## 1. 结构化数据和非结构化数据

- **结构化数据**：
  - 是指格式固定且组织良好的数据，通常存储在关系数据库中，便于检索和管理。
  - 示例：成绩单、表格、数据库中的记录。
  - 例如，成绩单可以用一个二维表格表示，每行代表一个学生，每列代表一个科目。
- **非结构化数据**：
  - 是指没有预定义的数据模型，不易于使用传统数据库工具进行分析和处理。
  - 示例：视频、图像、自然语言（如文本）。
  - 例如，视频文件包含图像、声音和时间序列数据，无法用传统表格格式表示。

## 2. 三基色

- **红（Red）、绿（Green）、蓝（Blue）**
  - 这三种颜色可以通过不同的组合和强度形成各种其他颜色。
  - 在数字图像处理中，通常使用RGB模型。

## 3. 常见的可视化工具

- **ECharts**：
  - 由百度开源，基于JavaScript的图表库，支持多种图表类型，易于集成。
- **D3.js**：
  - 一个用于创建动态、交互性数据可视化的JavaScript库，提供了操作文档对象模型（DOM）的强大功能。
- **PowerBI**：
  - 由微软开发的商业分析工具，支持数据连接、处理、可视化和报告生成。

## 4. 可视化效果的评价

- **有用**：能有效传达信息，帮助用户理解数据。
- **有效**：能够准确无误地表达数据内容。
- **满足用户需求**：图表设计符合用户的预期和使用场景。

## 5. Excel的相对和绝对定位

- **相对定位**：
  - 格式：一个字母+一个数字（如B2）。
  - 特点：可以横向和纵向拖拽，引用会根据拖拽位置自动调整。
- **绝对定位**：
  - 格式：$一个字母+$一个数字（如$B$2）。
  - 特点：固定行和列，引用不会随拖拽位置变化。
  - 混合绝对定位示例：$B2（横向固定，纵向可拖拽）、B$2（纵向固定，横向可拖拽）。

## 6. 折线图、柱状图、饼图、散点图包含的元素和数据

- **折线图、柱状图**：
  - 数据格式：`[1, 2, 3, 4]`
  - 图表元素：Title、xAxis、yAxis、series、grid、legend
- **饼图**：
  - 数据格式：`[{name: 'name1', value: 'value1'}, {name: 'name2', value: 'value2'}]`
  - 图表元素：Title、series、legend（无xAxis、yAxis）
- **散点图**：
  - 数据格式：`[[x1, y1], [x2, y2]]`
  - 图表元素：Title、xAxis、yAxis、series、grid、legend

## 7. 对一组元素中的部分元素单独设置其属性，使用回调函数

- 示例：
  - **散点图点的大小设置**：

```
series: [{
  type: 'scatter',
  data: [[10, 20], [30, 40]],
  symbolSize: function (data) {
    return data[1]; // 根据y值设置大小
  }
}]
```

  - **柱状图中柱子的颜色设置**：

```
series: [{
  type: 'bar',
  data: [10, 20, 30],
  itemStyle: {
    color: function (params) {
      var colorList = ['#c23531', '#2f4554', '#61a0a8'];
      return colorList[params.dataIndex];
    }
  }
}]
```

## 8. D3.js和ECharts.js的异同和特点

- **相同点**：
  - 都可以使用SVG和Canvas进行绘制。
- **不同点**：
  - **ECharts**：
    - 先配置再绘制，使用配置项（option）来定义图表。
    - 适合快速生成常见图表，封装较高。

- **D3.js**：
  - 按步骤绘制，通过直接操作DOM元素来创建图表。
  - 更灵活，适合高度定制化的图表，具有链式调用特点。

## 9. 基础JS函数的使用：map、sort

- **map**：
  - 用于创建一个新数组，数组中的每个元素是原数组元素调用函数后的返回值。
  - 示例：

```
let arr = [1, 2, 3];
let newArr = arr.map(x => x * 2); // [2, 4, 6]
```

- **sort**：
  - 用于对数组元素进行排序，默认按Unicode顺序排序。
  - 示例：

```
let arr = [3, 1, 4, 2];
arr.sort((a, b) => a - b); // [1, 2, 3, 4]
```

## 10. 复杂dict的定义和使用

- **定义和使用示例**：

```
let dict = {
  'events': [
    {
      'time': {'hour': 23, 'minute': 35, 'second': 12},
      'year': {'yearnumber': 2024, 'leapyear': true},
      'name': 'aaa',
      'sequence': [1, 2, 3, 4, 45]
    },
    {
      'time': {'hour': 22, 'minute': 30, 'second': 45},
      'year': {'yearnumber': 2023, 'leapyear': false},
      'name': 'bbb',
      'sequence': [6, 7, 8, 9, 10]
    }
  ]
};

// 在字典中定位minute的数值：
let minute = dict['events'][0]['time']['minute']; // 35

// 在数组中定位minute的数值：
let minute2 = dict['events'][1]['time']['minute']; // 30
```

## 11. 前后端用HTTP进行交互

- 前后端通过HTTP协议进行数据交互，通常通过GET和POST请求来传递数据。
  - **GET请求**：用于请求数据，参数通常附加在URL后面。
  - **POST请求**：用于提交数据，参数包含在请求体中。
- 示例（使用Fetch API）：

```
// GET请求
fetch('/api/data')
  .then(response => response.json())
  .then(data => console.log(data));

// POST请求
fetch('/api/data', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ key: 'value' })
})
  .then(response => response.json())
  .then(data => console.log(data));
```

## 12. Flask 路由定义

- 使用Flask框架定义路由和处理函数。
  - 示例：

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/url/url1', methods=['POST', 'GET'])
def myfunc():
    if request.method == 'POST':
        # 处理POST请求
        pass
    else:
        # 处理GET请求
        pass

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

## 13. Flask模板使用

- **模板文件**：HTML文件，包含模板变量和表达式。
- **模板变量定义**：使用`{{ }}`语法。
- **安全渲染**：使用`{{ variable|safe }}`。
- **渲染模板**：使用`render_template`函数。

- 示例：

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    data = {'name': 'Flask'}
    return render_template('index.html', data=data)

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

- HTML模板文件`index.html`：

```
<!DOCTYPE html>
<html>
<head>
    <title>Flask Template</title>
</head>
<body>
    <h1>Hello, {{ data.name }}</h1>
</body>
</html>
```

## 14. JS获取网页元素的方法

- **getElementById**：通过元素的ID获取元素。
  - 示例：

```
let element = document.getElementById('myId');
```

- **getElementsByName**：通过元素的name属性获取元素列表。
  - 示例：

```

```

js
let elements = document.getElementsByName('myName');
```

- **getElementsByClassName**：通过元素的类名获取元素列表。
  - 示例：

```
let elements = document.getElementsByClassName('myClass');
```

- **querySelector**：通过CSS选择器获取第一个匹配的元素。
  - 示例：
```

```
let element = document.querySelector('.myClass');
```

- **querySelectorAll**：通过CSS选择器获取所有匹配的元素。
  - 示例：

```
let elements = document.querySelectorAll('.myClass');
```

## 15. PowerBI

- **Filter**：用于查询和过滤数据。
- **Union函数**：用于合并表格，但列的数量、类型、名称必须一致。
  - 示例：

```
UNION(
  SELECTCOLUMNS(Table1, "Column1", Table1[Column1]),
  SELECTCOLUMNS(Table2, "Column1", Table2[Column1])
)
```

## 16. 前后端分离的特点

- **特点**：
  - 前端和后端通过API进行通信，前端专注于用户界面和交互，后端处理业务逻辑和数据存储。
  - 提高开发效率和可维护性，前后端可以独立开发和部署。
  - **不是特点**：使用模板（模板通常用于后端渲染，前后端分离则前端自行处理渲染）。

## 17. 绘图方法（ECharts示例）

- **ECharts初始化和配置**：
  - 示例：

```
<!DOCTYPE html>
<html>
<head>
  <script
src="https://cdn.jsdelivr.net/npm/echarts/dist/echarts.min.js"></script>
</head>
<body>
  <div id="main" style="width: 600px;height:400px;"></div>
  <script>
    // 初始化图表实例
    var myChart = echarts.init(document.getElementById('main'));

    // 配置项
    var option = {
      title: {
        text: 'ECharts 示例'
      },
      tooltip: {},
      legend: {
        data:['销量']
      },
```

```
      xAxis: {
        data: ["衬衫","羊毛衫","雪纺衫","裤子","高跟鞋","袜子"]
      },
      yAxis: {},
      series: [{
        name: '销量',
        type: 'bar',
        data: [5, 20, 36, 10, 10, 20]
      }]
    };

    // 使用配置项生成图表
    myChart.setOption(option);
  </script>
</body>
</html>
```

- **初始化图表实例**：`echarts.init`函数
- **配置项**：包含`title`（标题）、`tooltip`（提示框）、`legend`（图例）、`xAxis`（x轴）、`yAxis`（y轴）、`series`（数据系列）

这样，我们就详细介绍了每一点的内容和示例。希望对你有所帮助！