北京郵電大學
Beijing University of Posts and Telecommunications

Web Programming

Web开发技术基础

第7章 异步更新与REST API

计算机学院

授课人：王尊亮

第7章 异步更新与REST API

Alpine.js

{REST}

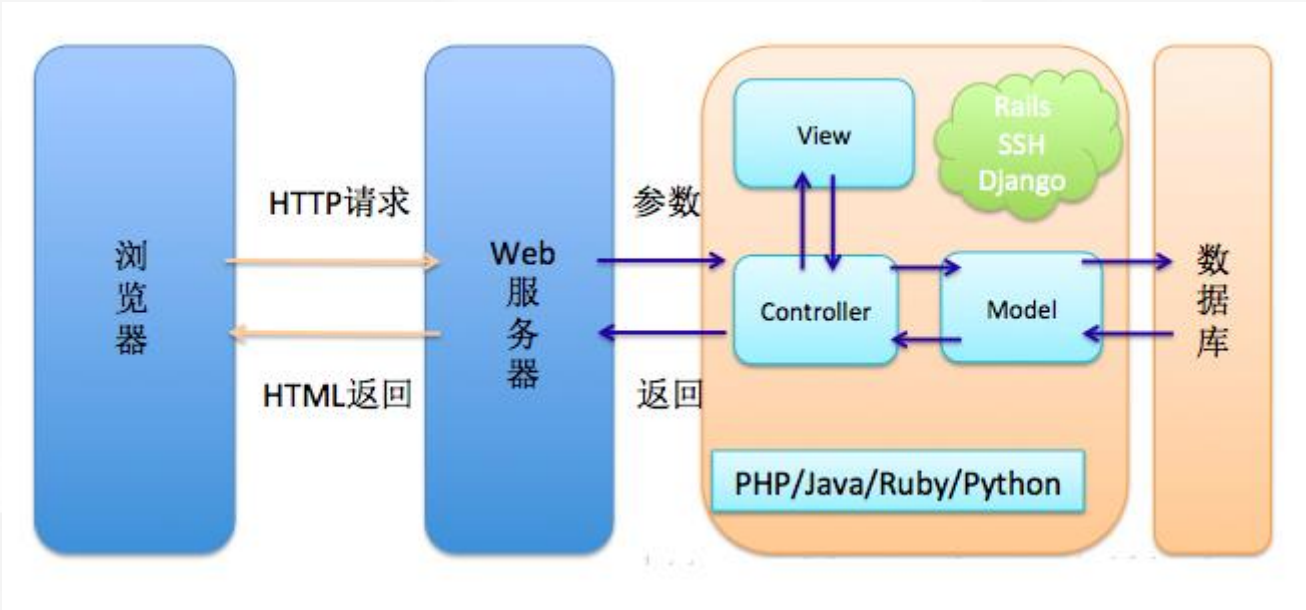
RESTful
Web Services

7.1.1 AJAX

Web系统日益复杂，MVC、ORM等概念被引入到Web开发中来。

2004年Ruby on Rails，2004年Struts，2005年 Django，2006年ThinkPHP……

2001年Hibernate，2001年iBATIS（MyBatis的前身）



后端MVC框架时代

后端开发工程师

\+

前端开发 or 美工

用户每次请求后都需要重新加载一个新页面

7.1.1 AJAX

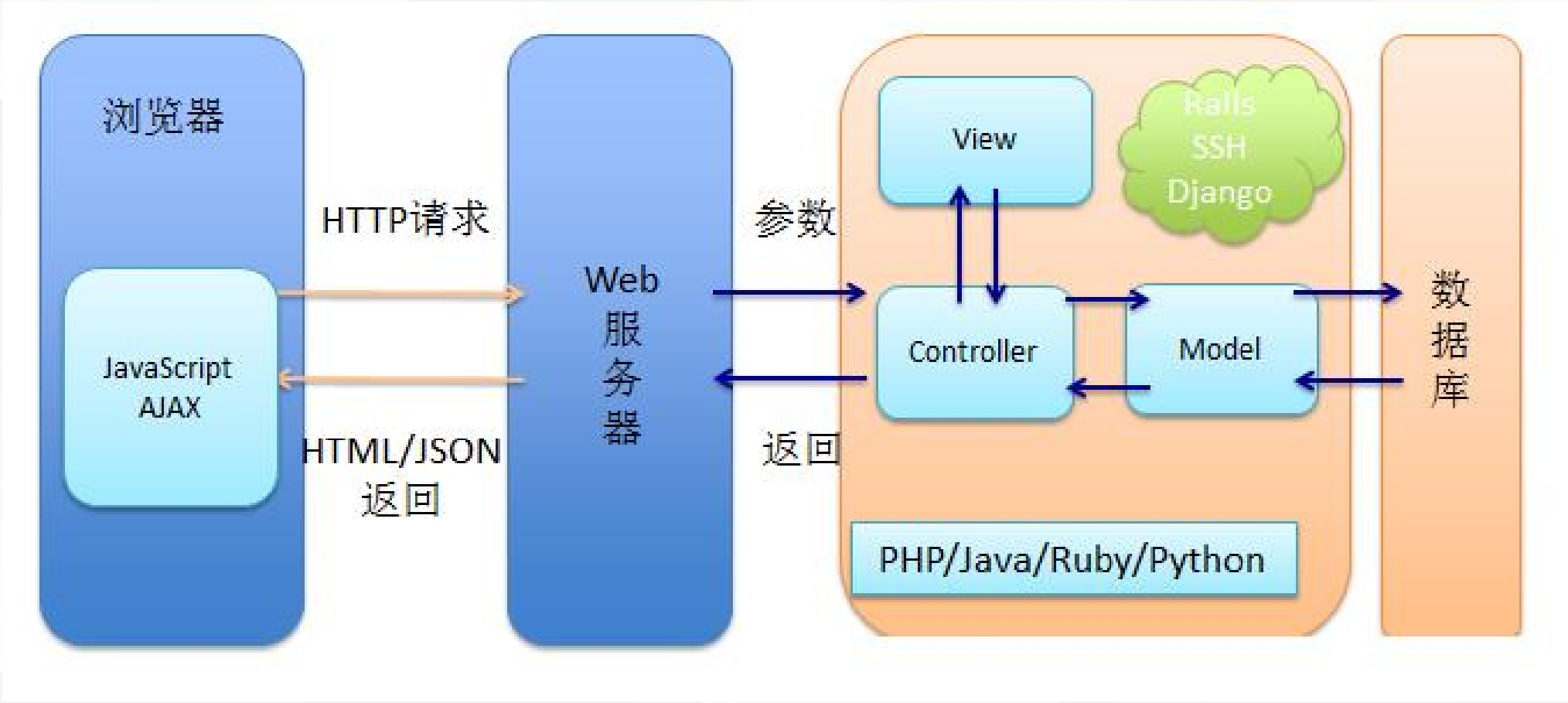

Baidu 百度
北京邮电
北京邮电大学
北京邮电大学招待所
北京邮电大学考研网


编辑于昨天 18:04
赞同 17
3 条评论
分享
收藏
喜欢


* 邮件地址
buptnetwork
@
163.com
该邮件地址已被注册
推荐您注册手机号码@163.com
免费注册
您还可以选择：
buptnetwork@yeah.net
(可以注册)
buptnetwork@163.com
(已被注册)


槽的。
3，国内没有汽车文化的积累。看TG的车评，很多都是机械层面的，三个主持的汽车知识也很丰富。国内的车评，跟广告也没什么区别。
发布于 2015-11-30
添加评论
感谢
分享
收藏
没有帮助
举报
作者保留权利
更多
我来回答这个问题
写回答...

7.1.1 AJAX



前后端半分离的方案（AJAX+JSON）

7.1.1 AJAX

- AJAX概念
  - Asynchronous Javascript And XML（异步JavaScript和XML）
  - AJAX不是一种新的编程语言，而是一种用于创建更好更快交互性更强的Web应用程序的技术。
  - 使用Javascript向服务器提出请求并处理响应而不阻塞用户，即可以在不重新加载整个网页的情况下，对网页的某部分进行更新。传统的网页（不使用 AJAX）如果需要更新内容，必须重载整个网页页面。
  - AJAX的核心对象是XMLHTTPRequest。示例： http://www.runoob.com/try/try.php?filename=tryajax_first

XHR即XMLHTTPRequest

Fetch是ES6中新增的全局函数，是XHR的替代方案。

7.1.1 AJAX

- 原生AJAX使用

```
var xmlhttp;
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
```

创建 XMLHttpRequest 对象

7.1.1 AJAX

- 原生AJAX使用

open(*method*,*url*,*async*)

method：请求的类型；GET 或 POST

url：文件在服务器上的位置

async：true（异步）或 false（同步），默认为true，JavaScript 无需等待服务器的响应，在等待服务器响应时执行其他脚本，当响应就绪后对响应进行处理

send(*string*)

string：仅用于 POST 请求

向服务器发送请求

7.1.1 AJAX

- 原生AJAX使用
  - GET请求示例：

```
xmlhttp.open("GET","demo_get2.html?fname=Henry&lname=Ford",true);
xmlhttp.send();
```

  - POST请求示例：
  - 如果需要像 HTML 表单那样 POST 数据，需使用 setRequestHeader() 来添加 HTTP 头
  - xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");

向服务器发送请求

7.1.1 AJAX

- 原生AJAX使用
  - POST请求示例：

```
xmlhttp.open("POST","ajax_test.html",true);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xmlhttp.send("fname=Henry&lname=Ford");
```

向服务器发送请求

7.1.1 AJAX

- 获取服务器响应
  - 获得来自服务器的响应，需使用 XMLHttpRequest 对象的 responseText 或 responseXML 属性。
  - responseText：获得字符串形式的响应数据。
  - responseXML：获得 XML 形式的响应数据
- 何时获取响应：
  - onreadystatechange：readystate发送变动时调用此函数

7.1.1 AJAX

onreadystatechange	存储函数（或函数名），每当 readyState 属性改变时，就会调用该函数。
readyState	存有 XMLHttpRequest 的状态。从 0 到 4 发生变化。 • 0: 请求未初始化 • 1: 服务器连接已建立 • 2: 请求已接收 • 3: 请求处理中 • 4: 请求已完成，且响应已就绪
status	200: "OK" 404: 未找到页面

```
xmlhttp.onreadystatechange=function()
  { if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
    }
  }
```

7.1.1 AJAX

- Jquery中有多个ajax操作方法：
  - **load()、 $.get()、 $.getJSON() 、 $.post() 、$.ajax()**
  - load()
    - 从服务器加载数据，并把返回的数据放置到指定的元素中

```
$("button").click(function(){
        $("#div1").load("demo_test.txt");
});
```

7.1.1 AJAX

- Jquery中有多个ajax操作方法：
  - **load()、 $.get()、 $.getJSON() 、 $.post() 、$.ajax()**
  - **$.get()** ：使用 HTTP GET 请求从服务器加载数据
    - 语法：
    - $.get(*URL， data， function(data, status， xhr)，dataType)*
    - URL：提交地址，必需
    - data：提交时携带的数据，可选
    - function：回调函数，data--返回数据，status—状态
    - xhr*包含 XMLHttpRequest 对象*
    - *dataType*：服务器响应的数据类型，默认自动判断

7.1.1 AJAX

- Jquery中有多个ajax操作方法：
  - **load()、 $.get()、 $.getJSON() 、 $.post() 、$.ajax()**
  - **$. post()** ： 使用 HTTP POST请求从服务器加载数据
    - 语法：
    - $.post(*URL， data， function(data, status，xhr)，dataType)*
    - URL：提交地址，必需
    - data：提交时携带的数据，可选
    - function：回调函数，data--返回数据，status—状态
    - xhr*包含 XMLHttpRequest 对象*
    - *dataType*：服务器响应的数据类型，默认自动判断

7.1.1 AJAX

- Jquery中有多个ajax操作方法：
  - **load()、 $.get()、 $.getJSON() 、 $.post() 、$.ajax()**
  - **$. ajax() ：该方法通常用于其他方法不能完成的请求。**
    - 语法：
    - $.ajax(*{name:value, name:value, ... }*)
    - type: “GET“ 、”POST”等
    - url：提交地址
    - dataType：预期返回数据类型
    - success(*result,status,xhr*) 成功时的处理函数
    - error(*xhr,status,error*)如果请求失败要运行的函数

7.1.2 Fetch

Fetch 是JavaScript新增的一个API，用于在 JavaScript 脚本里面发出 HTTP 请求，其目的是提供一个更理想的XMLHTTPRequest替代方案。

```
fetch("http://example.com/movies.json")
  .then((response) => response.json())
  .then((data) => console.log(data));
```

示例1：使用fetch获取数据

```
const data = { username: "example" };

fetch("https://example.com/profile", {
  method: "POST", // or 'PUT'
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.stringify(data),
})
  .then((response) => response.json())
  .then((data) => {
    console.log("Success:", data);
  })
  .catch((error) => {
    console.error("Error:", error);
  });
```

示例2：使用fetch提交数据

https://wangdoc.com/webapi/fetch

https://developer.mozilla.org/zh-CN/docs/Web/API/Fetch_API/Using_Fetch

7.1.2 Fetch

https://wangdoc.com/webapi/fetch

fetch()的功能与 XMLHttpRequest 基本相同，但有三个主要的差异。

（1）fetch()使用 Promise，不使用回调函数，因此大大简化了写法，写起来更简洁。

（2）fetch()采用模块化设计，API 分散在多个对象上（Response 对象、Request 对象、Headers 对象），更合理一些；相比之下，XMLHttpRequest 的 API 设计并不是很好，输入、输出、状态都在同一个接口管理，容易写出非常混乱的代码。

（3）fetch()通过数据流（Stream 对象）处理数据，可以分块读取，有利于提高网站性能表现，减少内存占用，对于请求大文件或者网速慢的场景相当有用。XMLHTTPRequest 对象不支持数据流，所有的数据必须放在缓存里，不支持分块读取，必须等待全部拿到后，再一次性吐出来。

关于Primise的说明：https://wangdoc.com/es6/promise

7.1.3 Axios网络请求库

https://www.axios-http.cn/docs/intro

Axios 是一个基于JavaScript中的Promise（异步编程方案）的网络请求库，对AJAX请求进行了进一步封装。

主要特性：

- 从浏览器创建 XMLHttpRequests
- 从 node.js 创建 http 请求
- 支持 Promise API
- 拦截请求和响应
- 转换请求和响应数据
- 取消请求
- 自动转换JSON数据
- 客户端支持防御XSRF

```
// 向给定ID的用户发起请求
axios.get('/user?ID=12345')
  .then(function (response) {
    // 处理成功情况
    console.log(response);
  })
  .catch(function (error) {
    // 处理错误情况
    console.log(error);
  })
  .then(function () {
    // 总是会执行
  });
```

```
axios.post('/user', {
    firstName: 'Fred',
    lastName: 'Flintstone'
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

7.1.3 Axios网络请求库

Axios API

- axios.request(config)
- axios.get(url[, config])
- axios.delete(url[, config])
- axios.head(url[, config])
- axios.options(url[, config])
- axios.post(url[, data[, config]])
- axios.put(url[, data[, config]])
- axios.patch(url[, data[, config]])

```
// 发起一个post请求
axios({
  method: 'post',
  url: '/user/12345',
  data: {
    firstName: 'Fred',
    lastName: 'Flintstone'
  }
});
```

7.1.3 Axios网络请求库

Axios支持拦截器：

在请求或响应被 then 或 catch 处理前拦截它们。

```
// 添加请求拦截器
axios.interceptors.request.use(function (config) {
    // 在发送请求之前做些什么
    return config;
  }, function (error) {
    // 对请求错误做些什么
    return Promise.reject(error);
  });

// 添加响应拦截器
axios.interceptors.response.use(function (response) {
    // 2xx 范围内的状态码都会触发该函数。
    // 对响应数据做点什么
    return response;
  }, function (error) {
    // 超出 2xx 范围的状态码都会触发该函数。
    // 对响应错误做点什么
    return Promise.reject(error);
  });
```

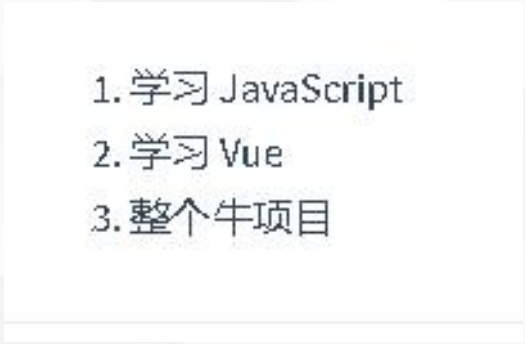
7.2 轻量级前端数据绑定框架

- AJAX仅解决了不重新加载整个网页的情况下，向服务器发起请求获取数据，然后对网页的局部进行更新。
- 但现代Web应用对前端还有更高要求：
  - 如何方便的修改DOM?
  - 如何方便的编写DOM事件响应?
  - 如何方便的实现DOM元素之间的逻辑联动?
  - 有没有可能不经过后端自己切换显示的“页面”，即页面内部路由?
  - 如何实现负责前端页面的多人协作工程化开发?
  - ……

7.2 轻量级前端数据绑定框架

- 前端MV*框架时代：Vue.js vs Angularjs vs Reactjs

- 轻量级前端数据绑定框架
  - Alpine.js
  - petite-vue



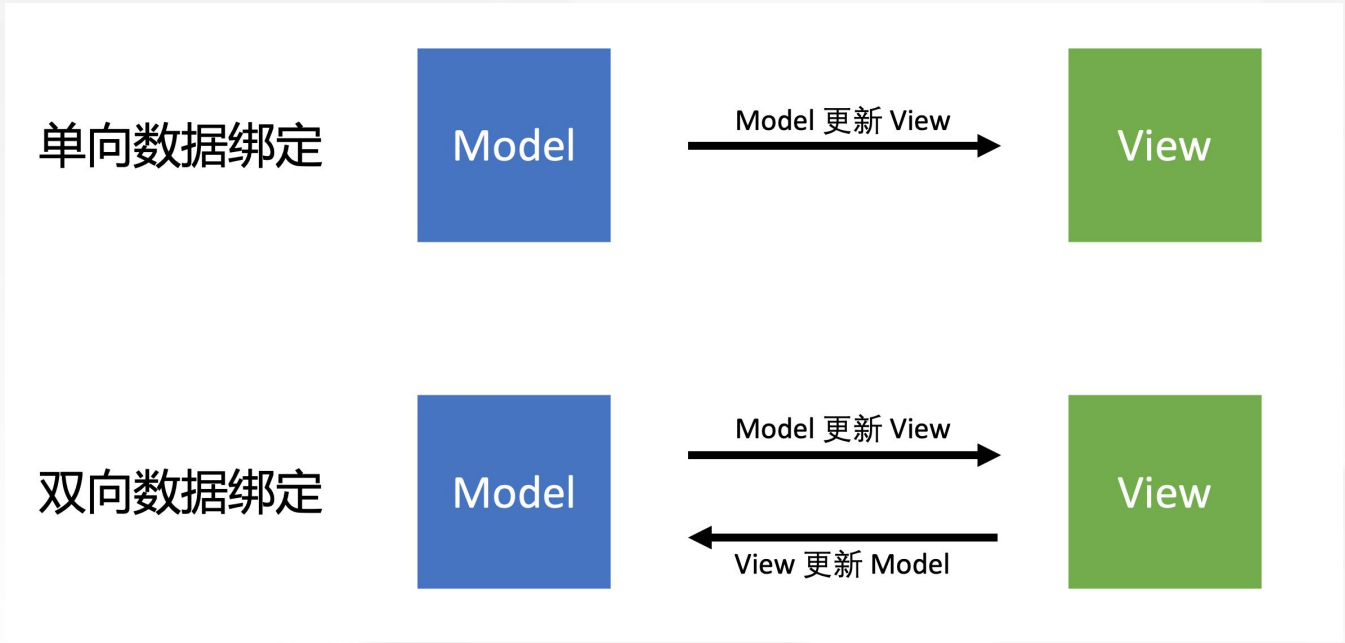
Alpine.js

```
<div id="app-4">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>
```

```
var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
      { text: '学习 JavaScript' },
      { text: '学习 Vue' },
      { text: '整个牛项目' }
    ]
  }
})
```

7.2.1 前端数据绑定

数据绑定：VIEW和MODEL之间的映射关系




```
<div id="app">
    <p>{{ message }}</p>
    <input v-model="message">
</div>

<script>
new Vue({
  el: '#app',
  data: {
    message: 'Runoob!'
  }
})
```

Runoob!

Runoob!

https://www.runoob.com/try/try.php?filename=vue2-v-model

7.2.2 Alpine.js与petite-vue

https://alpinejs.dev/ https://www.alpinejs.cn/

Alpine.js 通过很低的成本提供了与 Vue 或 React 这类大型框架相近的响应式和声明式特性。

可以直接在网页中引入AlpineJS，不涉及安装、构建、前端路由等复杂的前端工程化内容，可以用来替代JQuery。

- 动态的更新DOM
- 条件显示和隐藏DOM节点
- 绑定用户输入
- 监听事件并做出处理

```
<script src="//unpkg.com/alpinejs" defer></script>

<div x-data="{ open: false }">
    <button @click="open = true">Expand</button>

    <span x-show="open">
      Content...
    </span>
</div>
```

7.2.2 Alpine.js与petite-vue

https://alpinejs.dev/ https://www.alpinejs.cn/

Alpine.js 提供了15个类似Vue的指令用来简化DOM操作。

x-data

Declare a new Alpine component and its data for a block of HTML

```
<div x-data="{ open: false }">
    ...
</div>
```

x-bind

Dynamically set HTML attributes on an element

```
<div x-bind:class="! open ? 'hidden' : ''">
  ...
</div>
```

7.2.2 Alpine.js与petite-vue

https://alpinejs.dev/ https://www.alpinejs.cn/

Alpine.js 提供了15个类似Vue的指令用来简化DOM操作。

x-on

Listen for browser events on an element

```
<button x-on:click="open = ! open">
  Toggle
</button>
```

x-text

Set the text content of an element

```
<div>
  Copyright ©

  <span x-text="new Date().getFullYear()"></span>
</div>
```

7.2.2 Alpine.js与petite-vue

https://alpinejs.dev/ https://www.alpinejs.cn/

Alpine.js 提供了15个类似Vue的指令用来简化DOM操作。

x-html

Set the inner HTML of an element

```
<div x-html="(await axios.get('/some/html/partial')).data">
  ...
</div>
```

x-model

Synchronize a piece of data with an input element

```
<div x-data="{ search: '' }">
  <input type="text" x-model="search">

  Searching for: <span x-text="search"></span>
</div>
```

7.2.2 Alpine.js与petite-vue

https://alpinejs.dev/ https://www.alpinejs.cn/

Alpine.js 提供了15个类似Vue的指令用来简化DOM操作。

x-show

Toggle the visibility of an element

```
<div x-show="open">
  ...
</div>
```

x-transition

Transition an element in and out using CSS transitions

```
<div x-show="open" x-transition>
  ...
</div>
```

7.2.2 Alpine.js与petite-vue

https://alpinejs.dev/ https://www.alpinejs.cn/

Alpine.js 提供了15个类似Vue的指令用来简化DOM操作。

x-for

Repeat a block of HTML based on a data set

```
<template x-for="post in posts">
  <h2 x-text="post.title"></h2>
</template>
```

x-if

Conditionally add/remove a block of HTML from the page entirely.

```
<template x-if="open">
  <div>...</div>
</template>
```

7.2.2 Alpine.js与petite-vue

https://alpinejs.dev/ https://www.alpinejs.cn/

Alpine.js 提供了15个类似Vue的指令用来简化DOM操作。

x-init

Run code when an element is initialized by Alpine

```
<div x-init="date = new Date()"></div>
```

x-effect

Execute a script each time one of its dependancies change

```
<div x-effect="console.log('Count is '+count)"></div>
```

7.2.2 Alpine.js与petite-vue

https://alpinejs.dev/ https://www.alpinejs.cn/

Alpine.js 提供了15个类似Vue的指令用来简化DOM操作。

x-ref

Reference elements directly by their specified keys using the `$refs` magic property

```
<input type="text" x-ref="content">

<button x-on:click="navigator.clipboard.writeText($refs.co
  Copy
</button>
```

x-cloak

Hide a block of HTML until after Alpine is finished initializing its contents

```
<div x-cloak>
  ...
</div>
```

7.2.2 Alpine.js与petite-vue

petite-vue，轻量级Vue，类似Alpine.js。https://github.com/vuejs/petite-vue

```
<script src="https://unpkg.com/petite-vue" defer init></script>

<!-- anywhere on the page -->
<div v-scope="{ count: 0 }">
  {{ count }}
  <button @click="count++">inc</button>
</div>
```

petite-vue is an alternative distribution of Vue optimized for progressive enhancement. It provides the same template syntax and reactivity mental model as standard Vue. However, it is specifically optimized for "sprinkling" a small amount of interactions on an existing HTML page rendered by a server framework

7.2.3 Alpine Todolist Demo

https://gitee.com/buptnetwork/vue-todo-demo(alpine版+vue版)

引入了alpinejs和axios

7.2.3 Alpine Todolist Demo

https://gitee.com/buptnetwork/vue-todo-demo

```
<button class="btn btn-primary float-right" @click="addTodo()">Add</button>
<table id="todoTable" class="table">
    <thead>
    <tr>
        <th>Index</th>
        <th>Content</th>
        <th>Operations</th>
    </tr>
    </thead>
    <tbody>
    <template x-for="(todo, index) in todos">
        <tr>
            <td x-text="index+1"></td>
            <td x-text="todo.content"></td>
            <td>
                <button class="btn btn-xs btn-success"
                        @click="editTodo(todo)">Edit
                </button>
                <button class="btn  btn-xs btn-danger"
                        @click="removeTodo(todo)">Delete
                </button>
            </td>
```

7.2.3 Alpine Todolist Demo

```
<div class="container" x-data="tasks()" x-init="loadTodos()">
    <a href="/">alpine版</a> | <a href="/vue">vue版</a>
    <h2>AlpineJS+Axios TodoList</h2>
    <button class="btn btn-primary float-right" @click="addTodo()
    <table id="todoTable" class="table">
        <thead>
        <tr>
            <th>Index</th>
            <th>Content</th>
            <th>Operations</th>
        </tr>
        </thead>
        <tbody>
        <template x-for="(todo, index) in todos">
            <tr>
                <td x-text="index+1"></td>
                <td x-text="todo.content"></td>
                <td>
                    <button class="btn btn-xs btn-success"
                            @click="editTodo(todo)">Edit
```

```
function tasks() {
    return {
        todos: [],
        content: '',
        show_modal: false,
        cur_todo_id: '',
        loadTodos() {
            let self = this;
            axios.get('/todo/')
                .then(function (res) {
                    self.todos = res.data
                })
                .catch(function (error) {
                    console.log(error);
                })
        },
```

7.2.3 Alpine Todolist Demo

```
addTodo() {
    let self = this;
    self.show_modal = true
    self.content = ''
    self.cur_todo_id = ''
},
editTodo(todo) {
    let self = this;
    self.show_modal = true
    self.content = todo.content;
    self.cur_todo_id = todo.id;
},
```

弹出新建/编辑模态框

```
saveTodo() {
    let self = this;
    if (!self.cur_todo_id) {
        axios.post('/todo/', {
            content: self.content,
        })
            .then(function (res) {
                self.loadTodos();
                self.show_modal = false;
            })
            .catch(function (error) {
                console.log(error);
            })
    } else {
        axios.put('/todo/' + self.cur_todo_id, {
            content: self.content,
        })
            .then(function (res) {
                self.loadTodos();
                self.show_modal = false;
            })
            .catch(function (error) {
                console.log(error);
```

发送新建/编辑请求

7.3 REST API设计

Web API:

使用**HTTP协议**通过网络调用的API，简而言之，Web API就是一个Web系统，通过访问URI可以与服务器完成信息交互，或者获得存放在服务器的数据信息等。常见的是XML、JSON等格式的返回数据，也称为“XML over HTTP”或“JSON over HTTP”。

REST API:

REST即表述性状态转移（英文：Representational State Transfer，简称REST）是Roy Fielding在2000年他的博士论文中提出来的一种软件架构风格。严格意义上的REST API应为遵循REST规定的约束和原则的一类Web API，目前很多不是严格遵循规定约束的API也被叫做REST风格的API，也就是广义的REST API。

7.3 REST API设计

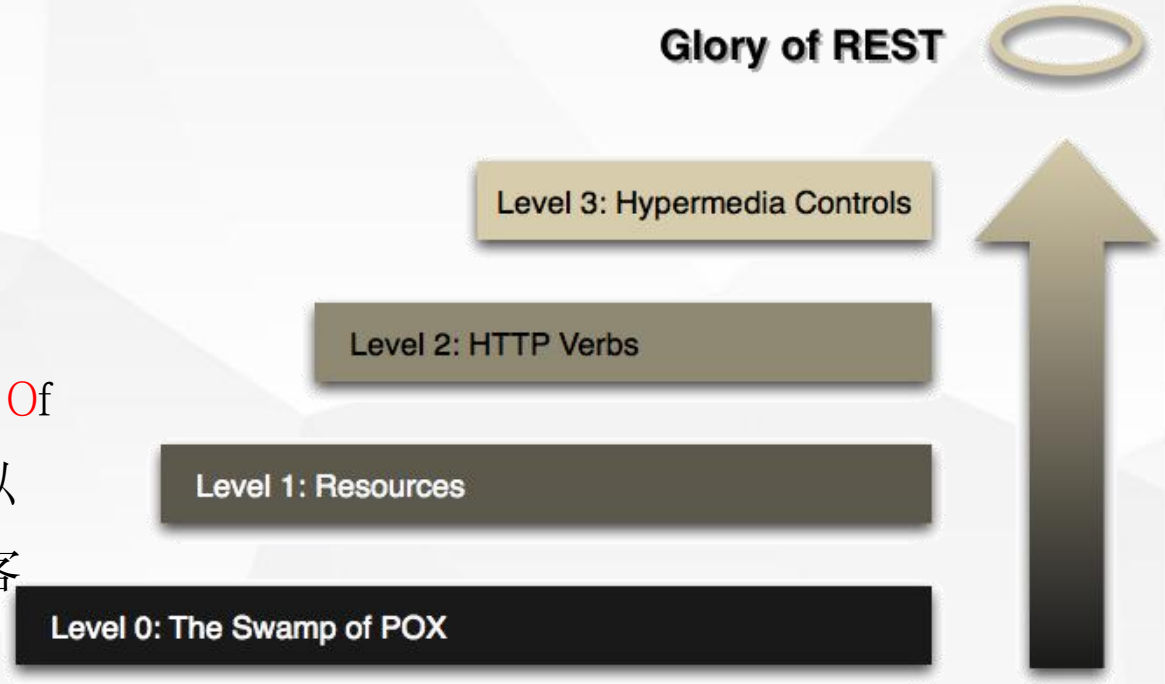
Martin Fowler（敏捷开发模式创建人之一）在文章中介绍了表示REST API设计级别（成熟度）的Richardson Maturity Model：

REST LEVEL 0:仅使用HTTP作为传输方式，实际上只是远程方法调用（RPC）的一种具体形式。SOAP 和 XML-RPC 都属于

REST LEVEL 1:引入资源的概念,每个资源有对应的标识符

REST LEVEL 2:引入HTTP动词和HTTP状态码（GET/POST/PUT/PATCH/ DELETE）表示对资源的操作

REST LEVEL 3:引入HATEOAS概念（Hypermedia As The Engine Of Application State），**在资源的表达中包含了链接信息**，可以告诉客户端下一步可以做什么和可以操作资源的URI,使得客户端无需事先知道所有操作的URI



7.3 REST API设计

HATEOAS的优点：

客户端只需要知道URI的固定入口即可根据需要访问到所有的资源，实现了客户端和服务器更好的解耦，只要固定入口不变，其它资源的URI发生变化时，不会受到影响。

HATEOAS的问题：

增加的内嵌资源的URI并不一定是前端想要的和方便使用的，特别是angular、vue等前端框架都有一套自己的路由机制，所以HATEOAS目前未得到广泛使用

HATEOAS示例

7.3 REST API设计

设计优美的Web API：

- **易于使用**：应方便用户使用，右图是一个不好用的示例，当获取好友列表时，仅返回好友id，客户端为了显示好友的名字，需要再反复调用多次根据id获取姓名的接口API，会加大网络传输消耗及延时，也增加了客户端的处理工序
- **便于更改**：升级更新时，尽量不影响正在使用的用户，或方便客户端调整
- **健壮性好**：安全可靠，如有些API直接暴露了后端数据库表项的ID，且缺乏权限验证，可能会导致数据被非法访问或操作
- **不怕公之于众**：丑陋的API会使服务提供者的技术水平受到质疑

```
{
  "friends": [
      234342,
      93734,
      197322,
        :
        :
  ]
}
```

7.3 REST API设计

设计端点：指用于访问API的URI。基本的设计原则是容易记忆，URI包含的功能一目了然。

- 短小简洁的URI
  - http://api.example.com/service/api/search 
  - http://api.example.com/search 
- 清晰易懂的URI
  - http://api.example.com/sv/u 
  - http://api.example.com/products/12345 
  - http://api.example.com/shangpin/123456
  - http://api.example.com/商品/123456 

7.3 REST API设计

设计端点：指用于访问API的URI。基本的设计原则是容易记忆，URI包含的功能一目了然。

- 不要大小写混用，标准做法是全部小写，如确实有多单词时采用脊柱法连接较为主流
  - http://api.example.com/v1/users/12345/profile-image　　脊柱法 ←
  - http://api.example.com/v1/users/12345/profile_image　　蛇形法
  - http://api.example.com/v1/users/12345/profileImage　　小驼峰法
  - http://api.example.com/v1/users/12345/ProfileImage　　大驼峰法
- 使用不会暴露服务器端架构的URI
  - http://api.example.com/cgi-bin/get_user.php?user=100
- CRUD操作不要体现在URI中，应使用HTTP的方法

7.3 REST API设计

HTTP方法的使用：URI和HTTP方法之间的关系可以认为是**操作对象和操作方法**的关系。

- URI：表示资源，用于描述某种具体的数据信息
- HTTP方法：表示对资源进行怎么的操作

方法名	说明
GET	获取资源
POST	新增资源
PUT	更新已有资源
DELETE	删除资源
PATCH	更新部分资源
HEAD	获取资源的元信息

7.3 REST API设计

POST：在REST中POST一般用来生成新的资源，非幂等。

https://api.example.com/v1/friends

https://api.example.com/v1/friends/12345

数据在服务器端完成注册，能够通过新的URI进行访问

7.3 REST API设计

PUT：用发送的数据完全替换原有的资源信息。幂等

POST	https://api.example.com/v1/friends	将新数据注册到URI之下
PUT	https://api.example.com/v1/friends/12345	更新指定的数据本身

PATCH：只是更新资源的某部分数据，如只更新用户信息中的昵称字段

DELETE:删除指定的资源

7.3 REST API设计

响应数据格式：XML、JSON。

有些Web API支持客户端选择返回数据格式，如：

http://api.example.com/v1/users?format=json

响应消息的封装格式：

方案一：充分利用HTTP的状态码，1xx：消息，2xx:成功，3xx:重定向，4xx:客户端引起的错误，5xx:服务器端引起的错误，REST推荐做法，状态码含义周知，缺点是不够灵活。

方案二：自定义统一的返回结构，下图为一种示例，success表示是否执行成功，code返回状态码，message返回提示信息，result中存储返回的业务结果数据。

7.3 REST API设计

状态码：

状态码	名称	说明
200	OK	请求成功
201	Created	请求成功，新的资源已创建
202	Accepted	请求成功
204	No Content	没有内容
300	Multiple Choices	存在多个资源
301	Moved Permanently	资源被永久转移
302	Found	请求的资源被暂时转移
303	See Other	引用他处
304	Not Modified	自上一次访问后没有发生更新
307	Temporary Redirect	请求的资源被暂时转移

7.3 REST API设计

状态码：

400	Bad Request	请求不正确
401	Unauthorized	需要认证
403	Forbidden	禁止访问
404	Not Found	没有找到指定的资源
405	Method Not Allowed	无法使用指定的方法
406	Not Acceptable	同 Accept 相关联的首部里含有无法处理的内容
408	Request Timeout	请求在规定时间内没有处理结束
409	Conflict	资源存在冲突
410	Gone	指定的资源已不存在
413	Request Entity Too Large	请求消息体太大
414	Request-URI Too Long	请求的 URI 太长

7.3 REST API设计

状态码：

415	Unsupported Media Type	不支持所指定的媒体类型
429	Too Many Requests	请求次数过多
500	Internal Server Error	服务器端发生错误
503	Service Unavailable	服务器暂时停止运行

7.3 REST API设计

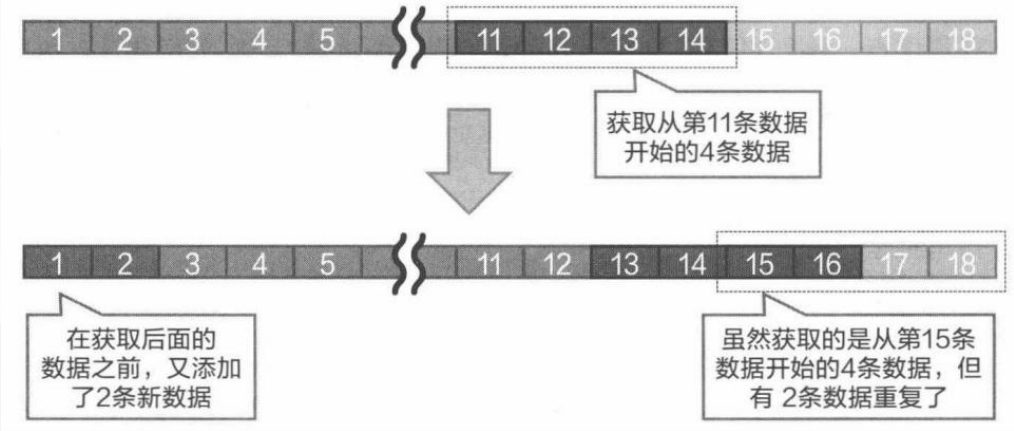
分页：

使用相对位置

http://api.example.org/books/?limit=100&offset=400 offset为偏移量，limit为请求最大条数

http://api.example.org/books/?per_page=10&page=5 page为请求的页码，per_page为每页条数

当有数据插入或删除时，采用相对位置获取数据的两个请求结果可能会有重复或不连续，如分页读取头条新闻时



使用绝对位置：http://api.example.org/books/?limit=100&since_id=12345 或since_timestamp

7.3 REST API设计

过滤:

http://api.example.org/books/?name=xxx

排序:

http://api.example.org/books/?sort=-publish_date,+name

响应数据的字段:

有些Web API支持通过查询参数来指定返回特定的字段，如:

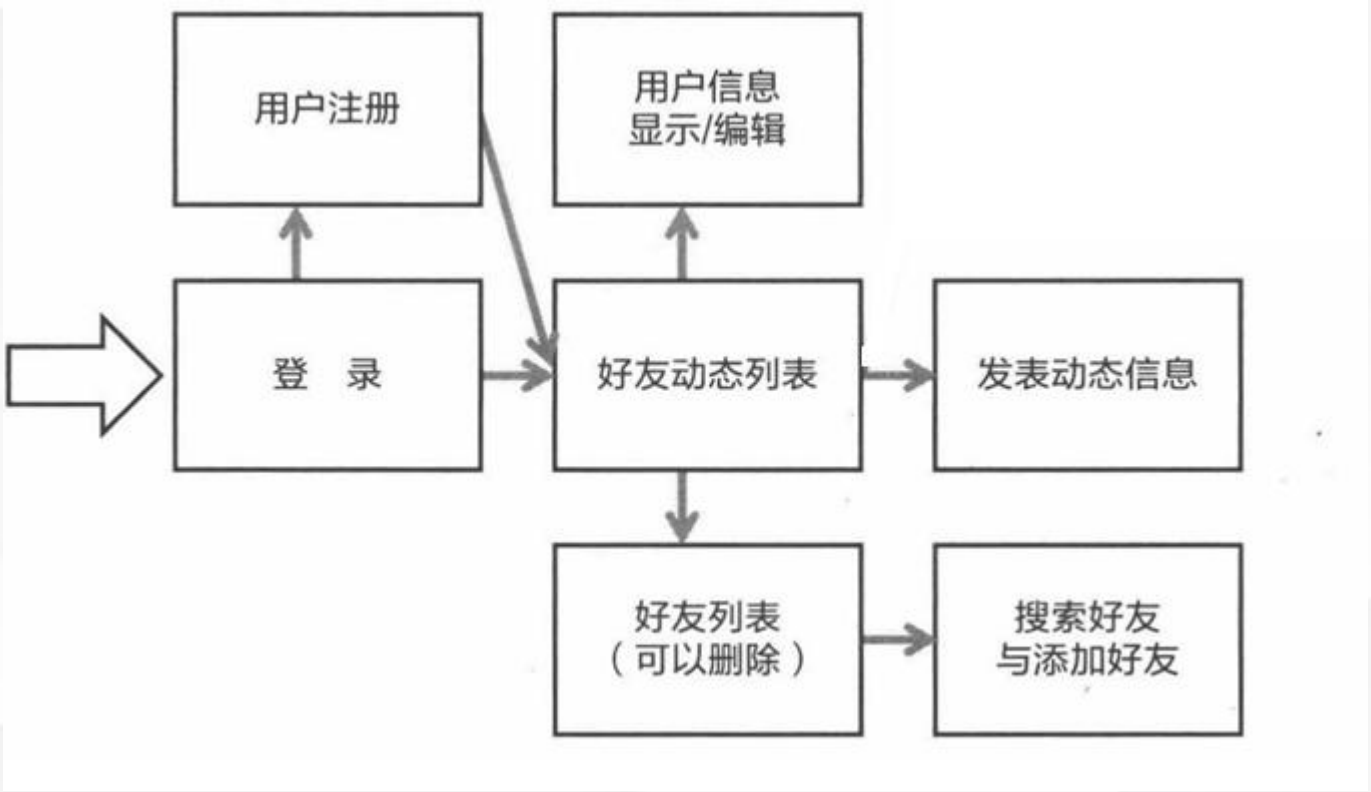
http://api.example.com/v1/users/12345/?fields=name,age

7.3 REST API设计

思考，针对如下的简单社交服务系统应如何设计接口？

主要功能点：

1. 注册登录
2. 关注好友
3. 发表动态



一个简单的社交服务系统的主要页面功能

7.3 REST API设计

部分接口

目的	端点	方法
获取用户列表	http://api.example.com/v1/users	GET
新用户注册	http://api.example.com/v1/users	POST
获取特定用户的信息	http://api.example.com/v1/users/:id	GET
更新用户信息	http://api.example.com/v1/users/:id	PUT/PATCH
删除用户信息	http://api.example.com/v1/users/:id	DELETE
获取当前用户的好友列表	http://api.example.com/v1/users/:id/friends	GET
添加好友	http://api.example.com/v1/users/:id/friends	POST
删除好友	http://api.example.com/v1/users/:id/friends/:id	DELETE

7.3 REST API设计

部分接口

目的	端点	方法
发表动态信息	http://api.example.com/v1/blogs	POST
删除动态信息	http://api.example.com/v1/blogs/:id	DELETE
编辑动态信息	http://api.example.com/v1/blogs/:id	PUT
获取好友的动态列表	http://api.example.com/v1/users/:id/friends/blogs	GET
获取特定用户的动态信息	http://api.example.com/v1/users/:id/blogs	GET

7.3 REST API设计

Gitee的OpenAPI https://gitee.com/api/v5/swagger/

示例：列出仓库的所有Issues

https://gitee.com/Tencent/APIJSON/issues