



# JSON RPC-Gửi các cấu trúc dữ liệu phức tạp đến & từ server

---

Cao Tuấn Dũng  
Trịnh Tuấn Đạt



# Nội dung

---

- 1. Giới thiệu và cài đặt
- 2. Cấu hình phía Server
- 3. Cấu hình phía Client
- 4. Tạo các lời gọi synchronous từ xa
- 5. Tạo các lời gọi asynchronous từ xa
- 6. Truyền và trả về các cấu trúc dữ liệu phức tạp



# 1. Giới thiệu và cài đặt

---



# 1.1. RPC và JSON

---

- Ý tưởng: Client code như gọi 1 method trên server (không qua 1 URL)
- Thuận lợi
  - Cú pháp client đơn giản hơn
    - Không cần sử dụng tường minh đối tượng XmlHttpRequest
  - Cú pháp server đơn giản hơn
    - Không cần các phương thức doGet
  - Có thể truyền và nhận dữ liệu trực tiếp
- Khó khăn
  - Yêu cầu code thêm ở cả client & server
  - Trói buộc code server với Java (do không dùng URL)



## 1.2. JSON-RPC

---

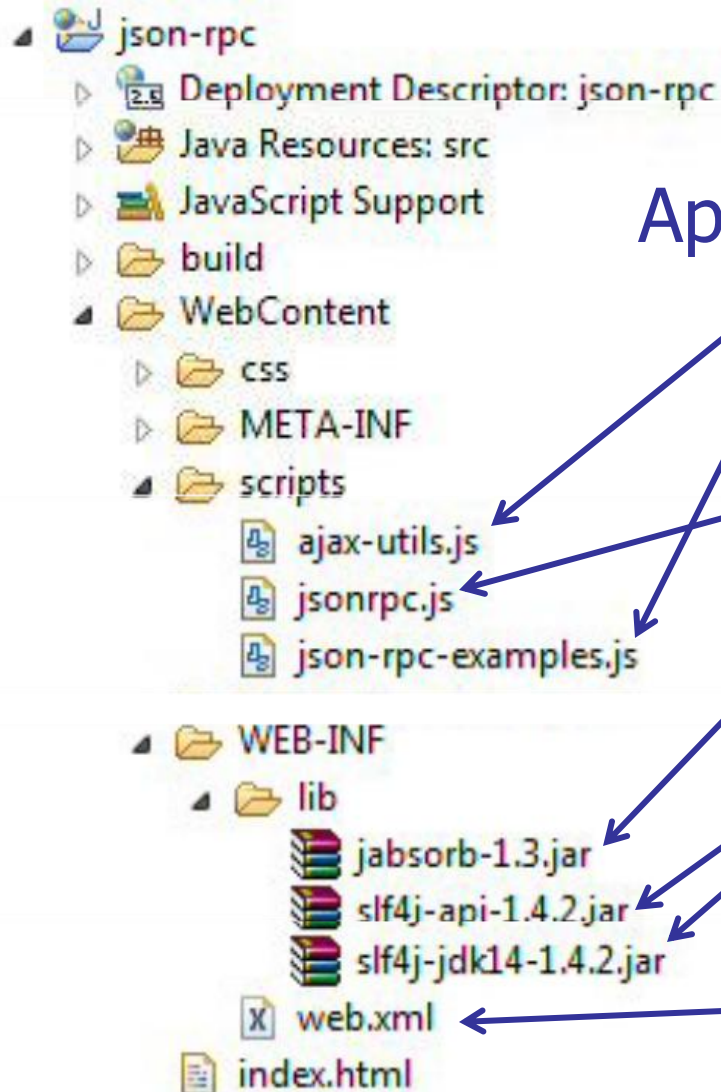
- JSON-RPC
  - Chuẩn tắc cho các lời gọi từ xa với JSON data
  - <http://json-rpc.org/>
- jabsorb
  - Cài đặt của JSON-RPC
    - "JavaScript to Java Object Request Broker"
  - Dễ sử dụng
  - Khả năng hạn chế, ít tài liệu
- Một số thực thi khác của JSON-RPC
  - <http://json-rpc.org/wiki/implementations>
- Các RPC-to-Java toolkits khác:
  - Google Web Toolkit
  - Direct Web Remoting (<http://directwebremoting.org/>)



## 1.3. Cài đặt và thiết lập

- Download jabsorb
- <http://jabsorb.org/Download>
  - Bản "minimal" version: tải file jabsorb-1.x.jar và jsonrpc.js
  - Bản "src" version: slf4j-xxxx.jar (2 files)
- Cài đặt:
  - Server:
    - Đặt 3 file JAR vào thư mục WEB-INF/lib
      - jabsorb-1.x.jar, 2 file JAR cho slf4j
    - Sửa file web.xml, thực hiện ánh xạ tới JSON
  - Client:
    - Thiết lập thêm file jsonrpc.js vào mỗi page
- Tài liệu:
  - <http://jabsorb.org/Manual>
  - <http://jabsorb.org/Tutorial>

# Ví dụ trong Eclipse project



If you use  
ServletContextListener to  
register handler, then you  
also need <listener> entry.



## 2. Cấu hình phía Server

---





## 2.1. Servlet mapping

---

```
<servlet>
    . . .
    <servlet-class>org.jabsorb.JSONRPCServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>org.jabsorb.JSONRPCServlet</servlet-name>
    <url-pattern>/JSON-RPC</url-pattern>
</servlet-mapping>
```

- Có thể cấu hình JSON-RPC gzip response khi có thể
  - Cho phép tăng hiệu năng
  - Không dùng khi test
  - Tra cứu tài liệu để biết cách cấu hình



## 2.2. Đăng ký Handler Object trên server

- Ý tưởng:
  - Khởi tạo 1 đối tượng server và đăng ký nó
    - Sử dụng thuộc tính context-listener hoặc load-on-startup
  - Với synchronous class, code ở client sẽ gọi `rpcClient.objName.methodName(args)`
    - Hữu ích khi test các tương tác, không nên dùng trong ứng dụng cuối cùng
  - Với loại gọi asynchronous, code ở client gọi `rpcClient.objName.methodName(callbackFunction, args)`
- Lựa chọn khác:
  - Nếu chỉ là 1 phương thức static: dùng 1 Class
  - Ngược lại, tạo 1 object handler cho từng user: lưu trong user session

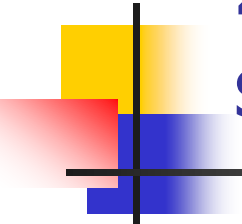


## 2.3. Object Handler mẫu

---

```
public class JsonRpcTester {  
    public double getRandomNumber() {  
        return(Math.random());  
    }  
  
    public double getRandomNumber(double range) {  
        return(range * Math.random());  
    }  
  
    public City getCity(String cityName) {  
        return(CityUtils.getCity(cityName));  
    }  
}
```

- Chú ý: Đây là đối tượng thông thường, với các phương thức thông thường (không phải là 1 servlet)



## 2.4. Đăng ký Handler Object trên server: sử dụng Servlet Context Listener

```
package coreservlets;

import javax.servlet.*;
import org.jabsorb.*;

public class JsonRpcInitializer implements
    ServletContextListener {

    public void contextInitialized(ServletContextEvent event) {
        JsonRpcTester rpcTester = new JsonRpcTester();
        JSONRPCBridge globalBridge =JSONRPCBridge.getGlobalBridge();
        globalBridge.registerObject("rpcTester", rpcTester);
    }

    public void contextDestroyed(ServletContextEvent event) {
    }
}
```



## 2.4. Đăng ký đối tượng Handler Object trên server: web.xml

---

```
<!-- Run JsonRpcInitializer when app starts
up. -->
<listener>
  <listener-class>
    coreservlets.JsonRpcInitializer
  </listener-class>
</listener>
```



## 3. Cấu hình phía Client

---



## 3.1. Đăng ký RPC Client trong browser

---

### ■ Idea:

- Client tải file jsonrpc.js
- Client thực hiện:
  - `rpcClient = new JSONRpcClient( address );`
    - Địa chỉ từ URL pattern thiết lập trong web.xml
- (e.g., /JSON-RPC)
- Với loại gọi synchronous
  - `rpcClient.serverObjName.methodName(args)`
    - Tuy không bao giờ deploy ứng dụng sử dụng loại gọi synchronous, nhưng rất hữu ích trong test tương tác với Firebug.
- Với loại gọi asynchronous
  - `rpcClient.serverObjName.methodName(callback, args)`

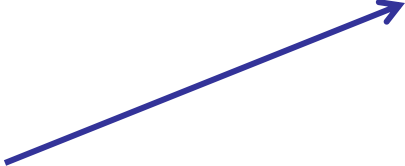


### 3.1. Đăng ký RPC Client trong browser: Ví dụ

---

```
var rpcClient;
```

```
window.onload = function() {  
    rpcClient = new JSONRpcClient("JSON-RPC");  
}
```



URL tương đối. Ví dụ này giả thiết page nằm ở mức trên cùng của ứng dụng Web. Nếu ở folder con, phải dùng ../JSON-RPC.





## 4. Tạo các lời gọi synchronous từ xa

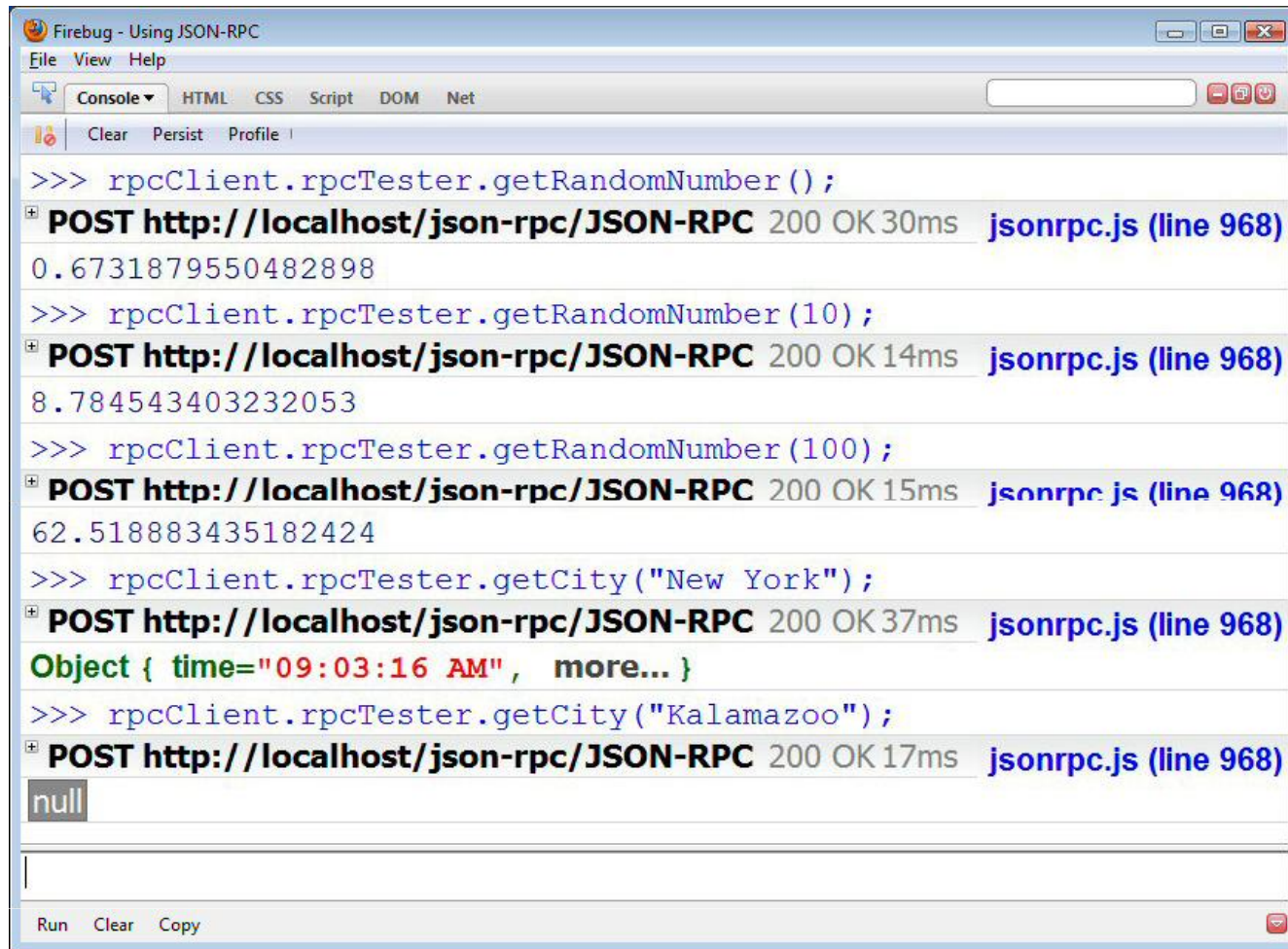
---



## 4. Synchronous RPC Calls

- Ý tưởng:
  - Tạo các lời gọi hàm thông thường
    - `var result = rpcClient.serverObj.methodOfServerObj(...);`
- Tham số của phương thức và các giá trị trả về
  - Có thể truyền numbers, strings, arrays, hoặc objects.
  - Không cần **escape** string khi truyền lên server
  - Beans được trả về từ server sẽ được **JSONified**
  - Truyền object lên server khó khăn hơn 1 chút, phải là JSONObject
- Note: sử dụng lời gọi asynchronous
  - Lời gọi Synchronous chỉ dùng trong thực hành và test code phía server
  - *Luôn sử dụng lời gọi asynchronous trong thực tế, nếu không JavaScript sẽ bị treo cho đến khi nhận kết quả từ server.*

## 4. Synchronous RPC Calls: Ví dụ kiểm thử



The screenshot shows the Firebug console window titled "Firebug - Using JSON-RPC". The console displays a series of JavaScript commands and their corresponding JSON-RPC responses. Each command is preceded by a prompt ">>>>". The responses are preceded by a plus sign "+" and show the HTTP method, URL, status, and response time. The responses are as follows:

```
>>> rpcClient.rpcTester.getRandomNumber();
+ POST http://localhost/json-rpc/JSON-RPC 200 OK 30ms jsonrpc.js (line 968)
0.6731879550482898

>>> rpcClient.rpcTester.getRandomNumber(10);
+ POST http://localhost/json-rpc/JSON-RPC 200 OK 14ms jsonrpc.js (line 968)
8.784543403232053

>>> rpcClient.rpcTester.getRandomNumber(100);
+ POST http://localhost/json-rpc/JSON-RPC 200 OK 15ms jsonrpc.js (line 968)
62.518883435182424

>>> rpcClient.rpcTester.getCity("New York");
+ POST http://localhost/json-rpc/JSON-RPC 200 OK 37ms jsonrpc.js (line 968)
Object { time="09:03:16 AM", more... }

>>> rpcClient.rpcTester.getCity("Kalamazoo");
+ POST http://localhost/json-rpc/JSON-RPC 200 OK 17ms jsonrpc.js (line 968)
null
```

The console window includes a menu bar (File, View, Help), a toolbar with tabs for Console, HTML, CSS, Script, DOM, and Net, and buttons for Clear, Persist, and Profile. At the bottom, there are buttons for Run, Clear, and Copy.



## 4. Synchronous Call: Ví dụ (JavaScript)

---

```
function showRandomNumber1(resultRegion) {  
    var randomNumber =  
        rpcClient.rpcTester.getRandomNumber();  
    htmlInsert(resultRegion, "Number is " +  
                                                randomNumber);  
}
```



## 4.2. Synchronous Call: Ví dụ (HTML)

---

```
<script src="./scripts/ajax-utils.js"
```

```
<script src="./scripts/json-rpc-examples.js"  
        type="text/javascript"></script>
```

```
<script src="./scripts/jsonrpc.js"  
        type="text/javascript"></script>
```

...

```
<fieldset>
```

```
<legend>Synchronous Server Call</legend>
```

```
<form action="#">
```

```
<input type="button" value="Show Random Number"  
        onclick='showRandomNumber1("ran-num-result-1")' />
```

```
</form>
```

```
<div id="ran-num-result-1" class="ajaxResult"></div>
```

```
</fieldset>
```

## 4.Synchronous Call: Ví dụ (Kết quả)





## 5. Tạo các lời gọi asynchronous

---



# 5. Asynchronous RPC calls

---

- Ý tưởng
  - Khi gọi các hàm từ xa, truyền JavaScript callback function như tham số đầu tiên của lời gọi
  - Hàm callback function có 2 tham số:
    - Dữ liệu thực sẽ đến server
    - 1 exception
  - Callback sẽ được gọi với readyState 4.
    - Không làm "đờ" browser cho đến khi nhận response
    - Exception sẽ không được định nghĩa nếu mọi thứ đều OK
- Tham số và giá trị trả về
  - Tương tự như trước
  - Numbers & strings: truyền trực tiếp
  - Objects phải tuân thủ 1 số điều kiện về kiểu dữ liệu



## 5. Ví dụ tạo Asynchronous calls (Javascript)

```
function showRandomNumber2(inputField, resultRegion) {
    if (isNaN(range)) {
        range = 1;
    }
    var callback = function(randomNumber, exception) {
        if(exception) {
            alert(exception.msg);
        } else {
            htmlInsert(resultRegion, "Number is " +
                randomNumber);
        }
    };

    rpcClient.rpcTester.getRandomNumber(callback, range);
}
```



## 5. Ví dụ tạo Asynchronous calls (HTML)

---

```
<fieldset>
```

```
<form action="#">
```

```
  <label for="ran-num-range">Range (default 1):</label>
```

```
  <input type="text" id="ran-num-range"/><br/>
```

```
  <input type="button" value="Show Random Number"
```

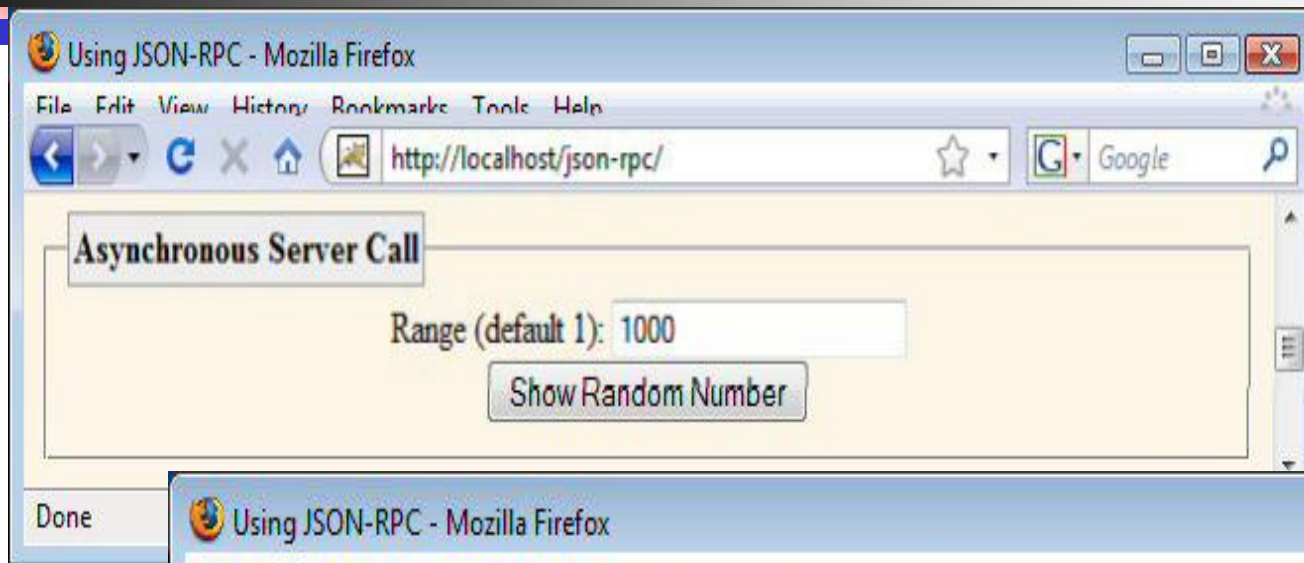
```
    onclick='showRandomNumber2 ("ran-num-range", "ran-num-  
    result-2") ' />
```

```
</form>
```

```
<div id="ran-num-result-2" class="ajaxResult"></div>
```

```
</fieldset>
```

## 5. Ví dụ tạo Asynchronous calls (Kết quả)





## 6. Truyền và trả về các cấu trúc dữ liệu phức tạp

---



## 6.1. Complex data

---

- Dữ liệu gửi tới server:
  - Số và xâu không cần xử lý đặc biệt
    - Thực hiện tự động
  - Đối tượng chỉ có thể được gửi nếu server đợi 1 JSONObject
- Dữ liệu trả về từ server:
  - Số và xâu không cần xử lý đặc biệt
    - Không cần từ khóa **escape**
  - Đối tượng trả về tự động được truyền cho JSONObject constructor
    - Dễ dàng trả về bean, không cần đổi kiểu tường minh



## 6.2. Ví dụ trả về Complex data (JavaScript)

---

- Đọc giá trị của 1 textfield:

```
function getRawValue(id) {  
    return  
        (document.getElementById(id).value) ;  
}
```



## 6.2. Ví dụ trả về Complex data (JavaScript)

```
function showCity(inputField, resultRegion) {
    var cityName = getRawValue(inputField);
    var callback = function(city, exception) {
        if(exception) {
            alert(exception.msg);
        } else {
            var result;
            if (city) {
                result = "<ul>" +
                    "<li>Time: " + city.time + "</li>" +
                    "<li>Population: " + city.population + "</li>" +
                    "</ul>";
            } else {
                result = "Unknown City";
            }
            htmlInsert(resultRegion, result);
        }
    };
    rpcClient.rpcTester.getCity(callback, cityName);
}
```



## 6.2. Ví dụ trả về Complex data (Server code)

---

```
package coreservlets;

public class JsonRpcTester {
    public double getRandomNumber() {
        return(Math.random());
    }

    public double getRandomNumber(double range) {
        return(range * Math.random());
    }

    public City getCity(String cityName) {
        return(CityUtils.getCity(cityName));
    }
}
```



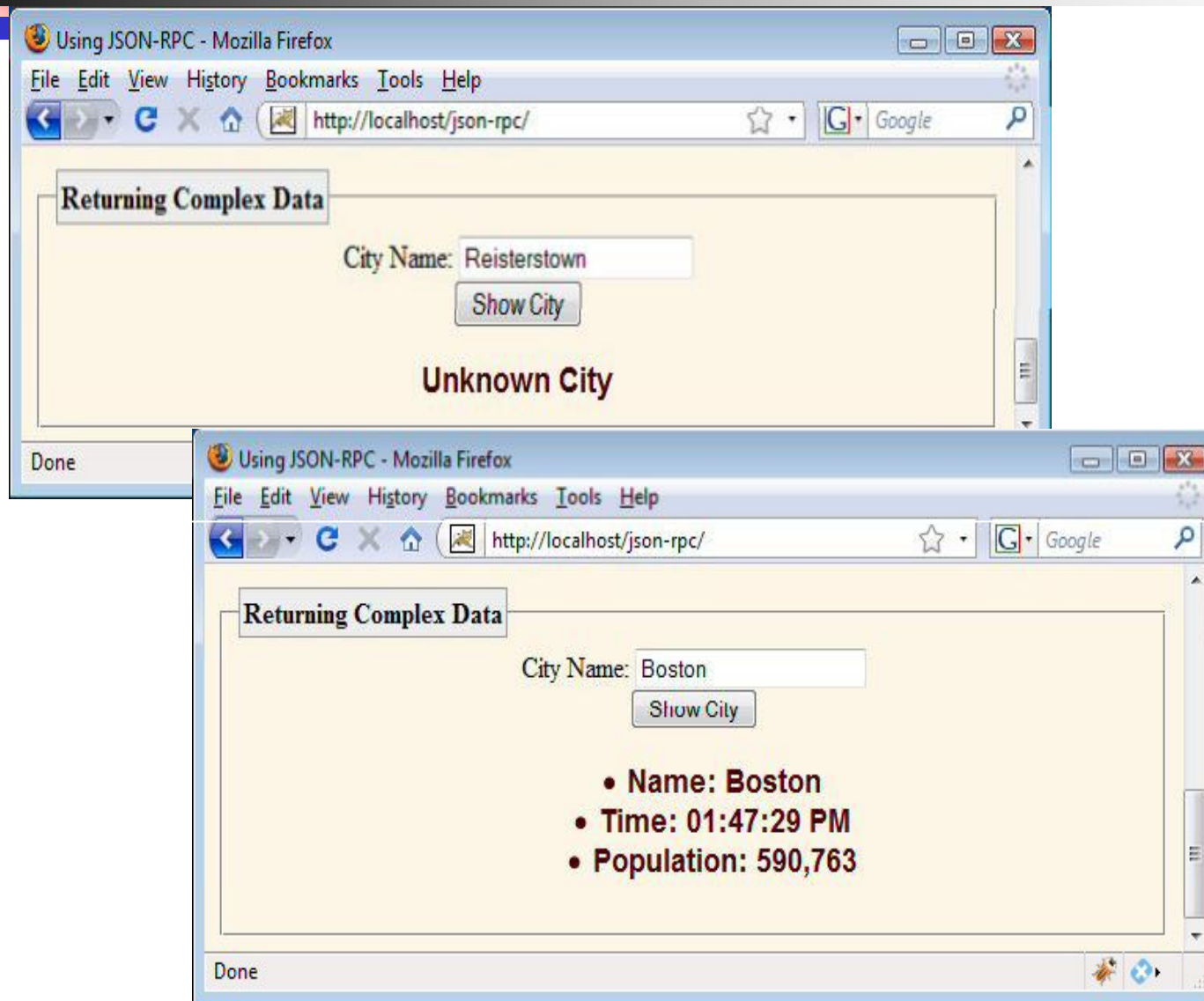


## 6.2. Ví dụ trả về Complex data (HTML)

---

```
<fieldset>
<legend>Returning Complex Data</legend>
<form action="#">
  <label for="city-name">City Name:</label>
  <input type="text" id="city-name"/>
  <br/>
  <input type="button" value="Show City"
    onclick='showCity("city-name", "city-result")' />
</form>
<p/>
<div id="city-result" class="ajaxResult"></div>
</fieldset>
```

## 6.2. Ví dụ trả về Complex data (Kết quả)





## 6.3. Complex data: trả về lists hoặc arrays

---

- Code phía server:
  - Chỉ ra kiểu trả về là List<Type> hoặc Type[]
- Code phía client:
  - Lấy ra thuộc tính list (không phải trực tiếp giá trị trả về)

```
var callback = function(cityList, exception) {  
    if(exception) {  
        alert(exception.msg) ;  
    } else {  
        doSomethingWith(cityList.list) ;  
    }  
};
```



## 6.3. Cập nhật lại JsonRpcTester

---

```
public class JsonRpcTester {

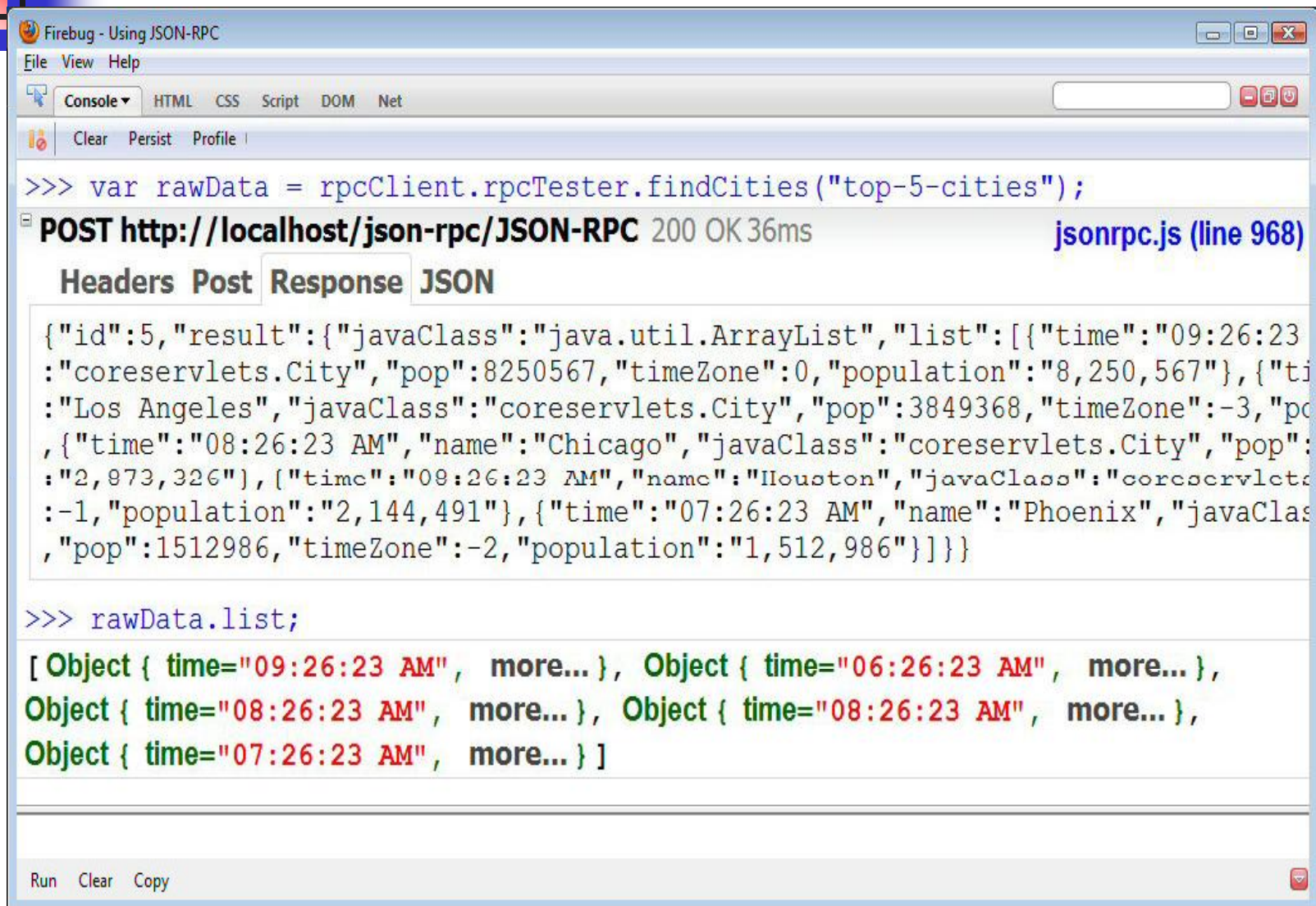
    public double getRandomNumber() {
        return(Math.random());
    }

    public double getRandomNumber(double range) {
        return(range * Math.random());
    }

    public City getCity(String cityName) {
        return(CityUtils.getCity(cityName));
    }

    public List<City> findCities(String description) {
        return(CityUtils.findCities(description));
    }
}
```

## 6.3. Kiểm tra kết quả



The screenshot shows the Firebug console window titled "Firebug - Using JSON-RPC". The "Console" tab is selected, displaying a series of log messages. The first message is a JavaScript command: `>>> var rawData = rpcClient.rpcTester.findCities("top-5-cities");`. This is followed by a log entry for a POST request to `http://localhost/json-rpc/JSON-RPC` with a status of 200 OK and a response time of 36ms. The log entry is labeled `jsonrpc.js (line 968)`. Below this, the "Response" tab is selected, showing the JSON response: `{ "id": 5, "result": { "javaClass": "java.util.ArrayList", "list": [ { "time": "09:26:23 AM", "name": "Los Angeles", "javaClass": "coreservlets.City", "pop": 3849368, "timeZone": -3, "population": "3,849,368" }, { "time": "08:26:23 AM", "name": "Chicago", "javaClass": "coreservlets.City", "pop": 2873326, "timeZone": -5, "population": "2,873,326" }, { "time": "08:26:23 AM", "name": "Houston", "javaClass": "coreservlets.City", "pop": 2144491, "timeZone": -1, "population": "2,144,491" }, { "time": "07:26:23 AM", "name": "Phoenix", "javaClass": "coreservlets.City", "pop": 1512986, "timeZone": -2, "population": "1,512,986" } ] } }`. Below the JSON response, the command `>>> rawData.list;` is shown, followed by the resulting array of objects: `[ Object { time="09:26:23 AM", more... }, Object { time="06:26:23 AM", more... }, Object { time="08:26:23 AM", more... }, Object { time="08:26:23 AM", more... }, Object { time="07:26:23 AM", more... } ]`. The bottom of the console shows "Run", "Clear", and "Copy" buttons.

```
Firebug - Using JSON-RPC
File View Help
Console HTML CSS Script DOM Net
Clear Persist Profile
>>> var rawData = rpcClient.rpcTester.findCities("top-5-cities");
POST http://localhost/json-rpc/JSON-RPC 200 OK 36ms jsonrpc.js (line 968)
Headers Post Response JSON
{"id":5,"result":{"javaClass":"java.util.ArrayList","list":[{"time":"09:26:23 AM", "name": "Los Angeles", "javaClass": "coreservlets.City", "pop": 3849368, "timeZone": -3, "population": "3,849,368"}, {"time": "08:26:23 AM", "name": "Chicago", "javaClass": "coreservlets.City", "pop": 2873326, "timeZone": -5, "population": "2,873,326"}, {"time": "08:26:23 AM", "name": "Houston", "javaClass": "coreservlets.City", "pop": 2144491, "timeZone": -1, "population": "2,144,491"}, {"time": "07:26:23 AM", "name": "Phoenix", "javaClass": "coreservlets.City", "pop": 1512986, "timeZone": -2, "population": "1,512,986"}]}}
>>> rawData.list;
[ Object { time="09:26:23 AM", more... }, Object { time="06:26:23 AM", more... }, Object { time="08:26:23 AM", more... }, Object { time="08:26:23 AM", more... }, Object { time="07:26:23 AM", more... } ]
Run Clear Copy
```