Contents

1 Web Services vs RPC

- Web Services
  - The communication platform between two different or same platform applications that allows to use their web method
- Remote Procedure Call
  - Programming language embedding
  - Data exchange stays transparent for the programmer
  - RPC is located above UDP or TCP in the protocol stack
  - Is mostly implemented as a part of the actual application

Currently most Web services architectures adopt RPC as their architectural style. But because of the complexity of RPC, there are bottlenecks of RPC-style Web services in Web-scale applications. REST not only can make full use of Web features, but also has the advantage of simplicity. So REST becomes a new alternative to RPC for Web services architecture. In this paper, at first the brief introductions of RPC and REST are provided. Then two kinds of architectural styles are analyzed and compared from the perspectives of scalability, coupling, and security. In the end the development trend of Web services architecture is prospected. Web Service a higher level representation of RPC. A web service is a specific implementation of RPC. At its lowest level, all a web service is, is connecting to a socket, using the HTTP protocol to negotiate sending a payload that is executed in a remote space (it may even be on the same computer, for all the consumer knows). All those abstractions are at its core RPC.

2 SOAP vs REST

SOAP is a **protocol** which was designed before REST and came into the picture. The main idea behind designing SOAP was to ensure that programs built on different platforms and programming languages could exchange data in an easy manner.
REST is an **architectural style** that was designed specifically for working with components such as media components, files, or even objects on a particular hardware device. Any web service that is defined on the principles of REST can be called a RestFul web service. A Restful service would use the normal HTTP verbs of GET, POST, PUT and DELETE for working with the required components.

3 What is the difference between URI and URL?

- URI is an abstract resource identifier (may be a unique name of the resource – URN or it‘s location – URL)
- URL describes a location of the resource and the protocol used to access it

✓ `http://www.tu-chemnitz.de/informatik`
✓ `http://tu-chemnitz.de/informatik`
✓ `http://www.tu-chemnitz.de:443/informatik`
✓ `http://www.tu-chemnitz.de/informatik?show=all?group=true`
✓ `http://www.tu-chemnitz.de/informatik?show=all%20group=true`
× `c:/windows/php.ini`
× `ftp://www.tu-chemnitz.de/informatik?show=all&group=true`
× `ftp://bob:pass@www.tu-chemnitz.de/informatik`
× `mailto://user@example.org`

4 Status Codes

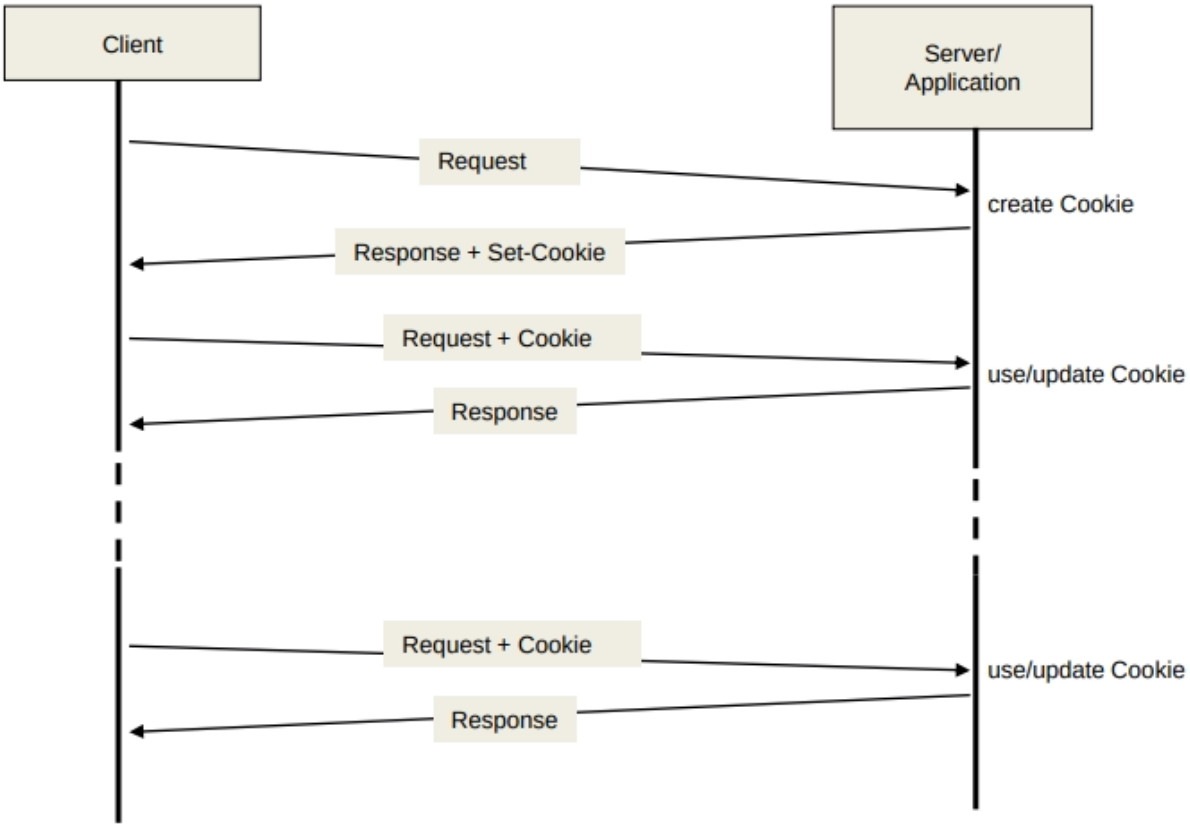
- 20X Success
  - 200 OK
- 30x Redirection
  - 301 Moved Permanently
  - 302 Found (Moved Temporarily)
  - 303 See other

- 40x Error
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
- 50x Server error
  - 500 Internal Server Error
  - 10x Information
  - 101 Switching protocols

5 Cookies

A HTTP Cookie

- is a small text information stored by the webbrowser on the computer of the user
- it extends the stateless HTTP protocol by means to remember stateful information
- A mechanism to store a small amount of data (up to 4KB) at the client [RFC6265]
- A cookie is associated with a specific web site
- Cookie is sent in HTTP header
- Cookie is sent with each HTTP request
- Can last for only one session (until browser is closed) or can persist across sessions
- Can expire some time in the future
- used for: Session management (usually supported by Session objects in programming languages), Personalization, Tracking



5.1 HTTP Response and Request

Response:

```
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2011 02:20:42 GMT
Server: Apache/2.2.31
Connection: Keep-Alive
Keep-Alive: timeout=3, max=100
Content-Type: text/html
Set-Cookie: lastUser=Schwertfischkan-Can-Pimmelgott;expires=Tue, 29 Dec 2019
    13:30:00 GMT;Max-Age=259200;Path=/scripts/guestbook.php
```

Request:

```
GET /scripts/guestbook.php HTTP/1.1
Host: vst.informatik.tu-chemnitz.de
User-Agent: Mozilla/5.0
Accept: text/html
Connection: Keep-Alive
Cookie: lastUser=Schwertfischkan-Can-Pimmelgott
```

6 W3C

6.1 How To set standards

1. If Members express an interest through a Member Submission and the W3C monitors signs of interests, a Workshop will be organized.

2. If a topic has enough interest after a Workshop, the Director announces the development of a proposal for new Working Groups
3. Three type of Working Group participants: Member representatives, invited experts, Team representatives. The working Group sets expectations about deliverables (e.g. tec reports, tutorials)
4. Working groups create specifications and guidelines that undergo cycles of revision and review as they advance to W3C Recommendation status. In the end the Advisory Committee reviewst he technical report and if there is support, the W3C publishes it as a Reommendation. Consensus and Transparency are the ultimate goal of this process.

6.2 Organization

W3C Team besteht auf 60 Forschern. Direktor ist Tim Berners-Lee. Das W3C besteht aktuell aus 429 Mitgliedern. Diese Mitglieder sind zumeist wirtschaftliche Unternehmen, aus verschiedenen Branchen, z.B. Amazon, Apple, Google & Microsoft. Jedes Mitglied im W3C stellt einen Repräsentanten im Advisory Commitee. Dieser plant und reviewed Themen des W3C. Darüber hinaus wählen sie die Technical Architecture Group.

6.3 Entscheidungen & Entwicklungsprozesse

Was bedeuten die Entscheidungen bzw. Standards des W3C?

- Das W3C kann keine ISO-Normen festlegen, weil keine zwischenstaatlich anerkannte Körperschaft.
- Vom W3C festgelegten Standards geeignet, die Grundlage für ISO-Normen zu bilden.
- Aus diesem Grund spricht das W3C von „Recommendations“

Die Entwicklungsprozesse für W3C-Empfehlungen sind im Regelfall gleich aufgebaut.

1. Zunächst erfolgt ein Arbeitsentwurf (Working Draft). In dieser Stufe besteht für Mitglieder und die Öffentlichkeit die Möglichkeit, den Entwurf zu kommentieren.
2. Ist dieser Arbeitsentwurf fertiggestellt, wird ein letzter Aufruf gestartet (Last Call Working Draft). Ist dieser Zeitpunkt vorbei, ist keine Kommentierung des Arbeitsentwurfes mehr möglich.
3. In der nächsten Arbeitsstufe wird der Empfehlungskandidat (Candidate Recommendation) vorgestellt. Die Arbeitsgruppe wird beim Empfehlungskandidaten schon Implementierungen der Technologie vornehmen. Erfahrungen dieser Arbeitsstufe werden gesammelt und dokumentiert.
4. Der Empfehlungskandidat wird schlussendlich zum Empfehlungsvorschlag (Proposed Recommendation). Dieser Vorschlag wird dem beratenden Ausschuss zur Abstimmung durch die Mitglieder übergeben. Erst, wenn die Zustimmung vorliegt, ist eine neue Empfehlung geboren.

7 Metadata

What is meta data?

- First of all Metadata is data about data.
- The HTML Head contains metadata about the document, which is not dispalayed on the webpage.
- The meta tag is used in an HTML document to provide high level metadata about the web page: information that describes the web page in a meaningful way that can be understood by web crawlers and browsers.

What kind of meta data can be provided?

- Page description
- Page authorship
- Details about page title etc...

For example:

```
<meta charset="utf-8">
<meta name="author" content="Chris Mills">
<meta name="description" content="The MDN Web Docs Learning Area aims to provide
complete beginners to the Web with all they need to know to get
started with developing web sites and applications.">
<meta property="og:title" content="Mozilla Developer Network">
```

8 Data Attributes

- The data-*attributes are new in HTML5 and can be used to store meta data
- The data-* attribute is used to store custom data private to the page or application.
- The data-* attribute gives us the ability to embed custom data attributes on all HTML elements.
- The stored (custom) data can then be used in the page's JavaScript to create a more engaging user experience (without any Ajax calls or server-side database queries)

The data-* attribute consist of two parts:

1. The attribute name should not contain any uppercase letters, and must be at least one character long after the prefix "data-"
2. The attribute value can be any string

HTML Syntax:

```
<article
id="electric-cars"
data-columns="3"
data-index-number="12314"
data-parent="cars">
...
</article>
```

Javascript Access:

```
const article = document.querySelector('#electric-cars');
article.dataset.columns // "3"
article.dataset.indexNumber // "12314"
article.dataset.parent // "cars
```

In HTML4 attributes come in key/value pairs. You must assign a value to a attribute otherwise it becomes invalid. But in HTML5 if we don't provide value to an attribute then the value becomes a empty string.

As data-* attributes are formed the same way as any other custom attribute you can use data-* attributes in HTML4. But make sure you assign a value to it.

```
<html>
  <head>
    <title>QNimate</title>
  </head>
  <body>
    <!-- This is invalid in HTML4 --> <p data-blog-name-qnimate>QNimate</p>
    <!-- This is valid in HTML4 --> <p data-blog-name="qnimate">QNimate</a>
  </body>
</html
```

9 Web Components

- main features: HTML imports, HTML template, custom elements, shadow DOM

9.1 Elements

9.1.1 Defining a new element

The `customElements global is used for defining a custom element and teaching the browser about a new tag. Call =customElements.define()` with the tag name you want to create and a JavaScript `class` that extends the base `HTMLElement`.

- example: defining a mobile drawer panel, `<app-drawer>`

```
class AppDrawer extends HTMLElement {
  constructor() { // constructor arguments can also be defined
    super(); // always call super() first
    // click listener on <app-drawer> elemnt itself
    this.addEventListener('click', e => {
      if (this.disabled) {
        return;
      }
      this.toggleDrawer();
    });
  }

  // A getter/setter for on 'open' property
  get open() {
    return this.hasAttribute('open');
  }
  set open(val) {
    if (val) {
      this.setAttribute('open', ''); // refelect prop as an HTML attr
    } else {
      this.removeAttribute('open');
    }
  }
  get disabled() {
    return this.hasAttribute('disabled');
  }
  set disabled(val) {
```

```
    if (val) {
      this.setAttribute('disabled', ''); // refelect prop as an HTML attr
    } else {
      this.removeAttribute('disabled');
    }
  }

  toggleDrawer() {
    ...
  }
}

window.customElements.define('app-drawer', AppDrawer);
```

- the custom element created above can now be used just like native HTML elements i.e. `<app-drawer></app-drawer>`
  - instances of it can be declared on the page, created dynamically via JS, event listeners can be attached an so on
- `this` inside a class definition refers to the DOM itself
  - the entire DOM API is available inside the element code for example `this.children` to inspect its direct children or `this.querySelectorAll('.items')` to query nested nodes

9.1.2 Naming rules

- names of custom elements must contain a dash `"-"`
- the same name can only be registered once
- custom elements cannot be self-closing

9.2 Custom element reactions

A custom element can define special lifecycle hooks for running code during interesting times of its existence, these are called custom element reactions

Name	Called when
constructor	instance of the element is created or upgraded; useful for initializing state, setting up event listeners or creating a shadow dom
connectedCallback	called everytime the element is inserted into the DOM; useful for running setup code, such as fetching resources or rendering
disconnectedCallback	called everytime the element is removed from the DOM
attributeChangedCallback(attrName, oldVal, newVal)	called when an observed attribute has been added, removed, updated or replaced; also called for initial values when an element is created/upgraded; only attributes listed in the observerdAttributes property will receive this callback
adoptedCallback	the custom element has been moved into a new document

- to the above example `static get observedAttributes() { return ['disabled', 'open']}` needs to be added to the class to have `attributeChangedCallback` called for changes in those attributes

9.3 Creating an element that uses Shadow DOM

The Shadow DOM provides a way for an element to own, render and style a chunk of DOM that's separate from the rest of the page. You could for example hide away an entire within a single tag:

```
// chat app's implementation details are hidden away in Shadow DOM
<chat-app></chat-app>
```

To use Shadow DOM in a custom element, call `this.attachShadow` inside the constructor:

```
// Create template in js
let tmpl = document.createElement('template');
tmpl.innerHTML = `
  <style>:host { ... }</style> <!-- look ma, scoped styles -->
  <b>I'm in shadow dom!</b>
  <slot></slot>
`;
// or via HTML template tag
// <template id="shopping-template">
//   <b>I'm in shadow dom</b>
//   <slot></slot>
// </template>

customElements.define('x-foo-shadowdom', class extends HTMLElement {
  constructor() {
    super(); // always call super() first in the constructor.

    // Attach a shadow root to the element.
    let shadowRoot = this.attachShadow({mode: 'open'});
    shadowRoot.appendChild(tmpl.content.cloneNode(true));
  }
  ...
});
```

Example usage:

```
<x-foo-shadowdom>
  <p><b>User's</b> custom text</p>
</x-foo-shadowdom>

<!-- renders as -->
<x-foo-shadowdom>
  #shadow-root
    <b>I'm in shadow dom!</b>
    <slot></slot> <!-- slotted content appears here -->
</x-foo-shadowdom>
```

Demo:

```
<b>I'm in shadow dom!</b>
<p><b>User's</b> custom text</p>
```

Example from tutorial slides:

- have HTML template tag defined `<template id`"shopping-template"></template>
- instantiate Shadow DOM

- in constructor, select the template
- create a shadow DOM
- copy the template notes to the shadow root

```
let tmpl = document.querySelector('#shopping-template');
let shadowRoot = this.attachShadow({mode: 'open'});
shadowRoot.appendChild(tmpl.content.cloneNode(true));
```

- use `this.shadowRoot.getElementById()` and so on to access Shadow DOM nodes

9.4 Shadow DOM

Shadow DOM is just normal DOM with two differences: 1) how it's created/used and 2) how it behaves in relation to the rest of the page. Normally, you create DOM nodes and append them as children of another element. With shadow DOM, you create a scoped DOM tree that's attached to the element, but separate from its actual children. This scoped subtree is called a shadow tree. The element it's attached to is its shadow host. Anything you add in the shadows becomes local to the hosting element, including <style>. This is how shadow DOM achieves CSS style scoping.

9.5 Creating shadow DOM

A **shadow root** is a document fragment that gets attached to a "host" element. The act of attaching a shadow root is how the element gains its shadow DOM. To create shadow DOM for an element, call `element.attachShadow()`:

```
const header = document.createElement('header');
const shadowRoot = header.attachShadow({mode: 'open'});
shadowRoot.innerHTML = '<h1>Hello Shadow DOM</h1>'; // Could also use
    appendChild().
// header.shadowRoot === shadowRoot
// shadowRoot.host === header
```

Shadow DOM is particularly useful when creating **custom elements**. Use shadow DOM to compartmentalize an element's HTML, CSS, and JS, thus producing a "web component".

Example - a custom element attaches shadow DOM to itself, encapsulating its DOM/CSS:

```
customElements.define('fancy-tabs', class extends HTMLElement {
  constructor() {
    super(); // always call super() first in the constructor.

    // Attach a shadow root to <fancy-tabs>.
    const shadowRoot = this.attachShadow({mode: 'open'});
    shadowRoot.innerHTML = `
      <style>#tabs { ... }</style> <!-- styles are scoped to fancy-tabs! -->
      <div id="tabs">...</div>
      <div id="panels">...</div>
    `;
  }
  ...
});
```

There are a couple of interesting things going on here. The first is that the custom element creates its own shadow DOM when an instance of `<fancy-tabs>` is created. That's done in the constructor(). Secondly, because we're creating a shadow root, the CSS rules inside the <style> will be scoped to <fancy-tabs>.

9.6 Template

Template = A document or file having a preset format, used as a starting point for a particular application so that the format does not have to be recreated each time it is used.

- To create a templated content, declare some markup and wrap it in the <template> element:

```
<template id="mytemplate">
  <img src="" alt="great image">
  <div class="comment"></div>
</template>
```

Wrapping content in a <template> gives us few important properties.

1. Its content is effectively inert until activated. Essentially, your markup is hidden DOM and does not render.
2. Any content within a template won't have side effects. Script doesn't run, images don't load, audio doesn't play,... until the template is used.
3. Content is considered not to be in the document. Using document.getElementById() or querySelector() in the main page won't return child nodes of a template.

To use a template, you need to activate it. Otherwise its content will never render. The simplest way to do this is by creating a deep copy of its .content using document.importNode(). The .content property is a read-only DocumentFragment containing the guts of the template.

```
var t = document.querySelector('#mytemplate');
// Populate the src at runtime.
t.content.querySelector('img').src = 'logo.png';

var clone = document.importNode(t.content, true);
document.body.appendChild(clone);
```

<template> standardizes the way we do client-side templating. Making the entire web authoring process more sane, more maintainable.

9.7 Imports

Imports are a tool for loading related HTML/CSS/JS. Include import on page:

```
<head>
  <link rel="import" href="/path/to/imports/stuff.html">
</head>
```

The URL of an import is called an import location. In fact, the content of an import is called an import document. You're able to manipulate the guts of an import using standard DOM APIs! To access the content of an import, use the link element's `.import` property

`var content = document.querySelector('link[rel="import"]').import;` The HTML Template element is a natural fit for HTML Imports