Python Flask 数据可视化

原创 太阳雪 Python技术 3月17日

数据可视化是数据处理中的重要部分，前面我们了解了 Flask 的开发和部署，如何用 Flask 做数据可视化呢？今天我们来了解一下。

Python 语言极富表达力，并且拥有众多的数据分析库和框架，是数据分析的首选；

echarts，最初由百度团队开发，现在已独立成 Apache 旗下一款国际化产品，是基于 Web 的数据可视化框架，API 简单明了，应用极为广泛；

Python 和 echarts 的完美结合就是 pyecharts

pyecharts 简介

pyecharts 使得可以用 Python 语言，完成 echarts 中对图表的各种操作，并且让编写代码更便利

pyecharts 中的概念和 echarts 是相通的，对于刚接触的同学，无论从 pyecharts 还是 echarts 开始了解都可以

图表类

pyecharts 中的图表都是类，都继承自 `Base` 基类，构造函数接受一个 `init_opts` 参数，用于设置图表的属性

以下是常用 API 接口：

- `add_js_func`：将 js 脚本附加在图表 Html 中
- `set_global_opts`：设置图表属性
- `render`：渲染出图表的 Html 文件
- `dump_options_with_quotes`：将图表所有设置导出为 json，用于前后分离

全局配置

pyecharts 将图表中和数据无关的属性，集中在全局配置中，也就是这些配置是服务于整个图表的，比如 标题、图例、工具栏、数据提示框、区域缩放等，每种配置项，都是一个 `BasicOpts` 的子类，通过图标对象的 `set_global_opts` 方法设置，例如：

```
from pyecharts.charts import Bar

bar = Bar()
bar.set_global_opts(
```

```
    title_opts=opts.TitleOpts(
        title="Bar-基本示例",
        subtitle="我是副标题",
        pos_left= "center",
        pos_top="top"),
    legend_opts=opts.LegendOpts(
        pos_top="60"
))
```

系列配置

`系列`（series）是很常见的名词。在 echarts 里，`系列`（series）是指：一组数值以及他们映射成的图。“系列”这个词原本可能来源于“一系列的数据”，而在 echarts 中取其扩展的概念，不仅表示数据，也表示数据映射成的图。所以，一个 `系列` 包含的要素至少有：一组数值、图表类型（`series.type`）、以及其他的关于这些数据如何映射成图的参数。

pyecharts `系列配置` 和 `全局配置` 类似，用于对图表中 `系列` 进行设置，比如设置 `系列` 样式、坐标系、颜色、形状、特殊点，以及等。

例如，柱状图上不显示标签：

```
from pyecharts.charts import Bar

bar = Bar()
bar.set_series_opts(label_opts=opts.LabelOpts(is_show=False))
```

pyecharts 安装

首先安装 pyecharts：

```
pip install pyecharts
```

安装完后，在 Python 交互式环境(REPL)中，可以查看版本信息：

```
>>> import pyecharts
>>> print(pyecharts.__version__)
1.7.0
```

Flask 集成

前面我们了解了 Flask 的开发，对于一个应用来说，需要有 `视图函数`，`模板`、和 `路由`，echarts 是一个前台框架，只

要将页面做成模板，然后将数据写入模板就好，这样确实是可以做的，不过 pyecharts 已经处理了大部分工作，只要在 Python 中开发代码就好了。

pyecharts 和 Flask 集成，四种形式，分别是 模板渲染、前后分离、定时全集更新 和 增量数据更新

模板渲染

模板渲染是比较方便的，可以不用写前台页面，因为 pyecharts 已经定义了很多模板，以及模板宏，调用很方便。

第一步 下载 pyecharts 的模板

可以从 github 的 pyecharts 项目中获取，https://github.com/pyecharts/pyecharts

如果用 pip 安装的 pyecharts ，可以在安装环境中的模块目录下找到，即 Python home 中的 `Lib/site-packages/pyecharts/render/templates`

第二步 将模板放入项目目录下

在我们的 Flask 应用的目录的 templates 模板下，创建 pyecharts 目录，来存放复制的 pyecharts 模板。

这样可以避免与 Flask 应用中我们自建的模板混淆。

第三步 渲染图表

我们将业务逻辑写入都写在 Flask 启动脚本 `app.py` 中：

```
from flask import Flask  # 引入 Flask
from jinja2 import Markup, Environment, FileSystemLoader
from pyecharts.globals import CurrentConfig

CurrentConfig.GLOBAL_ENV = Environment(loader=FileSystemLoader("./templates/pyecharts"))

from pyecharts import options as opts
from pyecharts.charts import Bar

app = Flask(__name__)

def bar_base() -> Bar:  # -> 表示要返回的是类型
    c = (
        Bar()
        .add_xaxis(["衬衫", "羊毛衫", "雪纺衫", "裤子", "高跟鞋", "袜子"])
        .add_yaxis("商家A", [5, 20, 36, 10, 75, 90])
        .add_yaxis("商家B", [15, 25, 16, 55, 48, 8])
        .set_global_opts(
            title_opts=opts.TitleOpts(
```

```
                title="Bar-基本示例",
                subtitle="我是副标题"
            )
        )
    )
    return c

@app.route("/")
def index():
    c = bar_base()
    return Markup(c.render_embed())
```

- 首先引入 Flask、jinjia2 和 pyecharts
- 为全局变量设置 jinjia2 环境，指定模板路径为 `/templates/pyecharts` 即我们存放 pyecharts 模板的路径。这样不会影响 Flask 的默认模板路径
- 定义图表工厂方法，返回一个图表实例，图表实例支持点串联操作
- `add_xaxis` 添加 X 轴显示的项目
- `add_yaxis` 添加 Y 轴数据分类和数值，相当于分组，可以添加多个
- `set_global_opts` 设置图标的全局配置
- 视图函数中，用图表工厂方法 `bar_base` 创建一个图表实体，返回 `render_embed` 经过 jinjia2 的渲染结果
- `render_embed` 返回的是合成好的 `html` 可以直接返回给前台做展示

前后分离

模板渲染虽然方便，但是不够灵活，比如要修改已有页面，加上一个图表，这是可以考虑用前后分离的方式

前两步和 模板渲染 中的一样

第三步 创建前台页面

创建一个 html 文件 `index.html`，存放在 templates 文件夹下，内容和 echarts 一样，主要是需要引用 echarts 框架，和 jQuery 框架(其他的Ajax框架均可)，定义显示图表的 Dom，最后在页面加载完成回调方法中，通过 ajax 请求后台数据，异步将获取到的图标数据设置到图表中:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>我的图表</title>
    <script src="https://cdn.bootcss.com/jquery/3.0.0/jquery.min.js"></script>
    <script type="text/javascript" src="https://assets.pyecharts.org/assets/echarts.min.js"></script
</head>
<body>
    <div id="bar" style="width:1000px; height:600px;"></div>
    <script>
```

```
        $(
            function () {
                var chart = echarts.init(document.getElementById('bar'), 'white', {renderer: 'canvas
                $.ajax({
                    type: "GET",
                    url: "/barChart",
                    dataType: 'json',
                    success: function (result) {
                        chart.setOption(result);
                    }
                });
            }
        )
    </script>
</body>
</html>
```

第四步 编写后台相应方法

前台页面中定义了 ajax 请求路径是 barChart，我们就写一个处理该请求的视图方法：

```
@app.route("/barChart")
def bar_chart():
    c = bar_base()
    return c.dump_options_with_quotes()
```

- 定义图表的方式和 模板渲染一样
- 视图方法中，用工厂方法创建视图对象，返回 `dump_options_with_quotes` 的结果
- `dump_options_with_quotes` 将图标的配置集成为前台需要的格式，返回 JSON 数据

最后启动 Flask 应用，在localhost:5000 就能看到效果

前后分离的方式更常用，可以让前台的展示发挥最大的优势，Flask 后台提供图表需要的数据和设置

定时全量更新

有很多场景需要实时更新图表内容，实现方式是将 前后分离 的方式，获取后台图标配置的请求写成定时调用的，将得到的图标数据通过 `setOption` 设置到图表对象中。

后台视图方法每次重新根据查询条件，获取新的数据，设置到图表对象中，再用 `dump_options_with_quotes` 将设置导出，返回给前台

定时增量更新

增量更新在数据监控的场景中很常用，实现方式和全量更新有些差别

首先需要得到一个图表的设置，这个和全量更新一样

然后将获取增量数据的方法作为定时的，在回调函数中，为图标设置增量数据，与全量更新不同的是只更新 `系列数据`，echarts 会处理好图表的变化，包括动画效果

前台获取增量数据并更新的方法：

```
function getDynamicData() {
    $.ajax({
        type: "GET",
        url: "/lineDynamicData",
        dataType: "json",
        success: function (result) {
            old_data.push([result.name, result.value]);
            chart.setOption({
                series: [{data: old_data}]
            });
        }
    });
}
```

`old_data` 图表数据的应用：

```
old_data = chart.getOption().series[0].data;
```

如果需要同时将最早的数据清除掉，只需要将需要去除的数据从 `old_data` 中删除就行：

```
old_data.shift();  // 清楚最早的一个数据
```

后台数据处理

根据图表数据要求，每次前台请求增量数据时，将最新的数据返回

这里需要注意到是增量数据范围，即**怎么确定增量数据**

常用 `数据产生时间` 或者 `数据 id` 作为增量条件，例如图表展示的是在线用户数变化曲线，在线用户数，会定时存放在库表中，每条记录都有个 `id`，每次请求增量数据时，将已经获取到数据的最大的 `id` 值作为请求参数，后台就可以获取该主键值后面的数据，作为增量数据。

渲染图片

在有些场景下，需要生成图表图片，Python 有很多图表处理工具，可以做图像生成。

对 echarts 来说，也有生成图片的功能，不过需要在浏览器中，pyecharts 作为 Python 和 echarts 的桥梁，支持后端生成图表图片。

pyecharts 提供了 `selenium`，`phantomjs` 和 `pyppeteer` 三种方式渲染图片，其原理是用无头浏览器，渲染图表页面后，用 echarts 生成图片功能，生成图片。

这里我们用 `selenium` 做演示

安装 snapshot-selenium

snapshot-selenium 是 pyecharts + selenium 渲染图片的扩展，selenium 需要配置 browser driver，推荐使用 Chrome 浏览器，可以开启 headless 模式，具体配置可参考 selenium-python相关介绍。

使用

pyecharts 使用 `make_snapshot` 直接生成图片，支持生成图片相关的配置，如 echarts html 文件名，输出文件名，浏览器种类等：

```
from pyecharts import options as opts
from pyecharts.charts import Bar
from pyecharts.render import make_snapshot

from snapshot_selenium import snapshot

def bar_chart() -> Bar:
    c = (
        Bar()
        .add_xaxis(["衬衫", "毛衣", "领带", "裤子", "风衣", "高跟鞋", "袜子"])
        .add_yaxis("商家A", [114, 55, 27, 101, 125, 27, 105])
        .add_yaxis("商家B", [57, 134, 137, 129, 145, 60, 49])
        .reversal_axis()
        .set_series_opts(label_opts=opts.LabelOpts(position="right"))
        .set_global_opts(title_opts=opts.TitleOpts(title="Bar-测试渲染图片"))
    )
    return c

make_snapshot(snapshot, bar_chart().render(), "bar0.png")
```

- 先引入 `make_snapshot` 和 `snapshot`
- 定义图表工厂方法

- 调用 `make_snapshot` 导出图片，第一个参数是渲染扩展工具，第二个是生成的 Html 文件路径，第三个参数是生成的图片文件路径
- 由于是通过无头浏览器中模拟的，图表复杂或者数据多时，渲染可能较慢，可以通过 `make_snapshot` 命名参数 `delay` 来设置等待时间，默认为 2 秒

总结

今天介绍了使用 pyecharts 实现数据可视化的方法，并描述了如何与 Flask 集成，以及几种生成图表的方式，可以尝试一下，以便做出更好玩更有用的 Flask 应用。

参考

- pyecharts: http://pyecharts.org/#/zh-cn/
- https://selenium-python.readthedocs.io/installation.html#drivers
- https://www.echartsjs.com/examples/zh/index.html

> 示例代码：https://github.com/JustDoPython/python-100-day/tree/master/flask_pyecharts