加载 JSON、数据处理、交互性

编程基础 – 2022 年秋季修订版 2022.11.29.b

JSON： JavaScript Object Notation

常见数据交换格式

- 图像和其他复杂媒体通常以二进制格式发送/接收
- 人类不太易读，因为二进制
- 示例：**jpeg**、**mp3**、**mp4**
- 数字和文本数据通常以文本格式发送/接收

- 通常可被人类读取
- 示例： **csv**、 **json**
- 这些格式中的任何一种都可用于在客户端（Web 浏览器）和服务器之间交换数据，或在本地计算机上加载和保存数据。
- 你也可以编造自己的文件格式，写自己的解析器（解释器让你的数据进出你的 for mat），但从现有格式开始通常更容易……JSON 是一种几乎可以满足任何需求的格式！

使用 JSON 加载和保存 JavaScript 数据 对象（属性，而不是方法）

- JavaScript Object Notation 可以存储我们迄今为止探索过的任何类型的 JavaScript 数据（信息，而不是对信息采取行动的代码）。
- 这些 JavaScript 对象可以根据需要复杂或简单。你在 JavaScript 中看到的一些例子：
- 简单数组： **[1，2，3]**
- 一个存储多个数组的对象，而数组又存储多个对象：

{list1： [{subitem1：“A”}，{subitem2：“B”， subitem3： “C”}]， list2：
[{subitem1：“C”}，{subitem2：“D”}]， list3： [{subitem1：“A”}，
{subitem2：“E”}]}

- JSON 以几乎与您习惯在 JavaScript 中看到的相同格式保存数据。

另一种格式 JSONP 允许保存 JavaScript 代码（方法）和数据。无论如何，使用这种格式需要创建 安全异常，因此它的使用量远远少于 JSON。

“键” = 属性名称

“值” = 存储在属性中的值

- JavaScript Object 属性及其值的组合称为 **键/值对**
- 在编写代码时，您通常可以互换使用术语“key”和“property”，人们会知道您的意思！
- 类似于**变量**与**属性**和**函数**与**方法**。我们希望您使用术语来表明您了解自己在做什么，但我们不会挑剔这些细节！
- 在**阅读** JavaScript 语言解释时，您最常会遇到术语“键/值对”，所以请记住它的含义！

- 这个术语甚至内置在 JavaScript 和 D3 中！
- 您可能还记得我们使用 **.keys**（） 从示例城市数据中提取唯一值
- 这是 您需要使用正确术语的另一个实例;JavaScript 比你的导师挑剔得多！

JSON 要求在每个级别的每个**键**（属性名称）周围加上**引号**

JavaScript 对象

```
让 myData = { 列表 1：[
      {分项 1：“A”},
      {分项 2a： [3，4]， 分项 2b： “C”}
      ]，列表 2：
[
      {子项 1： {a：7， b：true}},
      {子项 2：“D”}
      ]
}
```

中文别名编码

```
{
“列表 1”：[
      {“分项 1”：“A”},
      {“分项目 2a”： [3，4]，
        “分项 2b”： “C”}
      ],
“列表 2”：[
      {“子项 1”：{“a”： 7，“b”： true}},
      {“子项目 2”：“D”}
      ]
}
```

否则，它基本上与创建 JavaScript 对象相同！

JSON 文件格式不支持 代码注释 或 变量赋值

有效的 JavaScript 对象

```
让 myData = { 列表 1：[
      {分项 1：“A”},
      {分项 2a： [3，4]， 分项 2b： “C”}
      ]， // 结束列表 1 列表 2：[
      {子项 1： {a：7， b：true}},
      {子项 2：“D”}
      ] / / 结束列表 2
} // 结束 myData
```

有效的 JSON 编码

```
{
“列表 1”：[
      {“分项 1”：“A”},
      {“分项目 2a”： [3，4]，
        “分项 2b”： “C”}
      ],
“列表 2”：[
      {“子项 1”：{“a”： 7，“b”： true}},
      {“子项目 2”：“D”}
      ]
}
```

请记住在将对象转换为 JSON 时删除这些额外的内容。

演示：JSON 只能存储值，不能存储变量引用

```
让幸运数字 = [1， 2， 3]; let myObject = {name： “Fred”， numbers：
luckyNumbers};
luckyNumbers[0] = 7; let myJSON = JSON.stringify
（myObject）；幸运数字[2] = 9; let myRerestoreObject =
JSON.parse（myJSON）； 幸运数字[1] = 8; console.log
（myObject.numbers）; console.log
（myRerestoreObject.numbers）；
```

myObject.numbers 包含一个引用

将 myObject 的副本存储为 JSON

更改特定 数组 元素

将 JSON 解析为 新对象

更改特定 数组 元素

myObject.numbers 是 [7， 8， 9]

更改特定 数组 元素

myRerestoreObject.numbers 是 [7, 2， 3]

JSON.stringify（） “冻结”了这些值，以便存储的数据中没有变量引用，并且每个存储的键都有一个固定的值！ 与原始数据源的任何连接都将永久断开。

用于术语项目的 JSON

从本周开始，您需要将部分或 全部 数据转换为 JSON 格式，以便将其用于可视化效果。

- 在术语项目的最终版本之前，您可以选择在 JSON 代码中仅包含可视化所需的数据。

- 换句话说，您可以手动重组数据以适应分配，如果这有助于您了解正在执行的操作。
- 但是，我们建议您 尽可能 多地练习操作数据的技术，以便为 最终学期的项目要求做好准备

对于术语项目的最终版本， **所有原始数据*都需要最终出现在 JSON 文件中**，即使其中大部分数据在被 JavaScript 加载后被转换（或忽略）!

**例外：您可以排除收集不一致的数据，即在项目中途停止收集或更改收集方法的数据。*

项目框架

- 下载并打开 **term_project_framework.zip**
- 将此框架用于练习 #10 以及您的术语项目的所有未来尝试
- 确保您拥有学期项目所需的所有部件 ' 您可以 删除尚不需要的部件
- 最后， 您必须 删除所有未使用的部件
- 对于使用此框架的所有提交， 您必须：

- 删除说明性注释
- 为每个主要代码部分添加您自己的解释性注释
- 对于最终学期的项目提交，您还必须为程序中的每个主要功能编写完整的功能文档
- 这包括我们在您在决赛中使用的框架中提供的任何内容！

索引.html

- 请务必通读 所有 评论
- 按照说明操作后删除评论
- 如果您忘记了什么，您可以随时打开入门器的另一个副本！
- 为页面提供基本框架，但您可以自定义布局
- 确保您的可视化位于页面顶部， 任何标题和文本都在下方！
- 请记住，**d3.min.js** 和任何 **css**（可选）必须在页面**<正文>**之前加载
- 尽早加载 CSS 可确保页面正确呈现
- 尽早加载 D3 可确保当我们到达页面底部时它已准备就绪
- 我们在页面正文之后加载了 **主要.js** 以确保加载其他所有内容

主.css

- 包括一些我们用于其他项目的样式
- 使用这些是 可选的。 您可以将 此 CSS 替换为您自己的 CSS
- 请务必至少查看此 CSS。请记住，如果您使用此文件中提供的类名，则分配这些类的任何内容也将采用这些样式！
- 注意示例 /* 代码注释 */
- 您应该在 CSS 中提供代码注释，以解释如何在您的项目中使用它。 ' 如果您决定不使用外部 CSS 文件，请务必删除此文件
- 请务必同时从**索引**中删除 **<link>** 标记.html

data.json

- 包含热图练习中的示例 JSON da ta 的文件
- 我们还将在克利夫兰点图示例中使用此数据

- 如果您在从文件加载数据时遇到问题，您可以暂时将数据分配给变量，就像到目前为止所做的那样。
- 对于您的学期项目最终，您需要弄清楚如何完成这项工作
- 对于 **try1/final**，您不需要使用框架的这一部分 ' 将此文件中的数据替换为您自己的 JSON 格式数据。
- 您也可以为该文件指定一个更能描述您的项目的名称，例如 **myStudyHabits.json**
- 文件名应遵循类文件名约定（不能有空格！），并且必须具有 扩展名 。**JSON** 以便正确读取

d3.min.js

- 与往常一样，您无需 查看此文件...多
- 但是，您应该确保始终使用 **v7.6.1**
- 您可以在文件第 1 行的代码注释中找到版本号
- 这是术语项目所需的版本

主.js

- 为您设置一些常用的**配置变量**
- 在这里，我们创建了一个名为 **margin** 的对象，其中包含四个边框的四个**属性**。
- 这是 D3 创建者通常的工作方式，但如果您愿意，可以继续使用四个单独的变量
- 提供要绘制到 **的 svg**
- （可选）提供存储在变量**可视化**中的 **svg 组**，供您绘制到
- 您需要为此取消注释代码
- 由您决定如何设置边距和偏移绘图
- **async function（）** 代码块是我们不会要求您解释的代码部分。您需要了解的只是：
- 它加载 **data.json**（或你在引号中为文件名提供的任何名称）
- 它将数据 **解析** 为 JavaScript **对象**
- 它将该数据对象作为**参数**传递给函数 **buildVisual**（或您提供的任何函数名称）
- 它接受从它调用的函数返回的任何数据，并将其分配给全局**数据**

函数 构建可视化（数据）

- 接收加载的 JSON 数据的函数
- 调用我们提供的一系列“存根函数”，以保持井井有条
- 您可以更改其中任何一个的函数名称或参数
- 最后返回与发送时相同的数据，确保全局变量 数据将具有数据的副本，以备以后再次需要它

“存根 函数”

- 包含最少信息的函数。
- 更像是描述程序在每个阶段应该做什么 ' **组织数据** 应该获取您的数据并在需要时对其进行过滤和排序。
- 如果你不需要他的函数，请删除对它的调用并删除函数
- 请注意，此函数**首先在我们的 buildVisualization** 函数中调用，尽管它是在框架中编写**的**。如果这对您更有意义，您可以更改这些函数的顺序！
- **buildScales** 应该获取发送给它的数据，并构建像 **xScale** 和 **yScale** 这样的东西。
- 请注意，我们为您常用的一些尺度提供了全局变量。请记住，在函数中创建的任何变量都是**局部变量**，因此如果需要在其他地方使用，则必须将其作为**参数传递给**另一个函数或从函数**返回**。
- **drawVisualization** 实际上应该 使用您之前设置的所有内容来绘制可视化
- 我们提供了一个建议的第二个参数绘制，您可以使用它来告诉函数是要绘制到 **svg** 还是可视化 组，还是绘制到您设置的其他 DOM 元素！

代码块中的调试器

- 到目前为止，您已经能够检查 变量并在绘制可视化后检查可视化的情况
- 这是因为你所做的一切都是**在全球背景下**进行的。
- 换句话说，所有重要变量都可以通过 JavaScript 控制台访问，因为它们不是在函数中创建的。
- 使用术语项目框架时，大部分工作都在函数内进行
- 我们为您提供了一些全局变量，但如果您想调查绘制代码中发生的情况，则需要使用调试器！
- 调试器可能令人生畏，但 确实值得 了解
- 如果没有别的，您可以使用 **调试器;** 关键字，以在遇到问题的代码块中间停止程序的执行 。
- 然后，当前代码块中存在的所有局部变量仍将在 JavaScript 控制台上可供您使用，这可能是检查它们的更温和的方法！

克利夫兰点图示例：简单交互性

cdp_framework_starter.zip

- 下载、 解压缩和测试
- 基于您的术语项目框架构建，但具有增强功能
- 考虑类似的增强功能，以帮助您更好地了解自己的项目。不要只是复制我们的！

函数 构建可视化（数据）

- 接收由 **d3.json** 加载的数据
- 调用**组织**数据，传递数据并将其返回**渲染数据**
- 对于您的项目，如果您需要执行任何数据转换（例如筛选、减少或排序），则可以在 **organizeData** 中执行此操作
- 使用有组织的数据版本调用 **buildScales**
- 如果您根本不需要组织数据，则可以仅使用**数据**作为参数
- 调用 **drawVisualization**，传递有组织的数据以及绘制到的位置
- 可视化实际上是这个程序中的全局变量，但我们在这里明确传递它是为了明确可以 _改变 **drawVisualization** 绘制的内容。
- 最后，**返回**数据，以便可以将其存储在全局**数据**中
- 在这里，我们知道我们希望在组织数据时处理数据，因此我们返回 **renderData**。如果我们希望能够使用原始数据，我们可以返回**数据**
- 在这个特定的例子中，实际上没有区别，因为我们还没有操纵我们的数据！

函数 **绘制可视化**（数据、绘图）

- 请注意，我们选择将绘图分解为几个函数
- 此函数只是绘制 X 轴和 Y 轴，然后调用另一个函数 **plotPoints**（） 来完成大部分绘制
- 为什么要这样做？我们希望能够重绘用户所选月份的温度数据，但我们不需要每次都重绘整个可视化效果！
- 这是一种高级的工作方式，可以向我们展示您真正了解您在可视化中正在执行的操作（当然，只要 您不只是 复制此示例）
- 如果您对此概念有疑问，如果添加任何交互性，也可以每次重绘整个可视化效果。
- 记住 D3 的 **selection.remove**（） 如果你需要这样做！
- 还要记住，交互性在术语项目上 是完全可选的 。只有当您认为它会使您的可视化更清晰时（或者您的教师建议您这样做！

复习挑战#1： 组织数据（）

- 现在，我们的城市名称实际上没有任何顺序

- 函数 **organizeData**（）目前只返回与在 ' 中发送的数据相同的数据 尝试编写一个 **sort**（） 函数，以便数据按字母顺序排列：
- 芝加哥出现在我们 Y 轴的顶部 ' 圣何塞出现在我们 Y 轴底部的 g 中

- *数组*提醒。**sort**（） 辅助函数形式：

```
函数 （item1， item2）{if （交换条件）{ 返
    回 1 } 返回 -1
}
```

审查挑战#1：解决方案

```
数据。sort（function （a， b）{ if （a.城市。toLowerCase（） > b.城市。
    toLowerCase（））{ return 1
    } 返回 -1
})
```

- 注意：虽然我们返回数据是为了遵循框架的格式，但从技术上讲，我们的全局**数据**已经更新。
- 为什么？因为 **Array.sort（）** 对数据进行“就地”排序，而不是制作数组的副本！

交互式代码

- 在 main 的底部 **.js** 您将找到应该使该项目具有交互性的代码
- 这段代码的第一部分将我们的**<select>**菜单上的**“change”**事件（我们通过 **id=“month”**标识）连接到一个名为 **filterAndRedraw** 的函数 ' 现在我们可以看到为什么我们需要全局变量了！
- 请记住， **事件处理程序** 函数只接收一个参数，它是自动生成的参数， 而不是 您指定的参数！
- 您只能指定将“捕获”此参数的变量。我们在这里称之为 **事件**。
- 你最终可能甚至没有使用这个变量，除非提醒你已经编写了一个事件处理函数。
- 在 **filterAndRedraw** 的代码中，我们展示了另一种寻址触发事件的对象的方法（而不是使用 **document.getElementById（）**）。
- 如果您有多个文档元素触发相同的功能，这将特别方便！
- 因此，我们希望 **filterAndRedraw（）** 处理的任何数据都需要是**全局的**。
- 有更复杂的方法来处理这种情况，但它们超出了本课程的范围。全局变量是一种适合初学者的工作方式，适合本课程！

filterAndRedraw（）是怎么回事？

- 现在，如果您使用弹出菜单：
- 选择**“2015 年全年”**以外的任何内容会导致散点图消失。
- 此外，即使你切换回“整个 2015 年”，这些点仍然消失了！
- 请注意，即使点消失，X 轴和 Y 轴也会显示 n！
- 回想一下我们如何将绘图分解为几个单独的函数
- 此处调用的 **clearPoints**（） 函数使用您在创建热图时学到的语法删除了所有点。
- 但是， **plotPoints**（） 函数似乎没有绘制任何东西，即使我们知道它最初有效！
- 做一些侦探工作来找出问题所在
- 您可以通过阅读这部分代码来弄清楚这一点
- 您还应该使用常用工具来检查错误、确认变量值等。

复习挑战#2： 过滤器和重绘（）

- 现在的问题是，对 **plotPoints**（）的调用正在发送一个过滤的空数组 ， 以提供绘制点的数据。 编写代码，内容如下：
- 如果用户选择月份**“all”**，则**筛选的值**应与**数据**相同
- 否则，**请**过滤数据，以便**过滤后**仅包含所选月份的数据 ' 数组提醒.**filter**（） 辅助函数：

```
函数（值）{ if （包含项的条件）{返回真
    }
    返回假
}
```

- 问：为什么我们需要在我们正在编写的条件逻辑之外声明过滤以确定其值？

审查挑战#2：解决方案

- 这里有条件逻辑的一个可能版本：

```
if （monthChoice == “all”） { filter = data
```

```
} else { filter = data.filter（function （value） { return （value.月 ==
        月选择）}）
}
```

- 请注意，我们已将过滤器的条件逻辑减少到一行。
- 你的过滤器函数必须返回 **true** 或 **false**，但它可以以任何你知道的方式执行此操作
- 在这种情况下，我们写了一个**布尔语句**：与 **if**（） 的括号内的内容相同，但它只是直接提供**真**或**假**
- 如果这对您更有意义，您可以编写更适合初学者的 **if/else** 版本！

回顾挑战#3：连接点！

- 很难看到每个城市的低温和高温之间的联系。
- 为了练习使用数据并创建实际的简单克利夫兰点图，让我们连接每个城市的点：
- 对于每个**数据**项，创建一**行**类**词干**
- 确保不要在可视化中选择其他行;你会得到奇怪的结果！
- 绘制每条线，以便连接该数据的**低温**和**高温**

- 将线定位在 Y 轴上，使其与它所代表**的城市**对齐 nts
- 确保线条最终位于点 后面 而不是点前面
- 提醒：**“线”**对象有 5 个必需属性： x1、**x2、y1、y2、笔触**
- 提示：您需要的所有其他信息都可以在用于绘制此可视化效果其他部分的代码中找到
- 提示：为了更清楚地看到您的作品，您可能需要从我们刚刚启用的弹出菜单中选择一个月，这样您只能看到每个城市的 1 对点！

审查挑战#3：解决方案

```
绘图。selectAll（“line.stems”）
    .数据（数据）
    .连接（“行”）
    .分类（“词干”， 真）
    .attr（“x1”， function （value） { return xScale
        （value.温度[0]）
    })
    .attr（“x2”， function （value） { return xScale
        （value.温度[1]）
    })
    .attr（“y1”， function （value） { return yScale
        （value.城市）
    })
    .attr（“y2”， function （value） { return yScale
        （value.城市）
    })
    .attr（“笔触”， “灰色”）
    .attr（“笔触宽度”， “1px”）
```

- 要在圆圈**后面**绘制线条，需要在代码中的圆圈**之前**绘制线条
- 您是否记得使用选择**类**的模式以及 所需的形状类型？

- 为了提醒自己这一点的重要性，请尝试将 **line.stems** 更改为仅行。您应该会看到一些问题！

*作为 **以后的额外挑战**，请考虑扩展和重新设计每个城市的刻度，以一直延伸到 x 轴以形成虚线，这可能会提高每个城市数据行的可读性。*

自定义克利夫兰点图

- 完整的解决方案可以下载 **为 cdp_framework_completed.zip**
- 我们建议您花时间尝试了解如何创建该版本，而不仅仅是使用成品
- 仅稍微 自定义此代码的术语项目将不会获得好分数！你需要真正使项目成为你自己的项目。
- 一个好方法是从您知道的基础知识中构建克利夫兰点图，而不仅仅是遵循我们的食谱
- 这里唯一 真正新 的东西是添加一个交互式组件，我们甚至希望看到其中的定制。

- 考虑示例中弹出菜单以外的其他内容是否最适合您的数据。
- 也许复选框？单选按钮？这真的取决于你要 修改什么！

优先事项是清晰度和创造性编码！

- 我们应该能够轻松比较有助于说明您的假设的数据
- （无论 它最终是真的还是假的;我们只需要能够根据我们所看到的得出这个结论！）
- 使用一些方法来增强您的项目，以尽可能清晰地显示您的数据。
- 例如，如果对于**“2015 年全年”**选项，我们只显示当年每个城市的最低和最高温度，而不是所有温度，不是看起来更好吗？
- 这是你可以用你在本课程中学到的东西做的事情。挑战在于以与您的数据配合的方式将这些部分组合在一起！
- 另一个想法：如果我们想在单独的线上显示每个月的温度，所以每个 **city** 将是一个波段，但每个月将是波段内的一个点，该怎么办？
- 对于此 特定可视化，您最终会在 Y 轴上显示 120 条数据，这会非常混乱，因此这可能 不 是一个好主意。
- 但是， 这又是您可以通过结合您在本课程中学到的东西来做的事情。您只需要逐步弄清楚如何开发每个部分！

- 在继续下一件事之前，请确保一件事有效，最终您将 100% 了解您的项目！