世界乒乓球员排名网站说明文档

——基于 D3 实现的数据可视化

姓名：李浩

班级：F1603703

学号：516030910453

日期：2017/07/30

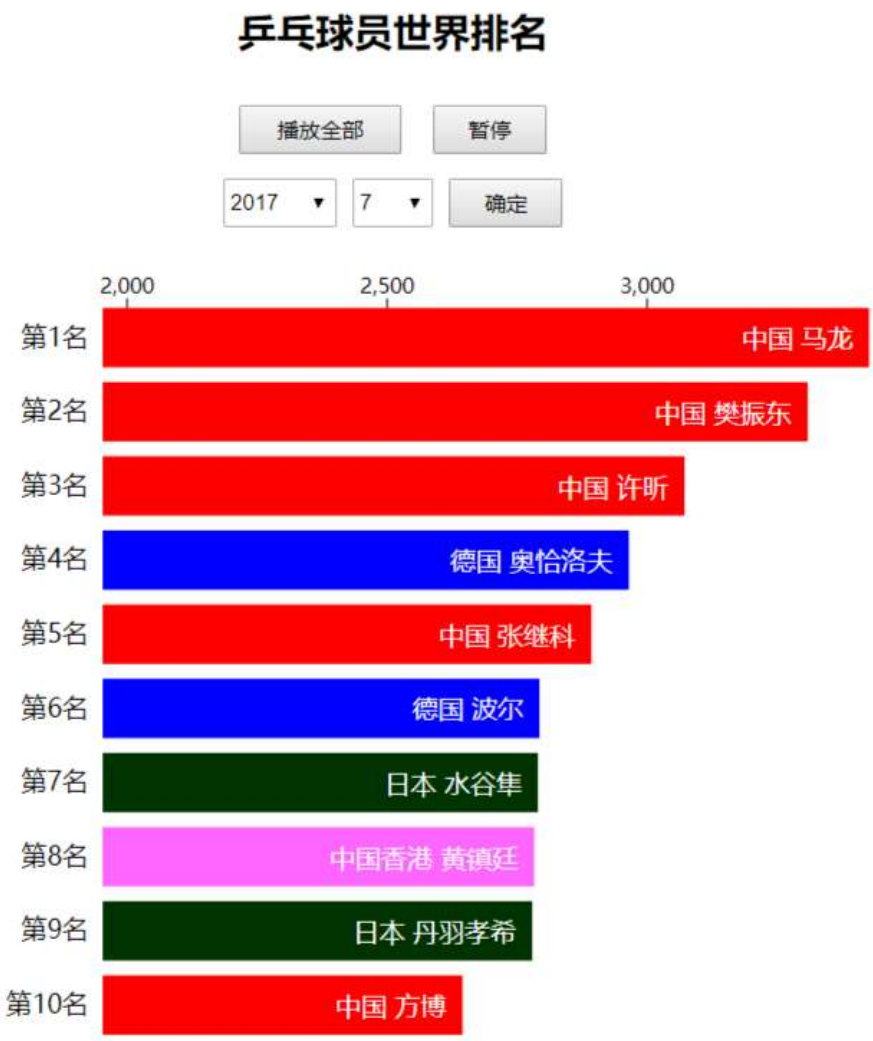
目录

一、概述

1. 程序功能概述

本程序是使用 d3 开发的一个交互式的乒乓球员世界排名网站。

运行时截图如下：



用户可以在下拉菜单中选择月份，点击“确定”按钮后图表即会更新为相应月份的数据。当用户的鼠标悬停在条形图上时，会出现一个跟随鼠标的悬浮的提示框显示该运动员该月的积分值。点击“播放全部”按钮可以让图表从 2001 年 1 月自动播放到 2017 年 7 月，点击“停止”按钮即可停止播放。

2. 如何运行本程序

本程序的文件目录如下：

文件名	主要功能
index.html	网页源代码
js.js	Javascript 脚本
style.css	样式表文件
data.csv	程序运行时将要加载的外部数据
js.test.js	单元测试文件

要想运行本程序，需要自行搭建一个简易服务器（否则无法访问外部数据），目前我已经把该程序部署到远程服务器上，因此你也可以直接访问 http://128.199.247.171:3179/来查看本程序（可能需要*翻墙*）。

3. 开发工具

```
WebStorm 2017.1.4
Build #WS-171.4694.29, built on June 7, 2017
JRE: 1.8.0_112-release-736-b21 amd64
JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o
```

4. 开发调试平台

```
Windows 10 10.0
```

二、获取原始数据

1.数据来源

新浪竞技风暴的世界体育排名页面中有 2001 年 1 月至今的国际乒联男子排名。网址：*http://info.sports.sina.com.cn/rank/ittf.php?action=search1&gender1=m&y*

ear=2017&month=7。

2. 爬虫的设计

由于数据量很大且有序，我使用 python 编写了一个简易爬虫爬取了所有数据。该爬虫基于 urllib2，使用正则表达式匹配相应数据，将爬取的数据以制表符为分隔符保存在 data.txt 中。

编写该爬虫遇到的最大阻力在于编码问题，最初我一直使用的是 utf-8 格式编码，可是爬取的数据全是乱码，后来查看原网页源代码时发现该网页的编码格式是 gb2312，修改获取网页源码的代码为 response.read().decode('gb2312', 'ignore').encode('utf-8')后才解决编码问题。

爬虫的源代码及爬取的原始数据均在 Spider 文件夹内。

三、数据处理

1. 概述

直接爬取的数据由于格式问题无法被 js 读取，因此需要进行必要的数据处理。

我处理该数据使用的是 Microsoft Office Excel 2016。

2. 处理过程

（1）使用 Excel 2016 打开爬虫生成的文件"data.txt"，选择以制表符为分隔符生成表格

（2）由于我们只需要每个月份前十名的数据，我们使用 Excel 的数据筛选功能，只保留排名那一栏前十名的数据

（3）添加表头

（4）将文件保存为 csv 格式

四、数据可视化

1. 准备工作

在 WebStorm 中新建工程，编写 html 和 css 代码，完成数据可视化的前期准备工作。

2. 开发过程

我在开发本网站的过程中遵循先易后难，逐层递进的方法。整体的开发分为以下三个阶段：

（1）制作某一个月的静态页面，数据写入程序，添加各种标签，搭建整体框架

（2）将画图的代码封装成函数，从外部读取数据，实现简单的过渡效果

（3）完善其它功能，如“播放全部”按钮，悬浮提示框等

3. 特色之处

（1）使用边距规范

为了避免频繁使用 d3 的 `translate` 方法降低代码可读性，我在本程序中定义变量 `svg` 来引用`<svg>`中包含的`<g>`元素。这里使用了一个偷换概念的小技巧，可以灵巧地处理边距问题，为坐标轴的设计带来了极大的便利。

（2）动画过渡效果

程序加载外部数据时以运动员的姓名为主键，这使得两个月之间条形图的渐变动画十分自然。也正是由于条形图以运动员姓名为主键，你可以一直关注某一个运动员，可以看到他的运动生涯的起起伏伏。

事实上，条形图中为每一个运动员维护了一个`<rect>`元素，每当新运动员进入榜单时，都会从最下面进入；当其排名或积分发生变化时，和他关联的`<rect>`元素的位置和长度都会有平稳的过渡效果；当其离开榜单时，则会从最下方离开。

（3）文字说明及悬浮提示框

程序使用一个两个<text>元素指示条形图代表的运动员。其中一个跟随条形图运动，始终显示在条形图<rect>的末尾，另一个为交互式的文本框，当鼠标移入的时候才会显示，而且它一直跟随鼠标的移动而移动，直到鼠标离开该条形图才消失。

这种交互式的文本框的原理是灵活地使用了<text>样式中的 opacity 标签，在程序运行的开始便定义一个<text>元素，并将其设置为完全透明，当鼠标移入某一个条形图时，触发相应事件，改变该<text>的位置、文字和透明度。

五、单元测试

1. 环境配置

本程序的单元测试采用的是 mocha+chai 的方式，首先在网上下载 node.js(with npm)并安装，我在程序目录下建立了名为 test 的文件夹，并在该目录下新建了两个名为”test_component.js”和”test_component.test.js”的脚本文件，第一个文件中放置待测试的代码，第二个文件中书写各种断言测试代码。

在上级目录键入 npm 的配置文件 package.json，需要注意的是，改配置文件中 d3 的版本号为 3.x，否则默认安装的 4.x 的版本与程序中的某些代码不兼容。然后在该目录下输入 npm install 即可完成安装，此时该目录下新增了一个名为 node_modules 的文件夹，里面储存的便是我们刚刚安装的库。

由于新版本的 WebStorm 内置了 mocha 测试框架，因此其配置十分简单。在 Run 的下拉菜单中选择 Edit Configurations，在弹出的窗口中点击“+“按钮，在下拉菜单中选择 mocha，输入配置信息即可。

2. 测试代码

例如我测试读取的数据的条数是否正确时的代码如下：

```
var expect = require('chai').expect;
var data_month = require("./test_component.js");
```

```
describe('数据条数的测试', function() {
    it('数据一共应为 10 条', function() {
        expect(data_month.length).to.be.equal(10);
    });
});
```

其中 test_component.js 中是读取数据的代码，并返回读取后的数据。

3. 测试结果

测试的结果和预期不符。

如上例，测试结果为：

```
AssertionError: expected undefined to equal 10
Expected :10
Actual   :[undefined]
```

实际运行的结果并不是预期的 10，而是[undefined]，分析原因可能是该单元测试需要读取外部数据，而 js 脚本在没有搭建服务器的条件下无法完成与外部数据的交互。

在我测试到这一步时已经耗费了大量时间（之前踩了太多的坑），限于时间无法解决这个问题。

六、代码重构

1. 重命名

为了提高代码的可读性与可维护性，我做了很多命名修改，以下是部分：

原名称	新名称	作用
play	buttonPlayAll	“播放全部“按钮
stop	buttonStop	“停止“按钮
year_temp	play_year	播放全部时当前年份

r_data_y	rankDataYear	某一年的全部数据

2. 封装成函数

例如：

原始代码中需要向悬浮框添加鼠标事件，最初时需要添加一次，图表变化后还需要添加一次；需要给条形图添加该代码，还需要给条形图上的静态文字添加该代码。即近乎相同的代码需要被写 4 遍，因此我在代码重构时将这一段代码封装成一个函数：`function addActivity(ele) { // function body }`。在需要给条形图添加该事件时只需调用 `addActivity(bars);`在给文字添加该事件时也只需要调用 `addActivity(tags);`这种设计大大减少了程序大小（减少了 30 多行），而且提升了可读性。