总设计

页面总体由几部分组成：

1. Navigator导航栏部分
2. SlideShow 幻灯片模块
3. Main 核心内容部分

…

# Navigator架构设计

页面上的导航栏，一个navigatorLoader对象作为核心对象，之后通过navigatorCell对象声名导航格，每一个导航格的声明占一篇代码，统一写到一个文件夹里好维护。一篇声明代码的标准格式如下：

// navigator-cell-xxx.ts

const navigatorCell = new NavigatorCell("content", [], function () {  
 console.log("这里是回调函数触发事件");  
})  
  
export default navigatorCell;

## Navigator-loader.ts

创建一个类用于提供多种管理导航格的方法，例如添加，删除，修改，查看等等，维护一个列表用于储存当前正在被管理的导航格对象，从而动态生成导航。

动态生成导航需要根据添加的项异步刷新导航（setTimeout(xxx,0)）

## Navigator-cell.ts

每一个导航格的具体单元的类，这个类的实例对象表示一个导航格。这个类的实例对象要维护以下信息：导航格的内容，导航格的二级导航，导航格的响应回调函数

## Navigator-adapter.ts

作为将navigatorLoader对象显示在html文档中的中介而存在，是一个显示的适配器，主要根据html文档需要调整输出策略，需要维护navigatorLoader实例对象，提供一个display()方法

# SlideShow

在主页面上显示的图片轮播

此处注意，所有图片轮播图片必须预加载

## SlideShow.js

标准图片轮播提供类

# Main模块

需要考虑两点：跳转到页面之后会根据所选中的人物再跳转。

一个HTML页面有多个<template>之后根据当前页面需要显示的东西实现显示和隐藏

多个同样的页面，例如某人的详细信息，应该用类接管

## 简介：

无需使用JS生成，直接使用HTML和CSS简易布局即可

## 社区荣誉：

给定一个空页面即可

## 网格管理

贴张图片就OK

需要图片解析：图像解析算法流程如下：

输入：一个png图像用于解析，图像需求如下：图像是一个地图，每一个地区是一个区块，每一个区块都只有一个颜色，背景为透明。一个json文件用于记录一个颜色块和其信息的对应关系，包含name，desc等属性

1. 使用offscreencanvas载入图像，将图像大小固定在内容容器之内且不失真，使用2d上下文
2. 使用像素处理
3. 遍历像素，根据json文件描述对各个块进行分类，将位置信息存入数组
4. 通过最大最小值算出数组边界值，确定区块范围
5. 为每一个颜色区块分别建立一个canvas，canvas尺寸为区块大小，并为其设置一个id，id为当前颜色区块对应信息关系中的name属性
6. 将建成的canvas放入容器中，使用容器内的绝对定位，定位位置为区块的左上方位置，并使用CSS将canvas中透明的部分修剪掉
7. 在其他模块中通过获取id的方法来获取对应的区块

注意：这里一定要显示一个让用户等待的标识

Svg矢量图解析流程如下：

输入：

* 一个 SVG 地图，地图中每个社区由一个 <g> 组表示，该 <g> 组内包含两部分：
  1. 一组图案路径（不规则图形），通常由一个嵌套 <g> 内的 <path> 元素构成
  2. 一行文本（社区名称），由另一嵌套 <g> 内的 <text> 元素构成
* 输出数据将存储到一个 Map 数据结构中，键为社区名称，值为该社区对应的 <g> 元素及其解析后的路径数据（包括图案路径和边缘路径）。

算法流程：

1. 清空已有数据
   * 读取 SVG 根元素（属性 svgElement）。
   * 清空该类中用来存储解析结果的数组（this.items）。
2. 遍历所有社区组
   * 使用 querySelectorAll('svg > g') 遍历 SVG 根元素直接子元素，每个 <g> 表示一个社区。
   * 对每个社区组：
     + 从中取出第一个 <g>（作为图案路径所在的容器，记为 pathGroup）。
     + 取出第二个 <g>（作为文本所在的容器，记为 textGroup）。
3. 提取路径与文本数据
   * 在 pathGroup 内：
     + 选择第一个 <path> 元素作为图案路径（patternPath）。
     + 选择第二个 <path> 元素作为边缘路径（edgePath）。
   * 在 textGroup 内：
     + 选择 <text> 元素，获取其文本内容作为社区名称，并去除多余的空格。
4. 解析路径数据
   * 对于 patternPath 和 edgePath，提取其 d 属性字符串。
   * 调用辅助方法 parsePathData(d: string): PathOperation[]，将路径字符串解析为一组路径操作指令数组。
5. 存储解析结果
   * 如果提取的社区名称非空，则创建一个社区数据对象：
     + 键：社区名称。
     + 值：包含三个部分：整个 <g> 组、解析后的图案路径数据和边缘路径数据。
   * 将此社区数据对象存入 this.items 数组（或存入一个 Map 数据结构，如 name2Map），以便后续通过社区名称获取对应社区的 SVG <g> 元素。
6. 结束流程
   * 遍历完成后，所有社区数据均已解析并存储在 this.items 中，其他模块可以通过获取对应的社区名称（键）来快速查找其 SVG 图案数据。
7. 用户等待标识
   * 在调用解析方法之前或解析过程中，通过显示加载动画或提示信息，告知用户正在进行地图解析，解析完成后隐藏等待标识。

解析说明：

* 解析路径字符串  
  使用正则表达式匹配每个路径命令字母及其后续数值参数，将它们按顺序转换为一个 PathOperation 数组，确保每个操作包含命令字母和对应的参数数组。这一步骤为后续计算边界、绘制路径等操作提供数据支持。
* 分离路径和文本  
  根据 SVG 的标准结构，每个社区 <g> 中分成两个部分：图案路径部分与文本部分。解析时，分开获取二者的数据，可确保社区名称与图案数据一一对应，从而实现最终存储与调用的准确性。
* 数据存储  
  将每个社区的名称（经过 trim 处理）作为键，将整个 <g> 元素及解析后路径数据作为值，存入集合中（例如 Map<string, SVGElement> 或 Array<MapDriverReturnType>）。这样设计方便其他模块通过社区名称直接定位到对应的 SVG 图案数据，增强了数据重用和维护性。
* 用户提示  
  在解析过程中，因为 SVG 数据量可能较大，会有一定的处理延时，因此需在解析开始前提供等待提示，并在解析结束后取消提示，提升用户体验。

## 两委成员：

需要构建两个页面，分别显示所有信息和详情信息，一个页面就是一个类，每个类需要对应的template

详情信息中给定一个返回按钮

注册一个container元素，用于作为多页面容器，每一个页面必须要给定一个render()方法用于渲染至父元素，每一个页面必须实现SubPage接口，此接口定义一个render()方法。

Html：一堆template和一个container

JS：

没个两委的具体信息可以根据一个或多个json文件给出

Commissioner类：相当于DAO类，维护每一个委员的个人信息，有name，image，info等属性

CommissionerListEvent类：是CommissionerList类的父类，用于编写事件

CommissionerList类：用于维护两委列表页面的信息，需要基本属性：commissionerList: Commissioner[]，currentPage:number，itemInAPage:number，totalPage:number等，需要维护基本方法render();

CommissionerDetails类：用于维护详情信息页面，需要维护基本方法render();

## 为民服务：

空页面即可

## 事项清单：

和两委一样

## 证据出具：

和两委一样

给出网络接口以获取证据出具文件，html格式即可

# 兼容性分析

CSS：变量，::sloted，等。总结：CSS3，CSS4的极少数模块。仅支持CSS3的浏览器可能不正常运行。

JS：

viewTransition API，SVG API，customElement，