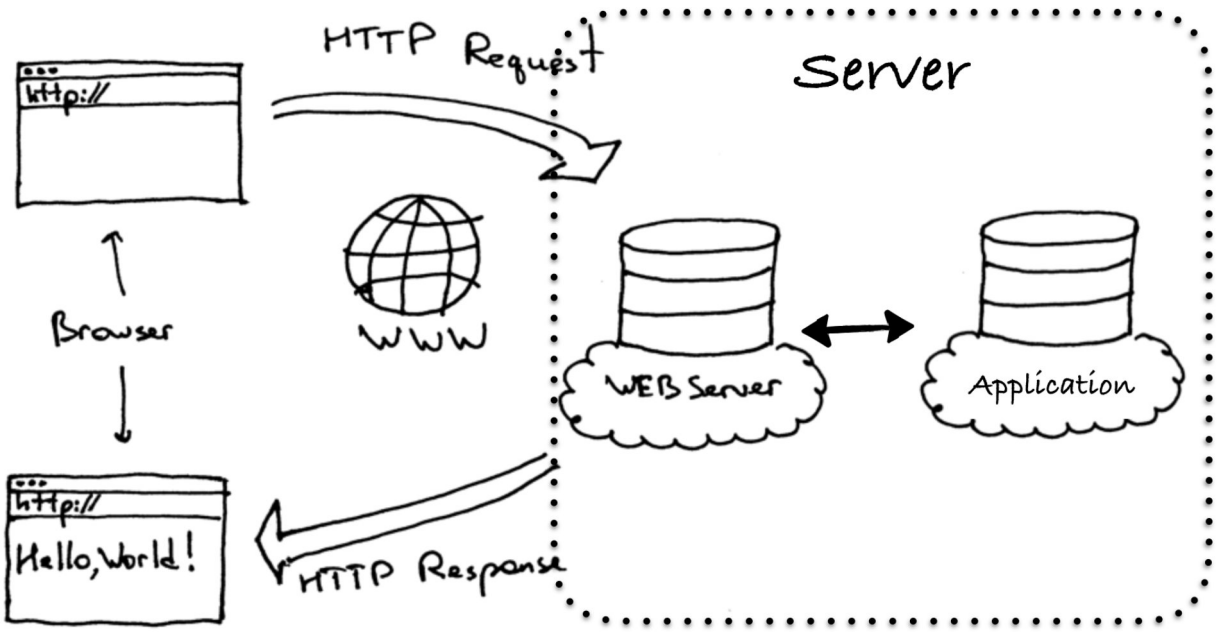
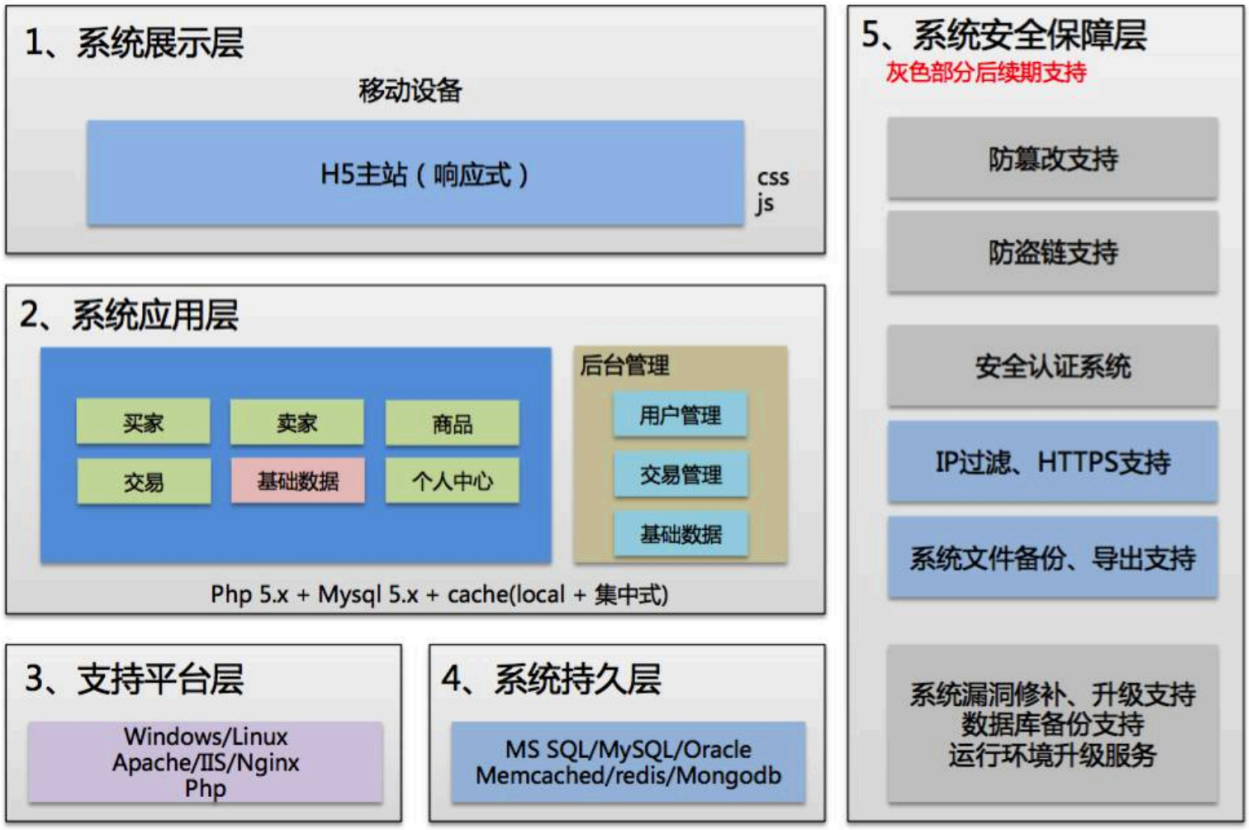
Web Application



注：
web服务器：一般指像nginx，apache这类的服务器，他们一般只能解析静态资源。
应用服务器：一般指像tomcat，jetty，resin这类的服务器可以解析动态资源也可以解析静态资源，但解析静态资源的能力没有web服务器好。
一般只有Web服务器才能被外网访问，应用服务器只能内网访问。

一个典型的web应用的架构示意图

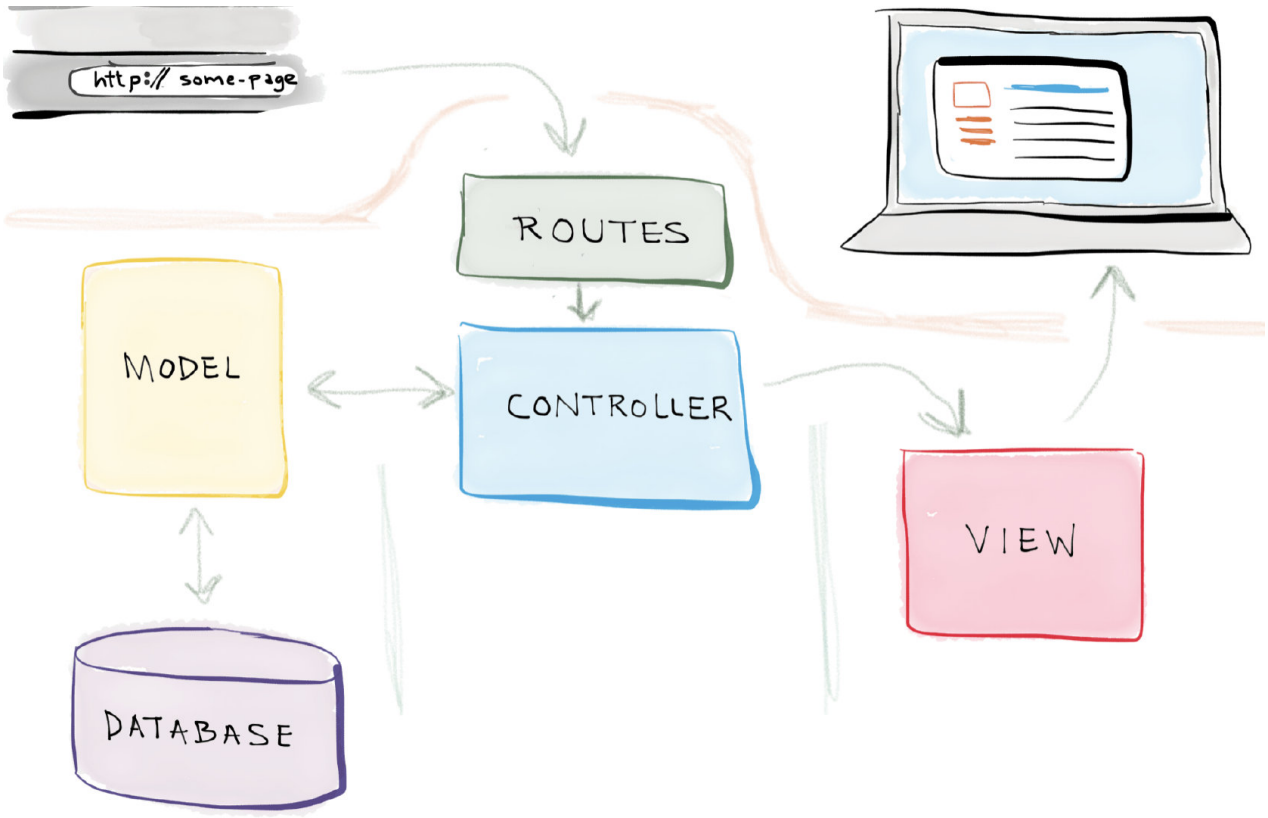


前后端交互

MVC 模式/ 前后端逻辑混合开发模式/ 服务端渲染

服务器端直接生成页面。

MVC 是一种经典的设计模式，全名为 Model-View-Controller，即 模型-视图-控制器。



模型（Model）：是用于封装数据的载体，与数据库交互

```
@app.route('/')
def main_page():
    """Searches the database for entries, then displays them."""
    db = get_db()
    cur = db.execute('select * from entries order by id desc')
    entries = cur.fetchall()
    return render_template('index.html', entries=entries)
```

视图（View）：决定了界面到底长什么样子，以前多用模版语言

```
{% for entry in entries %}
<li>
  <h2>{{ entry.title }}</h2>
  <div>{{ entry.text|safe }}</div>
</li>
{% else %}
<li><em>No entries yet. Add some!</em></li>
{% endfor %}
```

控制器（Controller）：负责转发请求，对请求进行处理，例如，用户发送一个 HTTP 请求，此时该请求首先会进入控制器，然后控制器去获取数据并将其封装为模型，最后将模型传递到视图中进行展现

```
@app.route('/')
def main_page():
    pass
```

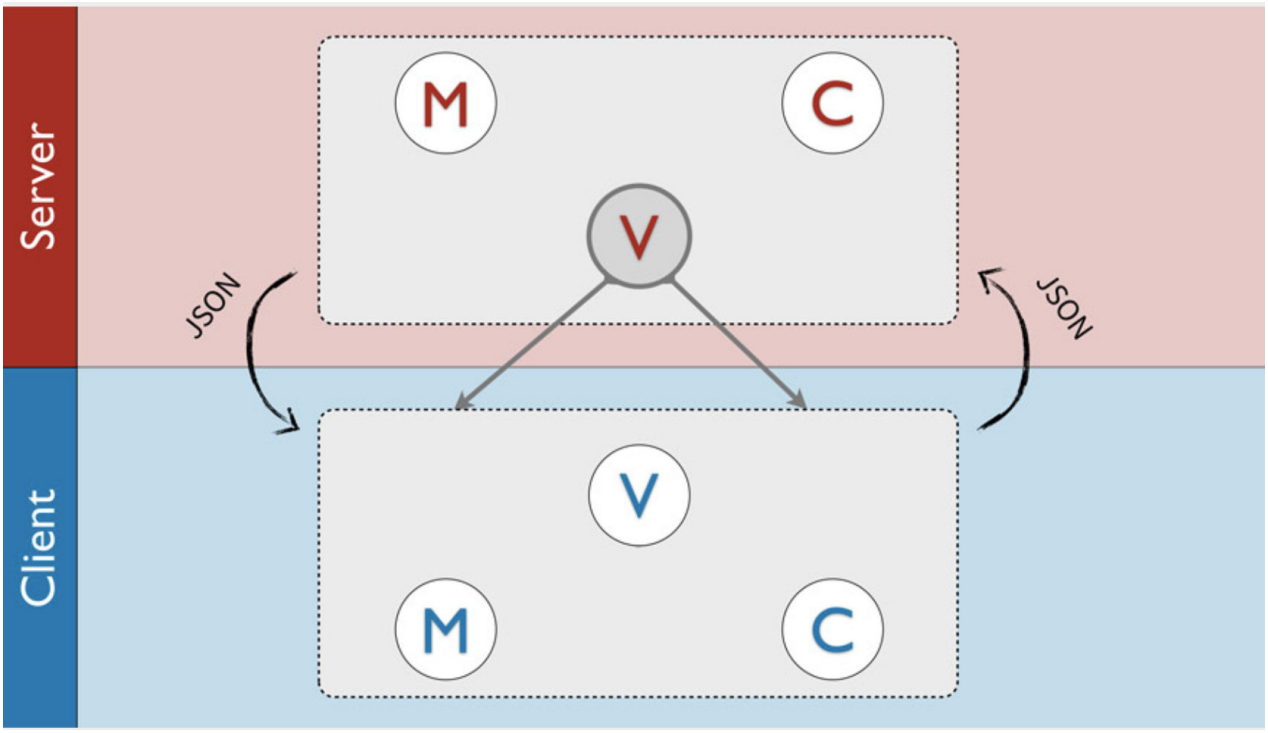
- 缺点

```
1. 前后端耦合度太高。有的数据来自AJAX，有的数据印在DOM上；有的业务逻辑在前端，有的在Model层，更多的是在View层，且后端开发人员需要了解前端的页面结构来填充逻辑代码。
2. 不易维护，由于对于一个页面的维护需要牵扯到两端的开发人员来共同进行维护，在需求变更后容易出现bug。
3. View强依赖于Model，对于后端来讲前端视图组件是无法复用的。
```

接口分离/ajax跨域请求分离模式

- 前后端分离目的:职责分离，分工合作
- 核心思想

  前端Html页面通过Ajax调用后端的数据API并使用JSON数据进行交互。



后端	前端
提供数据	接收数据，返回数据
处理业务逻辑	处理渲染逻辑
代码跑在服务器上	代码跑在浏览器上

```
from flask import Flask, jsonify
app = Flask(__name__)
tasks = [
    {
        'id': 1,
        'title': u'Buy groceries',
        'description': u'Milk, Cheese, Pizza, Fruit, Tylenol',
        'done': False
    },
    {
        'id': 2,
        'title': u'Learn Python',
        'description': u'Need to find a good Python tutorial on the web',
        'done': False
    }
]

@app.route('/todo/api/tasks', methods=['GET'])
def get_tasks():
    return jsonify({'tasks': tasks})
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

- 优点

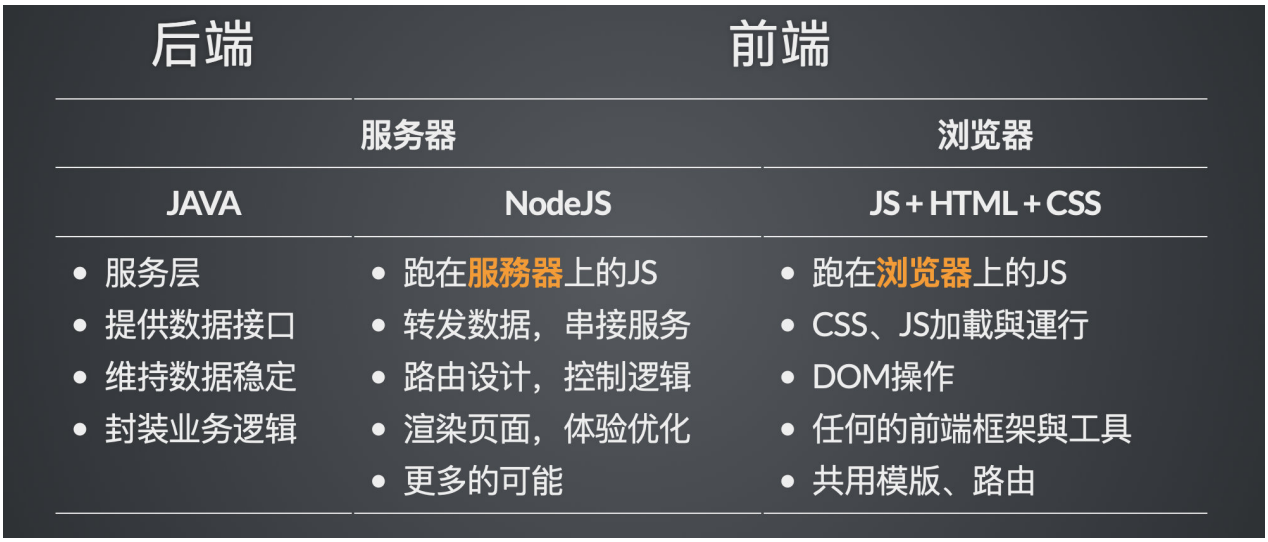
```
1．前后端的逻辑不需要混合在一起，两端的开发人员基本不需要参与对方的代码，大大提升了整体的开发效率，也方便定位问题。
2．在部署方面前后端可以分别部署，有利于前后端独立调试。
```

- 缺点

```
1．有些板块渲染需要等到ajax请求数据返回后才能进行完全的展示
2．在性能方面，ajax请求的暴涨会影响渲染性能。
3．请求的处理需要异步编程，影响代码可读性
```

同构渲染（同时在服务器和客户端渲染页面）

增加了Nodejs作为前后端的中间层，由Nodejs服务器端提供尽量快速的首页首次加载，其他的代码在用户继续浏览站点时加载。渲染发生在服务器端，只有部分DOM的更新在浏览器完成。




```
优点：
1．node的异步特性能有效提升前端加载速度（多文件时可以并行渲染，先渲染完先输出）。
2．node本身内置服务器功能，几行代码就可以启动一个服务器，免去了对apache，wamp等服务器的依赖。
3．前后端彻底分离，node端只要启动一个http-proxy进行api请求转发类似于nginx的代理功能，也不存在跨域问题。

缺点：
1．增加了学习、使用前端门槛；对异步编程要求较高
2．node的单线程机制在部署的时候需要启动一个监控进程，在node挂了后能自动重启（需配置supervisor）。
3．在node端调试相对比较麻烦。
```

Tips:

- 目前工业界多采用第三种架构模式，本课程中比较推荐使用第二种（简单、方便分工合作）。后端负责处理数据、生成JSON API；前端通过Ajax调用JSON API获得数据，渲染视图，负责交互逻辑。
- 后端（应用服务器）
  - Python（推荐）: Flask, Tornado, Django
  - Java
  - Node
- 前端（web服务器）
  - 简单起见，我们可以直接把应用服务器也作为资源服务器


```
@app.route("/")
def index():
    return render_template("index.html")

@app.route('/api/tasks', methods=['GET'])
def get_tasks():
    return jsonify({'tasks': tasks})
```

  - 前端独立启动一个静态资源服务器（e.g. IIS/Apache/Python，见week1,week2的内容）

数据接口 API

API(application programming interfaces)，即应用程序编程接口。API是一套协议，规定了与外界的沟通方式：如何发送请求和接受响应。通过API，计算机可以读取、编辑网站数据。

- JSON是数据打包的一种格式，形式上类似于python中的字典格式，但并不像字典具备操作性。且是格式有一些形式上的限制：只能使用双引号作为key或者值的边界符号，不能使用单引号，而且“key”必须使用边界符（双引号）

```
//JSON数据格式
{
  "clusterInfo":
  {
    "id":1324053971963,
    "resourceManagerVersionBuiltOn":"Tue Dec 13 22:12:48 CST 2011",
    "hadoopVersion":"0.23.1-SNAPSHOT",
    "hadoopVersionBuiltOn":"Tue Dec 13 22:12:26 CST 2011"
  }
}
```

- RESTful 风格

HTTP请求发送与处理示例

- 后端（python示例）

```
# 处理GET请求
@app.route('/api/anomaly')
def getAnomaly():
    starttime = request.args.get('starttime', 1486609046000, type=int)
    endtime = request.args.get('endtime', 1486706225000, type=int)
    a=calcAnomaly(starttime,endtime)
    return jsonify(a)

# 处理POST请求
@app.route('/api/ipga', methods=['POST'])
def ipga():
    datagoal = request.form.getlist("timelinegoal[]")
    startnum = request.form.get('startnum')
    endnum = request.form.get('endnum')
    srciparr = request.form.getlist("iparr[]")
    a=GAips(datagoal,startnum,endnum,srciparr)
    return jsonify(a)
```

- 前端：Javascript示例

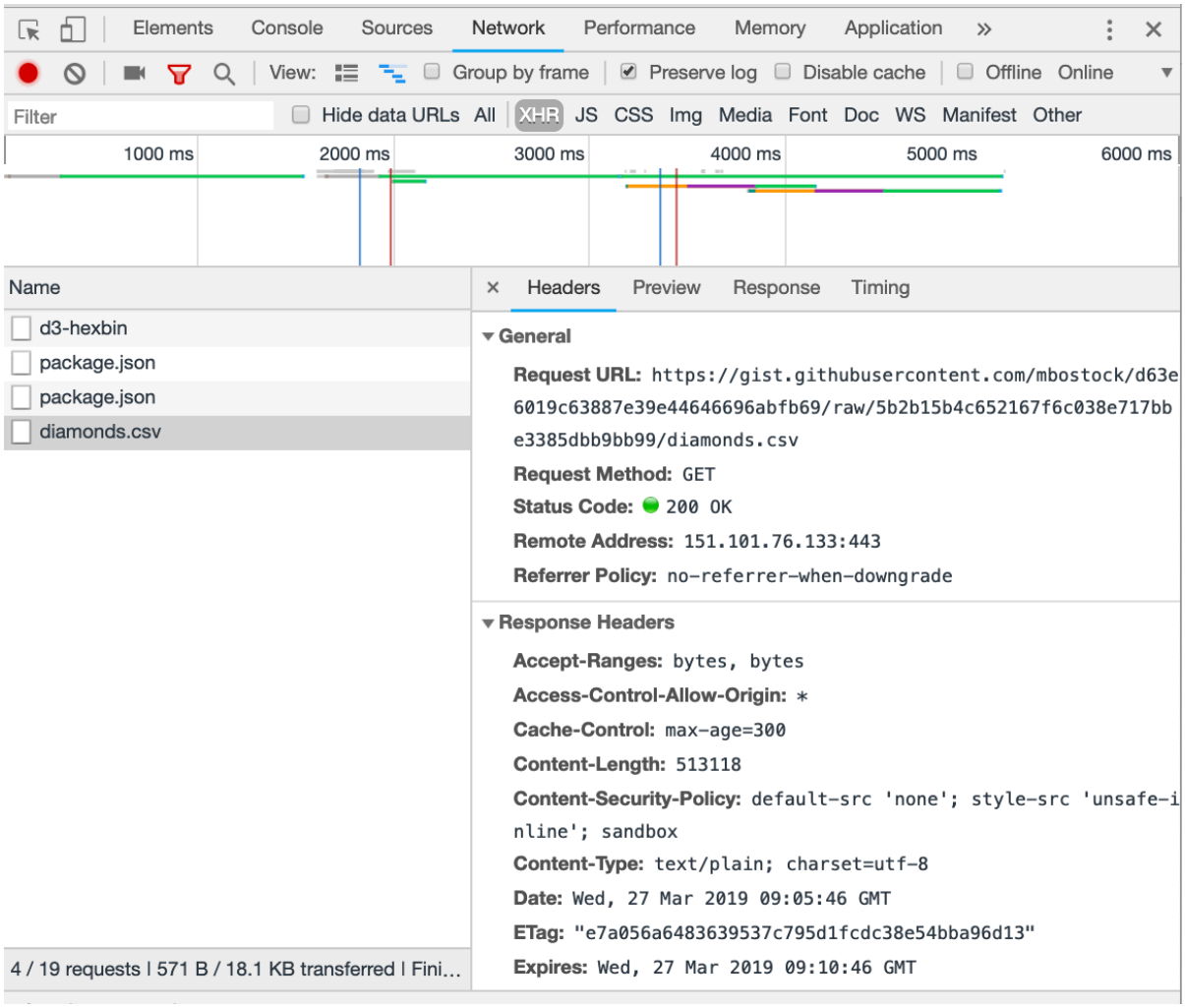
```
var xhr = new XMLHttpRequest();
xhr.open("POST", "/searchguard/api/v1/auth/login", true);
xhr.setRequestHeader("Content-type", "application/json");
xhr.setRequestHeader("kbn-version", "5.3.0");
xhr.send(JSON.stringify({
  "username" : user.name,
  "password" : user.p
}));
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    if (xhr.status == 200) {
```

```
      window.location.href = user.nextUrl;
    }
  }
};
```

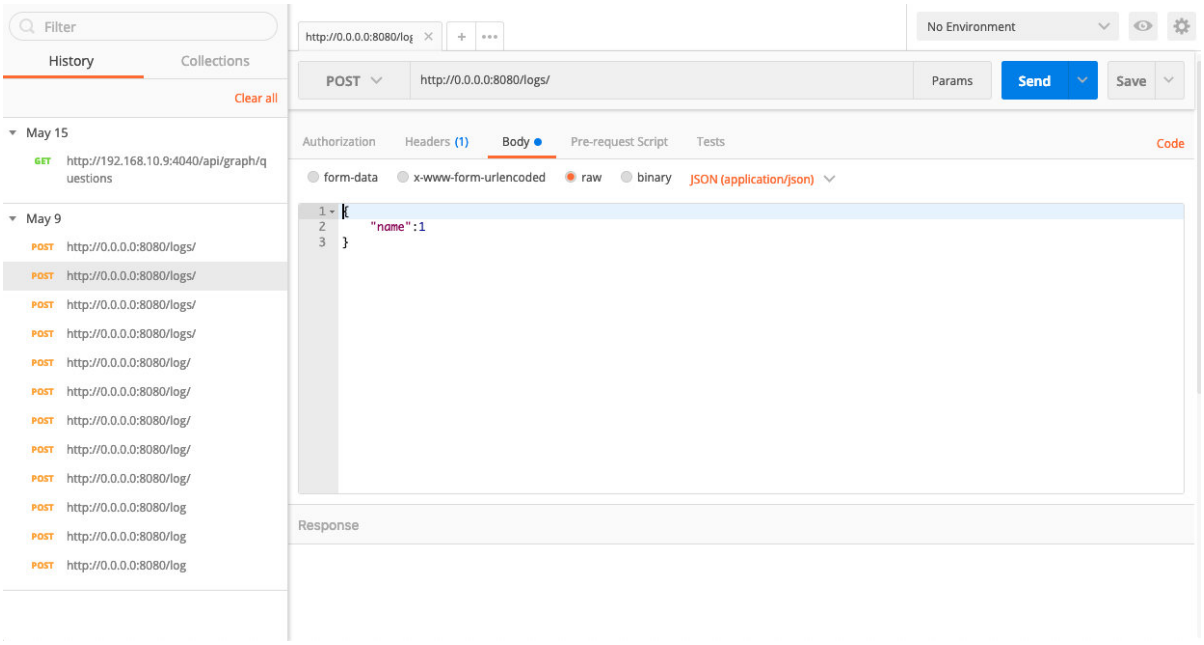
- 工具库：jquery.ajax; fetch; d3-fetch

前后端通信的检查工具

- Chrome 控制台



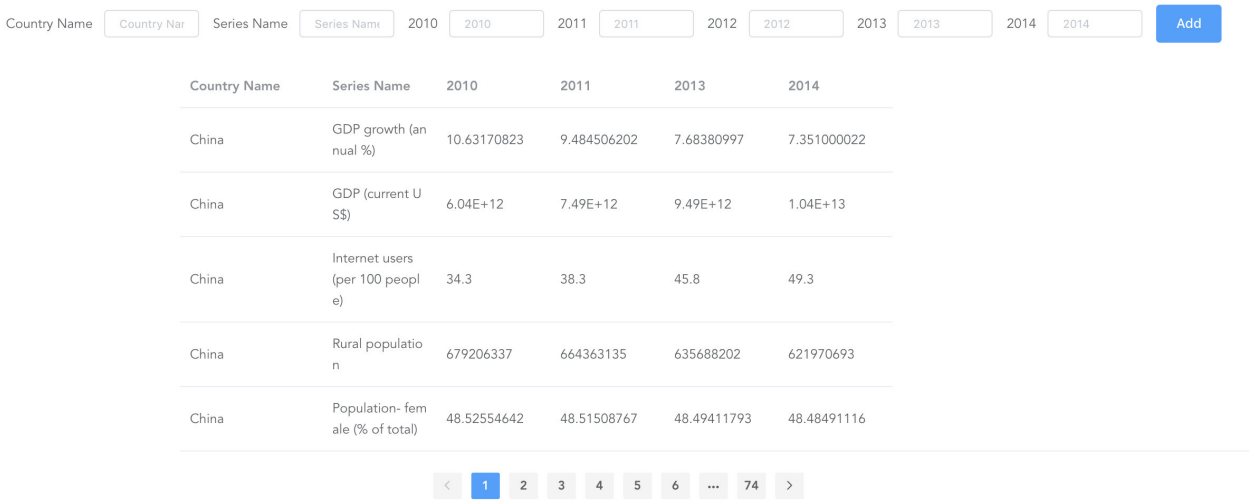
- Postman



- 代理服务器(http proxy): 请求在特定容器内(线上App，微信)，无法通过inspector获得时，Proxy在网络传输中间层，可完全掌控数据。可用于作开发工具(对web自动注入js实现对页面控制；获取打点上报请求)；也可以是攻击手段

练习

表格展示和添加（GET，POST练习）



- 数据：ftp://public.sjtu.edu.cn/material/data/countryData/
- 功能
  - 表格分页加载与显示,每页显示5条数据（后端分页，前端展示）
  - 增加行,服务端数据（e.g.数据文件）应相应改变。

Reference

- 用Flask实现一个RESTful API服务器端
- 精读前后端渲染之争
- 淘宝前后端分离实践
- RESTful API 设计参考

- API设计之道