Remote Procedure Call and Remote Method Invocation

CMPE 273 Enterprise Distributed Systems

RPC

- 1984: Birrell & Nelson
- Mechanism to call procedures on other machines
- Remote Procedure Call

- Combine Socket programming and procedure call

Regular procedure calls

- Machine instructions for call & return but the compiler really makes the procedure call abstraction work:
  - Parameter passing
  - Local variables
  - Return data
- x = f(a, “test”, 5);

RPC implementation

- Create **stub functions** to make it appear to the user that the call is local
- Stub function contains the function's interface
- Writing application is simplified
  - RPC hides all network code into stub functions details
  - Sockets, port numbers, byte ordering
- RPC: presentation layer in OSI model

RPC Parameter Passing

- Pass by value: copy data to network message
- Pass by reference: does not make sense without shared memory
- Copy items referenced to message buffer
    1. Send them over
    2. Unmarshal data at server
    3. Pass local pointer to server stub function
    4. Send results back

Complex data structures: copy & reconstruct

RMI

- Distribute objects across different machines to take advantage of hardware and software
- Developer builds network service and installs it on a machine
- User requests an instance of a class using URL syntax
- User uses object as if it were a local object

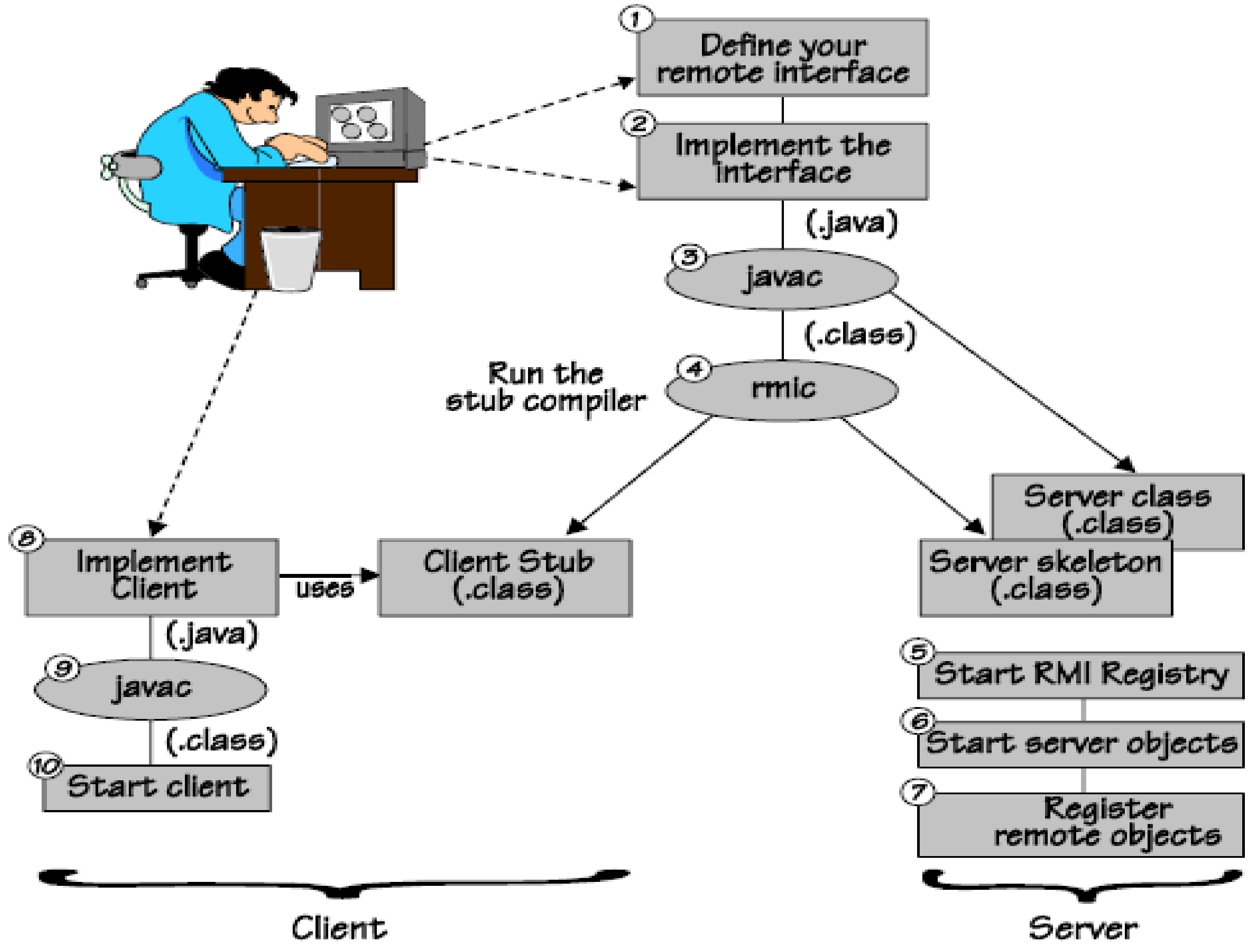
RMI

- **RMI is built for Java only!**
  - No goal of OS interoperability (as CORBA)
  - No language interoperability(goals of SUN, DCE, and CORBA)
  - No architecture interoperability
- **No need for external data representation**
  - All sides run a JVM
- **Benefit: simple and clean design**

RMI Operations

- Stub operation
  - Package identifier of remote object
  - Package method identifier
  - Marshal parameters
  - Send package to server skeleton
- Skeleton Operation
  - _____________ parameters
  - Calls method or exception
  - Marshall method return
  - Send package to client stub

How to write an RMI Application



Naming service

- Object registry does this: **rmiregistry**
- Server: Register object(s) with Naming.bind("ObjectName", obj);
- Client: Contact rmiregistry to look up name
- MyInterface test = (MyInterface)Naming.lookup("rmi://www.sjsu.edi/ObjectName");
  rmiregistry returns a remote object reference.
- Lookup gives reference to local stub.
- Invoke remote method(s):test.func(1, 2, "hi");

Simple RMI Example

- The interface for the Remote Object
  - The interface should extend java.rmi.Remote and all its methods should throw java.rmi.RemoteException

```
/* The RMI server will make a real remote object that
implements this, then register an instance of it with some
URL */
public interface countRMI extends java.rmi.Remote {
    int sum() throws java.rmi.RemoteException;
    void sum (int _val) throws java.rmi.RemoteException;
    public int increment() throws RemoteException;
}
```

```
  public int sum() throws RemoteException
  { return sum;
  }

  public  void sum(int val) throws RemoteException
  { sum = val;
  }

  public int increment() throws RemoteException
  { sum++;
    return sum;
  }
}
```

RMI Client

- Look up the object from the host using Naming.lookup cast it to the appropriate type and use it like local object

```
// CountRMIClient.java  RMI Count client

import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;

public class CountRMIClient
{ public static void main(String args[])
  { // Create and install the security manager
    System.setSecurityManager(new RMISecurityManager());

    try
    { CountRMI myCount = (CountRMI)Naming.lookup("rmi://"
                               + args[0] + "/" + "my CountRMI");

      // Set Sum to initial value of 0
      System.out.println("Setting Sum to 0");
      myCount.sum(0);
```

Shark.sjsu.edu

Local call

Remote Objet/Server

- Remote Object
  - This class must extend UnicastRemoteObject and implement the remote object interface defined earlier
  - The constructor should throw Remote Exception
- The RMI Server
  - The server builds an object and register it with a particular URL
  - Use Naming.rebind (replace any previous binding) or Naming.bind (throw AlreadyBoundException if a previous binding exists)

Remote Object Implementation

```
// CountRMIImpl.java, CountRMI implementation
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class CountRMIImpl extends UnicastRemoteObject
       implements CountRMI
{ private int sum;

  public CountRMIImpl(String name) throws RemoteException
  {
    super();
    try
    { Naming.rebind(name, this);
      sum = 0;
    } catch (Exception e)
    { System.out.println("Exception: " + e.getMessage());
      e.printStackTrace();
    }
  }
```

Name = “my CounteRMI”

Compiling/Running

Compile the Client/Server Programs

```
prompt> javac -d \CorbaJavaBook.2e\classes CountRMI.java
prompt> javac -d \CorbaJavaBook.2e\classes CountRMIImpl.java
prompt> javac -d \CorbaJavaBook.2e\classes CountRMIClient.java
prompt> javac -d \CorbaJavaBook.2e\classes CountRMIServer.java

prompt> rmic -d \CorbaJavaBook.2e\classes CountRMIImpl
```

Run the Client/Server Programs

```
prompt> start rmiregistry
prompt> start java CountRMIServer
prompt> java CountRMIClient <server-hostname>
```

If you're running it locally,
use localhost as the hostname.

The Output:

```
Setting Sum to 0
Incrementing
Avg Ping = 3.275 msec
Sum = 1000
```

Introduction to JavaScript

An Overview

What is JavaScript?

JavaScript was initially created to “make webpages alive”.

- The programs in this language are called *scripts*. They can be written right in the HTML and execute automatically as the page loads.
- Scripts are provided and executed as a plain text. They don’t need a special preparation or a compilation to run.
- When JavaScript was created, it initially had another name: “LiveScript”. But Java language was very popular at that time, so it was decided that positioning a new language as a “younger brother” of Java.
- But as it evolved, JavaScript became a fully independent language, with its own specification called ECMAScript and now it has no relation to Java at all.

Where does it executes?

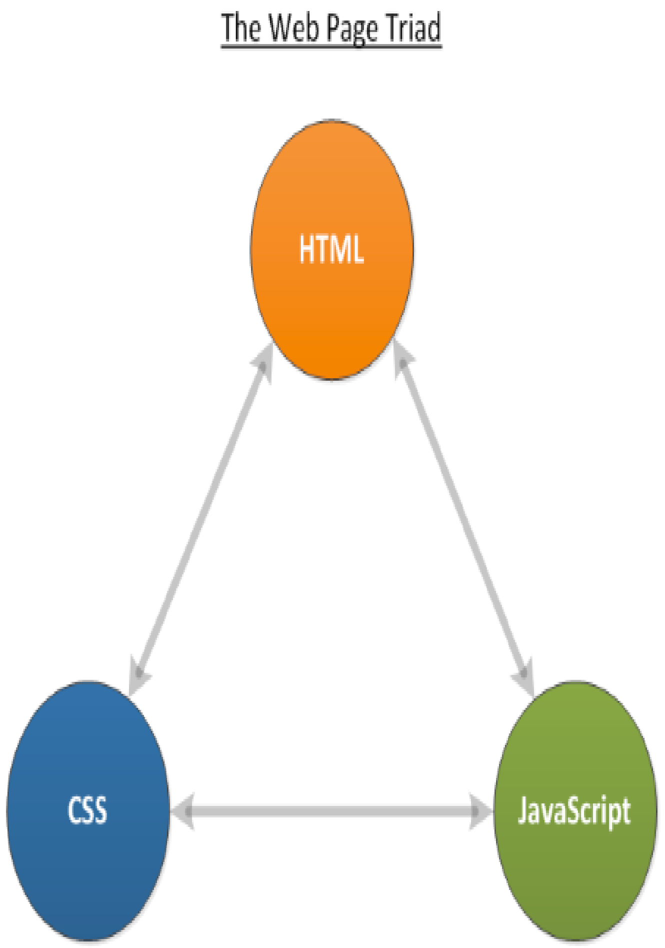
- At present, JavaScript can execute not only in the browser, but also on the server, or actually on any device where there exists a special program called the JavaScript engine.
- The browser has an embedded engine, sometimes it’s also called a “JavaScript virtual machine”.

Different engines have different “codenames”, for example:

- V8 – in Chrome and Opera.
- SpiderMonkey – in Firefox.
- There are other codenames like “Trident”, “Chakra” for different versions of IE, “ChakraCore” for Microsoft Edge, “Nitro” and “SquirrelFish” for Safari etc.

Web Page Triad

- HTML describes the page, including the text, graphics, etc
- CSS is used to control and customize the look of the web page, including the colors, fonts, etc.
- JavaScript is used to add a dynamic component to the web page and make most elements on the page programmable.

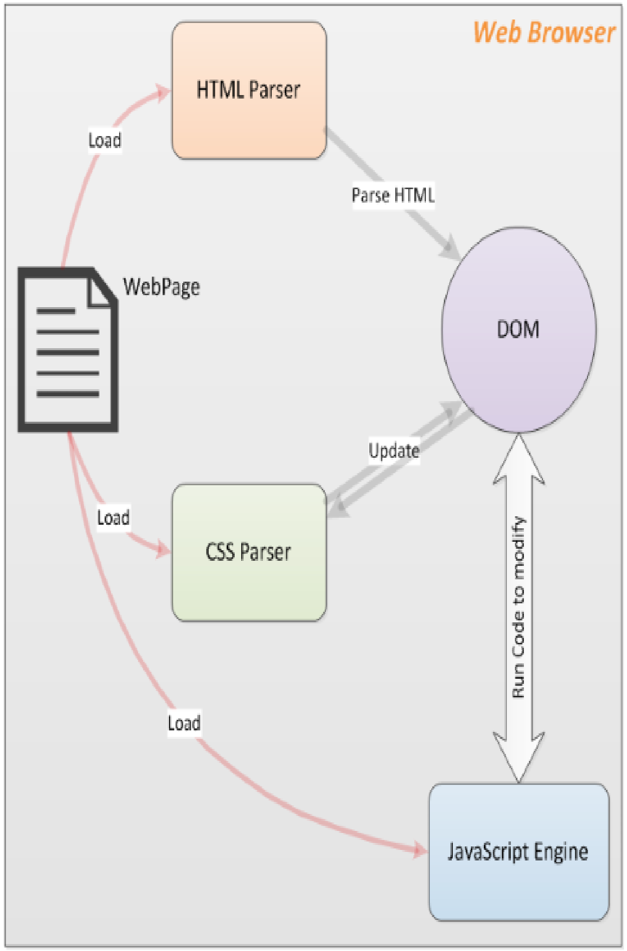


How does JavaScript Works?

- When the web browser loads a web page, the HTML parser begins parsing the HTML code and creating the DOM.

- Whenever the parser encounters a CSS or JavaScript directive (inline or externally loaded), it gets handed over to the CSS parser or the JavaScript engine as required.

- The JavaScript engine loads external JavaScript files and inline code, but does not run the code immediately.

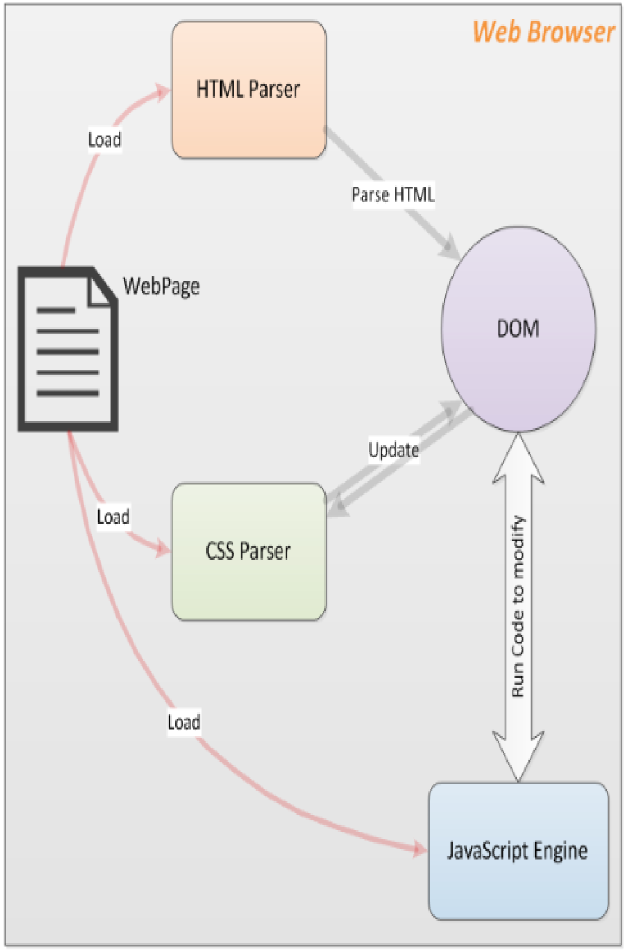


JavaScript working continue

- It waits for the HTML and CSS parsing to complete.

  Once this is done, the JavaScript is executed in the order they were found on the web page: variables and functions are defined, function invocations are executed, event handlers are triggered, etc.

- These activities result in the DOM being updated by the JavaScript and is rendered instantly by the browser.



Loading JavaScript in a webpage

- Load an external javascript file into a web page as follows:

```
<script type="text/javascript"
src="/path/to/javascript"></script>
```

- You can specify the complete URL if the javascript is from a different domain from the web page as follows:

```
<script type="text/javascript"
src="https://code.jquery.com/jquery-
3.2.1.min.js"></script>J
```

- JavaScript can be directly embedded in the HTML. The following causes the web page to popup an alert box when it is loaded.

```
<script type="text/javascript">
alert("Page is loaded");
</script>
```

Nature of JavaScript

It's dynamic
Many things can be changed.

It's dynamically typed
Variables and object properties can always hold values of any type.

It's functional and object-oriented
JavaScript supports two programming language paradigms: functional programming (first-class functions, closures, partial application via bind(), built-in map() and reduce() for arrays, etc.) and object-oriented programming (mutable state, objects, inheritance, etc.).

Nature of JavaScript continue

It’s deployed as source code

JavaScript is always deployed as source code and compiled by JavaScript engines. Source code has the benefits of being a flexible delivery format and of abstracting the differences between the engines.

It’s part of the web platform

JavaScript is such an essential part of the web platform (HTML5 APIs, DOM, etc.) that it is easy to forget that the former can also be used without the latter. However, the more JavaScript is used in non-browser settings (such as Node.js)

Advantages of JavaScript

- **Speed.** Being client-side, JavaScript is very fast because any code functions can be run immediately instead of having to contact the server and wait for an answer.
- **Simplicity.** JavaScript is relatively simple to learn and implement.
- **Versatility.** JavaScript plays nicely with other languages and can be used in a huge variety of applications.
- **Server Load.** Being client-side reduces the demand on the website server.

Some Disadvantages of JavaScript

- **Security.** Because the code executes on the users' computer, in some cases it can be exploited for malicious purposes. This is one reason some people choose to disable JavaScript.
- **Reliance on End User.** JavaScript is sometimes interpreted differently by different browsers. Whereas server-side scripts will always produce the same output, client-side scripts can be a little unpredictable.

Javascript features

Understanding es5, es6, es7 syntax

Introduction

If you are wondering what’s es6, es7 and es8 then for your info es stands for ECMAScript. It is a simple standard for adding new features to JavaScript. So es6 , es7 and es8 stands for the new released versions. Likewise es6 was released in 2015, es7 in 2016 and so on.

Arrow function =>

- ES6 has introduced arrow functions which have three main benefits.
- a *concise* syntax, *implicit returns (return stmt optional), don’t rebind the value of **this*** (lexically scoping) when you use a arrow function inside of another function, which is really helpful for when you’re doing things like click handlers and so on.
- var add = function (x,y) { return x + y; }
  var add = (x,y) => { return x + y; }

Async in JavaScript

Callback (An es5 feature)

- Callback is a function that is passed as an argument to other function and call inside the function.

  For the purpose of understanding we are doing a mimic of actual rest call with some dummy data!!

Promises (An es6 feature)

- Promises promotes an elegant way of handling async calls.
- Promise takes two arguments resolve and reject in its arrow function.
- A promise is resolved when there are no errors and rejected if there is any error.
- Function .then() gives response data when promise is resolved.
- Function .catch() catches exception if promise is rejected.

Async and await(An es7 feature)

- Makes async code look like sync code.
- Internally uses promises for its working.
- Simplifies the way of handling async programming.
- To make a function async just add keyword ‘async’ in-front of it.
- await() can be called inside an async function and it simply waits for async function to complete.