27th Nov '15

JS based RPC with Protocol Buffers

github.com/hsk81/protobuf-rpc-js
www.npmjs.com/package/protobuf-rpc

Hasan Karahan <hasan.karahan@blackhan.com>

RPC Layers: Service, Transport, Encoding

Service: Reflector, Calculator

Transport: WebSockets, AJAX

Encoding: Binary, JSON, B64, ..

RPC Layers: Service, Protocol, Transport

Service: Reflector, Calculator

Transport: WebSockets, AJAX

Encoding: Binary, JSON, B64, ..

RPC Service: reflector.proto and calculator.proto

```
package Reflector;

message AckRequest {
    string timestamp = 1;
}
message AckResult {
    string timestamp = 1;
}
service Service {
    rpc ack(AckRequest) returns(AckResult);
}
```

```
package Calculator;

message AddRequest {
    int32 lhs = 1; int32 rhs = 2;
}
message AddResult {
    int32 value = 1;
}
service Service {
    rpc add(AddRequest) returns(AddResult);
    rpc sub(SubRequest) returns(SubResult);
}
```

RPC Service: reading the *.proto files in JS

```
var ReflectorFactory =
    ProtoBuf.loadProtoFile({
        root: 'path/to/protocol',
        file: 'reflector.proto'
});

var Reflector =
    ReflectorFactory.build('Reflector');
```

```
var CalculatorFactory =
    ProtoBuf.loadProtoFile({
        root: 'path/to/protocols',
        file: calculator.proto'
});

var Calculator =
    CalculatorFactory.build('Calculator');
```

RPC Service: api.proto streamlines imports

```
import public “reflector.proto”;
import public “calculator.proto”;
```


```
var ApiFactory =
    ProtoBuf.loadProtoFile({
        root: 'path/to/protocols',
        file: api.proto'
});

var Api = ApiFactory.build();
assert(Api.Reflector);
assert(Api.Calculator);
```

RPC Service: reflector-svc and calculator-svc

```
var reflector_svc = new ProtoBuf.Rpc(
    Api.Reflector.Service, {
        url: 'ws://localhost:8088'
    }
);
```

```
var calculator_svc = new ProtoBuf.Rpc(
    Api.Calculator.Service, {
        url: 'ws://localhost:8088'
    }
);
```

Same WebSocket URL possible, but not required!

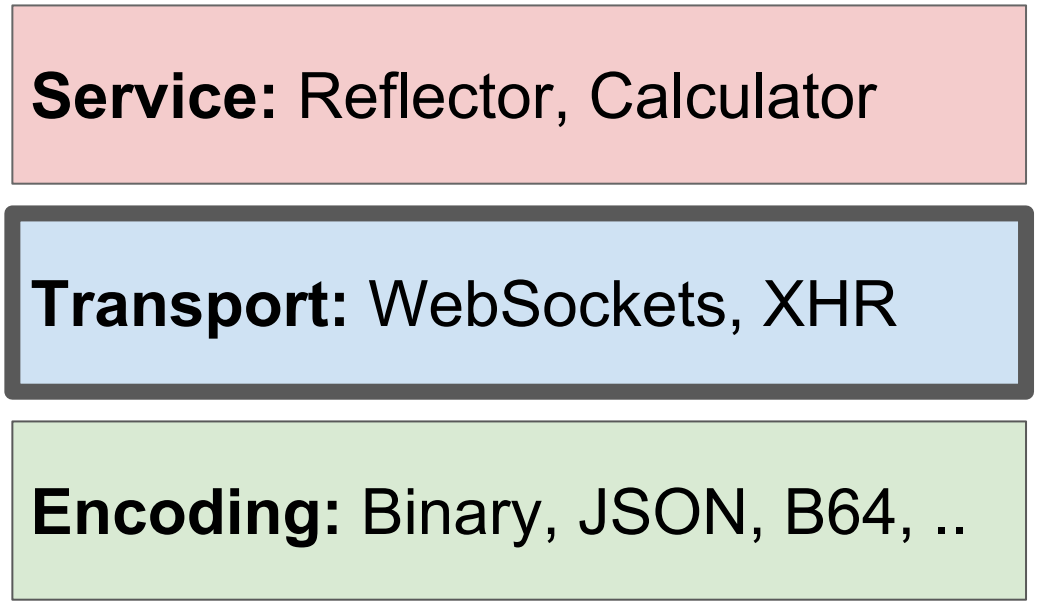
RPC Service: reflector-srv.ack & calculator-srv.add

```
var req = new Api.Reflector.AckRequest({
    timestamp: new Date().toISOString()
});

reflector_svc.ack(req, function (err,res) {
    if (err !== null) throw err;
    assert(res.timestamp);
});
```

```
var req = new Api.Calculator.AddRequest({
    lhs: 2, rhs: 3
});

calculator_svc.add(req, function (err,res) {
    if (err !== null) throw err;
    assert(res.value);
});
```

The callbacks *function (err, res)* are by default asynchronous, but they don't have to be: Their actual behaviour depends on the transport layer!

RPC Layers: Service, Protocol, Transport



RPC Transport: WebSockets vs XMLHttpRequest

```
var reflector_svc = new ProtoBuf.Rpc(
  Api.Reflector.Service, {
    transport: function () {
        this.open = function (url) {
            this.socket = new WebSocket(url);
            this.socket.binaryType = 'arraybuffer';
        };
        this.send = function (buf, msg_cb, err_cb) {
            this.socket.onmessage = function () {..};
            this.socket.onerror = function () {..};
            this.socket.send(buf);
        };
    },
    url: 'ws://localhost:8089'
  }
);
```

```
var calculator_svc = new ProtoBuf.Rpc(
  Api.Calculator.Service, {
    transport: function () {
        this.open = function (url) {this.url = url};
        this.send = function (buf, msg_cb, err_cb) {
            var xhr = new XMLHttpRequest();
            xhr.open('POST', this.url, true);
            xhr.onreadystatechange = function () {
              // if ok: msg_cb(this.reponse)
            };
            xhr.send(new Uint8Array(buf));
        };
    },
    url: 'http://localhost:8088'
  }
);
```

RPC Transport: WebSockets vs XMLHttpRequest

```
var reflector_svc = new ProtoBuf.Rpc(
  Api.Reflector.Service, {
    transport: function () {
        this.open = function (url) {
            this.socket = new WebSocket(url);
            this.socket.binaryType = 'arraybuffer';
        };
        this.send = function (buf, msg_cb, err_cb) {
            this.socket.onmessage = function () {..};
            this.socket.onerror = function () {..};
            this.socket.send(buf);
        };
    },
    url: 'ws://localhost:8089'
  }
```

WebSockets: binary and asynchronous

```
var calculator_svc = new ProtoBuf.Rpc(
  Api.Calculator.Service, {
    transport: function () {
        this.open = function (url) {this.url = url};
        this.send = function (buf, msg_cb, err_cb) {
            var xhr = new XMLHttpRequest()
            xhr.open('POST', this.url, true);
            xhr.onreadystatechange = function () {
              // if ok: msg_cb(this.reponse)
            };
            xhr.send(new Uint8Array(buf));
        };
    },
    url: 'http://localhost:8088'
  }
```

async

XHR: binary (or text) and **async/sync!**

RPC Transport: WebSockets vs XMLHttpRequest

```
var reflector_svc = new ProtoBuf.Rpc(
  Api.Reflector.Service, {
    transport: function () {
        this.open = function (url) {
            this.socket = new WebSocket(url);
            this.socket.binaryType = 'arraybuffer';
        };
        this.send = function (buf, msg_cb, err_cb) {
            this.socket.onmessage = function () {..};
            this.socket.onerror = function () {..};
            this.socket.send(buf);
        };
    },
    url: 'ws://localhost:8089'
  }
```

WebSockets: binary and asynchronous

```
var calculator_svc = new ProtoBuf.Rpc(
  Api.Calculator.Service, {
    transport: function () {
        this.open = function (url) {this.url = url};
        this.send = function (buf, msg_cb, err_cb) {
            var xhr = new XMLHttpRequest()
            xhr.open('POST', this.url, false);
            xhr.onreadystatechange = function () {
              // if ok: msg_cb(this.reponse)
            };
            xhr.send(new Uint8Array(buf));
        };
    },
    url: 'http://localhost:8088'
  }
```

sync!

XHR: open(POST, url, **async/sync**)

RPC Transport: WebSockets vs XMLHttpRequest

```
var reflector_svc = new ProtoBuf.Rpc(
    Api.Reflector.Service, { url: '..',
        transport: ProtoBuf.Rpc.Transport.Ws
    }
);

var req = new Api.Reflector.AckRequest({
    timestamp: new Date().toISOString()
});
reflector_svc.ack(req, function (err, res) {
    if (err !== null) throw err;
    console.log(res.timestamp); // second
});
console.log('ACK sent: waiting..'); // first
```

WebSockets: ProtoBuf.Rpc.Transport.Ws

```
var calculator_svc = new ProtoBuf.Rpc(
    Api.Calculator.Service, { url: '..',
        transport: new ProtoBuf.Rpc.Transport.Xhr({
            sync: true|false})
    }
);
var req = new Api.Calculator.AddRequest({
    lhs: 2, rhs: 3
});
calculator_svc.add(req, function (err, res) {
    if (err !== null) throw err;
    console.log(value); // first!
});
console.log('ADD sent _and_ received!'); // second
```

XHR: ProtoBuf.Rpc.Transport.Xhr (**a/sync**)

RPC Layers: Service, Protocol, Transport

Service: Reflector, Calculator

Transport: WebSockets, AJAX

Encoding: Binary, JSON, B64, ..

RPC Encoding: Request and Response Frames

```
message Rpc {
    message Request {
        string name = 1;
        uint32 id = 2;
        bytes data = 3;
    }
    message Response {
        uint32 id = 2;
        bytes data = 3;
    }
}
```

```
rpc_encode = function () {
    // encode rpc-request frame
};
rpc_decode = function () {
    // decode rpc-response frame
};
msg_encode = function () {
    // encode rpc-request data
};
msg_decode = function () {
    // decode rpc-response data
};
```

RPC Encoding: Binary vs JSON

```
var reflector_svc = new ProtoBuf.Rpc(
    Api.Reflector.Service, {
        encoding: function () {
            this.rpc_encode = function (msg) {
                return msg.toBuffer(); };
            this.rpc_decode = function (cls, buf) {
                return cls.decode(buf); };
            this.msg_encode = function (msg) {
                return msg.toBuffer(); };
            this.msg_decode = function (cls, buf) {
                return cls.decode(buf); };
        },
        url: '..'
    }
);
```

Binary Encoding

```
var calculator_svc = new ProtoBuf.Rpc(
    Api.Calculator.Service, {
        encoding: function () {
            this.rpc_encode = function (msg) {
                return msg.encodeJSON(); };
            this.rpc_decode = function (cls, buf) {
                return cls.decodeJSON(buf); };
            this.msg_encode = function (msg) {
                return msg.encodeJSON(); };
            this.msg_decode = function (cls, buf) {
                return cls.decodeJSON(buf); };
        },
        url: '..'
    }
);
```

JSON Encoding

RPC Protocol: Hex vs Base64

```
var reflector_svc = new ProtoBuf.Rpc(
    Api.Reflector.Service, {
        encoding: function () {
            this.rpc_encode = function (msg) {
                return msg.encodeHex(); };
            this.rpc_decode = function (cls, buf) {
                return cls.decodeHex(buf); };
            this.msg_encode = function (msg) {
                return msg.encodeHex(); };
            this.msg_decode = function (cls, buf) {
                return cls.decodeHex(buf); };
        },
        url: '..'
    }
);
```

Hex Encoding

```
var calculator_svc = new ProtoBuf.Rpc(
    Api.Calculator.Service, {
        encoding: function () {
            this.rpc_encode = function (msg) {
                return msg.encode64(); };
            this.rpc_decode = function (cls, buf) {
                return cls.decode64(buf); };
            this.msg_encode = function (msg) {
                return msg.encode64(); };
            this.msg_decode = function (cls, buf) {
                return cls.decode64(buf); };
        },
        url: '..'
    }
);
```

Base64 Encoding

RPC Encoding: Binary, JSON, Hex vs Delimited

```
var reflector_svc = new ProtoBuf.Rpc(
    Api.Reflector.Service, {
        encoding: ProtoBuf.Rpc.Encoding.Binary,
        url: '..'
    }
);
```

```
var reflector_svc = new ProtoBuf.Rpc(
    Api.Reflector.Service, {
        encoding: ProtoBuf.Rpc.Encoding.Hex,
        url: '..'
    }
);
```

Binary/Hex Encoding

```
var calculator_svc = new ProtoBuf.Rpc(
    Api.Calculator.Service, {
        encoding: ProtoBuf.Rpc.Encoding.JSON,
        url: '..'
    }
);
```

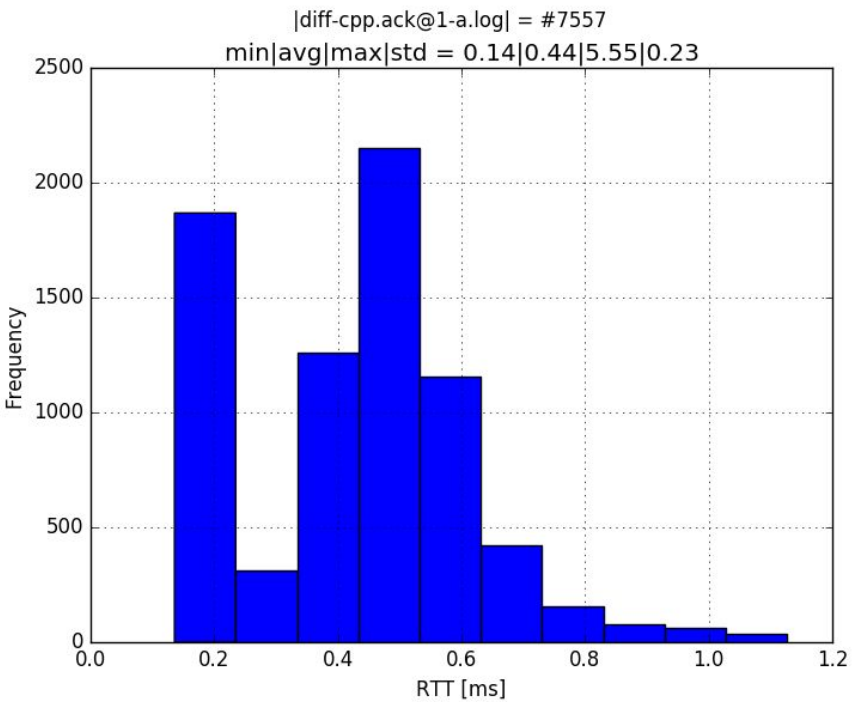
```
var calculator_svc = new ProtoBuf.Rpc(
    Api.Calculator.Service, {
        encoding: ProtoBuf.Rpc.Encoding.Base64,
        url: '..'
    }
);
```

JSON/Base64 Encoding

RPC Examples: Client vs Server

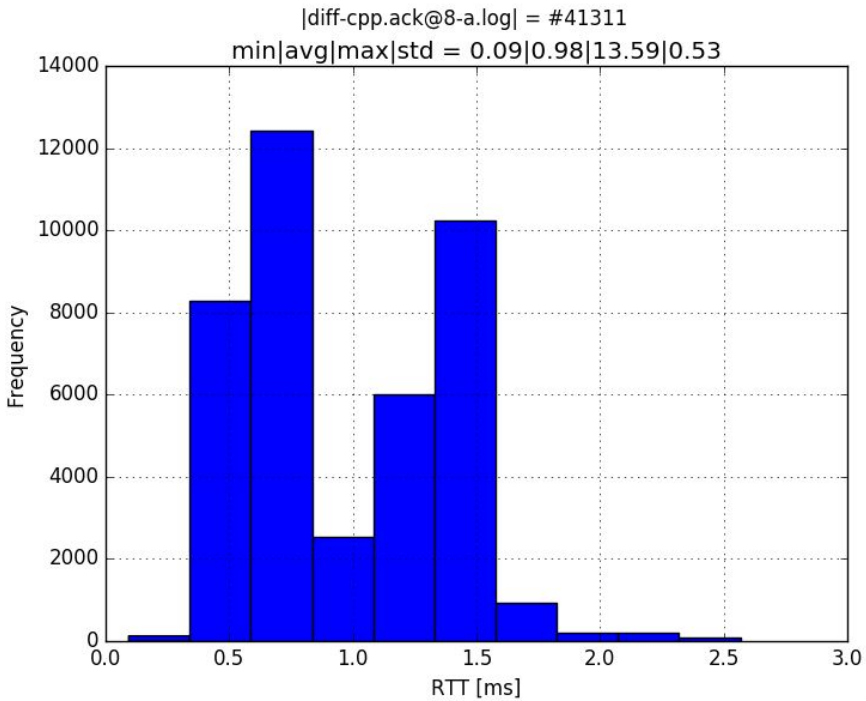
- **NodeJS** Client: examples/js
  - support for Transport.{Ws, Xhr}
  - support for Encoding.{Binary, JSON}
- **Browser** Client: examples/js-www
  - support for Transport.{Ws, Xhr}
  - support for Encoding.Binary
- **Dizmo** Client: com.dizmo.protobuf
  - support for Transport.{Ws, Xhr}
  - support for Encoding.Binary
- **NodeJS** Server:
  - support for Transport.{Ws, Xhr}
  - support for Encoding.{Binary, JSON}
- **QT/C++** Server:
  - support for Transport.Ws
  - support for Encoding.Binary
- **Python** Server:
  - support for Transport.{Ws, Xhr}
  - support for Encoding.Binary

Performance: NodeJS Client vs QT/C++ Server



- **RTT (round trip time):**
  - the time it takes to invoke an RPC request and then to receive a response:
  - on avg. **440** micro-seconds;
- **Throughput (number of message):**
  - total number of messages over measurement period (10 seconds):
  - #7557: i.e. **755.7** calls per second;
- **Client setting:**
  - only a single *setInterval* working, till cut off after 10 seconds with *process.kill*;

Performance: NodeJS Client vs QT/C++ Server



- **RTT (round trip time):**
  - the time it takes to invoke an RPC request and then to receive a response:
  - on avg. **980** micro-seconds;
- **Throughput (number of message):**
  - total number of messages over measurement period (10 seconds):
  - #41311: i.e. **4131.1** calls per second;
- **Client setting:**
  - in total 8 *setInterval* working, till cut off after 10 seconds with *process.kill*;

Questions

?