Adv Web Dev Arch - Lecture 3

Amir Amintabar, PhD

高级网页开发架构——第3讲

阿米尔・阿明塔巴尔, 博士

Outline

- 1- Review
- 2-Why we need Web APIs
  - Example:
  - Challenges of launching an online movie store
- 3- Anatomy of Web APIs
- 4- Postman
- 5- Intro to RESTful **architecture** style
  - The five key principles of RESTful service-enabled applications
  - Microservices
  - Example of an educational application with Microservice Architecture

Part 1

大纲

- 1- 回顾
- 2- 为什么我们需要 Web API
  - 示例：
  - 上线一个在线电影商店的挑战
- 3- Web API 的结构
- 4- Postman
- 5- RESTful 架构 入门 **指南**
  - RESTful 服务化应用的五大核心原则
  - 微服务
  - 采用微服务架构的教育类应用程序示例

第一部分

Review: Labs

- Lab0: The four review quizzes were based on week1 content: COMP4537-1- review HTML CSS JS Hosting.pptx

- Lab1: Were based on the week 2 of lecture material: COMP4537-2 Architecture JSON LocalStorage.pptx

复习：实验课

- 实验课0：四次复习测验均基于第一周内容：COMP4537-1——复习HTML CSS、JS与托管.pptx

- 实验1：基于第2周授课内容：COMP4537-2 架构 JSON 本地存储.pptx

Review: Labs

- You should have a basic understanding of SQL-based database, JavaScript, HTML and CSS *based on your prior coursework*:

Web Development 1
COMP1537: https://www.bcit.ca/outlines/20211087560/

- Create JavaScript classes and objects.
- Create JavaScript code that handles various types of events on th
- Apply the standard three-tiered web architecture (data, applicatio app with that architecture.
- Utilize various storage mechanisms on both the client-side and se requirements.
- Design and implement a Progressive Web Application (PWA).
- Apply core JavaScript concepts such as functions as objects, vario control constructs, and how JSON is utilized.

Web Development 2
COMP2537: https://www.bcit.ca/outlines/20211087580/

- Utilize techniques to dynamically serve web content such as image
- Perform DOM manipulation via jQuery on the server-side.
- Utilize a routing framework on the server-side.
- Use asynchronous programming on the client and server

Relational Database Design and SQL
COMP1630: https://www.bcit.ca/outlines/20211086178/

- Use SQL - DDL to implement a relational database.
- Use SQL for data manipulation such as the basic Select statement.
- Use SQL for advanced manipulation such as Group BY, Having, Correlated subqueries.
- Create stored procedures and triggers.
- Discuss techniques for transaction management and concurrency control.

.earning Resources

复习：实验课

- 您应已具备 SQL 数据库、JavaScript、HTML 和 CSS 的基础知识 根据您此前修读的课程：

网页开发 1
COMP1537：https://www.bcit.ca/outlines/20211087560/

- Create JavaScript classes and objects.
- Create JavaScript code that handles various types of events on th
- Apply the standard three-tiered web architecture (data, applicatio app with that architecture.
- Utilize various storage mechanisms on both the client-side and se requirements.
- Design and implement a Progressive Web Application (PWA).
- Apply core JavaScript concepts such as functions as objects, vario control constructs, and how JSON is utilized.

网页开发 p模块 2
COMP2537：https://www.bcit.ca/outlines/20211087580/

- Utilize techniques to dynamically serve web content such as image
- Perform DOM manipulation via jQuery on the server-side.
- Utilize a routing framework on the server-side.
- Use asynchronous programming on the client and server

关系型数据库设计与SQL（COMP1630）：
https://www.bcit.ca/outlines/20211086178/

- Use SQL - DDL to implement a relational database.
- Use SQL for data manipulation such as the basic Select statement.
- Use SQL for advanced manipulation such as Group BY, Having, Correlated subqueries.
- Create stored procedures and triggers.
- Discuss techniques for transaction management and concurrency control.

.earning Resources

Review:

- We face a choice: do we pause and potentially bore those who are already familiar with the content from previous courses, or do we move forward and assume that students without the background will catch up? How can we find a compromise here?

2021 COMP1537, https://www.bcit.ca/outlines/20211087560/

复习：

- 我们面临一个选择：是暂停讲解，从而可能令那些已在先前课程中接触过相关内容的学生感到乏味；还是继续推进教学进度，并假设缺乏相关背景知识的学生能够自行跟上？我们该如何在此二者之间寻求折中方案？

2021 年 COMP1537 课程，https://www.bcit.ca/outlines/20211087560/

Review: JSON

- Last we learned JSON is a way of **converting an object into a string**, so that we can serially store it or send it to other machines on the net. Then to pars it back to the same object

- // Object to String

```
let myJSON = JSON.stringify(myObj);
```

- // String to Object
- `let myObj = JSON.parse(myJSON);`

- `Q1: what are the JavaScript data types?`
- `Q2: what would be the result of typeof (null) ?`

复习：JSON

- 上一节我们学习了，JSON 是一种 **将对象转换为字符串** 的方式，以便我们能够按顺序存储该字符串，或将其发送至网络上的其他设备。随后再将其解析回原始对象。

- // 对象转字符串

```
声明我的 JSON =JSON.stringify(myObj);
```

- // 字符串转对象
- 声明 `myObj = JSON.parse(myJSON);`

- Q1：JavaScript 的数据类型有哪些?
- Q2：typeof (null) 的结果是什么?

Review: Web Storage

- We also learned about web storages
- LocalStorage and sessionStorage
- They are more **secure** compared* to cookies
- local storage is **permanent**, but session storage **expires**
- All pages, from same origin, can store and access the same data **within same browser** on your computer of course
- Web storage only handles string key/value pairs.
- **Q:** what could be the variable value in the code snippet below?
- let value = sessionStorage.getItem("key");
- A: it returns string…. or null
- * secure compared to cookie, not fully secure.

复习：Web 存储

- 我们还学习了 Web 存储机制
- localStorage 和 sessionStorage
- 与 Cookie 相比，它们更加 **安全***
- localStorage 是 **永久性** 的，而 sessionStorage 则会 **过期失效**
- 同一源下的所有网页均可在您计算机的同一浏览器内存储并访问相同的数据 **（当然）**
- Web 存储仅支持字符串类型的键值对。
- **问：** 以下代码片段中变量的可能取值是什么？
- 声明 值 = 从 sessionStorage 中获取项("key");
- 答：它返回一个字符串……或 null
- * 相较于 Cookie 更安全，但并非完全安全。

- **Q:** Now that we can only store strings in localStorage, is there a way to store an object in localStorage?

- A: yes, via JSON

- **问：** 既然 localStorage 只能存储字符串，那么有没有办法在 localStorage 中存储对象？

{v1>答：有，可通过 JSON 实现

Review: JavaScript (from COMP4537-1- review HTML CSS JS Hosting.pptx)

- Let
- const
- var
- Hoisting
- Set time out

复习：JavaScript（选自 COMP4537-1- HTML、CSS、JS 及托管复习.pptx）

- let
- const
- var
- 变量提升（Hoisting）
- 设置超时（setTimeout）

Review: JavaScript (from COMP4537-1- review HTML CSS JS Hosting.pptx)

- Q: what does

typeof []

return?

- **var**: Declares a **function-scoped** variable.
- **let**: Declares a **block-scoped**, variable.
- **const**: Declares a **block-scoped** const. const variables cannot be *re-assigned* after assigned once.

复习：JavaScript （来自 COMP4537-1- 复习 HTML CSS JS Hosting.pptx）

- 问题：

typeof []

返回什么?

- **var**：声明一个 **函数作用域** 变量。
- **let**：声明一个 **块级作用域** 的变量。
- **const**：声明一个 **块级作用域** 的常量。const 变量一旦赋值后便无法 重新赋值。

Scope of Declarations by var, let, const:

var、let、const 声明的作用域：

```
{
  var a = 10;
}
console.log(a);
```

```
{
  let a = 10;
}
console.log(a);
```

```
{
  const a = 10;
}
console.log(a);
```

```
{
  var a = 10;
}
console.log(a);
```

```
{
  let a = 10;
}
console.log(a);
```

```
{
  const a = 10;
}
console.log(a);
```

What does console.log(a) print on console ?

console.log(a) 在控制台打印什么?

10	**Uncaught ReferenceError: a is not defined**	**Uncaught ReferenceError: a is not defined**	**10**	**未捕获引用错误：a 未定义**	**未捕获的引用错误：a 未定义**
Function scoped	Block Scoped	Block Scoped	函数作用域	块级作用域	块级作用域

Q: What will console print in each of these scenarios ?

A

```
console.log (n)
var n = 150
```

B

```
var n = 150
console.log (n)
```

C

```
console.log (n)
let n = 150
```

问题：在以下每种场景中，控制台将打印什么?

A

```
console.log (n)
var n = 150
```

B

```
var n = 150
console.log (n)
```

C

```
console.log (n)
let n = 150
```

variable hoisting (only for variables declared by “var”)

- The JavaScript engine treats all variable declarations using “var” as if they are declared at the top of a functional scope (if declared inside a function, if not as if they are declared at top of the script)

```
<script>

    console.log(myVar);// will not issue error
    var myVar;
    myVar = 140;
</script>
```

JS engine moves declarations using **var** to top

Will not issue error, even though myVar was declared next line
It will simply print undefined

```
<script>

    console.log(myLet);// will issue error
    let myLet;
    myLet = 140;
</script>
```

Will issue error and terminate execution.
JS engine will NOT hoist variables declared by “let”

变量提升（仅适用于用“var”声明的变量）

- JavaScript 引擎会将所有使用“var”声明的变量视为在函数作用域顶部声明（如果在函数内部声明），否则视为在脚本顶部声明。

```
<script>

    console.log(myVar);// will not issue error
    var myVar;
    myVar = 140;
</script>
```

JS 引擎会将使用**var** 的声明移动到顶部

即使 myVar
是在下一行声明的，也不会报错
它只会输出 undefined

```
<script>

    console.log(myLet);// will issue error
    let myLet;
    myLet = 140;
</script>
```

将引发错误并终止执行。
JS 引擎不会提升由“let”声明的变量

- Q: When declaring a variable which one you should choose first? const, let or var?

- Q: What if you use a variable without declaring it in JavaScript? (btw worst practice)

A: first const, then let and never var.

A: becomes global scope

- Q：声明变量时，应优先选择哪一个？const、let 还是 var?

- Q：在 JavaScript 中，如果使用一个未声明的变量会怎样？（顺带一提，这是最差的实践方式）

14

=== vs ==

- Q1: Whats the diff between the two?
- The == (or !=) operator performs an automatic type conversion if needed.
- The === (or !==) operator will not perform any type conversion. It first compares if both sides have the same type, then compares the values

- Q2: Can a function return another function in js?

- Q3: Can a function accept another function as arguments in js?

=== 与==

- 问题1：两者的区别是什么？
- == （或!=）运算符在需要时会执行自动类型转换。
- === （或!==）运算符不会执行任何类型转换。它首先比较两边是否具有相同的类型，然后比较值。

- 问题2：JavaScript 中函数可以返回另一个函数吗？

- Q3：JavaScript 中的函数能否接受另一个函数作为参数？

Order of execution … Event Queue

- What would be the output of this script?

```
<script>
    console.log(1);
    setTimeout(function () {
        console.log(2);
    }, 1000);
    console.log(3);
</script>
```

1
3
2

- What if we replace 1000 with 0 ?

1
3
2

Study: https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

执行顺序……事件队列

- 这段脚本的输出结果是什么?

```
<script>
    console.log(1);
    setTimeout(function () {
        console.log(2);
    }, 1000);
    console.log(3);
</script>
```

1
3
2

- 如果我们将1000替换为0会怎样?

1
3
2

学习资料：https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

Code of conduct reminder

- Students' work will not be checked before assignment submission if the request is very general.

- Statements such as "I did not read the lecture notes or lab description etc." are not accepted as valid excuses for exemption from meeting lab requirements

- … If students miss a class/lecture/lab, it is their responsibility to determine what was missed … https://www.bcit.ca/outlines/20233046882/

行为准则提醒

- 如果请求非常笼统，则在作业提交前不会检查学生的作业。
  请求非常笼统。

- 诸如“我没有阅读讲义或实验说明等”之类的声明，不被视为免除实验要求的正当理由。

- ……如果学生错过了一堂课/讲座/实验课，他们有责任确定所遗漏的内容……
  https://www.bcit.ca/outlines/20233046882/

Part 2

Why do we need API servers?

第二部分

为什么我们需要 API 服务器?

- Let's start by asking a question:
  Q: What does a web server return when browsers send a request like these?
- Examples:
- www.who.int/index.**html**
- https://wordpress.com/index.**php**

A: they return **HTML pages** and their JavaScript and CSS

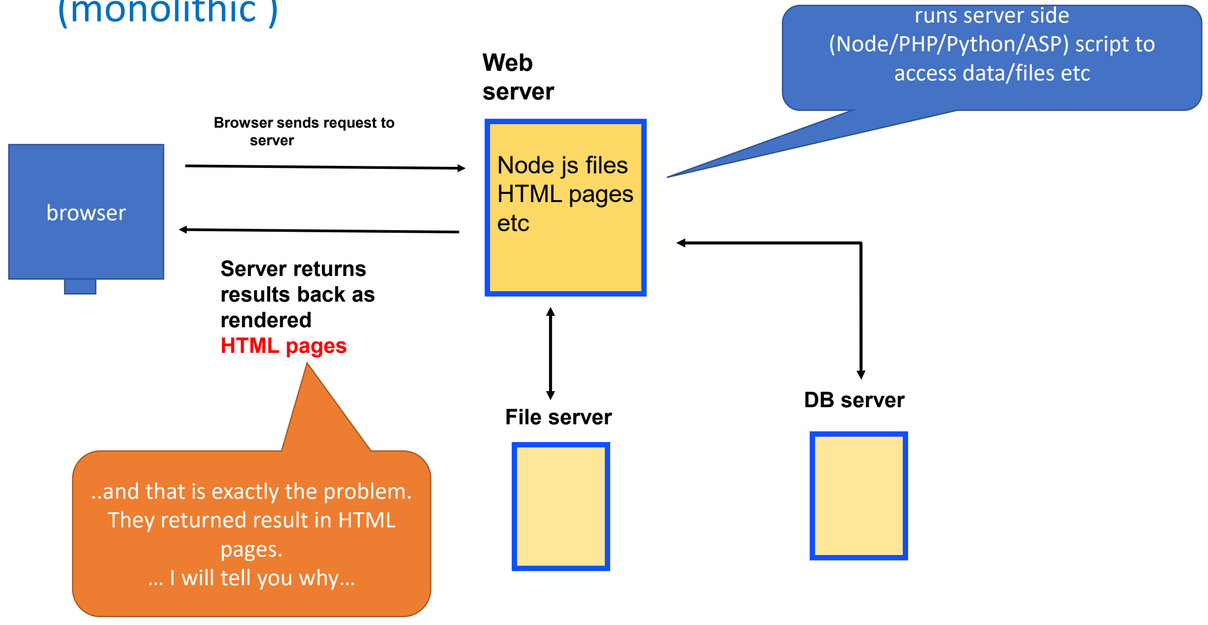
Press ctrl+u to see it for yourself

- 让我们从提出一个问题开始：
  Q: What does a web server return when browsers send a request like these?
- Examples:
- www.who.int/index.**html**
- https://wordpress.com/index.**php**

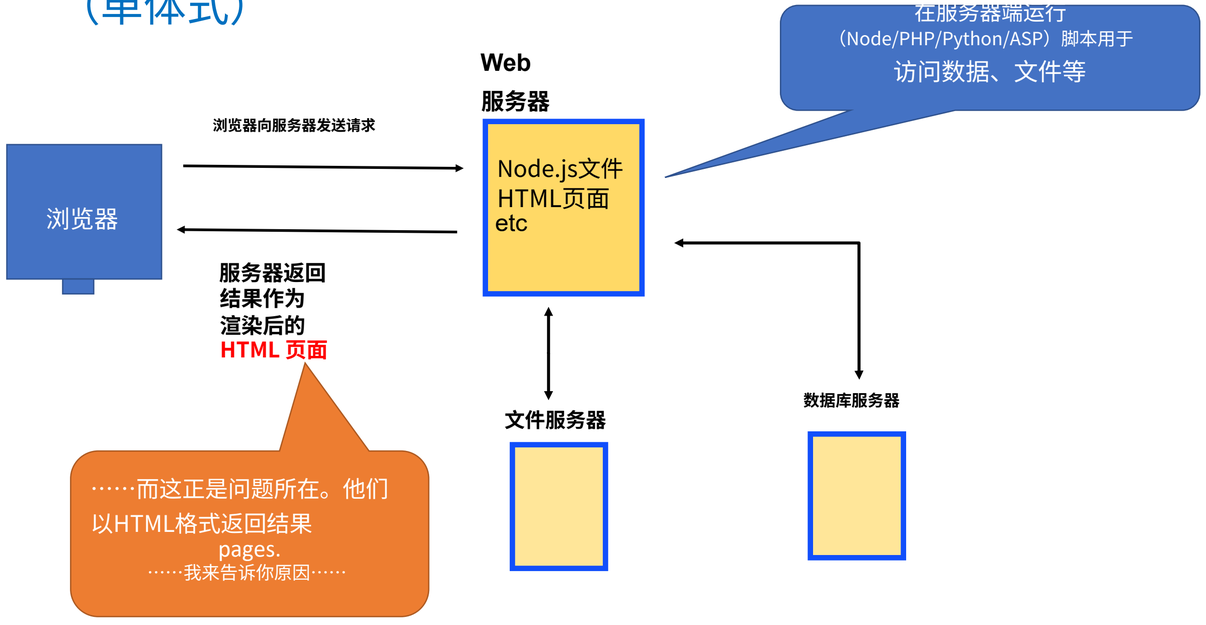
答：它们返回 **HTML 页面** 及其 JavaScript 和 CSS

按下 ctrl+u 亲自查看

The architecture of small web applications (monolithic)



小型Web应用程序的架构（单体式）



Imagine

You want to launch
an
online Movie store

想象

你想启动
an
在线电影商店

What are the main challenges?

You need a database of all existing movies.
You have two choices:
1- Create your own database which takes years!
2- or getting from websites like IMDB.com which already contain what you need.
… 100,000 move titles

Online movie store

主要挑战有哪些?

您需要一个涵盖所有现有电影的数据库。
你有两个选择：
1— 自行构建数据库，这可能耗时数年！
2— 或从 IMDb.com 等网站获取数据，这些网站已包含您所需的内容。
…… ,000 部电影片名
100

在线电影商店

challenge 1, HTML scraping

… so you chose using IMDb's data.

however

IMDB does not offer you any data format other than HTML which contains CSS, Tags etc.

But

you don't need all of that.. So you need to extract what you need (**Screen Scraping**!!!) you need to search for the Actor names, Date was published etc inside the HTML!...

Imagine that you need to extract movie titles by HTML scaping of IMDB web pages!

挑战 1：HTML 网页抓取

……因此，您选择了使用 IMDb 的数据。

然而

IMDb 并未提供除 HTML 以外的任何数据格式，而 HTML 中包含 CSS、各类标签等。

But

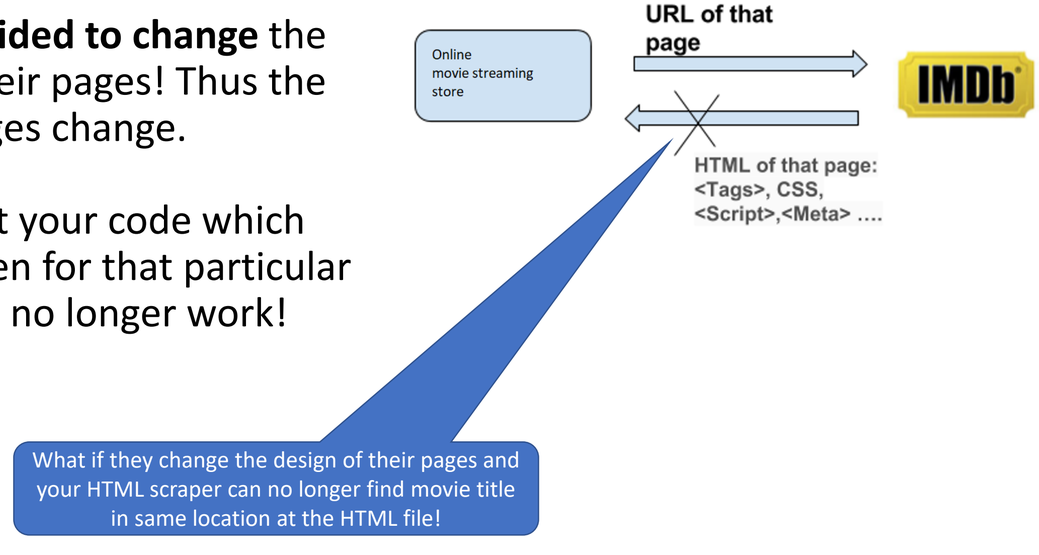
您并不需要所有这些内容……因此，您需要提取所需的信息（**屏幕抓取**！！！），即在 HTML 中搜索演员姓名、发布日期信息！……

试想一下，你需要通过 HTML 网页抓取技术从 IMDb 网站页面中提取电影标题！

challenge 2, what if UI of IMDB pages change?!

IMDb **decided to change** the look of their pages! Thus the HTML pages change.

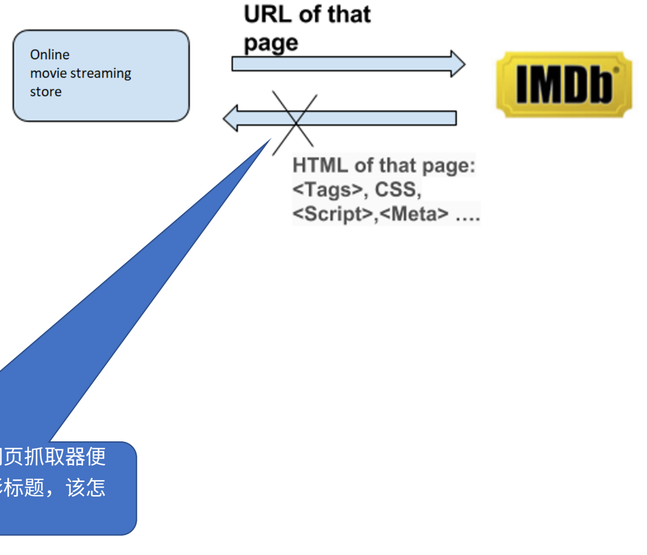
As a result your code which was written for that particular HTML will no longer work!



挑战 2：如果 IMDb 页面的用户界面发生变化，该怎么办？！

IMDb **决定更改** 其网页的外观！因此，HTML 页面也随之发生变化。

结果，您为该特定 HTML 编写的代码将不再有效！



Solution

IMDb provides two services (two URLs)

1- For **browsers and users**.

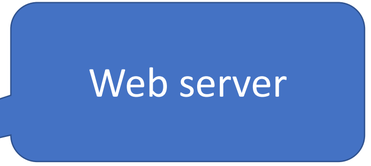
Returns HTML pages

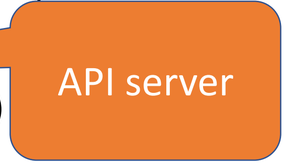
Returning result can change any time IMDb developer update the website

2- For **data concumer applications** (e.g. by omdbApi.com)

Returns Json/XML

The rule of thumb is OMDb does not change what this url return





解决方案

IMDb 提供两项服务（对应两个 URL）

1- 面向**浏览器及用户**。

返回 HTML 页面

返回结果可能随时发生变化，具体取决于 IMDb 开发人员对网站的更新

2- 面向**数据消费类应用程序**（例如 omdbApi.com）

返回 Json/XML

经验法则是 OMDb 不会更改此 URL 返回的内容

Calling API

http://www.omdbapi.com/?t=star+wars&y=2015&apikey=1668f32d


```
{
  "Title":"Star Wars: The Force Awakens",
  "Year":"2015",
  "Rated":"PG-13",
  "Released":"18 Dec 2015",
  "Runtime":"136 min",
  "Genre":"Action, Adventure, Fantasy",
  "Director":"J.J. Abrams",
  "Writer":"Lawrence Kasdan, J.J. Abrams, Michael Arndt, C
  "Actors":"Harrison Ford, Mark Hamill, Carrie Fisher, Ada
  "Plot":"Three decades after the Empire's defeat, a new t
  "Language":"English",
```


What API server returns (JSON)

VS

Browsers: Rendering HTML pages

https://www.imdb.com/title/tt2488496


```
<script>
    if (typeof uex == 'function') {
      uex("ld", "LoadIcons", {wb: 1});
    }
</script>

        <meta property="pageId" content="tt2488496" />
        <meta property="pageType" content="title" />
        <meta property="subpageType" content="main" />


        <link rel='image_src' href="https://images-na.ssl-images-amazon.com/images
        <meta property='og:image' content="https://images-na.ssl-images-amazon.com

        <meta property='og:type' content="video.movie" />
    <meta property='fb:app_id' content='115109575169727' />

    <meta property='og:title' content="Star Wars: The Force Awakens (2015)" />
    <meta property='og:site_name' content='IMDb' />
    <meta name='title' content="Star Wars: The Force Awakens (2015) - IMDb" />
        <meta name='description' content="Directed by J.J. Abrams.  With Daisy Ric
threat arises in the militant First Order. Stormtrooper defector Finn and spare pa
        <meta property="og:description" content="Directed by J.J. Abrams.  With Da
new threat arises in the militant First Order. Stormtrooper defector Finn and spar
        <meta name="keywords" content="Reviews, Showtimes, DVDs, Photos, Message B
        <meta name="request_id" content="1WZ0HAN3CHHHGBP4DX2C" />

<script>
    if (typeof uet == 'function') {
      uet("bb", "LoadCSS", {wb: 1});
    }
</script>
  <script>(function(t){ (t.events = t.events || {})["csm head pre css"] = new Date
```


What web server returns (HTML page)

调用API

http://www.omdbapi.com/?t=星球+大战&y=2015&apikey=1668f32d


```
{
  "Title":"Star Wars: The Force Awakens",
  "Year":"2015",
  "Rated":"PG-13",
  "Released":"18 Dec 2015",
  "Runtime":"136 min",
  "Genre":"Action, Adventure, Fantasy",
  "Director":"J.J. Abrams",
  "Writer":"Lawrence Kasdan, J.J. Abrams, Michael Arndt, C
  "Actors":"Harrison Ford, Mark Hamill, Carrie Fisher, Ada
  "Plot":"Three decades after the Empire's defeat, a new t
  "Language":"English",
```


API服务器返回的内容（JSON）

VS

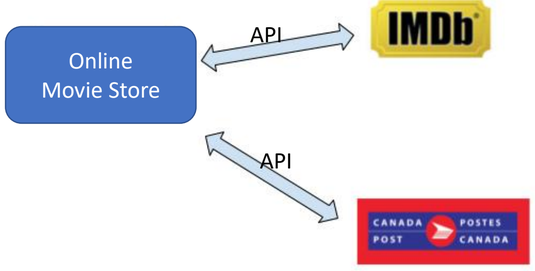
浏览器：渲染HTML页面

https://www.imdb.com/title/tt2488496


```
<script>
    if (typeof uex == 'function') {
      uex("ld", "LoadIcons", {wb: 1});
    }
</script>

        <meta property="pageId" content="tt2488496" />
        <meta property="pageType" content="title" />
        <meta property="subpageType" content="main" />


        <link rel='image_src' href="https://images-na.ssl-images-amazon.com/images
        <meta property='og:image' content="https://images-na.ssl-images-amazon.com

        <meta property='og:type' content="video.movie" />
    <meta property='fb:app_id' content='115109575169727' />

    <meta property='og:title' content="Star Wars: The Force Awakens (2015)" />
    <meta property='og:site_name' content='IMDb' />
    <meta name='title' content="Star Wars: The Force Awakens (2015) - IMDb" />
        <meta name='description' content="Directed by J.J. Abrams.  With Daisy Ric
threat arises in the militant First Order. Stormtrooper defector Finn and spare pa
        <meta property="og:description" content="Directed by J.J. Abrams.  With Da
new threat arises in the militant First Order. Stormtrooper defector Finn and spar
        <meta name="keywords" content="Reviews, Showtimes, DVDs, Photos, Message B
        <meta name="request_id" content="1WZ0HAN3CHHHGBP4DX2C" />

<script>
    if (typeof uet == 'function') {
      uet("bb", "LoadCSS", {wb: 1});
    }
</script>
  <script>(function(t){ (t.events = t.events || {})["csm head pre css"] = new Date
```

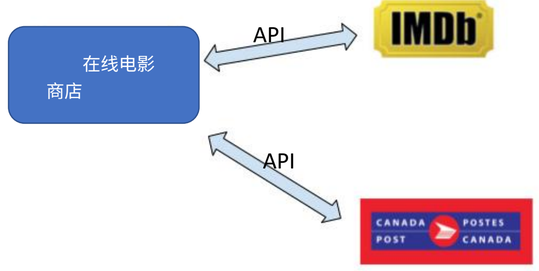

Web服务器返回的内容（HTML页面）

Now with the help of API you can even let your users to track their orders inside your website by partnering with Fedex or Canada post using their APIs.



In case someone needed physical DVD

现在，借助 API，您甚至可以在自己的网站内通过与联邦快递（FedEx）或加拿大邮政（Canada Post）合作并调用其 API，让用户实时跟踪订单状态。



以防有人需要实体 DVD

API

Application Programming Interface

- A user interface to access partners' data and services
- In web development, Web API often refers to the way by which we retrieve information from an online service.

API documentation :

- list of URLs, query parameters etc on how to make a request from the web API server,
- It also inform us what sort of response will be received for each query (json, text file, xml etc)

API

应用程序编程接口

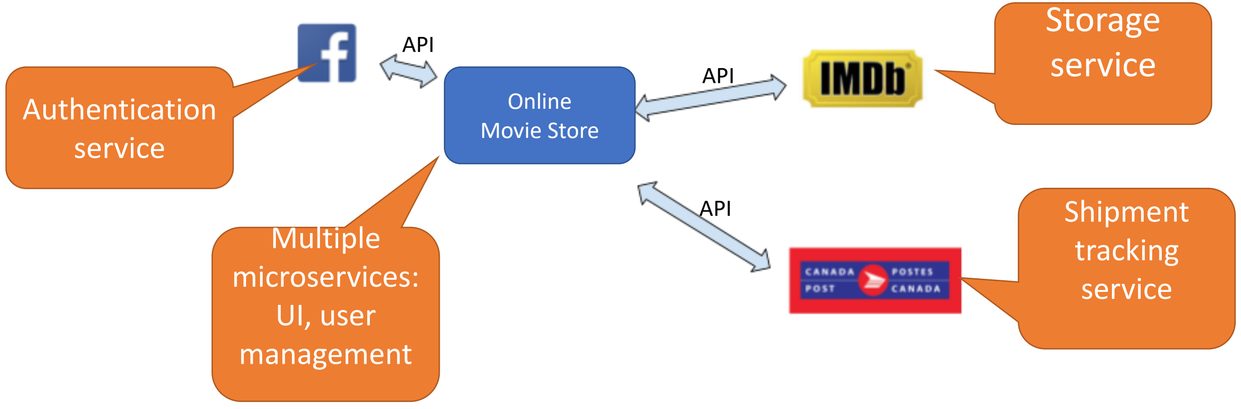
- 一种用于访问合作伙伴数据与服务的用户界面
- 在网页开发中，“Web API”通常指我们从在线服务中获取信息的方式。

API 文档：

- 一份关于如何向 Web API 服务器发起请求的 URL 列表、查询参数等说明；
- 同时还会说明针对每种查询将返回何种格式的响应（JSON、文本文件、XML 等）

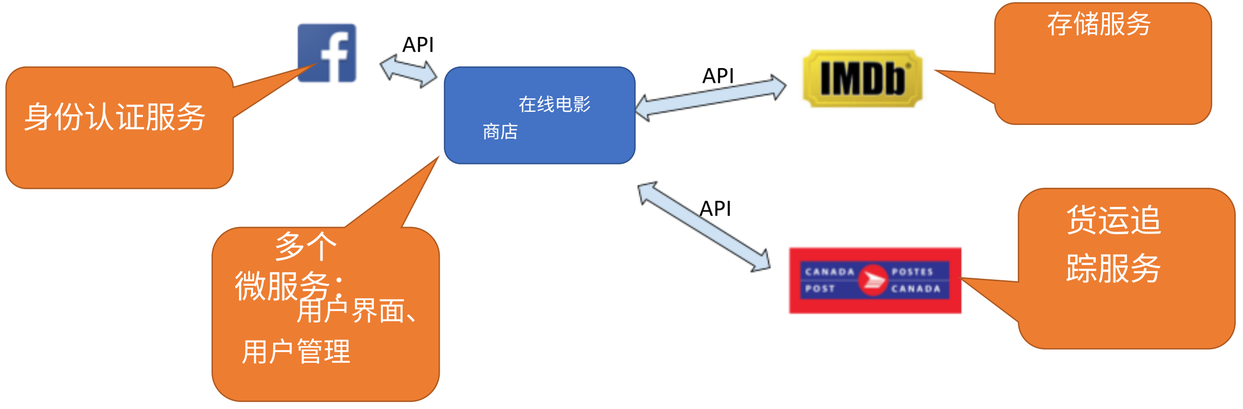
API and Microservices

You can even let your visitors buy Movie using their facebook account by integrating your online Movie store to facebook



API 与微服务

您甚至可以通过将您的在线电影商店与 Facebook 集成，让访客使用其 Facebook 账户直接购买电影。



Part 3

Anatomy of web APIs

第 3 部分

Web API 的结构解析

Anatomy of Web APIs

1- Headers: The additional details provided for communication between client and server. Some of the common headers are:

Request:
- *API-key*: the subscription key the client. This way the server knows who is making the request.
- *count*: the number of objects you want the
- pageNo: the page number etc

Response:
- *status*: the status of request or HTTP code.
- *content-type*: type of resource sent by server (e.g. html/img/...

2- Data: (also called body or message) contains info you want to send to the server.

Reference: https://www.freecodecamp.org/news/rest-api-tutorial-rest-client-rest-service-and-api-calls-explained-with-code-examples/

Web API 的结构组成

1- 请求头（Headers）：客户端与服务器之间通信时提供的附加信息。一些常见的请求头包括：

请求头（Request）：
- *API-key*：客户端的订阅密钥。服务器借此识别请求发起方。

- *count*：您希望获取的对象数量。
- pageNo：页码等

Response:
- *status*：请求的状态或 HTTP 状态码。
- *content-type*：服务器发送的资源类型（例如 html/img/...）。

2- 数据：（也称为正文或消息）包含要发送给服务器的信息。

参考：https://www.freecodecamp.org/news/rest-api-tutorial-rest-client-rest-service-and-api-calls-explained-with-code-examples/

Anatomy of Web APIs

3- Endpoint: it is the URL where the API Server is listening to.

4- HTTP Methods: determines the method of communication between client and server? Web API servers implements multiple 'methods' for different types of request, the following are most popular:

- **GET**: **Get** resource from the server.
- **POST**: **Create** resource to the server.
- **PUT**: **Update** existing resource on the server.
- **DELETE**: **Delete** existing resource from the server.

The two most common HTTP methods are: GET and POST.

Reference: https://www.freecodecamp.org/news/rest-api-tutorial-rest-client-rest-service-and-api-calls-explained-with-code-examples/

Web API 的结构解析

3- 端点（Endpoint）：即 API 服务器正在监听的 URL。

4- HTTP 方法：用于确定客户端与服务器之间的通信方式。Web API 服务器为不同类型的请求实现了多种“方法”，以下是最常用的几种：

- **GET**：**从服务器获取**资源。
- **POST**：**向服务器创建**资源。
- **PUT**：**更新**服务器上已存在的资源。
- **删除**：**删除**服务器上已存在的资源。

两种最常用的 HTTP 方法是：GET 和 POST。

参考文献：https://www.freecodecamp.org/news/rest-api-tutorial-rest-client-rest-service-and-api-calls-explained-with-code-examples/

GET vs POST method

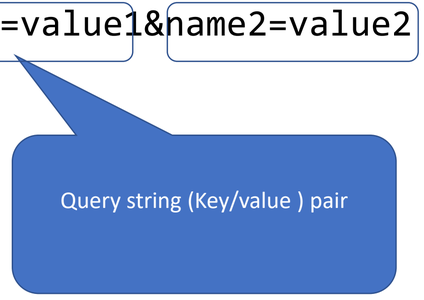
GET 与 POST方法

The two most common HTTP methods are: GET and POST.

两种最常用的 HTTP 方法是：GET 和 POST。

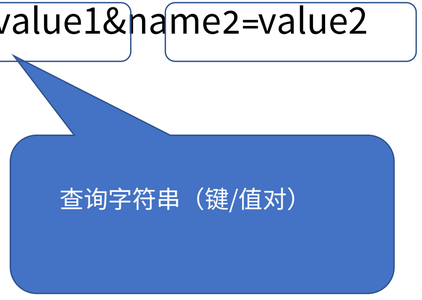
GET

- GET is used to request data from a specified resource.
- the query string (name/value pairs) is sent in the URL of a GET request
- `/test/demo_form.php?name1=value1&name2=value2`



GET

- GET 方法用于从指定资源请求数据。
- 查询字符串（名称/值对）通过 GET 请求的 URL 发送。
- /test/demo_form.php?name1=value1&name2=value2



GET

- GET requests can be **cached**
  - GET requests are re-executed but may not be re-submitted to server if the HTML response is already stored in the browser cache.
- GET requests **remain** in the browser **history**
- GET requests can be **bookmarked**
- GET requests should never be used when dealing with sensitive data
  - **Q**: Why?
- GET requests have **length restrictions** on data size
- GET requests **should** only be used **to request data** (not modify)

GET

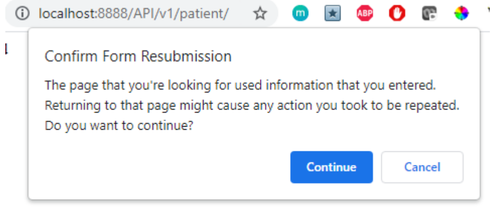
- GET 请求可以被**缓存**
  - GET 请求会重新执行，但如果 HTML 响应已存储在浏览器缓存中，则可能不会重新提交到服务器。
- GET 请求会**保留在**浏览器**历史记录中**
- GET 请求可以被**添加书签**
- 处理敏感数据时绝不应使用 GET 请求
  - **问**：为什么？
- GET 请求对数据大小有**长度限制**
- GET 请求**应**仅用于**请求数据**（而非修改）

POST

- POST is used to send data to a server to create/update a resource

- The data sent to the server with POST is stored in the request body of the HTTP request (not part of the URL)

- POST requests are **never cached**
- POST requests **cannot be bookmarked**
- POST requests **do not remain** in the browser **history**
- POST requests have **no restrictions** on data length
- If refreshing leads to resending a post request (using form), the browser warns the user

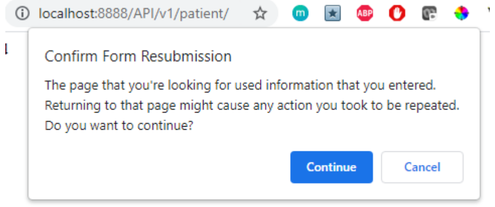


POST

- POST 用于向服务器发送数据以创建/更新资源

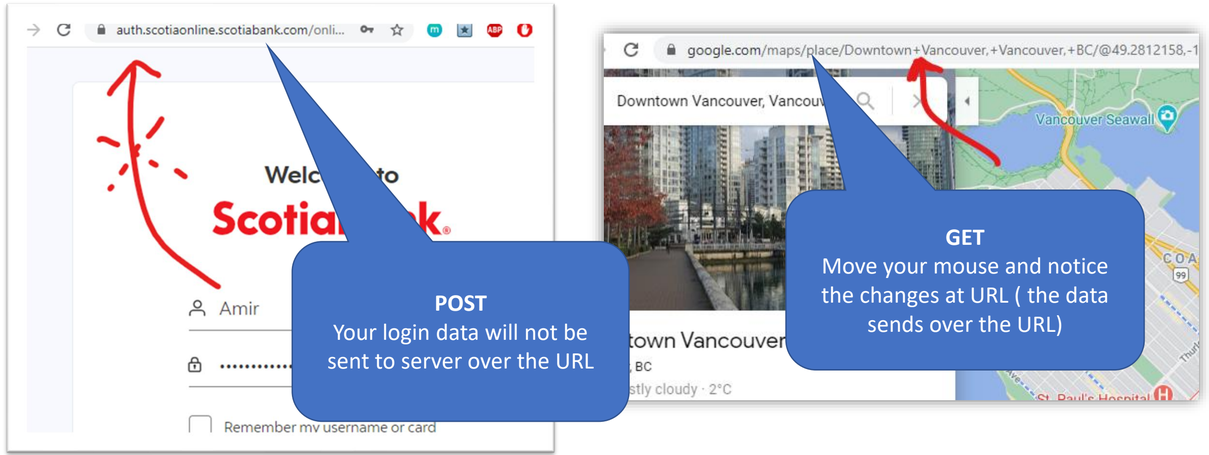
- 使用 POST 发送到服务器的数据存储在 HTTP 请求的请求体中（不在 URL 中）

- POST 请求 **从不被缓存**
- POST 请求 **无法被收藏**
- POST 请求 **不会保留** 在浏览器 **历史记录**中
- POST 请求对数据长度**没有限制**
- 如果刷新导致重新发送 POST 请求（使用表单），浏览器会向用户发出警告



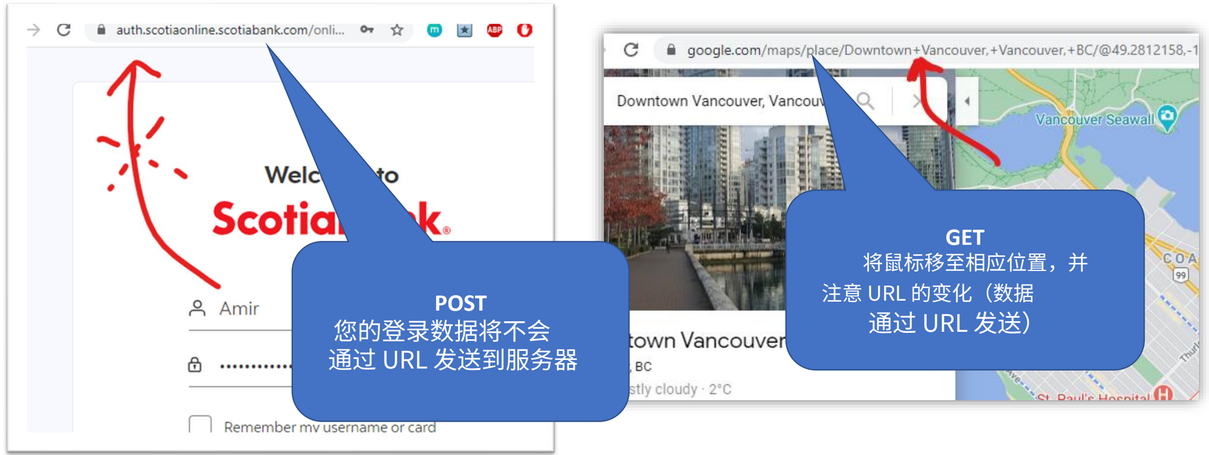
In class activity

- Which of the following web API request should be handled by GET which one should be handled by POST?
- 1- Google map search
- 2- Login to your online banking system



课堂活动

- 以下 Web API 请求中，哪些应由 GET 方法处理，哪些应由 POST 方法处理?
- 1- 谷歌地图搜索
- 2- 登录您的网上银行系统



In class activity

- Q: which method is secure, GET or POST?
- A: none! The data is not encrypted by default when sent over GET or POST, you need to encrypt data (or use secure protocols such as https etc)

课堂活动

- 问：GET 和 POST 哪种方法是安全的?
- 答：都不安全！通过 GET 或 POST 发送数据时，默认情况下数据不会被加密，你需要对数据进行加密（或使用 HTTPS 等安全协议）

	GET	POST
Pressing BACK button or browser/Reload (refreshing page)	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security (none are secure!)	GET is less safe compared to POST because data sent is part of the URL Never use GET when sending passwords	POST is a **little** safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Ref:w3school

	GET	POST
按下返回按钮或浏览器/重新加载（刷新页面）	无害	数据将被重新提交（浏览器应提醒用户数据将被重新提交）
已收藏	可以收藏	无法收藏
已缓存	可缓存	不缓存
历史记录	参数保留在浏览器历史记录中	参数不会保存在浏览器中 历史记录
数据长度限制	是的，在发送数据时，GET 方法会将数据附加到 URL 中；而 URL 长度是有限制的（最大 URL 长度为 2048 个字符）	无限制
数据类型限制	仅允许 ASCII 字符	无限制，二进制数据也允许
安全性（均不安全！）	GET 方法相比 POST 方法安全性较低 因为所发送的数据是 URL 的一部分 发送密码时切勿使用 GET 方法 密码	POST 方法比 GET 方法**略为**安全，因为 参数不会保存在 浏览器历史记录或Web服务器日志中
可见性	数据对URL中的每个人都是可见的	数据不会在URL中显示

参考资料：w3school

Part 4

How to test and try an API?

We can use postman

Download: https://www.getpostman.com/

第 4 部分

如何测试和试用 API?

我们可以使用 Postman。

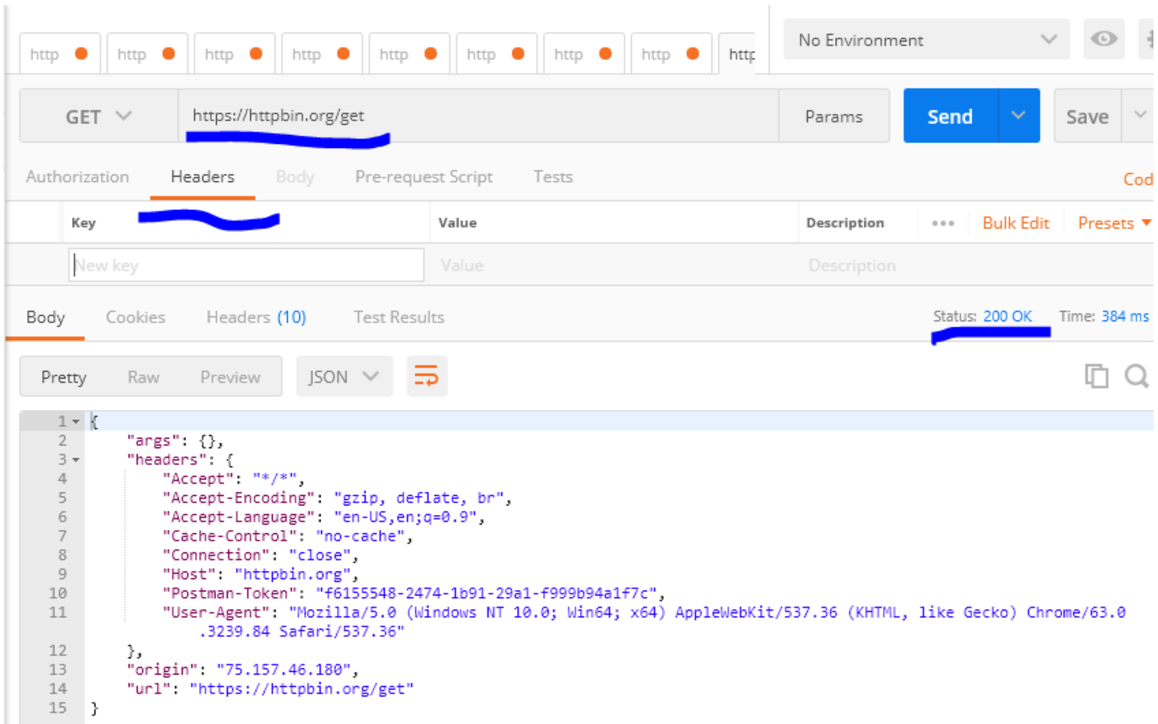
下载：https://www.getpostman.com/

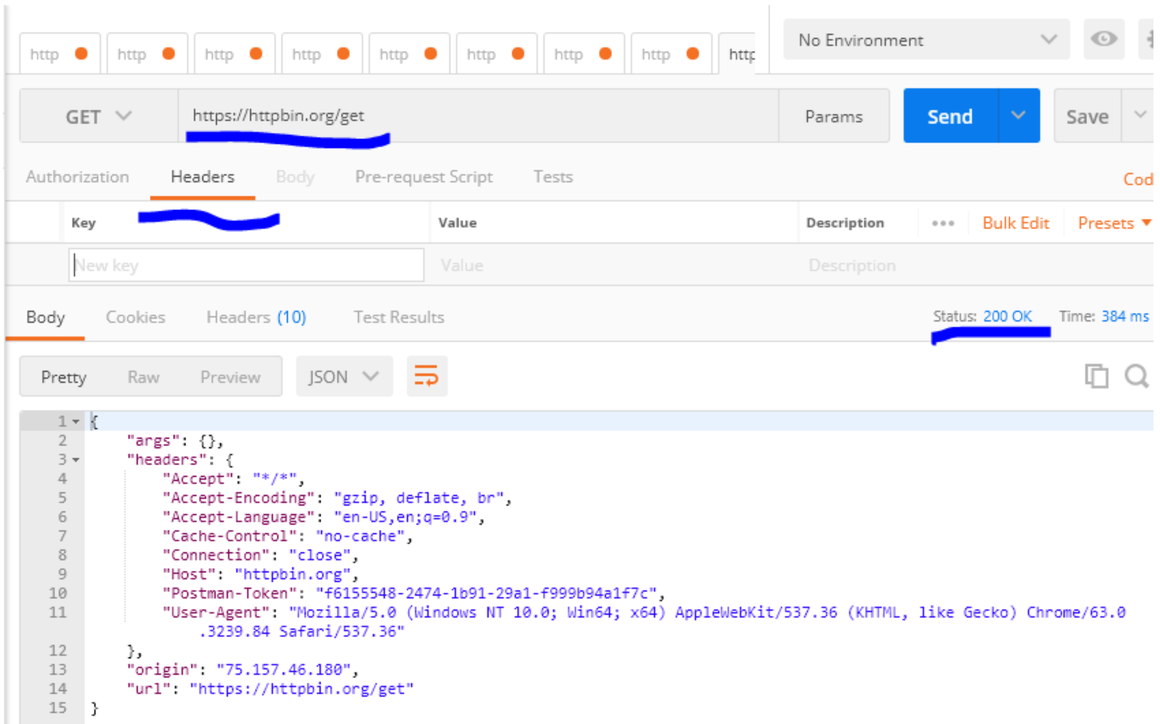
In class activity
Now use postman to try https://httpbin.org/get API

try these
https://github.com/public-apis/public-apis
Or even larger set at:
https://rapidapi.com/marketplace

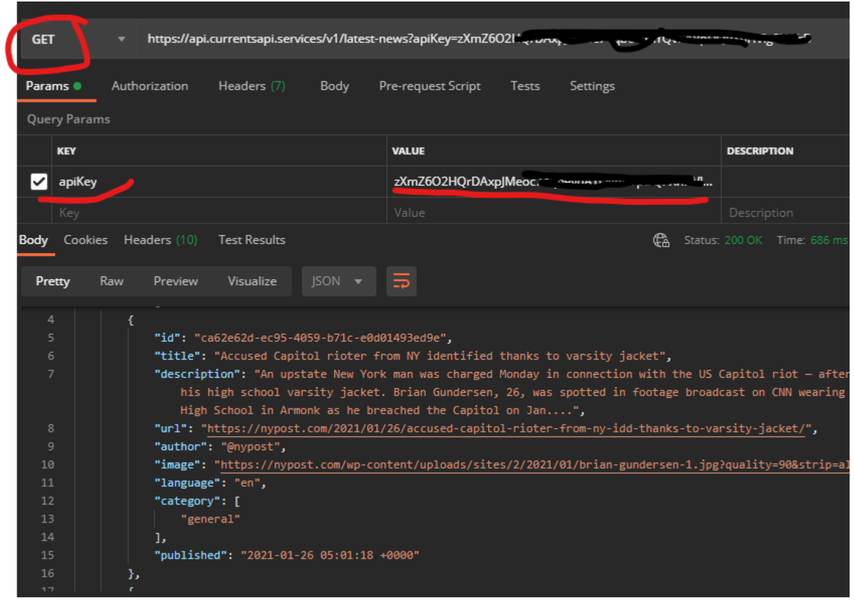
课堂活动：现在请使用 Postman 尝试调用
https://httpbin.org/get API

尝试这些
https://github.com/public-apis/public-apis
或者更大全集：
https://rapidapi.com/marketplace

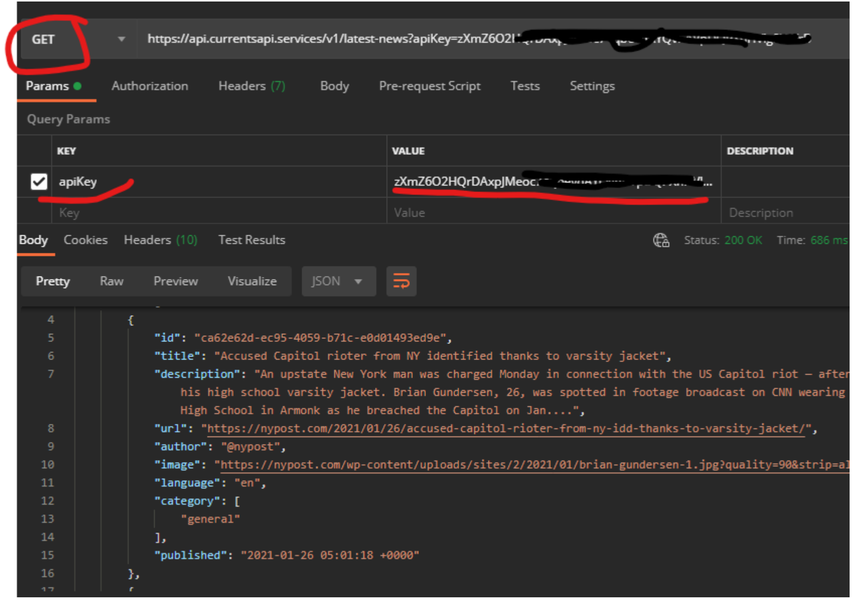

No Environment
GET
https://httpbin.org/get
Params
Send
Save
Authorization
Headers
Body
Pre-request Script
Tests
Key
Value
Description
Bulk Edit
Presets
New key
Value
Description
Body
Cookies
Headers (10)
Test Results
Status: 200 OK
Time: 384 ms
Pretty
Raw
Preview
JSON
{
"args": {},
"headers": {
"Accept": "*/*",
"Accept-Encoding": "gzip, deflate, br",
"Accept-Language": "en-US,en;q=0.9",
"Cache-Control": "no-cache",
"Connection": "close",
"Host": "httpbin.org",
"Postman-Token": "f6155548-2474-1b91-29a1-f999b94a1f7c",
"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0
.3239.84 Safari/537.36"
},
"origin": "75.157.46.180",
"url": "https://httpbin.org/get"
}


No Environment
GET
https://httpbin.org/get
Params
Send
Save
Authorization
Headers
Body
Pre-request Script
Tests
Key
Value
Description
Bulk Edit
Presets
New key
Value
Description
Body
Cookies
Headers (10)
Test Results
Status: 200 OK
Time: 384 ms
Pretty
Raw
Preview
JSON
{
"args": {},
"headers": {
"Accept": "*/*",
"Accept-Encoding": "gzip, deflate, br",
"Accept-Language": "en-US,en;q=0.9",
"Cache-Control": "no-cache",
"Connection": "close",
"Host": "httpbin.org",
"Postman-Token": "f6155548-2474-1b91-29a1-f999b94a1f7c",
"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0
.3239.84 Safari/537.36"
},
"origin": "75.157.46.180",
"url": "https://httpbin.org/get"
}

Example 2: https://currentsapi.services/en/docs/



示例 2：https://currentsapi.services/en/docs/



Part 5

What is RESTful API?

- It's service follows RESTful Architectural style

第五部分

什么是 RESTful API?

- 该服务遵循 RESTful 架构风格

RESTful Architecture

- **REST: Representational** State **Transfer** architectural design
- Is an architectural style; we will see
- how this **stateless** model helps **applications to scale easily** and
- how it separates data preparation and data consumption.

- Microservices (as a way of implementing the backend of RESTful API servers)
- Architects have started introducing the concept of microservices, aiming to reduce the complexity in designing by improving modularity
- systems by splitting the core components into small and independent pieces called Microservice that simply do a single task.

- Relation between Microservices and RESTful API:
  - The Microservices communicate via RESTful API calls

RESTful 架构

- **REST：表述性** 状态 **转移** 架构设计
- 是一种架构风格；我们接下来将看到
- 这种**无状态**模型如何助力**应用程序轻松扩展** 并
- 它如何分离数据准备与数据消费。

- 微服务（作为一种实现 RESTful API 服务器后端的方式）
- 架构师已开始引入微服务概念，旨在通过提升模块化程度来降低设计复杂度
  通过提高模块化程度来简化设计的复杂性
- 系统——将核心组件拆分为小型且独立的单元，称为微服务，每个微服务仅执行一项单一任务。

- 微服务与 RESTful API 的关系：
  - 微服务之间通过 RESTful API 调用进行通信

- Microservices – These are small and lightweight services that execute a single functionality. The Microservices Architecture framework has a number of advantages that allows developers to not only enhance productivity but also speed up the entire deployment process.

- The components making up an application build using the Microservices Architecture aren’t directly dependent on each other. As such, they don’t necessitate to be built using the same programming language.

- Therefore, developers working with the Microservices Architecture are free to pick up a technology stack of choice. It makes developing the application simpler and quicker.

• 微服务 – 这些是小型且轻量级的服务，用于执行单一功能。微服务架构框架具有诸多优势，可帮助开发人员不仅提高生产力，还能加快整个部署流程。

• 使用微服务 架构构建的应用程序组件彼此之间没有直接依赖关系。因此，它们 无需使用相同的编程语言进行构建。

• 因此，采用微服务架构的开发人员可以自由选择合适的技术栈。这使得应用程序的开发更简单、更快速。

46

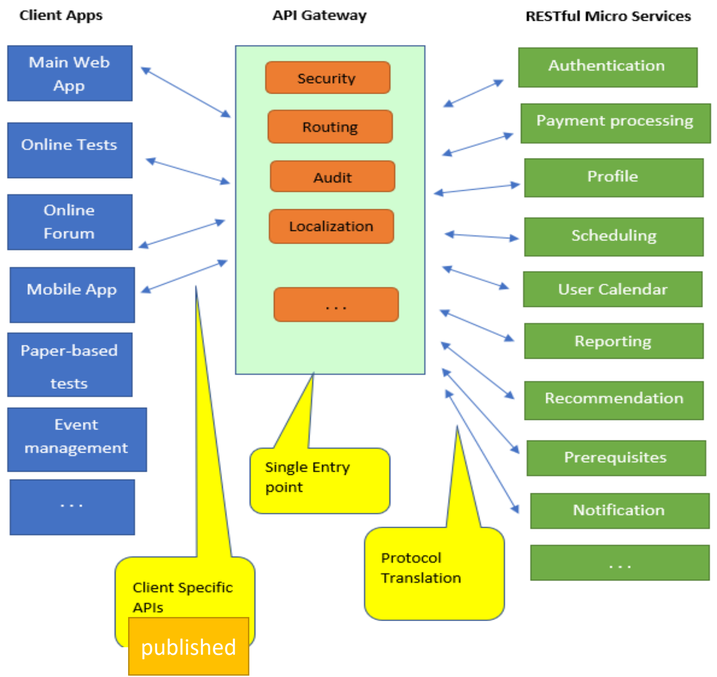
Example: Micro-service architecture diagram

Q: Guess application this diagram represent?
Online store? CRM?

RESTful API

Remarks:

1. Services communicate via API calls
2. Every request goes through API gateway
3. Gateway is responsible for routing, authentication, Auditing etc
4. Client specific APIs are the one published to developers
5. Each microservice could be developed in a different technology stack and could be hosted in a server geographically separated from other services.



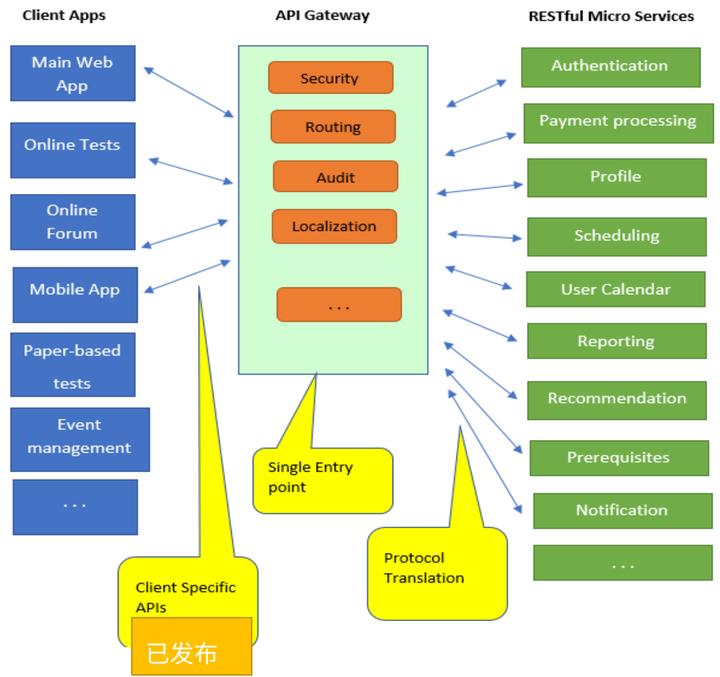
示例：微服务架构图

问：猜猜此图表示的应用程序表示什么？
在线商店？CRM？

RESTful API

备注：

1. 服务之间通过 API 调用进行通信
2. 每个请求都通过API网关
3. 网关负责路由、身份验证、审计等功能
4. 面向特定客户端的API是发布给开发者的接口
5. 每个微服务都可以采用不同的技术栈进行开发，并且可以托管在地理上与其他服务分离的服务器上。



RESTful is not an architectural pattern!

- REST is not an architectural pattern like Microservices or MVC.
- Instead, it's an architectural style or set of constraints for API communication between systems.
- The communication between components in an architectural patterns usually happen via **API** calls (Q: How else the communication can happen?)

RESTful 并非一种架构模式!

- REST 并非像微服务或 MVC 那样的架构模式。
- 相反，它是一种面向系统间 API 通信的架构风格，或一组约束条件。

- 在架构模式中，组件之间的通信通常通过 **API** 调用实现（问：除此之外，通信还能以何种方式进行？）

RESTful API Servers development principals

- **1. Client–Server**
- **2. Stateless**
- **3. Cacheable**
- **4. Uniform Interface (core constraint)**
- **5. Layered System**

- **Q:** what's the benefit to follow these principles when implementing (coding) our API server?

RESTful API 服务器开发原则

- **1. 客户端-服务器**
- **2. 无状态**
- **3. 可缓存**
- **4. 统一接口（核心约束）**
- **5. 分层系统**

- **问：** 在实现（编写代码）我们的 API 服务器时，遵循这些原则有何益处？

RESTful API Servers development principals

- **1. Client–Server**
- **Principle:** Separation of concerns — the client handles the user interface/experience, and the server manages data and business logic.
- **Benefit:** Improves portability (UI can change without backend changes) and scalability (servers optimized for logic, clients optimizes for UI).
- **Example: GitHub** — Web app, mobile app, and CLI clients all consume the same server APIs.

- **2. Stateless**
- Each request contains all the info needed; server doesn't store session state.
- Example: If your API server returns pages of a book, then to keep it *stateless* (and RESTful), the client must always specify the exact page. For instance, if you has just asked for page 50, to get content of page 51 , you cannot serve such request from client like **"give me the next page."** The server should respond **"I don't keep track of previous requests — you must tell me which page you want."**

- **3. Cacheable**
- The API server you implement must define if a response is cacheable or not (via HTTP headers like Cache-Control, ETag, Expires).
  the **client or intermediaries** (like browsers, CDNs, reverse proxies) can cache the response and reuse it.

RESTful API 服务器开发原则

- **1．客户端–服务器**
- **原则：** 关注点分离——客户端负责用户界面/用户体验，服务器负责数据管理和业务逻辑。

- **优势：** 提升可移植性（用户界面可独立变更，无需修改后端）和可扩展性（服务器针对业务逻辑优化，客户端针对用户界面优化）。
- **示例：** **GitHub**——网页应用、移动应用及命令行工具（CLI）客户端均调用同一套服务器 API。

- **2．无状态**
- 每个请求都包含所需的全部信息；服务器不存储会话状态。
- 例如：如果你的API服务器返回一本书的页面内容，为了保持其 无状态（并符合REST原则），客户端必须始终明确指定具体页码。比如，你刚刚请求了第50页，现在要获取第51页的内容，就不能向客户端发出类似 **"给我下一页"** 的请求。服务器应响应： **"我不会记录之前的请求—— 你必须告诉我你要哪一页。"**

- **3. 可缓存**
- 你实现的API服务器必须通过HTTP头部（如Cache-Control、ETag、Expires）定义响应是否可缓存。

  则 **客户端或中间节点**（如浏览器、CDN、反向代理）可以缓存该响应并重复使用。

- **4. Uniform Interface (core constraint)**
- Resource Identification via URIs → Each resource has a unique URL (e.g., /users/123).
- Manipulation through Representations → Resources are manipulated via JSON/XML representations.
- Self-Descriptive Messages → Requests and responses include metadata (e.g., HTTP verbs, headers).
- HATEOAS (Hypermedia as the Engine of Application State) → Responses include links for discoverability.(Q? I don't know what is means!!!)
- **Example**: GitHub REST API — /repos/{owner}/{repo}/issues returns issues plus links to related actions (close, comment, assign).

- **Benefit**: Simplifies interactions and makes APIs predictable.
- **Example**: GitHub REST API
- To get a user: GET /users/octocat
- To get that user's repos: GET /users/octocat/repos
- To get a specific repo: GET /repos/octocat/hello-world
- 👉 You don't need to read special documentation for every endpoint — the pattern is consistent and predictable.
- Nouns (resources) are always in the URL.
- Verbs (actions) are always HTTP methods (GET, POST, PUT, DELETE)

- **4．统一接口（核心约束）**
- 通过 URI 进行资源标识 → 每个资源均拥有唯一的 URL（例如：/users/123）。
- 通过表述进行资源操作 → 资源通过 JSON 或 XML 等表述形式进行操作。
- 自描述性消息 → 请求与响应均包含元数据（例如 HTTP 方法、请求头）。
- HATEOAS（超媒体即应用状态引擎）→ 响应中包含链接，以支持动态发现功能。（问？我不知道这是什么意思！！！

- **示例**：GitHub REST API — 调用 /repos/{owner}/{repo}/issues 接口将返回问题列表，并附带指向相关操作（如关闭、评论、指派）的链接。

- **优势**：简化交互过程，使 API 行为可预测。
- **示例**：GitHub REST API
- 获取用户信息：GET /users/octocat
- 获取该用户的仓库列表：GET /users/octocat/repos
- 获取指定仓库：GET /repos/octocat/hello-world
- 👉 您无需为每个端点单独查阅特殊文档——其设计模式统一且可预测。
- 名词（资源）始终出现在 URL 中。
- 动词（操作）始终对应 HTTP 方法（GET、POST、PUT、DELETE）。

- **5. Layered System**
- **Principle:** The system can be composed of multiple layers (e.g., client, proxy, gateway, server). Clients don’t know if they are talking to the origin server or an intermediary.
- **Benefit:** Adds flexibility, scalability, and security. Enables CDNs, load balancers, and gateways without changing the client.
- **Example: Amazon Web Services (AWS API Gateway)** — sits between clients and microservices, handling routing, throttling, and security.

- **5．分层系统**
- **设计原则：** 系统可由多个层级构成（例如客户端、代理、网关、服务器）。客户端无法分辨自己正在与源服务器还是中间组件通信。
- **优势：** 提升灵活性、可扩展性与安全性；可在不修改客户端的前提下，支持内容分发网络（CDN）、负载均衡器及网关。

- **示例： Amazon Web Services（AWS API 网关）** —— 部署于客户端与微服务之间，负责路由、限流及安全管控。

Benefits to making our API server RESTful?

- Client–Server → separation & evolvability (GitHub)
- Stateless → scalability & reliability (Netflix)
- Cacheable → performance & efficiency (Twitter)
- Uniform Interface → simplicity & predictability (GitHub API)
- Layered System → flexibility & security (AWS API Gateway)
- Code-on-Demand → extensibility (Web apps delivering JavaScript)

将我们的 API 服务器设计为符合 REST 风格有哪些优势?

- 客户端–服务器 → 分离与可演化性（GitHub）
- 无状态 → 可扩展性与可靠性（Netflix）
- 可缓存 → 性能与效率（Twitter）
- 统一接口 → 简洁性与可预测性（GitHub API）
- 分层系统 → 灵活性与安全性（AWS API 网关）
- 按需代码 → 扩展性（Web 应用程序交付 JavaScript）

Resources

- https://www.w3schools.com/
- RESTful Web API Design with Node.js 10, Third Edition, Valentin Bojinov
- chatGPT

资源

- https://www.w3schools.com/
- 使用 Node.js 10 进行 RESTful Web API 设计（第三版），Valentin Bojinov
- chatGPT