 gameRemote.js 파일을 생성 후 request라는 함수를 만들면 알아서 연결된다. args와 cb는 rpc call 때 사용하는 인자로 위 예에서는 2개의 인자를 사용하였으나 1개 이상이면 된다. (cb는 무조건 있어야 한다. 사용하고 싶지 않다면 null로 넘기고 함수 정의에는 cb를 빼버리면 된다.)

args는 오브젝트의 오브젝트, 배열의 오브젝트 등 이중 구조이면 안된다. 즉, 단일 오브젝트, 단일 배열이어야 한다. 그렇지 않으면 … Converting circular structure to JSON … 에러 메시지를 출력한다. 콜백 함수에 인자를 전달할 때에도 마찬가지이다.

인자의 routeParam은 route을 위해 필요한 값이다. 보통은 여기에 session에 설정한 값을 넣지만 그냥 user의 id 값, 날짜, 랜덤값 등을 넣기도 한다. 하지만 채널을 이용하고 싶다면 같은 채널의 사람은 꼭 같은 서버로 route 되게끔 해주어야 한다.((그렇지 않으면 같은 채널에 들어갈 수 없다) Route 설정은 다음과 같은 방식으로 이루어진다.

```
// app.js
var router = function(routeParam, msg, context, cb) {
  var gameServers = app.getServersByType('game');
```

```
  var id = gameServers[routeParam% gameServers.length].id;
  cb(null, id);
}

app.configure('production|development', function() {
  app.route('game', router); // router는 위에 정의된 함수
});
```

servers.json에 game 서버가 다음과 같이 정의되어 있다고 하자.

```
"game": [
  {"id": "game-server-1", "host": "127.0.0.1", "port": 7000},
  {"id": "game-server-2", "host": "127.0.0.1", "port": 7001}
]time
```

그러면 app.route()에 의해 위 두 서버 중 하나에 연결된다.

자세한 내용은 다음을 참고한다.

- RPC Invocation
- Rpc framework

Route, router

Route는 클라이언트가 서버로부터 메시지를 받는 위치 또는 특정 서비스를 구분하기 위해 사용하는 것이다. Javascript로 짜여진 클라이언트가 서버의 특정 서비스를 사용하려면 다음과 같은 형식을 이용한다.

```
window.pomelo.request( 'game.gameHandler.request', {
    protocol: ...
}, function( result ) {
    ...
} );
```

여기에서 game은 서버, gameHandler는 handler 파일 이름, request는 함수 이름이다. pomelo 서버 폴더의 app/servers 폴더에 game 폴더를 만들고 handler 폴더 밑에 gameHandler.js 파일을 생성 후 request라는 함수를 만들면 알아서 연결된다.

반대로 서버에서 클라이언트로 메시지를 보내려면 다음과 같이 한다.

```
channel.pushMessage('onChat', param);
```

여기서 channel은 pomelo app의 channelService를 이용하여 얻은 channel이다. 위 명령은 해당 채널에 속한 클라이언트에게 param 오브젝트를 보낸다. 이 param 오브젝트는 클라이언트의 다음 부분에서 메시지를 받는다.

```
pomelo.on('onChat', function (msg) {
});
```

Session

세션은 클라이언트와 연결이 이루어질 때 만들어지는 객체이다. 객체 형태는 다음과 같다.

```
{
    id : <session id> // readonly
    frontendId : <frontend server id> // readonly
    uid : <bound uid> // readonly
    settings : <key-value map> // read and write
    __socket__ : <raw_socket>
    __state__ : <session state>

    // ...
}
```

이중 uid 필드는 `session.bind(value, callback)`에 의해 bind될 때 `value`로 값이 정해진다.

Channel

플레이어의 컨테이너라 보면 된다. 단순히 플레이어들을 묶는 것이 아니라 같은 채널 안의 플레이어들에게 메세지를 푸쉬(브로드캐스팅)하기 위한 용도로 사용된다. 플레이어는 여러 개의 채널에 속할 수 있다. 유의해야할 점은 채널은 server-local하다는 점. 즉, Server A와 Server B는 채널 정보를 공유하지 않는다.

Channel은 `app.get('channelService').channels[“channel 이름 ”]`형태로 얻을 수 있다. 즉, channels는 여러 개의 “channel 이름 ” : 채널 객체로 구성된 객체이다. 이렇게 직접 얻을 수도 있으나 함수를 통해 얻는 것이 더 안전한 방법이다.

```
app.get('channelService').getChannel(roomname, true);
```

getChannel 함수의 원형은 다음과 같다.

```
ChannelService.prototype.getChannel = function(name, create) {
  var channel = this.channels[name];
  if(!channel && !!create) {
    channel = this.channels[name] = new Channel(name, this);
  }
  return channel;
};
```

Channel 객체의 함수는 공식 홈페이지를 참고한다.

Request, response, notify, push

Filter

Handler

Handler는 client의 요청을 처리하는 logic을 담고 있다. 다음과 같은 형태를 가진다.

```
handler.methodName = function(msg, session, next) {
  // ...
}
```

서버는 handler와 remote를 가질 수 있는데 이 중 remote 는 위에서 언급한 rpc invocation을 위해 사용된다.

Error handler

Component

Protocol

Pomelo는 기본적으로 socket.io 프로토콜을 사용하며 0.3 버전부터 TCP 또는 websocket 기반의 binary 프로토콜을 사용할 수 있다. Connector 또한 사용하는 프로토콜에 따라 두 가지로 나눌 수 있다. socket.io를 사용하는 sioconnector와 TCP와 websocket 기반의 hybridconnector가 있다. (이 두 connector는 `game-server/node_modules/pomelo/lib/connectors`에 정의되어 있으니 참고) 설정 방법은 다음과 같다.

```
// app.js
app.configure('production|development', 'connector', function(){
  app.set('connectorConfig', {
    connector: pomelo.connectors.hybridconnector,
    heartbeat: 3,
    useDict: true,
    useProtobuf: true,
    checkClient: function(type, version) {
        // check the client type and version then return true or false
    },
    handshake: function(msg, cb){
      cb(null, {/* message pass to client in handshake phase */});
    }
  });
});
```

각 옵션에 대한 설명은 아래 페이지를 참고.

- Pomelo 0.3 new features

Pomelo protocol에 대한 설명은 아래 페이지를 참고.

- Pomelo communications protocol

servers.json

특정 cpu에 바인드

0.5 버전부터 서버(프로세스)를 특정 cpu에 바인드시킬 수 있다. 아래와 같이 하면 된다.

```
{
    "development":{
        "connector":[
            {"id":"connector-server-1", "host":"127.0.0.1", "port":4050,
"clientPort": 3050, "frontend": true, "cpu": 2}
         ]
        "chat":[
            {"id":"chat-server-1", "host":"127.0.0.1", "port":6050, "cpu":
1}
        ]
        "gate":[
         {"id": "gate-server-1", "host": "127.0.0.1", "clientPort": 3014,
"frontend": true, "cpu": 3}
    ]
     }
}
```

단, 리눅스 시스템에서만 된다.

보안

연결 암호화

0.6 버전부터 hybridconnector 사용시 암호화를 지원한다. 이를 위해서 클라이언트에서는 pomelo.init 시 encrypt 속성을 추가하면 된다.

```
pomelo.init({
  host:'127.0.0.1',
  port:3014,
  encrypt:true
}, function() {
// do something connected
});
```

또한 서버의 app.js에서 connectorConfig 설정을 할 때 useCrypto 속성을 추가한다.

```
app.set('connectorConfig', {
  connector: pomelo.connectors.hybridconnector,
  heartbeat: 3,
  useDict: true,
  useProtobuf: true,
  useCrypto: true
});
```

서버 접속 시퀀스

Pomelo 서버에는 다음과 같은 순서로 접속한다.

1. Gate 서버의 handler에 connector 서버의 host, port 값 얻기 위한 쿼리 날림
2. Gate 서버의 handler에서 session 값 등을 이용하여 connector 서버 중 하나를 선택
3. 클라이언트에서 connector 서버의 host, port 값을 얻은 후 이 connector 서버의 handler로 쿼리.
4. Connector 서버의 handler에서 session bind 등의 처리.

그리고 router를 사용하는 경우 보통 connector 서버의 handler에서 session에 특정 값을 넣어 준다. 그리고 이 값을 이용하여 route를 하게 된다.

참고

- 공식 홈페이지
- Pomelo Wiki
- Promelo Wiki 한글판 (왜 여기에만 Tutorial 링크가 있지?)