# 目录

[摘 要 错误!未定义书签。](#_Toc22869)

[Abstract 错误!未定义书签。](#_Toc6240)

[目 录 错误!未定义书签。](#_Toc3951)

[第1章 概 述 错误!未定义书签。](#_Toc11035)

[1.1 Web开发背景 **错误!未定义书签。**](#_Toc26730)

[1.1.1 Web开发的主要方向](#_Toc21259) **[错误!未定义书签。](#_Toc21259)**

[1.1.2 2d绘图在Web开发中的地位 **错误!未定义书签。**](#_Toc21259)

[1.2 本类库的优势 5](#_Toc5498)

[1.2.1 现有的2d绘图类库](#_Toc21259) **[错误!未定义书签。](#_Toc21259)**

[1.2.2 本类库的设计主旨](#_Toc21259) **[错误!未定义书签。](#_Toc21259)**

[1.3 功能简介 7](#_Toc5633)

[第2章 2d计算机图形学相关技术 8](#_Toc6456)

[2.1 语言介绍 8](#_Toc27995)

[2.1.1 JavaScript语言介绍 8](#_Toc21259)

[2.1.2 TypeScript语言介绍 8](#_Toc18787)

[2.2 开发及运行环境 9](#_Toc18193)

[2.2.1 JetBrains PhpStorm介绍 9](#_Toc26392)

[2.2.2 Chrome Canary介绍 9](#_Toc19008)

[2.2.3 NodeJS介绍 9](#_Toc27583)

[2.2.4 TypeScript Compiler介绍 9](#_Toc27583)

[2.3 计算机图形学2d部分的原理介绍 11](#_Toc18193)

[2.3.1 二维坐标系统 11](#_Toc26392)

[2.3.2 二维图形学变换 11](#_Toc19008)

[2.3.3 线性插值算法（点和颜色） 11](#_Toc27583)

[2.3.4 贝塞尔曲线原理 11](#_Toc27583)

[2.3.5 填充算法 11](#_Toc27583)

[2.3.5.1 奇偶原则 11](#_Toc27583)

[2.3.5.2 非零绕数 11](#_Toc27583)

[2.3.6 剪裁算法及锯齿直线渲染 11](#_Toc27583)

[2.4 js对2d计算机图形学的实现 13](#_Toc18193)

[2.4.1 图元的实现 13](#_Toc26392)

[2.4.2 图形学转换的实现 13](#_Toc19008)

[2.4.3 颜色的实现 13](#_Toc27583)

[2.4.4 图像的引入 13](#_Toc27583)

[2.5 本章小结 15](#_Toc18193)

[第3章 类库总体设计 16](#_Toc10317)

[3.1 类库的体系结构设计 16](#_Toc18872)

[3.1.1 类库功能划分 1**错误!未定义书签。**](#_Toc26392)

[3.1.2 类库继承和依赖的逻辑 16](#_Toc19008)

[3.2 类库的功能模块设计 **错误!未定义书签。**](#_Toc6908)

[3.2.1 总体模块 **错误!未定义书签。**](#_Toc22642)

[3.2.2 图元 **错误!未定义书签。**](#_Toc22642)

[3.2.2.1 图元封装 1**错误!未定义书签。**](#_Toc27583)

[3.2.2.2 属性封装 1**错误!未定义书签。**](#_Toc27583)

[3.2.2.3 事件交互 1**错误!未定义书签。**](#_Toc27583)

[3.2.3 填充类封装 **错误!未定义书签。**](#_Toc515)

[3.2.3.1颜色类 22](#_Toc27583)

[3.2.3.2图案类 22](#_Toc27583)

[3.2.3.3渐变类 22](#_Toc27583)

[3.2.4 像素图像处理 **错误!未定义书签。**](#_Toc515)

[3.2.5 动画封装 22](#_Toc515)

[3.3 本章小结 23](#_Toc6908)

[第4章 类库详细设计与实现 22](#_Toc10147)

[4.1 图元绘制面向对象化实现 22](#_Toc14137)

[4.1.1 图形基类封装和继承关系 22](#_Toc22642)

[4.1.2 直线绘制封装 22](#_Toc22642)

[4.1.3 贝塞尔曲线封装 22](#_Toc515)

[4.1.3.1 二次贝塞尔 22](#_Toc27583)

[4.1.3.2 三次贝塞尔 22](#_Toc27583)

[4.1.4 圆的封装 22](#_Toc22642)

[4.1.5 椭圆封装 22](#_Toc515)

[4.1.6 矩形封装 22](#_Toc22642)

[4.1.7 圆角矩形封装 22](#_Toc515)

[4.1.8 文字封装 22](#_Toc515)

[4.1.9 自定义图元封装 22](#_Toc22642)

[4.1.10 文字文本属性封装 22](#_Toc22642)

[4.1.11 阴影边框属性封装 22](#_Toc22642)

[4.1.12 自定义图元封装 22](#_Toc22642)

[4.2 填充类型类封装实现 26](#_Toc4293)

[4.2.1 颜色类 26](#_Toc22642)

[4.2.1.1 英文颜色关键字 22](#_Toc27583)

[4.2.1.2 RGB，RGBA颜色封装 22](#_Toc27583)

[4.2.1.3 HSL，HSLA颜色封装 22](#_Toc27583)

[4.2.2 图案类 26](#_Toc22642)

[4.2.3 渐变类 26](#_Toc22642)

[4.2.3.1 线性渐变 22](#_Toc27583)

[4.2.3.2 辐射渐变 22](#_Toc27583)

[4.2.3.3 角度渐变 22](#_Toc27583)

[4.3 像素和图像处理实现 28](#_Toc6191)

[4.3.1 像素处理 28](#_Toc22642)

[4.3.2 图像及视频处理 28](#_Toc515)

[4.4 绘图动画封装及优化实现 31](#_Toc6191)

[4.4.1 动画依赖类 31](#_Toc22642)

[4.4.2 动画参数 31](#_Toc22642)

[4.4.3 动画运行时属性 31](#_Toc515)

[4.4.4 动画操作方法 31](#_Toc515)

[4.4.5 动画时间线 31](#_Toc515)

[4.5 运行时事件 34](#_Toc6191)

[4.5.1 画布整体运行时事件 31](#_Toc22642)

[4.5.2 图元运行时事件 31](#_Toc22642)

[4.5.2 动画运行时事件 31](#_Toc22642)

[4.6 本章小结 34](#_Toc6191)

[第5章 类库应用示例 35](#_Toc6998)

[5.1 类库的使用场景，方法，效果 **错误!未定义书签。**](#_Toc14857)

[5.1.1 类库使用场景 **错误!未定义书签。**](#_Toc22642)

[5.1.2 面对各种问题的使用方法 **错误!未定义书签。**](#_Toc515)

[5.1.2.1 建立场景 **错误!未定义书签。**](#_Toc27583)

[5.1.2.2 绘制对象 **错误!未定义书签。**](#_Toc27583)

[5.1.2.3 设置属性 **错误!未定义书签。**](#_Toc27583)

[5.1.2.4 添加交互 **错误!未定义书签。**](#_Toc27583)

[5.1.2.5 添加动画 **错误!未定义书签。**](#_Toc27583)

[5.1.3 使用的效果 **错误!未定义书签。**](#_Toc22642)

[5.2 类库的扩展插件写法 **错误!未定义书签。**](#_Toc19144)

[5.2.1 自定义图元插件的编写 **错误!未定义书签。**](#_Toc17464)

[5.2.2 自定义颜色的编写 **错误!未定义书签。**](#_Toc15956)

[5.3 本章小结 42](#_Toc2233)

[结 论 错误!未定义书签。](#_Toc5015)

[参考文献 错误!未定义书签。](#_Toc28629)

[致 谢 错误!未定义书签。](#_Toc28629)

# 目录（真）

[目录 1](#_Toc160127826)

[目录（真） 5](#_Toc160127827)

[第1章 概 述 7](#_Toc160127828)

[1.1 类库的体系结构设计 7](#_Toc160127829)

[1.1.1 Web开发的主要方向 7](#_Toc160127830)

[1.1.2 2d绘图在Web开发中的地位 7](#_Toc160127831)

[1.2 本类库的优势 8](#_Toc160127832)

[1.2.1 现有的2d绘图类库 8](#_Toc160127833)

[1.2.2 本类库的设计主旨 9](#_Toc160127834)

[1.3 功能简介 9](#_Toc160127835)

[第2章 开发环境 12](#_Toc160127836)

[2.1 语言介绍 12](#_Toc160127837)

[2.1.1 JavaScript语言介绍 12](#_Toc160127838)

[2.1.2 TypeScript语言介绍 12](#_Toc160127839)

[2.2 开发及运行环境 13](#_Toc160127840)

[2.2.1 JetBrains PhpStorm介绍 13](#_Toc160127841)

[2.2.2 Chrome Canary介绍 13](#_Toc160127842)

[2.2.3 Node.js介绍 14](#_Toc160127843)

[2.2.4 TypeScript Compiler介绍 15](#_Toc160127844)

[第3章 2D图形学原理及实现 16](#_Toc160127845)

[3.1 计算机图形学2d部分的原理介绍 16](#_Toc160127846)

[3.1.1 坐标系统 16](#_Toc160127847)

[3.1.2 齐次坐标 16](#_Toc160127848)

[3.1.3 二维图形学变换 17](#_Toc160127849)

[3.1.4 线性插值算法 20](#_Toc160127850)

[3.1.5 贝塞尔曲线原理 22](#_Toc160127851)

[3.1.6 绘图流水线 23](#_Toc160127852)

[3.1.7 剪裁及光栅化 23](#_Toc160127853)

[3.1.8 填充算法 24](#_Toc160127854)

[3.2 js对2d计算机图形学的实现 25](#_Toc160127855)

[3.2.1 图元的实现 25](#_Toc160127856)

[3.2.2 图形学转换的实现 26](#_Toc160127857)

[3.2.3 颜色的实现 27](#_Toc160127858)

[3.2.4 图像的引入 28](#_Toc160127859)

[3.3 本章小结 28](#_Toc160127860)

[第4章 类库总体设计 30](#_Toc160127861)

[4.1 类库的体系结构设计 30](#_Toc160127862)

[4.1.1 类库功能划分及封装设计 30](#_Toc160127863)

[4.1.2 类库继承和依赖的逻辑 33](#_Toc160127864)

[4.2 类库的功能模块设计 36](#_Toc160127865)

[4.2.1 总体模块 36](#_Toc160127866)

[4.2.2 图元 37](#_Toc160127867)

[4.2.3 填充类封装 39](#_Toc160127868)

[4.2.4 像素图像处理 41](#_Toc160127869)

[4.2.5 动画封装 41](#_Toc160127870)

[4.3本章小结 42](#_Toc160127871)

[参考文献 44](#_Toc160127872)

# 第1章 概 述

## 1.1 类库的体系结构设计

### 1.1.1 Web开发的主要方向

在当前快速发展的时代，多数产业正向高新科技靠拢，尤其是互联网领域。网络开发，特别是网络前端开发，已成为至关重要的一环。众所周知，2015年标志着网络开发的一个转折点，那一年前后技术经历了重大的改革和发展。前端开发从过去单一、局限的范畴转变为全面、复杂的领域。随着底层技术的突破，涌现出许多新兴的编程思想。企业在开发过程中也逐步更新观念，摒弃旧有的软件编程理念，转而采纳新的理念。其中最显著的变化是前端开发从客户端转向服务器端。随着本地JavaScript运行时环境的发展和优化，以及大量优秀的第三方包的加入，前端开发的架构逐渐从以HTML和CSS为主转变为以Node.js和其他框架为核心的后端架构。同时，底层技术也在不断迭代，2015年之后发布的JavaScript ES2015版本，将原本复杂且难以融入软件工程项目的JavaScript脚本语言转变为一个支持面向对象、模块化、异步多线程的编程语言，更好地利用客户端资源。这促使Web的整体发展从过去以静态页面为辅、服务器控制动态效果为主的传统架构，转变为实时与服务器通信，前端内容可以从后端完全分离的新型架构。大量框架的引入，其扩展性使得更多第三方插件得到更广泛的传播和应用，提升了代码的可复用性，甚至少量代码就能完成一个简单的应用。

### 1.1.2 2d绘图在Web开发中的地位

随着前端底层逻辑的不断发展，前端对客户端资源的使用及其安全性也在增强。最新一批的JavaScript API主要从以下几个方面控制客户机：以内部或外部硬件管理为主的控制，例如对外部硬件及其接口的控制，GPU的控制，音频输出的控制，虚拟存储及虚拟运行汇编语言，以及物理感应器的控制。此外，还通过多种手段对系统进行控制，例如与系统的其他软件交互，虚拟文件系统等。在这些API中，Gamepad API、USB API、XR API分别用于与游戏手柄、USB设备以及VR、AR等设备进行连接和交互。对GPU的控制主要通过WebGPU API，次要的权限通过WebGL和WebGL2 API。音频节点和WebAssembly则是人们熟悉的技术。

传统前端通常专注于UI界面的制作，而现代前端的UI界面仍然是非常重要的组成部分。追求美观大气的界面和满足用户需求的图形系统，在软件开发过程中至关重要。对优秀页面和效果的追求推动了前端计算机图形学的发展，使得相关图形学API受到重视。目前，市面上常见的动画制作和页面图形效果优化方法主要包括：利用CSS的动画和渐变模块，利用JS的多种渐变和动画API，如Animation API，以及利用2D绘图和SVG等绘图API来优化网页的2D动画效果，此外还有三种3D绘图API。

仔细分析这些动画制作方式，可以发现CSS动画和渐变实际上是通过设置属性在内部运行的。原生和框架中封装好的动画API也是通过设置属性在内部运行的。SVG则有专用的SMIL动画语言以及浏览器的支持和接口。而2D绘图API和3D绘图API没有直接支持动画运行，在开发过程中通常需要手写动画效果。这两种API是基于面向过程的，在管理和开发大型应用时表现复杂，且无法直接接入DOM与用户交互。因此，开发过程中的需求与现有的代码工具之间的不匹配导致了矛盾，即如何将这两种API转化为面向对象的、可与用户交互的、支持动画的工具。

## 1.2 本类库的优势

### 1.2.1 现有的2d绘图类库

面向对象语言中，将部分面向过程的底层API转化为面向对象的形式，是一个经久不衰的讨论话题。语言设计初期，必须考虑到语言功能的单一性、可复用性，以及API之间的交集复杂度，这些因素共同影响着耦合性的高低。然而，这样的设计往往会牺牲使用便利性。市面上已有的一些2D绘图类库如下所示：

1、Two.js：Two.js 是一个现代平面动画绘制类库，支持 SVG、Canvas 和 WebGL 多种渲染方式。它的特色在于提供了一个简单而强大的对象模型，便于开发者创建、管理图形对象，并进行变换、分组、嵌套和动画等操作。Two.js 还能将 SVG 文件导入其场景中，增加了使用的灵活性。

2、Konva：Konva 是一个适用于桌面和移动应用的 Canvas 2D 类库，提供了面向对象的 API，使开发者能够创建和操作图形、文本、图片、视频等。Konva 的优势在于其高性能渲染、事件处理、动画、滤镜、裁剪和拖放功能，以及对多个 Canvas 层的支持。Konva 还提供了与前端框架如 React、Vue 和 Angular 集成的扩展模块。

3、LeaferJS：LeaferJS 是一个功能强大的 Canvas 2D 类库，能够快速创建大量图形，并可与 AI 结合进行绘图。LeaferJS 提供了灵活的图形系统，允许开发者自定义图形的属性、样式、动画和事件。它还包含了渐变、阴影、混合、裁剪、剪贴板、导出等高级功能。

4、Paper.js：Paper.js 是基于 HTML5 Canvas 元素的矢量图形类库，提供了类似于 SVG 的 API，使得以对象方式操作图形成为可能。Paper.js 强大的数学引擎支持各种图形学操作，如路径、曲线、变换、布尔运算和几何运算等。此外，Paper.js 还提供了交互式脚本环境，允许开发者在浏览器中直接编写和运行代码。

### 1.2.2 本类库的设计主旨

尽管上述类库功能完善且成熟，但存在一些问题。

首先，它们并未充分实现与原生语法的对接。类库设计应尽可能与原生语法紧密对接，以便用户能更快速地学习和应用。类库设计者需考虑功能、性能、可读性、可维护性、可扩展性和兼容性等多个方面。其中，与原生语法的对接是一个关键因素，意味着类库的 API 应遵循原生语言的语法、风格和习惯，提供一致且自然的编程体验。

其次，这些类库在设计时过于注重抽象出更高层次的代码层级，而忽视了操作的原子性和可复用性。类库的 API 应提供简单、完整、独立和通用的操作，以确保编程体验的一致性和效率。保持原子性有助于避免数据不一致、资源泄漏、状态不同步等潜在错误和异常，同时增强类库的可组合性。保持可复用性则有助于扩展类库的功能和适用范围，因为可以将操作应用于不同的对象和数据上，实现多样的目的和效果。

综上所述，本类库的设计宗旨在于最大程度地与原生语法对接，并在面向对象化的基础上，保持操作的原子性和可复用性。同时，提供交互和动画功能，实现低复杂度封装，保持易用性和可扩展性。

综上所述，本类库设计的主旨就在于实现最大程度的和原生语法对接，以及在面向对象化的基础上，保持操作的原子性和可复用性，同时提供交互和动画功能。实现低复杂度封装且保持其易用性，可扩展性。

## 1.3 功能简介

本类库是一个JavaScript类库，用于在浏览器中绘制和操作2D图形。它的核心功能包括绘图及相关功能、必要的数学运算和其他工具类。按照设计主旨，类库仅封装最基础的代码以保持扩展性。此外，类库还包括与用户交互所需的事件类封装。其他功能则以面向对象的方式封装，不进行过多修改，并提供动画功能。详细模块划分如下：

* 图形学转换和数学实现： 负责图形变换处理，如平移、旋转、缩放等，使用数学算法确保图形正确显示。例如，计算旋转90度的矩形的新顶点坐标。
* 图形绘制面向对象化： 采用面向对象方法绘制图形，便于代码维护和扩展。创建图形对象，设置属性如颜色、线宽，并绘制。例如，绘制一个蓝色矩形。此模块包含三个子模块：
  + 图形样式：定义图形的样式，例如定义线宽、填充颜色、边框颜色等图形样式。
  + 文字样式：管理文本样式，如字体、大小、颜色。
  + 图形内操作：处理组合、裁剪等图形内部元素。
* 颜色封装： 管理图形颜色属性，提高可定制性。用户可设置填充颜色、边框颜色等。例如，设定矩形填充为红色，边框为黑色。
* 像素和图像处理： 处理像素级图像操作，如滤镜、亮度调整。用户可应用滤镜或调整色调、饱和度。例如，应用黑白滤镜或增加对比度。
* 动画： 实现图形平滑过渡和动态效果。创建平移、缩放、旋转动画。例如，制作矩形左至右的渐变动画。
* DOM事件： 处理图形相关DOM事件，如点击、拖动。用户可通过点击选择图形或拖动改变位置。例如，点击矩形更改颜色。如需进一步修改或调整，请告知。

具体功能模块图如图1.3.1。

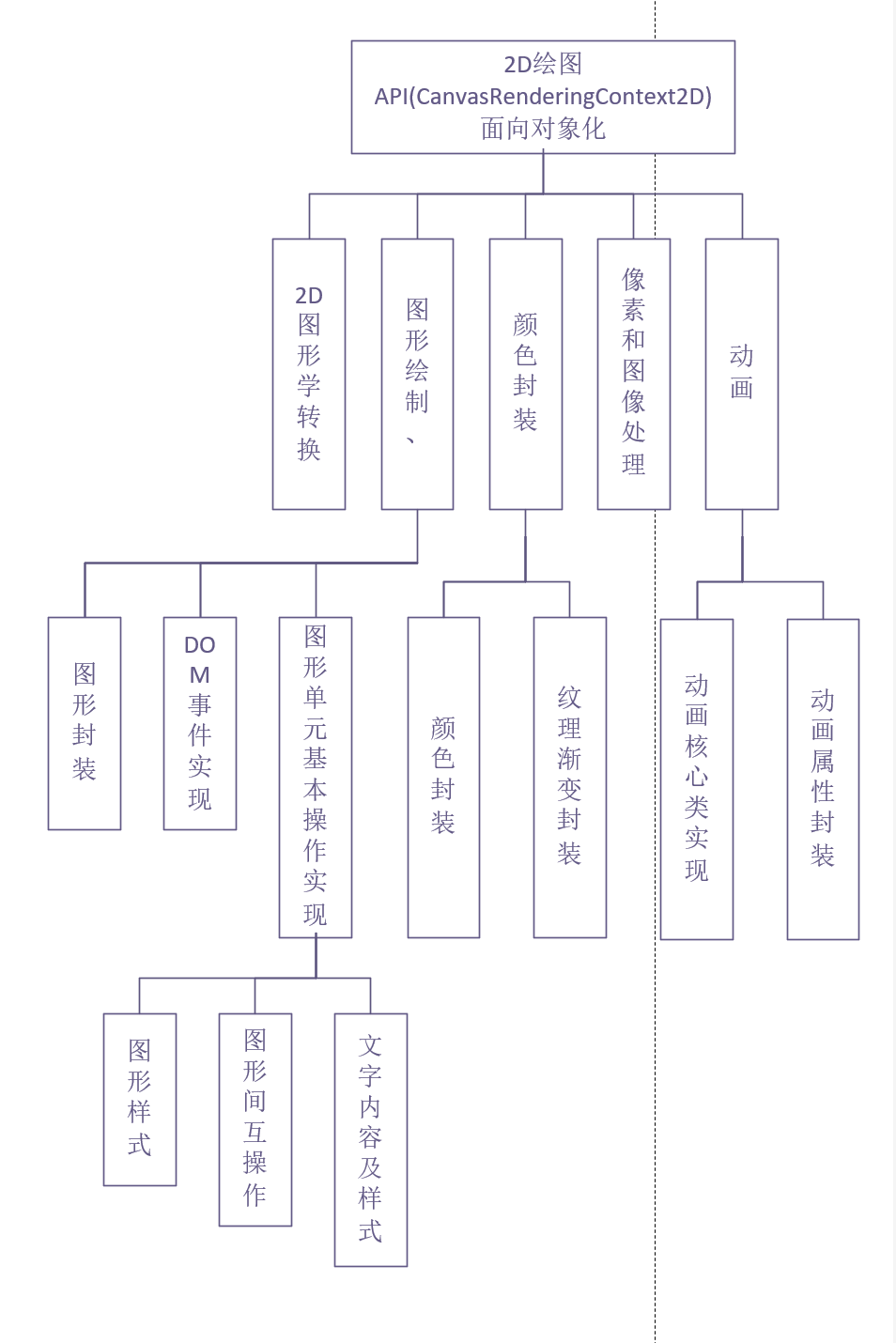


图1.3.1 功能模块图

# 第2章 开发环境

## 2.1 语言介绍

### 2.1.1 JavaScript语言介绍

JavaScript是一门由网景（Netscape）公司在1995年开发的客户端脚本语言，最初叫Mocha，后来改名为LiveScript，最后为了借助Java的热度而改名为JavaScript。它可以在浏览器中运行，用于处理简单的数据验证和交互。JavaScript在网景的Netscape Navigator浏览器中首次发布，后来又在Netscape Navigator 3中更新了版本。

为了使这门语言成为标准，网景将它提交给了ECMA（欧洲计算机制造协会）。然而，"JavaScript"这个名字实际上是Sun Microsystems公司（现在的Oracle）的注册商标，用来特指网景（现在的Mozilla）对这门语言的实现。由于商标的冲突，这门语言的标准版本被命名为ECMAScript，由TC39维护。

TC39在2008年启动了一个名为Harmony的项目，旨在制定下一代的ECMAScript标准，也就是ES.next。Harmony的目标是使JavaScript成为一种更适合开发复杂应用、库和代码生成器的语言，同时保持语言的简单、易用和互操作性。Harmony的特性集在2011年5月基本确定，经过多次的讨论、修改和测试，最终在2015年6月发布了正式的ES6规范。ES6是JavaScript语言的一个重大更新，它包含了许多新的特性和改进，如模块化、解构赋值、展开运算符、类等。ES6的规范是JavaScript语言的一个里程碑，它展示了语言的进化和创新，也为未来的版本奠定了基础。

ES6规范将ECMAScript分为三个部分：虚拟机、语言和标准库，以便更清晰地展示新的内容。ES6规范也更新了一些对象的命名规则，区分了标准对象和异质对象，以支持对象的自托管和互操作性。ES6规范还引入了一些新的语义种类，如静态语义和早期错误，以及命名算法和多态算法，以适应复杂的语法特性。ES6规范还提出了一些重要的语言特性，如Realms、Jobs、Proxies和元对象编程(MOP)，以支持多个全局命名空间、事件派发、虚拟对象和对象能力的隔离等功能。

### 2.1.2 TypeScript语言介绍

TypeScript 是一种由微软开发和维护的开源编程语言。它是 JavaScript 的严格语法超集，提供了可选的静态类型检查。

TypeScript 的起源可以追溯到微软和客户在开发大型应用程序时遇到的 JavaScript 缺点。即随着使用 JavaScript 编写的程序规模、范围和复杂性呈指数级增长，JavaScript 语言在表达代码之间关系方面的能力并没有相应提高。再加上 JavaScript 的奇特运行时语义，这使得在大规模情况下开发 JavaScript 变得难以管理。程序员最常犯的错误之一就是类型错误：即将一个值用在期望另一种值的地方。这可能是由于简单的拼写错误、对库的 API 表面理解不足、对运行时行为的错误假设或其他原因造成的。

TypeScript 的目标是成为 JavaScript 程序的静态类型检查器，即在代码运行之前运行并确保程序的类型正确。它旨在帮助开发者处理大型应用程序，同时可以将 TypeScript 转译成 JavaScript。由于 TypeScript 是 JavaScript 的严格语法超集，因此任何现有的 JavaScript 程序都是合法的 TypeScript 程序。

## 2.2 开发及运行环境

### 2.2.1 JetBrains PhpStorm介绍

PhpStorm 是一款商业性质的跨平台PHP IDE，构建在JetBrains的IntelliJ IDEA平台之上。它不仅提供了对PHP、HTML和JavaScript的编辑器，还具备实时代码分析、错误预防以及自动重构PHP和JavaScript代码的功能。PhpStorm的用户友好性使得用户能够编写整洁且易于维护的代码。

PhpStorm同时具有多种功能：

智能编码助手：PhpStorm 以其智能编码辅助功能而闻名。这一功能确保你遵循编码标准，提供相关建议，并在你输入代码时自动完成。

集成工具：该IDE与数据库、版本控制系统、远程部署等工具集成，简化了Web开发流程。

调试和测试：PhpStorm 集成了 Xdebug 和 Zend Debugger，支持 PHPUnit 进行单元测试。你可以在同一环境中开发、测试和调试代码。

可定制性：你可以通过插件、主题和自定义设置来调整 PhpStorm，使其适应不同的工作流程。

跨平台：PhpStorm 可在 Windows、macOS 和 Linux 上使用，无论你使用哪个操作系统都能得心应手。

PhpStorm 是一个强大且灵活的工具，它将帮助你在 Web 开发中事半功倍。

### 2.2.2 Chrome Canary介绍

Chrome Canary 是一款专为经验丰富的开发者设计的浏览器，每晚都会更新。Chrome Canary 是 Google Chrome 浏览器的一个分支版本，旨在为开发人员提供最新的功能和实验性特性。它的名字“Canary”源自矿工使用小鸟来检测瓦斯泄漏的传统做法。类似地，Chrome Canary 作为 Chrome 浏览器的“试验场”，允许开发人员测试新功能、修复错误并提供反馈。

以下是 Chrome Canary 的一些主要特色：

每日更新：Chrome Canary 每晚都会自动更新，以包含最新的代码和功能。这使开发人员能够第一时间体验到最新的改进和实验性功能。

开发者工具：Chrome Canary 集成了强大的开发者工具，包括审查元素、调试 JavaScript、性能分析等。这些工具有助于开发人员构建更好的网站和应用程序。

实验性功能：Chrome Canary 提供了一些实验性的功能，这些功能可能尚未在稳定版的 Chrome 中发布。开发人员可以在 Canary 中尝试这些功能，以便提供反馈和改进。

网页设置：Chrome Canary 具有人性化的网页设置，使用户能够自定义浏览器的行为和外观。

无痕浏览：与 Chrome 浏览器一样，Chrome Canary 也支持无痕浏览模式，确保用户的隐私和安全。

### 2.2.3 Node.js介绍

Node.js 是一种开源与跨平台的 JavaScript 的运行环境，能够使得 JavaScript 脱离浏览器运行。它允许用户通过 JavaScript 和一系列模块来编写服务器端应用和网络相关的应用。

Node.js 于 2009 年 由 Ryan Dahl 开发，它的设计灵感源自 Flickr 的一款上传进度栏。在上传过程中，浏览器并不清楚有多少文件已经发送到服务器，除非向服务器进行查询。为了更简便地处理这个问题，Ryan Dahl创造了 Node.js，将 JavaScript 从浏览器中解放出来，让它能够运行在服务器端。

事件驱动和非阻塞模型：Node.js 使用事件驱动和非阻塞的接口，可用于编写高并发状态下的程序。与其他语言最大的不同之处在于，Node.js 是非阻塞的，多条命令可以同时运行，通过回调函数得知命令已结束运行。同时，Node.js 提供了一系列核心模块，包括文件系统 I/O、网络（HTTP、TCP、UDP、DNS、TLS/SSL 等）、二进制数据流、加密算法等。这些模块的 API 形式简单，降低了编程的复杂度。

Node.js 主要用于编写像 Web 服务器一样的网络应用，支持实时、基于推送的架构。它在数据密集的即时应用程序方面表现出色。

同时Node.js提供了多种第三方模块，其中包括React，Vue，WebPack等，本文所讨论的TypeScript也是其中之一，其编译也是依赖于Node.js以及其他本地端JavaScript运行时来完成，例如Deno。

### 2.2.4 TypeScript Compiler介绍

TypeScript Compiler（简称为 tsc）是一款强大的工具，用于将 TypeScript 代码转换为可在浏览器或服务器上运行的 JavaScript 代码。

TypeScript 是由 Microsoft 开发的一种编程语言，它扩展了 JavaScript 的功能，使其更适合大型项目和团队协作。随着 JavaScript 项目的不断增长，开发人员需要更强大的工具来管理代码、提高可维护性并减少错误。于是，TypeScript 应运而生。

TypeScript 编译器会分析你的代码，检查变量、函数和对象的类型。这有助于在开发过程中捕获潜在的类型错误，提高代码质量。同时，TypeScript 引入了静态类型系统，允许开发人员在编码时指定变量的类型。这有助于减少运行时错误，并提高代码的可读性。TypeScript 编译器将 TypeScript 代码转换为普通的 JavaScript 代码，以便在浏览器或服务器上运行。这使得 TypeScript 成为一个跨平台的解决方案。即为实现兼容性的优先方案。

TypeScript 可以通过TypeScript 编译器编译成纯 JavaScript，适用于构建 Web 应用程序。它提供了强大的类型系统和现代的语法，使开发更高效。许多 Node.js 项目使用 TypeScript 编写，以利用其类型检查和模块化功能。TypeScript 在流行的前端框架（如 Angular、React 和 Vue）中得到广泛应用。它提供了更好的开发体验和更好的代码组织。TypeScript 可用于构建库、工具和插件，以提供更好的 API 和更好的文档。

# 第3章 2D图形学原理及实现

## 3.1 计算机图形学2d部分的原理介绍

### 3.1.1 坐标系统

在广义的仿射空间之中，选定一个特定的点作为参考点，基于此参考点确定基向量，就可以以明确的方式来表示所有点。这种原点和基向量的组合便称之为标架，也被称为坐标系统

为了描述图形，必须首先确定一个称为世界坐标系的合适的二维或三维笛卡儿坐标系。接着通过给出世界坐标系中的位置等几何描述来定义图形中的对象。例如，通过两个端点定义一条直线段，通过一组顶点位置定义一个多边形。

一般，在一个完整的绘图系统中，存在着多个坐标系统，这些坐标系统的原点被称之为基向量，一张完整的图像是由不同种坐标系共同作用的结果。例如一整个绘图系统拥有一个世界坐标系统，里面的对象拥有自己的对象坐标系统等。

### 3.1.2 齐次坐标

向量和点的表示相似，可能导致混淆。考虑由点**P0**和向量v1，v2，v3所定义的标架。通常会把位于(x，y, z)的点P用下面的列矩阵来表示：

其中x，y和z满足：

如果以这种方式来表示点，那么点的表示和向量的表示在形式上是一样的，因为向量w可以表示为：

齐次坐标（homogeneous coordinate）可以克服这个困难，它对三维空间中点和向量的表示都是四维的。在由点**P0**和向量v1，v2，v3所定义的标架中，任一点**P**可以唯一地表示为

那么可以用矩阵乘法把点P形式地表示为

等式右边的四维行矩阵就是点**P**在由点**P0**和向量v1，v2，v3所定义的标架下的齐次坐标表示( homogeneous-coordinate representation)。 等价地，也可以说P的表示是下面的列矩阵:

其次坐标所扩展的一位数被称之为齐次参数，齐次参数是一个非零值，当其次参数为1时，方可不改变坐标值，所以得出：

所以，一般的二维齐次坐标表示可以写为(hx,hy,h)。

### 3.1.3 二维图形学变换

#### 3.1.3.1仿射变换

变换（transformation）本质上是一个函数，它将一个点（或向量）映射到另一个点（或向量）。变换可以表示为函数形式：

上述变换的描述相对抽象。在实际应用中，即使函数f 可以用其他方式描述，也需要对曲线上的每个点执行变换。考虑使用四维齐次坐标的受限变换，点和向量都表示为四元组。对函数f施加限制，可以得到一类有用的变换。最重要的限制是线性条件。函数f是线性函数（ linear function），当且仅当对于任何标量α和β以及任何两个点（或向量）p和q，都有：

线性函数的重要性在于变换的线性组合等于线性组合的变换，因此只需知道 ( p ) 和 ( q ) 的变换，就可以求出它们的线性组合的变换，无需单独计算每个线性组合的变换。

使用点和向量的齐次坐标表示时，线性变换可以视为点（或向量）表示之间的映射，并且总是可以用矩阵乘法表示线性变换，形式如下：

其中u和v分别是变换前后的表示，M是一个方阵。变换的本质是坐标系的修改，导致顶点表示的变化。

使用齐次坐标时，M是一个4x4 矩阵，不会改变表示的第四个分量（w）。矩阵M的形式如下：

类似的，二维空间中使用齐次坐标时，M是一个3x3矩阵：

矩阵中的α为变换的自由度，一切变化从空间变化开始。

#### 3.1.3.2 平移

平移（ translation）变换把点沿着给定的方向移动固定的距离。通常只需要指定一个位移向量d就可以确定一个 平移变换，变换后的点由下式给出：

这个式子适用于对象上所有的点P，这个定义不依赖于任何标架或者表示。

旋转和平移都是刚体变换（ rigid-body transformation）。旋转和平移的任何组合都不能改变对象的形状和体积，它们只能改变对象的位置和方向。

#### 3.1.3.3 缩放

缩放（Scaling）是种仿射变换， 但不是刚体变换，通过缩放变换可以放大或者缩小对象。缩放不一定是均匀的，所以有必要考虑非均匀缩放。

缩放变换有一个不动点，通常是图形系统中的坐标原点。因此，为了确定一个缩放变换，通常应该指定不动点，缩放的方向和缩放因子α。当α> 1时，对象沿着指定的方向伸长；当0≤α< 1时，对象沿着指定的方向缩短。如果a是负的，那么就得到了以不动点为中心沿缩放方向的反射变换。缩放有两个自由度，一个用于设定不动点，一个用于设定各个基向量的分量。变换后的点由下式给出：

#### 3.1.3.4 旋转

 确定旋转（rotation）比确定平移要更困难些，因为需要指定的参数更多。 如图3.1.3.4.1所示在二维平面内围绕原点旋转一个点。

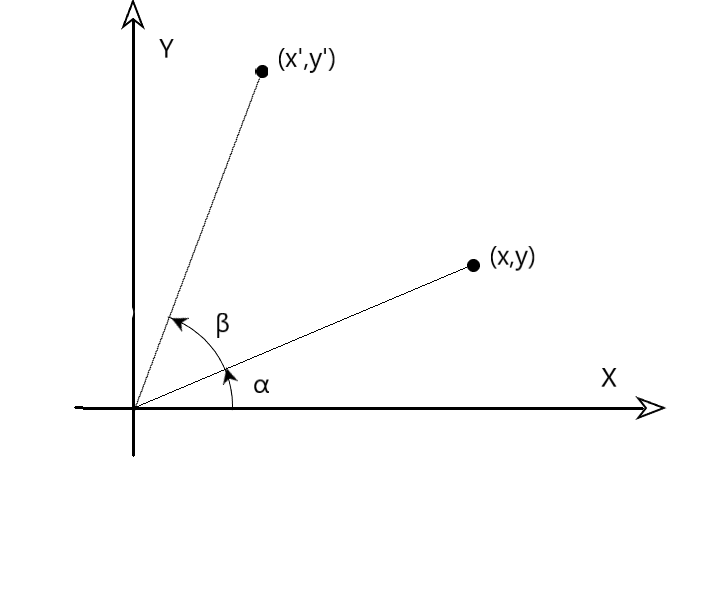


图3.1.3.4.1 二维旋转

在这个坐标下表示为(x，y)的点围绕原点旋转α后位于表示为(x’,y’)的位置。通过把(x，y)和(x’,y’)用极坐标表示，可以得到描述这个旋转的标准方程：

利用两角和的正弦和余弦公式，可得到

上述等式可以写成矩阵的形式

利用上述分析可以定义一般的三维旋转，并且这个定义是不依赖于坐标系的。旋转只有一个自由度，即变换角度。

#### 3.1.3.5 错切

从z轴正向观察位于原点且棱平行于坐标轴的立方体，若顶面向右拉伸、底面向左拉伸，则为沿x方向错切立方体。y和z坐标保持不变，故称此为x错切。角度θ定义x错切，用一个角度θ就可以表征一个x错切。其方程为：

由此可得错切矩阵

#### 3.1.3.6 复合变换

矩阵表达式允许通过计算变换矩阵乘积，将任意的变换序列组成复合变换矩阵（composite transformation matrix）。变换矩阵乘积称为矩阵合并（ concatenation）或复合（composition）。齐次列矩阵表示坐标位置，必须用矩阵前乘以表示任意变换序列。场景中多个位置相同顺序变换，先乘所有变换矩阵形成复合矩阵更高效。如果要对点位置P进行两次变换，变换后位置将用下式计算：

该坐标位置使用矩阵M来变换，而不是单独地先用M1然后用M2来变换。

### 3.1.4 线性插值算法

线性差值算法普遍应用于计算机的各种领域，计算机图形学中也有很多线性差值算法的实际应用。线性差值算法最早来源于参数方程，是一种变体的参数方程，所谓线性差值算法即为将一个事物的两个状态抽象出来，利用一个参数来表示从第一个点到第二个点的过渡过程，在一些其他的参考文献中有时会称之为混合函数（Mix-in），即为将两个不同的状态进行混合。

以仿射空间举例，在一坐标系中，有一点a坐标为(a1,a2)，有一点b坐标为(b1,b2)，设置一点c为a和b的中点，设置一点d为a和b连线上的一点，设置一点e为ab射线上的一点，其关系如下图所示：

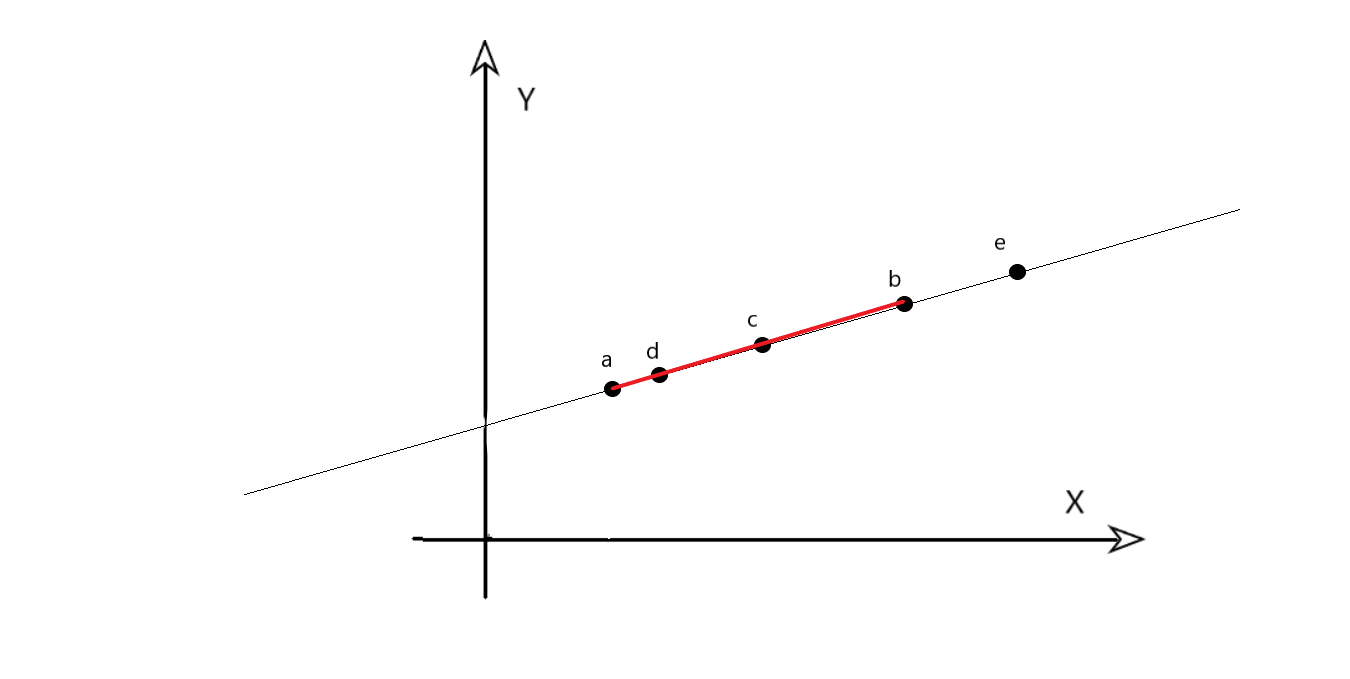


图3.1.4.1 线性插值图示

在此图当中，a和b为两个主要的点，其余的三个点都可以通过这两个点线性表出，其公式为：

其中，k为参数。例如c为a和b中点，可以表示为：

如果表示d的话，需要k<0.5，如果表示e的话，需要k>1。

这个方程之所以在计算机中拥有广泛的运用，关键原因有两个：

1、他将两个无关联的数据联系在一起，并结合成一个可利用的方程，

2、是他将一个连续的状态打散，形成无数个离散的数据，而计算机中很多的数据都是离散的。

例如计算机中的关键帧动画在执行的过程当中，即为将两个关键帧视为两个不同的点，通过调整两个关键帧之间的参数来调整两个关键帧之间的过渡，从而实现由两个完全没有联系的关键帧变为一个流畅的动画。在绘图流水线的绘制片元着色过程中，倘若所渲染的两个点颜色值不同，那么GPU也会默认的执行线性差值操作，即为将两个不同的rgb视为一对离散的量，通过此参数方程使其中间不断的过度过度成为一组渐变的颜色，其内部及时通过光栅化设置了此插值算法的参数的值。如下图，即为利用WebGL绘制的不同颜色顶点的三角形的颜色插值现象。

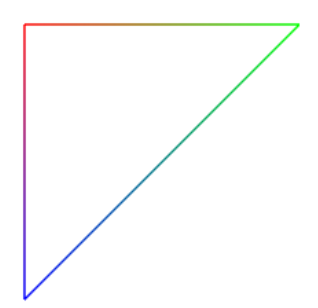


图3.1.4.2 线性插值实例

### 3.1.5 贝塞尔曲线原理

贝塞尔曲线是对上述线性差值算法的一个实际应用，由法国汽车制造商雷诺公司的工程师皮埃尔贝赛尔和雪铁龙公司的物理学家，数学家Paul de Casteljau所发明，其特点是形式简单，形状优美。

一个常规的二次贝塞尔曲线有一个开始点和一个结束点，还有一个控制点，其弧度由控制点所控制，如下图所示：

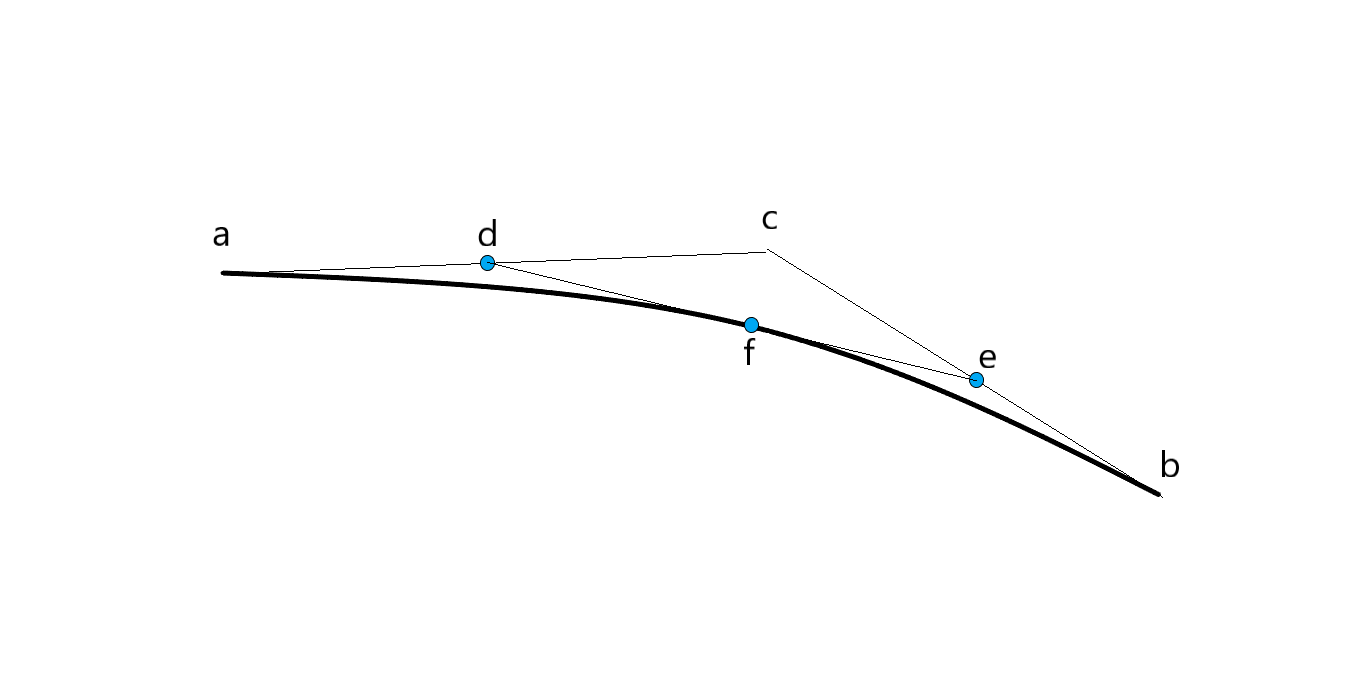


图3.1.5.1 贝塞尔曲线示例

设开始点为a，结束点为b，控制点为c，做一条从开始点到控制点的连线ac，再做一条从控制点到结束点的连线bc，取ac中点为d，取bc中点为e，做直线de取中点为f。贝塞尔曲线开始点和结束点的变化率与直线ac和直线bc的斜率相同，设开始点点a时的变化率为θ，结束点b时的变化率为φ，整条曲线的绘制过程即为从a点开始变化率不断由a向b变化的参数方程，这一条参数方程会恰好和直线de相切于f，假设这条直线绘制于笛卡尔坐标系中，那么曲线方程近似为：

向多个控制点扩展，即为多次贝塞尔曲线。假设给出n+1个控制点位置：Pk =（xk, yk, zk），这里k可以取0到n，这些坐标点将混合产生下列位置向量P(u)，用来描述P0和Pn间逼近Bezier多项式函数的路径：

其中，u为参数。

### 3.1.6 绘图流水线

常规3D场景中包含一组对象，每个对象由-组图元组成，每个图元又包含一组顶点。几乎可以认为是图元的类型和顶点集定义了场景的几何数据（geometry）。在一个复杂的场景里，可能会使用数千个甚至数百万个顶点来定义对象。为了生成存储在帧缓存中的图像，必须以相似的方式来处理所有这些顶点。如果把生成图像看成是处理对象的几何属性，可以用图2.3.6.1中所示的流程图来表示成像过程的几个主要步骤，即：顶点处理，裁剪和图元组装，光栅化，片元处理。

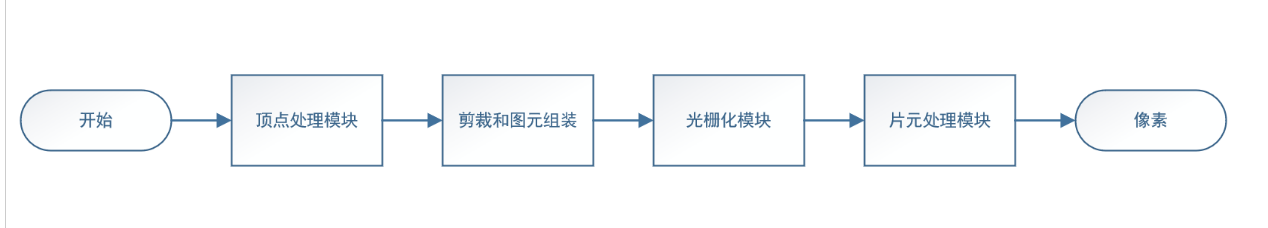


图3.1.6.1 绘制流水线

在JavaScript 2D绘图中，不过多涉及此类底层逻辑，大多都由底层逻辑和GPU自动完成。只有光栅化和剪裁时而用于理论学习之中。

### 3.1.7 剪裁及光栅化

1、 光栅和帧缓存

现代图形系统基本上都是基于光栅的。在这类系统中，输出设备上的图像实际上是由图形系统生成的像素（pixel）阵列，即光栅（raster）。每个像素对应图像中的一个具体位置或小区域，这些像素存储在帧缓存（framebuffer）中。帧缓存是图形系统的核心，其内的像素数量定义了分辨率（resolution），决定了图像的细节程度。帧缓存的深度（depth）或精度（precision）表示每个像素的比特数，决定了系统能表示的颜色种类。例如，1比特深度的帧缓存只能有两种颜色，而8比特深度的帧缓存能表示256（2^8）种颜色。

2、 剪裁

由于GPU资源有限，系统无法一次性成像整个场景，因此需要进行剪裁。人眼视网膜尺寸有限，大约对应90°视域。基于此，虚拟相机的视域也是有限的，在3D中通过裁剪体（clipping volume），在2D中通过裁剪面实现。裁剪体内的对象可以投影成像，而外部对象不会成像，被视为裁剪掉。跨越裁剪体边界的对象部分可见。裁剪针对图元而非顶点。透视除法后，图元的可见部分位于方形内，表示为：

其中w和h分别为宽度和高度。这个坐标系称为规范化设备坐标系（normalized device coordinates），范围也是对象的包围盒（bounding box）。在二维图形中，这个范围称为对象的包围矩形（bounding rectangle）。该坐标系中保留了生成正确图像的信息，但不依赖于应用程序定义的原始坐标单位或特定显示设备。

3、 光栅化

光栅化模块进一步处理裁剪模块得到的图元。例如，确定一个单色填充的三角形，光栅化模块需确定帧缓存中哪些像素位于三角形内部。光栅化模块为每个图元输出片元（fragment）。片元是携带颜色和位置信息的潜在像素，这些信息用于更新帧缓存中相应的像素。

### 3.1.8 填充算法

在2D绘图中经常会绘制封闭路径，一个封闭路径可以填充在2D图形学中有两种填充算法，第一种是奇偶原则算法（evenodd），第二种是非零绕数（non-zero）算法。这两种算法全部基于内外测试法：测试一个图形封闭性以及当前点是处于封闭图形内侧还是外侧的算法。

所谓奇偶原则即为假定p是多边形内部的某个点，从p点出发随意的射出一条射线与多边形的边相交的次数，如果是单数次则为在图形内，如果为双数次，即为在图形外。例如下图图二即为奇偶原则。

非零绕数法即为先设定顺时针为正数，逆时针为负数或反之，随机发出扫描线，检测扫描线被环绕的次数。如果是顺时针环绕则加一，如果是逆时针环绕则减一，在所有的边都检测完后如果环绕次数为零，则不在图形内，如果环绕次数为正数或负数，则在图形内。例如下图图一即为非零绕数原则。

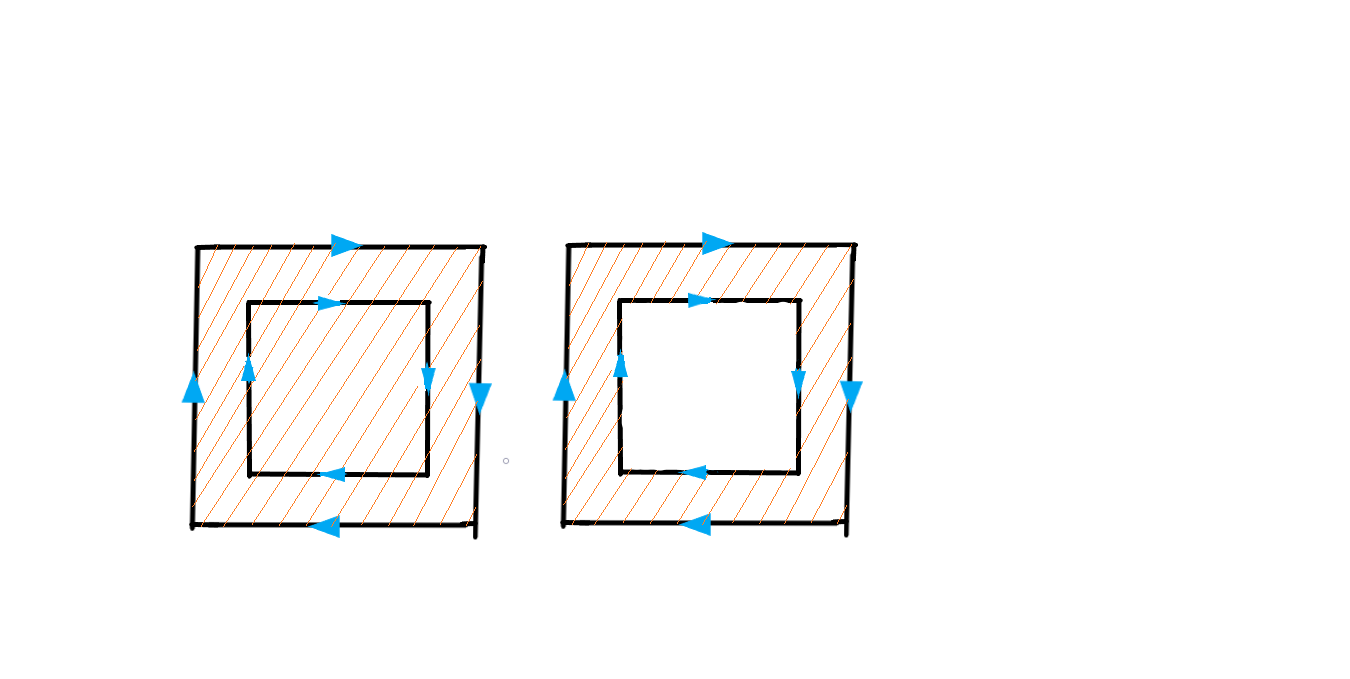


图3.1.8.1 填充算法图示

## 3.2 js对2d计算机图形学的实现

### 3.2.1 图元的实现

参考：MDN，JavaScript权威指南6，红宝书4

JS对2D API的实现采用路径的思想。然而和相比SVG有所不同，SVG中使用一个包含了字母和数字的字符串来描述路径，2D API是通过一系列方法调用来定义路径的。一旦定义了路径， 其他的诸如fill()这样的方法 就可以在该路径上操作了。而像fillStyle这样的上下文对象的属性则是指定了如何进行这些操作。以下方法可用于操作对象的路径：

|  |  |
| --- | --- |
| CanvasRenderingContext2D.beginPath() | 通过清空子路径列表来开始一个新路径。在创建新路径时调用此方法。 |
| CanvasRenderingContext2D.closePath() | 将笔的位置移回到当前子路径的起点。它试图从当前点绘制一条直线到起点。如果形状已经关闭或只有一个点，此函数不执行任何操作。 |
| CanvasRenderingContext2D.moveTo() | 将新子路径的起点移动到 (x, y) 坐标。 |
| CanvasRenderingContext2D.lineTo() | 将当前子路径中的最后一个点与指定的 (x, y) 坐标连接成一条直线。 |
| CanvasRenderingContext2D.bezierCurveTo() | 将一个三次贝塞尔曲线添加到当前路径。 |
| CanvasRenderingContext2D.quadraticCurveTo() | 将一个二次贝塞尔曲线添加到当前路径。 |
| CanvasRenderingContext2D.arc() | 在当前路径中添加一个圆弧。 |
| CanvasRenderingContext2D.arcTo() | 在当前路径中添加一个弧，其控制点和半径由前一个点通过直线连接。 |
| CanvasRenderingContext2D.ellipse() | 在当前路径中添加一个椭圆弧。 |
| CanvasRenderingContext2D.rect() | 创建一个矩形路径，其位置由 (x, y) 决定，大小由宽度和高度确定。 |
| CanvasRenderingContext2D.roundRect() | 创建一个带有指定位置、宽度、高度和圆角半径的圆角矩形路径。 |

表3.2.1.1 2D API提供图元绘制方法

### 3.2.2 图形学转换的实现

基于上一节所讨论的计算机坐标系的运行过程，将其结论带入2D绘图API中。在2D绘图API实现中，默认坐标系是以画布最左上角为坐标原点(0.0)。越往右X轴的数值越大，越往下Y轴的数值越大。在默认坐标系中，每一个点的坐标都是直接映射到一个CSS像素上(CSS像素之后再映射到一个或者多个设备像素)。画布中一些特定的操作和属性的设置(诸如抽取原始像素值以及设置阴影偏移量)都使用默认坐标系。然而，除了默认的坐标系之外，每个画布还有一个“当前变换矩阵”，作为图形状态的一部分，类似于上一节所讨论的对象标架。 该矩阵定义了画布的当前坐标系。当指定了一个点的坐标后，画布的大部分操作都会将该点映射到当前的坐标系中，而不是默认的坐标系。当前变换矩阵是用来将指定的坐标转换成为默认坐标系中的等价坐标。

在过去的实现中，变换矩阵是内置于2D绘图API中的，但随着DOMMatrix API的引入后，图形学转换的实现则更多通过DOMMatrix API进行，过去的内置矩阵实现以及内置转换函数只用于向下兼容性而存在。现在大部分API中使用DOMMatrix对象，并且将来也会使用。

DOMMatrix继承于DOMMatrixReadOnly，DOMMatrixReadOnly是一个对于DOMMatrix读取特征方法和属性的实现，可以访问一个矩阵中的值，并且可以和矩阵进行读取操作。DOMMatrix给了写的权限，提供一些可写入的方法，大致介绍如下：

|  |  |
| --- | --- |
| DOMMatrixReadOnly.flipX()  DOMMatrixReadOnly.flipY() | 返回一个新的 DOMMatrix，通过绕其 X（Y）轴翻转源矩阵创建。这相当于将矩阵乘以DOMMatrix(-1, 0, 0, 1, 0, 0)。原始矩阵不会被修改。Y轴则是乘以DOMMatrix(1, 0, 0, -1, 0, 0)。 |
| DOMMatrixReadOnly.inverse() | 返回一个新的 DOMMatrix，通过求源矩阵的逆矩阵创建。 |
| DOMMatrixReadOnly.multiply() | 返回一个新的 DOMMatrix，通过计算源矩阵与指定矩阵的点积创建 |
| DOMMatrixReadOnly.rotate()  DOMMatrixReadOnly.rotateAxisAngle()  DOMMatrixReadOnly.rotateFromVector() | 返回一个新的 DOMMatrix，通过指定的角度绕源矩阵的每个轴旋转。原始矩阵不会被修改。通过绕指定向量旋转给定角度，通过指定向量与 (1, 0) 之间的角度 |
| DOMMatrixReadOnly.scale() | 返回一个新的 DOMMatrix，通过在给定原点上为每个轴缩放源矩阵的指定量创建。 |
| DOMMatrixReadOnly.skewX() | 返回一个新的 DOMMatrix，通过在源矩阵的 X 轴上应用指定的倾斜变换来创建。原始矩阵不会被修改。 |
| DOMMatrixReadOnly.translate() | 方法会返回一个新的 DOMMatrix，其中包含通过使用指定的向量计算得出的矩阵来平移源矩阵。 |

表3.2.2.1 DOMReadOnlyMatrix API提供转换的方法

DOMMatrix方法与DOMReadOnlyMatrix类似，仅基于DOMReadOnlyMatrix的转换算法给出了DOMMatrix的转换算法，因此不一一列举。

### 3.2.3 颜色的实现

颜色学隶属于图形学分支，实现中也有多种形式，其中比较优秀且比较复杂的一种形式就是前端的CSS颜色表示模式。在最新CSS4数值和单位规范和CSS5数值单位模块草案中的<color>类型数据中，定义了全面的数值和单位表示法，其中包含颜色表示规范，这些颜色表示法构成了网络前端的基础颜色表示模式，无论是在类库的实现过程中还是2DAPI中都具有参考价值。同时这些CSS数值和单位规范隶属于CSS Houdini，可以适应未来编程环境。

<color> 值有多种可能的情况，其具体给定的值可以是以下之一：

1. 通过关键字：<named-color>（例如蓝色或粉红色）、<system-color> 和 currentcolor。
2. 通过十六进制表示法：RGB表示法的快速表示，<hex-color>（例如 #ff0000）。
   1. 三位十六进制表示法，即为#RGB
   2. 四位十六进制表示法，即为#RGBA
   3. 六位十六进制表示法，即为#RRGGBB
   4. 八位十六进制表示法，即为#RRGGBBAA
3. 通过颜色空间中的参数使用函数表示法：
   1. sRGB 颜色空间：hsl()、hwb()、rgb()；
   2. CIELAB 颜色空间：lab()、lch()；
   3. Oklab 颜色空间：oklab()、oklch()；
   4. 其他颜色空间：color()。
4. 通过混合两种颜色：color-mix()。
5. 通过指定两种颜色，第一种用于浅色方案，第二种用于深色方案：light-dark()。

### 3.2.4 图像的引入

图像是一组离散的数据，在网络前端之中，图像及其相关的数据是通过HTTP请求获得的，HTTP请求可以是浏览器内部发送或者通过API（类似于fetch）发送，所获得的离散数据应用于图形学绘制之中。

同样图像可以有多钟形式，

1. HTMLImageElement：HTML图片引用格式，通常由HTML或JS创建的<img />标签来创建，还可以通过自定义标签继承和初始化，在此处这种情况不列入考虑范围。
2. SVGImageElement：SVG矢量图引用格式，通常由HTML或JS创建的<svg />标签来创建。
3. HTMLVideoElement：HTML视频引用格式，通常由HTML和JS共同创建的<video />标签来创建。
4. HTMLCanvasElement：HTML画布引用格式，通常由HTML或JS创建的<canvas />标签来创建。
5. ImageBitmap：位图，由JS的createImageBitmap全局函数创建。createImageBitmap() 方法从给定的源创建一个位图，可选择裁剪以仅包含该源的一部分。
6. OffscreenCanvas：离屏画布，由JS的OffscreenCanvas构造函数创建，隶属于OffscreenCanvas API。
7. VideoFrame：视频帧，由JS的VideoFrame构造函数创建，隶属于Web Codecs API。
8. Blob：Blob图像数据表示，由JS创建，通常用处理前端数据块时的toBlob()方法和构造函数创建。
9. ImageData：2D API原生的像素数据表示类。

## 3.3 本章小结

本章主要讲解了计算机图形学的基础理论及其在JavaScript语言中的应用。图形学中的坐标系统和齐次坐标构成了图形绘制的基础，通过图形变换的矩阵运算，我们可以改变点和向量的位置，齐次坐标也会参与这些运算。接着，本章介绍了线性插值算法，并提供了一些相关案例，说明了本类库中的部分算法也基于此原理。此外，还阐述了一些图元的运算方法，以及图形学中绘制流水线的架构，特别是流水线中常用的两个部分。

在原理讲解之后，本章进一步展示了这些理论在实践中的应用，从基本图元的实现到2D绘图API的图元实现，以及2D绘图API所依赖的DOMMatrix API。还介绍了在网页中实现和表示颜色的CSS颜色规范，以及如何将图像等离散数据动态引入网页，以便在图形绘制过程中使用。这些理论知识和实践应用构成了本类库的基石。

# 第4章 类库总体设计

## 4.1 类库的体系结构设计

类库开发过程关键在于如何将已有的琐碎的条件进行规范，类型化，模式化，总体的目的在于在突出类库设计的主旨的前提下，尽可能全面地覆盖所有语法，并且尽量保持原子操作的特性，即尽量保持操作和操作之间耦合性弱，可复用性强。在这一前提下，将模块进行划分。

### 4.1.1 类库功能划分及封装设计

JS原生2D绘图API的功能比较复杂，包含了约30余个属性和50余个方法，以及5个相关类。

1、属性分类：文字属性、滤镜后处理属性、字体设置属性、图形学融合算法实现相关属性，抗锯齿属性、画线参数属性、阴影参数属性、绘制样式设置相关属性；

2、方法分类：路径方法，变换方法，像素操作方法，渐变方法，画线参数方法，路径操作方法，记忆方法，自定义图案方法。

本文所阐述的类库主要是在JS原生API的基础上进行进一步抽象，尽可能保持合理的逻辑，对原生API的属性、方法、相关类等进行全覆盖。结合需求分析结果，通过对现有属性、方法的分类和归纳发现，原生JS的2D绘图API中属性虽然类型繁多，但从功能的角度上来看，诸多属性存在一定的内在联系，因此可以采用面向对象的设计思想，将现有属性、方法以及相关类进行模块化的封装设计，具体设计思路如下。

1. 图元相关属性封装设计：分析发现，文字属性、滤镜后处理属性、字体设置属性、画线参数、阴影设置属性以及绘制样式设置都是图元相关属性，他们都描述了图元的状态信息，因此可以将其归于图元模块。同理，方法中的画线相关参数同样可以封装于其中。
2. 图元相关方法封装设计：在方法中，路径和变换两项与图元的绘制相关，因此可将这两类方法归为图元模块。

图元模块是类库中非常关键的模块。通过需求分析还发现，图元模块应该具备与用户交互的功能，因此在交互及其实现的过程中，需要引入路径操作等诸多步骤，路径操作也是图元模块所需要的一个类型。

图元模块的包含关系如下图所示：

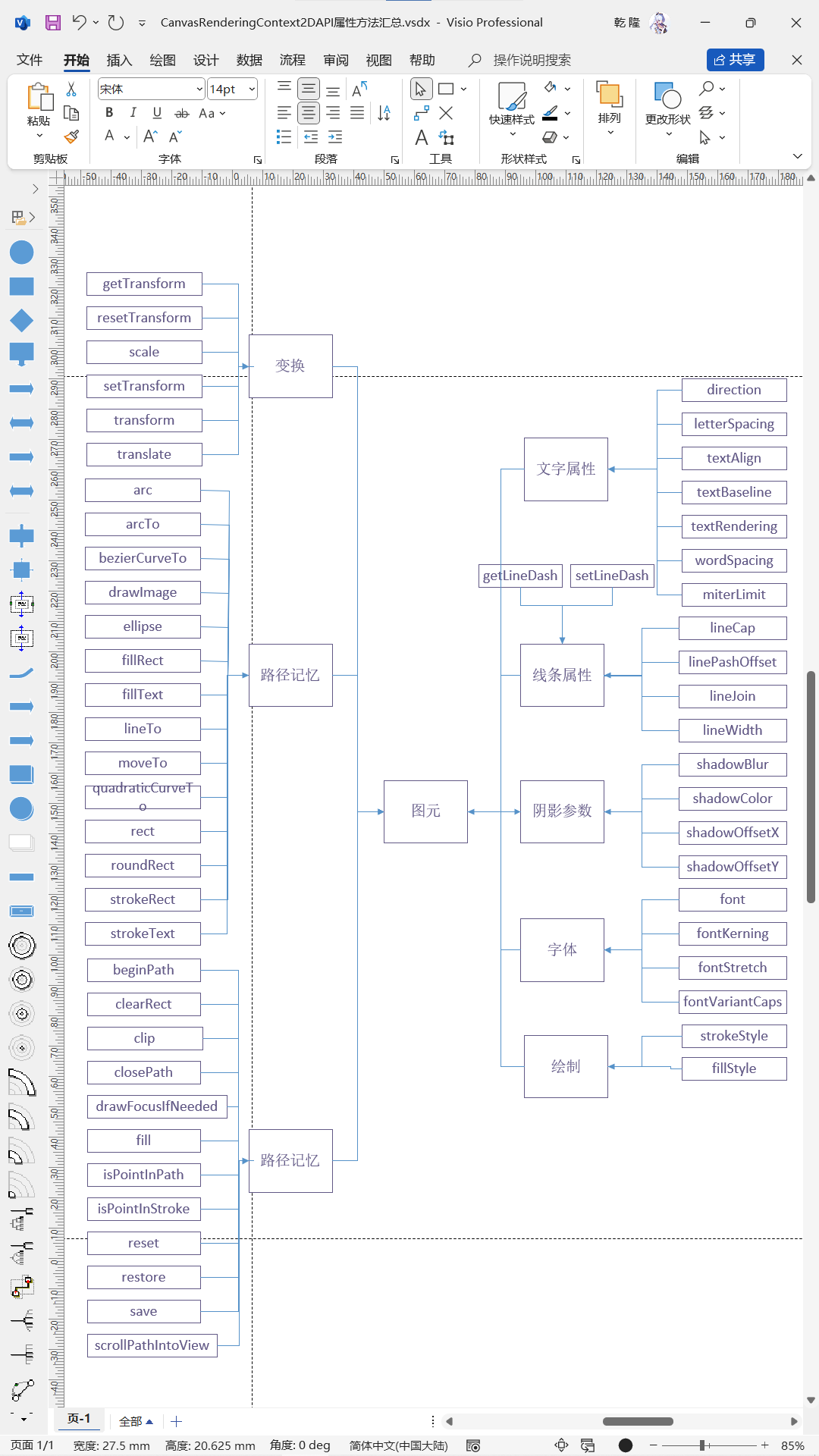


图4.1.1.1 图元模块规划分类

1. 可填充类型（Fillable）模块封装设计：在现有的方法中，渐变，自定义图案，结合绘图中的隐藏条件——颜色的封装，都有一个共同的特点，即都用于图元的着色。因此可以将以上属性和方法封装为一个模块，命名可填充类型（Fillable）模块。

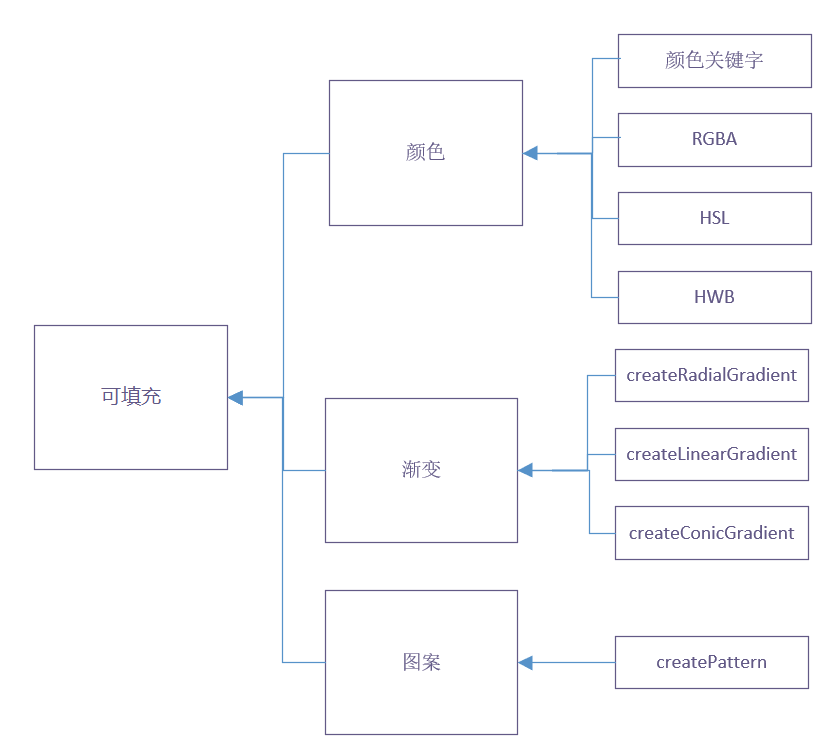


图4.1.1.2 可填充模块规划分类

1. 像素处理模块封装设计：像素操作的类型不尽相同，可以将相关属性和方法单独封装为一个模块，命名为像素处理模块。

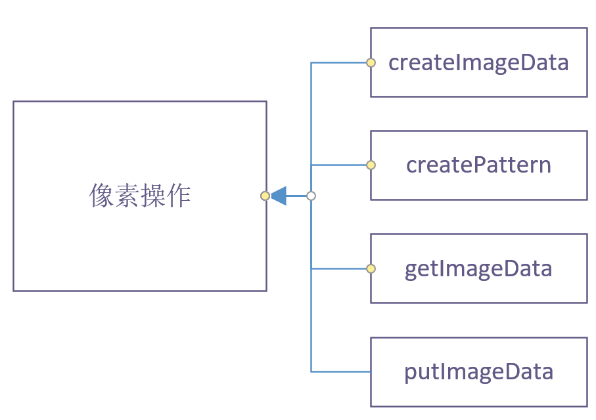


图4.1.1.3 像素操作模块规划分类

5、主要逻辑模块封装设计：剩余的融合操作，抗锯齿以及记忆功能，这几类操作是普遍用于调整画布的整体状态。在对象化2D绘图的过程中必然要进行对象化画布的操作，可以将这几类直接设置为画布的状态属性，分类于主要逻辑模块即可。

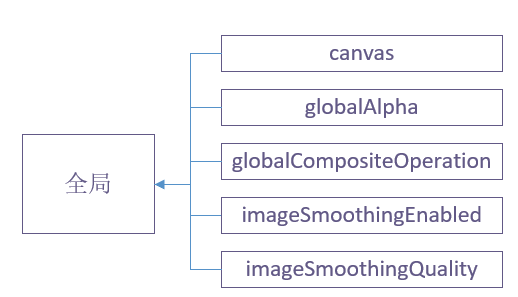


图4.1.1.4 全局模块规划分类

6、动画模块封装设计：通过需求分析发现，类库还需要尽可能完成对象化2D绘图后的动画功能，因此需要单独考虑动画模块。对于动画模块的设计和实现可以参考JS原生API的Animation API和相关API，例如Timeline API等。

综上所述，类库的功能通过抽象和归纳可以划分为以下几个模块：类库主要逻辑模块，图元模块，可填充类型模块，像素操作模块，动画模块。

### 4.1.2 类库继承和依赖的逻辑

面向对象编程具有封装、抽象、继承和多态等丰富特性，能够实现较为复杂的程序设计思路，因此也是当下主流的程序设计方法。利用面向对象思想进行类库的设计，需要考虑其继承和依赖的关系，以确保程序具有更好的可读性、可维护、可扩展以及低耦合等特性。

1、图元模块分析

图元模块的设计思路是以模块绘图为中心，属性和事件辅之。JS的2d原生语法提供了诸多的路径绘制方法，以这些绘制方法中的共有属性为基础，私有属性为扩展，可以设计一个图元基类，并基于此基类进行拓展，从而完成面向过程绘图向面向对象化的转化操作。

面向对象的设计思想要求继承JS所有对象的基类Object类。同时，为了实现用户交互的需求，还要继承事件类，即EventTarget类。本文所设计的类库中，每一个图元都有公用的属性和方法，因此都要继承于图元基类Graphic类。这样，每一个具体的图元类都可以确定其继承链，即某个图元类->Graphic类->EventTarget类->Object类。继承链图如下：

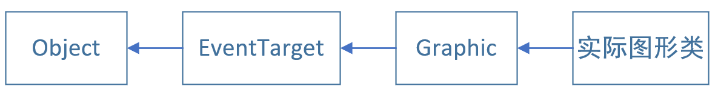


图4.1.2.1 图形类继承链

例如，矩形图元Rect类，它是一个图元，具有图元的共有属性和方法，因此继承于Graphic类；它也可以注册和触发交互事件，因此继承于EventTarget类；它作为一个对象，也继承于Object类。所以，Rect类实例对象的继承链为：Rect实例对象->Rect类->Graphic类->EventTarget类->Object类。

图元对象中可以设置的属性，根据2D绘图原生API的分类，可以分为以下几类：阴影属性，字体属性，文字格式属性，边框绘制属性。这四类属性是通用的，适用于所有的图元，因此可以将其设置于图元基类中。这四类属性的类型，都是相应的类，例如阴影属性的类型是阴影类，字体属性的类型是字体类。因此，这四类属性的类和图元基类之间存在直接依赖关系。

另外，图形变换相关功能也是图元对象中的一个重要属性。图形变换是绘图过程中常用的一种算法功能，它可以将一个图形进行刚体或非刚体的变换。图形变换的本质是利用变换矩阵作用于图形的点，从而改变整个图形的形态。因此，图形变换涉及到两个关键的概念，一个是图形的点，另一个是变换矩阵。图形的点可以用坐标数字或者JS API中的DOMPoint或DOMReadonlyPoint类来表示。变换矩阵可以用二维数组或者JS API中的DOMMatrix类来表示。为了实现图形变换的矩阵计算操作，可以利用JS API的DOMMatrix类中的相关方法，这是一种性价比高且与原生语法契合的实现方式。因此，图元基类的图形变换属性的类型是DOMMatrix类，图元基类与DOMMatrix类之间也存在直接依赖关系。

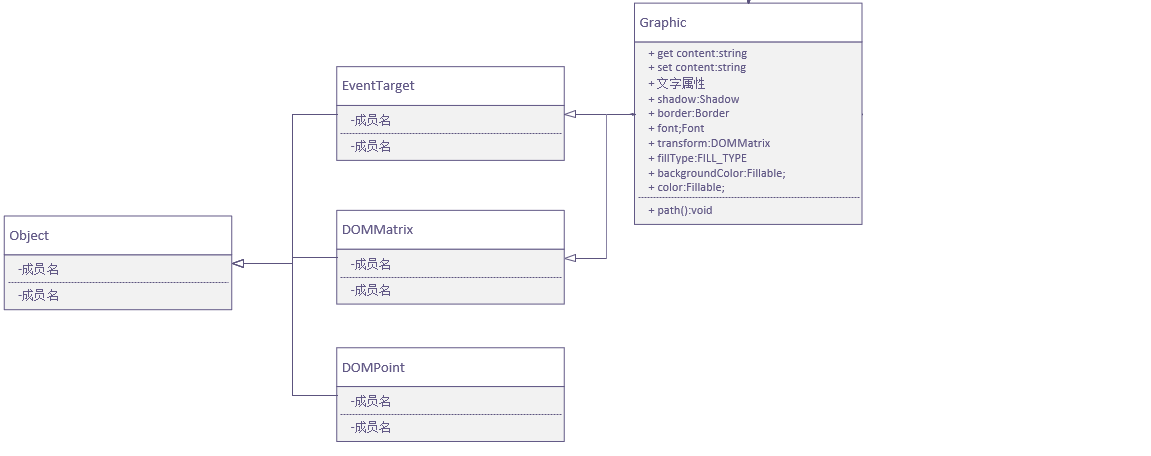


图4.1.2.2 图形类继承链

2、可填充类型模块分析

接下来是可填充类型类关系设计和介绍。所谓可填充类型类（Fillable），即为可以被填入图形的类型。在原声语法中有三种类型的值可以填充图形，一个是单一颜色，一个是多种渐变，一个是自定义图像图案。这三个类构成了整个Fillable类。

颜色类是可填充类型类中的一个重要组成部分。它包含了用于填充图形的颜色相关属性和方法。在原生语法中，颜色类的表示方法按照CSS中<color>值的标准来制定。可以细分为以下几种子类：

1、纯颜色静态属性：这是一些常见的颜色，如红色、绿色、蓝色等。这些颜色可以直接通过名称或预定义的常量来使用，作为Color类的属性而存在。

2、Rgb子类：这是基于RGB（红绿蓝）颜色模型的一种颜色子类，继承于Color类，其类型为rgb，可以通过指定红、绿、蓝三个通道的值来创建颜色。

3、Hsl子类：这是基于HSL（色相、饱和度、亮度）颜色模型的另一种颜色子类，继承于Color类，其类型为hsl，可以通过调整色相、饱和度和亮度三个属性的值来创建颜色。

除了颜色类，可填充类型类还包括渐变和图案。渐变是一种从一个颜色到另一个颜色逐渐过渡的效果，可以实现平滑的色彩变化。所以渐变一定对颜色类型有依赖关系。这三个渐变需要指定顶点，所以需要点的坐标，可能与DOMPoint和数字有关联。

1、线性渐变类（LinearGradient）：这是一种沿着一条直线的渐变效果，继承于渐变类，需要指定渐变的起点和终点的坐标，以及颜色的起始和结束值。

2、径向渐变类（RadialGradient）：这是一种从一个圆形向外扩散的渐变效果，继承于渐变类，需要指定渐变的圆心和半径，以及颜色的起始和结束值。

3、锥形渐变类（ConicGradient）：这是一种以一个圆形为中心，沿着角度的变化的渐变效果，继承于渐变类，需要指定渐变的圆心和起始角度，以及颜色的起始和结束值。

图案类是可填充类型类中的另一个重要组成部分。它允许用户自定义填充图形的图案，可以是用户自己设计的图像、其他画布元素、用户引入的视频或位图等。关联关系比较复杂，例如：

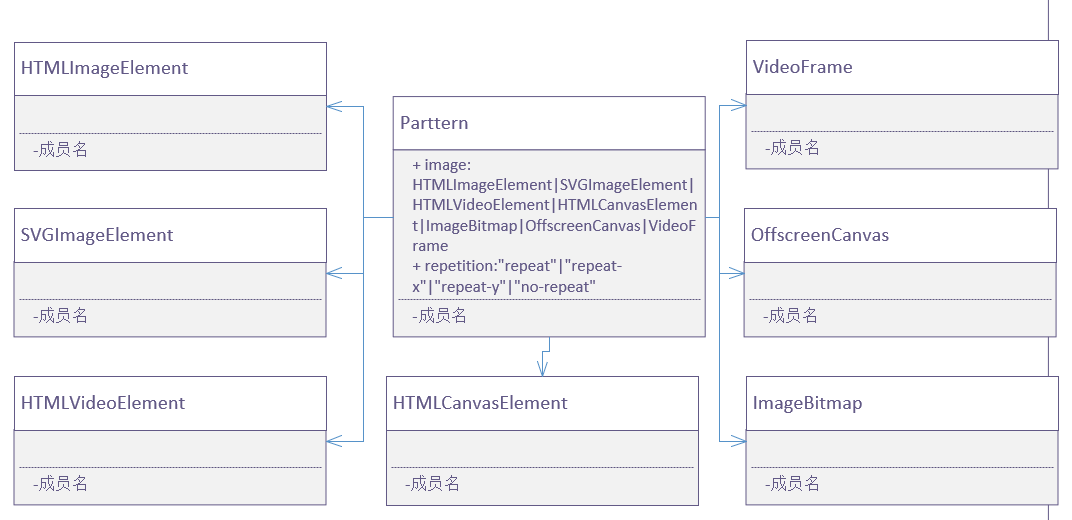
1、如果用户自定义图像作为图案的话，则此类便与HTMLImageElement引起关联，需要指定图像的源地址和重复方式。

2、如果用户引入视频的话，则此类便与HTMLVideoElement引起关联，需要指定视频的源地址和重复方式。

3、如果用户引入其他画布的话，则此类便与HTMLCanvasElement引起关联，需要指定画布的元素和重复方式。

4、如果是离屏画布，则是OffscreenCanvas类。

5、如果用户引入位图的话，则此类便与ImageBitmap引起关联。



4.1.2.1 Parttern类依赖图

3、像素操模块分析

像素操作是指对图形的每个像素的颜色值进行读取或修改的操作。像素的颜色由GPU所渲染，从GPU读入颜色值通常是需要严格的类型和内存的约束，所以需要用类型化数组（Typed Array）来表示。类型化数组是一种特殊的数组，它的每个元素都是固定的数据类型和大小，可以提高性能和效率。类型化数组根据计算机基础数据类型分为几类，例如：

1、Uint8Array和Uint8ClampedArray，它们的每个元素都是8位的无符号整数，范围是0-255。

2、16位：Uint16Array，它的每个元素都是16位的无符号整数，范围是0-65535。

3、32位：Uint32Array，它的每个元素都是32位的无符号整数，范围是0-4294967295。

4、64位：Float64Array，它的每个元素都是64位的浮点数，范围是-1.7976931348623157e+308到1.7976931348623157e+308。

像素数据类型通常为Uint8Array和Uint8ClampedArray，它们的区别主要在于处理超出范围的值的方式不同：如果给Uint8ClampedArray设置一个0-255范围之外的值，它将简单地默认为0或255，这样可以保证像素的颜色值不会出现异常。如果给Uint8Array设置一个0-255范围之外的值，它只会取数值二进制值的前8位，这样可能会导致像素的颜色值发生变化。

所以可以断定像素操作和类型化数组类有依赖关系，因为像素操作需要用类型化数组来存储和处理像素的颜色值。

4、动画模块分析

动画模块的需求是在最大程度贴合原声语法的基础上开发一个简易关键帧动画系统。于是动画中的诸多概念，例如时间线以及动画状态等接口可以直接使用原生接口进行扩展，于是动画模块依赖于Animation API和Document timeline API，动画模块需要创建一个动画类和其所依赖的API呈现直接依赖关系。动画类内部在实现的过程中需要获取屏幕刷新，所以需要用到requestAnimationFrame()方法来捕捉屏幕刷新的时机。

## 4.2 类库的功能模块设计

在前述内容中，模块的划分和分类已经明确，并且依赖与继承关系已经梳理清楚。继续前文的讨论，现在对每个模块的结构进行详细设计。

### 4.2.1 总体模块

首先，定义一个命名空间是必要的，这样做的目的是为编写的类提供一个清晰的环境，避免与JavaScript的全局变量发生冲突。JavaScript的机制允许在全局对象上进行操作，但如果不加以控制，全局声明和覆盖可能会修改预定义的类，从而引发意外的问题。随着版本的迭代，新类的增加可能会导致后期的问题。因此，定义一个对象或类，暂且称之为Mano，它不仅是框架的名称，也是XML命名空间。Mano封装了与类库相关的所有全局属性和方法，例如Canvas类。在框架设计初期，已经采用了这种代码风格，例如Graphic类，在JavaScript全局对象中，应该先引用Mano，然后使用Mano.Graphic。

Canvas类作为画布类型，在2D绘图中，所有操作都在画布上进行。画布需要维护的一个重要属性是当前已绘制的所有图元，以及其他全局状态，如是否开启全局抗锯齿，图元融合算法的选择等。方法中，渲染图元是一个关键操作，它负责渲染当前所有的图元。其他重要的方法包括添加和删除图元，清空画布，以及设置画布默认重绘颜色，这些类型可以是Color类。

### 4.2.2 图元

根据上述描述，对图元模块进行设计。首先，分析JavaScript已有的语法，EventTarget类已经继承自Object类，DOMMatrix和DOMPoint在高版本JavaScript中是内置语法，这些类的定义和引用被归入JavaScript原生语法包中。图元包的其他内容如下所述。

#### 4.2.2.1 图元封装

图元包的基础是Graphic类，它继承自EventTarget。接下来，定义每个图元的子类，所有子类都继承自Graphic基类。根据已有的图形算法，可以定义图元子类，例如，rect()方法可以封装成Rect类，其参数转化为属性。类似的子类还包括Arc、Ellipse、Line、BezierCurve、Rect、RoundRect和Text。为了允许用户自定义图形，添加了CustomGraphic子类。这些类和子类构成了图元类和子类的基本结构。

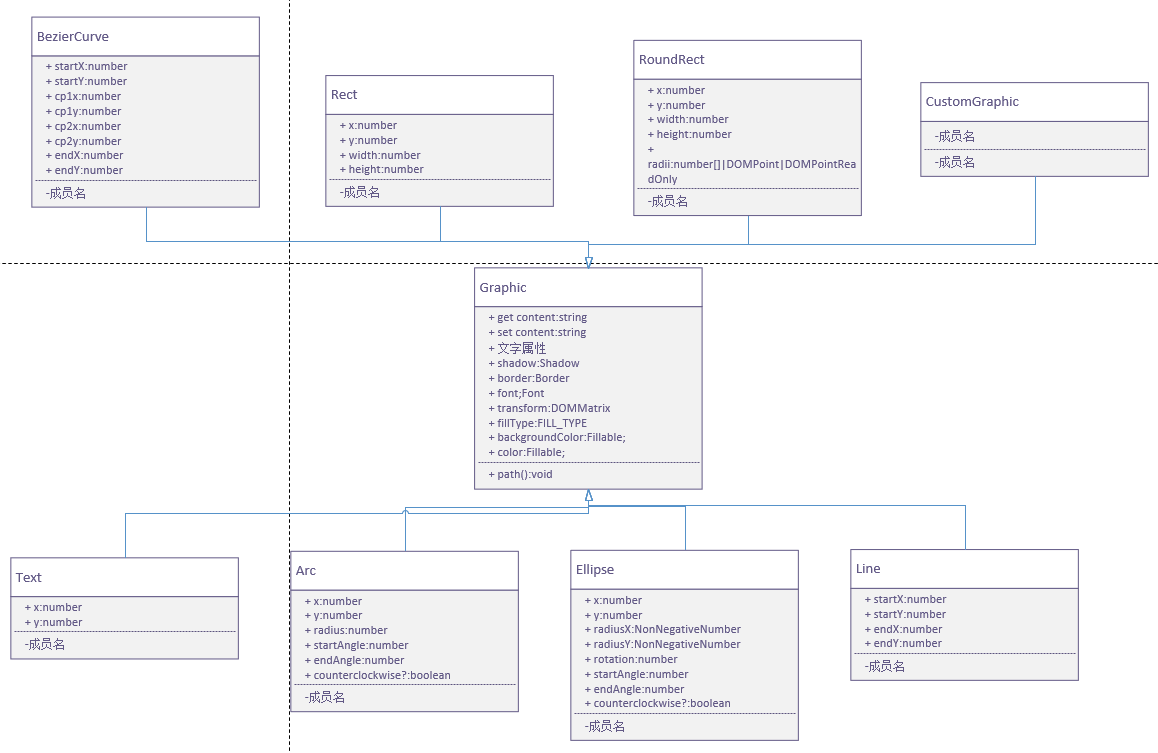


图4.2.2.1.1 图元封装类图

#### 4.2.2.2 属性封装

根据现有条件，可以确定图元基类的属性和方法。例如，set和get content属性用于设置图元内的文本内容；textFormat属性的类型为TextFormat类；shadow属性的类型为Shadow类；border属性的类型为Border类；font属性的类型为Font类；transform属性的类型为DOMMatrix类。这些属性都是控制样式的。fillType属性设置为FILL\_TYPE枚举类型，描述图元的绘制方式。backgroundColor属性和color属性为Fillable类，用于设置文字和背景的填充。后续迭代中，将更新其他需要的属性和方法。

根据以上描述，需要编写TextFormat类，用于控制文本格式，属性可以设置为控制文字居中，文字基线，文字渲染，单词空白，字母空白。

需要编写Shadow类，用于控制阴影，属性可以分为阴影模糊度，阴影颜色，阴影横轴偏移量，阴影纵轴偏移量。阴影颜色的类型可以设为颜色类。其余的属性类型可以设置为数字。

同时需要编写Border类，用于控制框样式。属性可以分为折线尾部渲染方法，折线折叠渲染方法，虚线线段长短及规律，线的粗细。

需要编写Font类，用于控制字体。属性主要为控制一些字体的细节，与CSS中字体控制类似。

FILL\_TYPE枚举类型主要用于绘制图元的两种不同的方法。有两个值，第一个是GRAPHIC\_FILL，第二个是GRAPHIC\_STROKE，前者通过内部填充算法填充图元内部的颜色，后者仅对图元的边框进行涂色，涂色后的边框遵从border属性的约束。同时需要在绘制的方法中设置内部着色算法，也可以通过枚举值的方法设置。

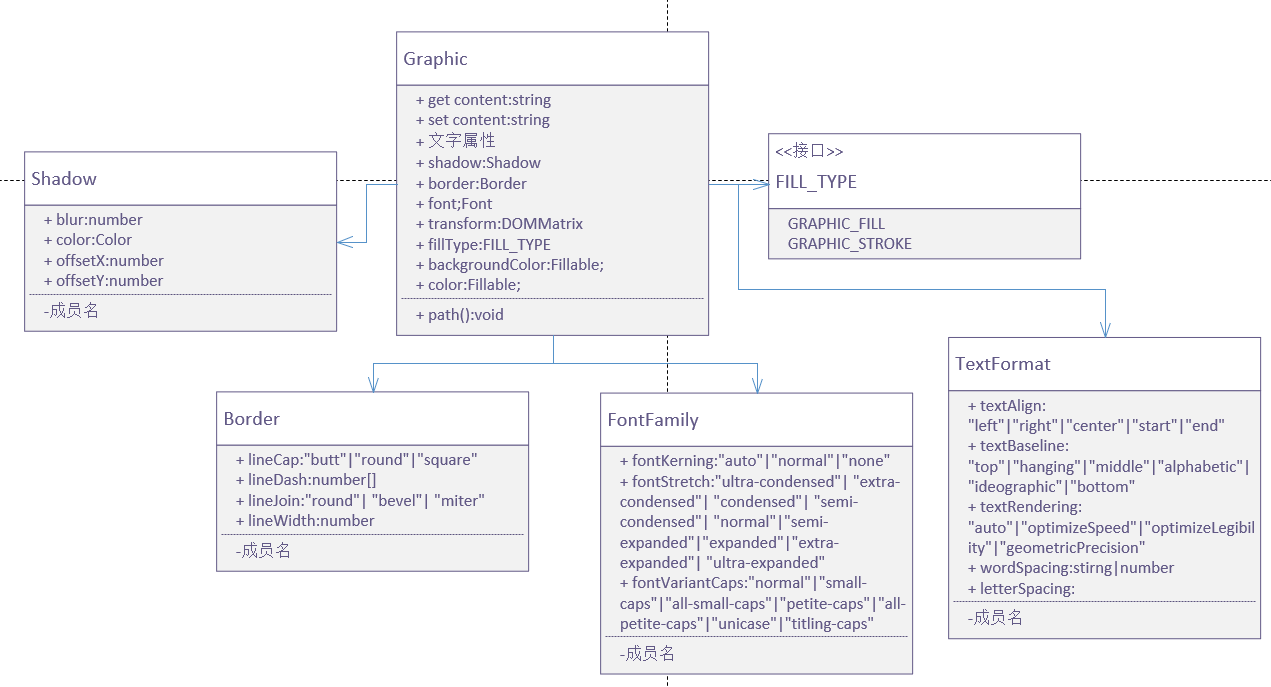


图4.2.2.2.1 属性封装类图

#### 4.2.2.3 事件交互

事件交互方面，需要与事件进行交互。前文中已经确定图元基类要继承于EventTarget类，EventTarget类是DOM2规范中的一个类，用于注册事件，触发事件以及维护事件。内部封装了addEventListener()，dispatchEvent()，removeEventListener()等DOM2事件核心方法。提供给用户的一个注册选项是利用DOM2规范中的事件注册方法。除此之外还有DOM0事件注册方法，DOM0事件名称以on开头，因此实现DOM0事件注册方法的一种方式是监测所有以on开头的属性的赋值和访问，并将同名事件注册至DOM2事件中。事件触发DOM2的事件注册触发。由于图元本身不是一个对象，因此不能直接注册事件，需要人为地将其封装成对象，并赋予可注册事件对象的特征，因此事件的触发需要手动进行。但在绘图API中，只有画布可以直接进行事件注册，因为它属于HTML元素对象。实现方法是将所有事件注册至画布对象中，利用路径检测方法检测当前点击坐标是否属于该图元，从而触发事件。不需要额外的类来支持，只需在画布上注册事件并绑定默认触发事件。每次触发事件时，自动执行一次默认触发事件，遍历所有图元和事件，如果图元数量较大，可以通过大致轮廓进行排除，优化时间复杂度。

### 4.2.3 填充类封装

可填充类的基类直接命名为Fillable。衍生出三个子类，分别用于表示颜色（Color类）、渐变（Gradient类）和图像（Pattern类）。

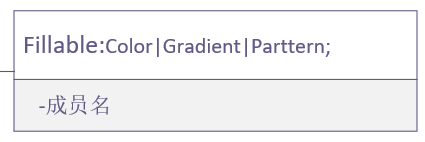


图4.2.3.1 可填充类类图

#### 4.2.3.1 颜色类

Color类具有多种表示方法，因此拥有多个子类。Color类包含所有关键字颜色的常量属性，并设置初始值。type属性用于区分子类类型。Rgb子类有四个属性，表示三个颜色分量和透明度，类型设为rgb，透明度默认为1。Hsl子类同理，属性表示三个颜色分量和透明度，类型设为hsl。如需封装其他颜色标识方法，可在后续添加。若需统一管理类型值，需额外添加枚举类型。

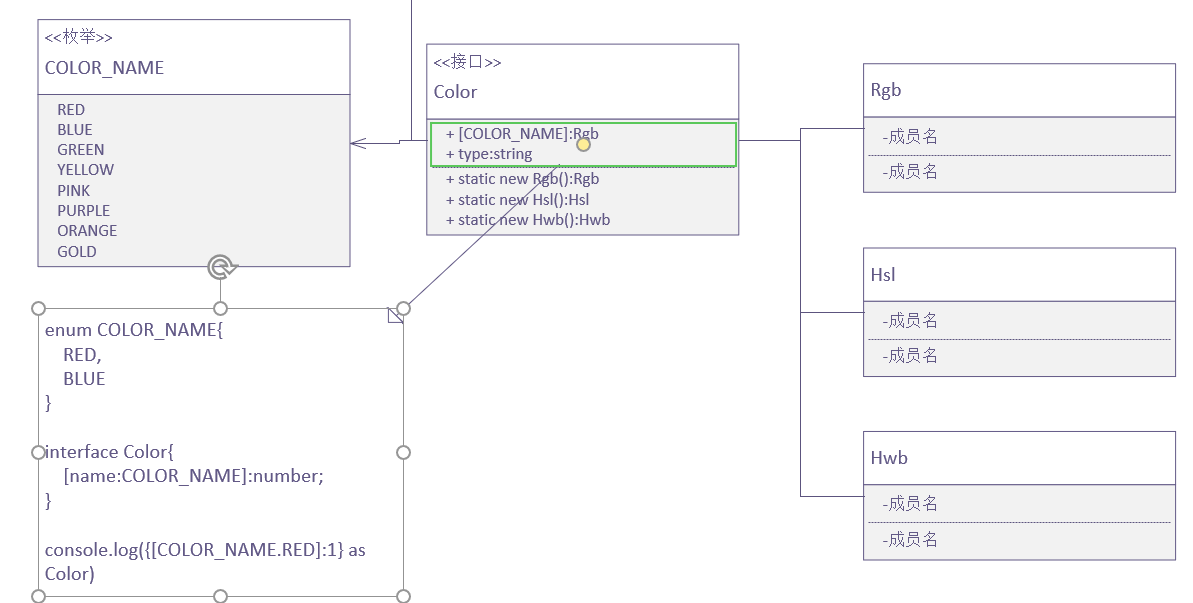


图4.2.3.1.1 颜色类类图

#### 4.2.3.2 图案类

图案类需有属性记录用户选择的图像，该属性类型包括用户可能选择的所有种类。另有方法负责添加图像，以及属性管理图像的重复方法，如双轴循环、x轴循环、不循环。



图4.2.3.2.1 图案类类图

#### 4.2.3.3 渐变类

渐变类涉及多种颜色及其偏移量，因此需定义方法以添加颜色和偏移量。同时，须有数组存储这些值。渐变类分为三种衍生类：

1、线性渐变类：需要两个属性表示起始和终止坐标，可用DOMPoint和数字表示。

2、辐射渐变类：包含四个属性，两个顶点坐标和两个半径值，均可用数字表示。

3、角度渐变类：需要两个属性表示中心点坐标和起始角度，均可用数字表示。

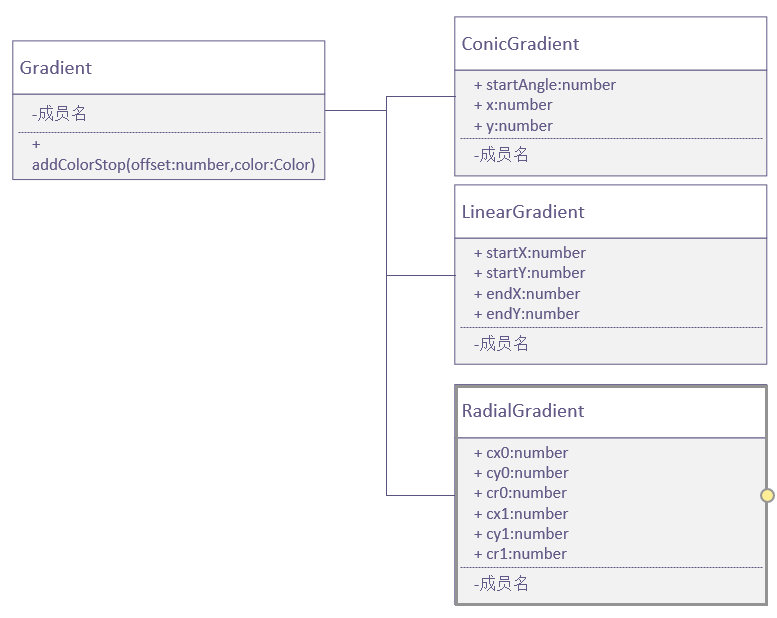


图4.2.3.1 渐变类类图

### 4.2.4 像素图像处理

像素图像处理的主要任务是记录像素数据，并提供修改途径。需要一个Uint8ClampedArray类型的属性来记录底层颜色数据，以便与底层接口对接。提供两个方法，一个用于获取像素颜色，另一个用于设置像素颜色，均需像素坐标点参数。写入像素颜色时，需新颜色参数，可为Color类或数字数组。需字符串表示底层颜色格式，该格式在画布创建时确定。还需记录高度和宽度值，这是两个数字类型属性。

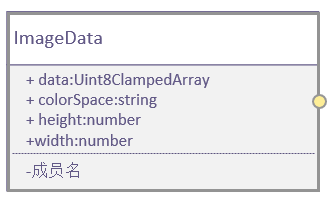


图4.2.4.1 像素类类图

### 4.2.5 动画封装

在封装动画模块的时候需要的不是封装一个可以完成一整套动画流程的动画包，动画其本质为一段时间内一个状态到另一个状态的变换过程，所以封装的实质是封装一个以时间为参数的不同状态之间的参数方程。将动画每一次渲染的时间t作为自变量传入后得出当前状态，每一次渲染的本质也是抹去画布后重新获取当前状态信息进行绘制。基于这一思想创建一个动画架构。动画封装由一个类和两个接口组成。类为动画模块的核心，封装了动画需要的常用功能，两个接口分别为动画的不同时刻的状态，和动画属性配置对象模型。

动画类属性大致如下：

1、状态属性：如是否结束、是否等待、是否准备好等，均为布尔值，播放状态为枚举值。

2、信息属性：如当前时间、动画ID、播放速率等，均为数字。

3、时间线：使用DocumentTimeline类型。

两个接口其中之一用于记录不同时刻动画的状态，动画的状态的接口则需要制定一个属性集合规范。动画即为在一定时间中，样式属性值从一个点平稳过渡到另外一个点。在用户指定动画时，类库需要规范接口从而约束用户每一次传入状态的准确性。

另一个接口是动画属性配置对象模型，用于规范用户对于动画配置信息的输入。用户可以配置的动画信息大概有以下：延迟时间，动画迭代次数，时长，这三个均为数字。还可以指定结束填充状态，动画运行方向，时间函数，动画状态累积方式等，这几个值为枚举类型。

两个接口所指定的信息可以用于控制时间t的走向，利用t即可生成动画。

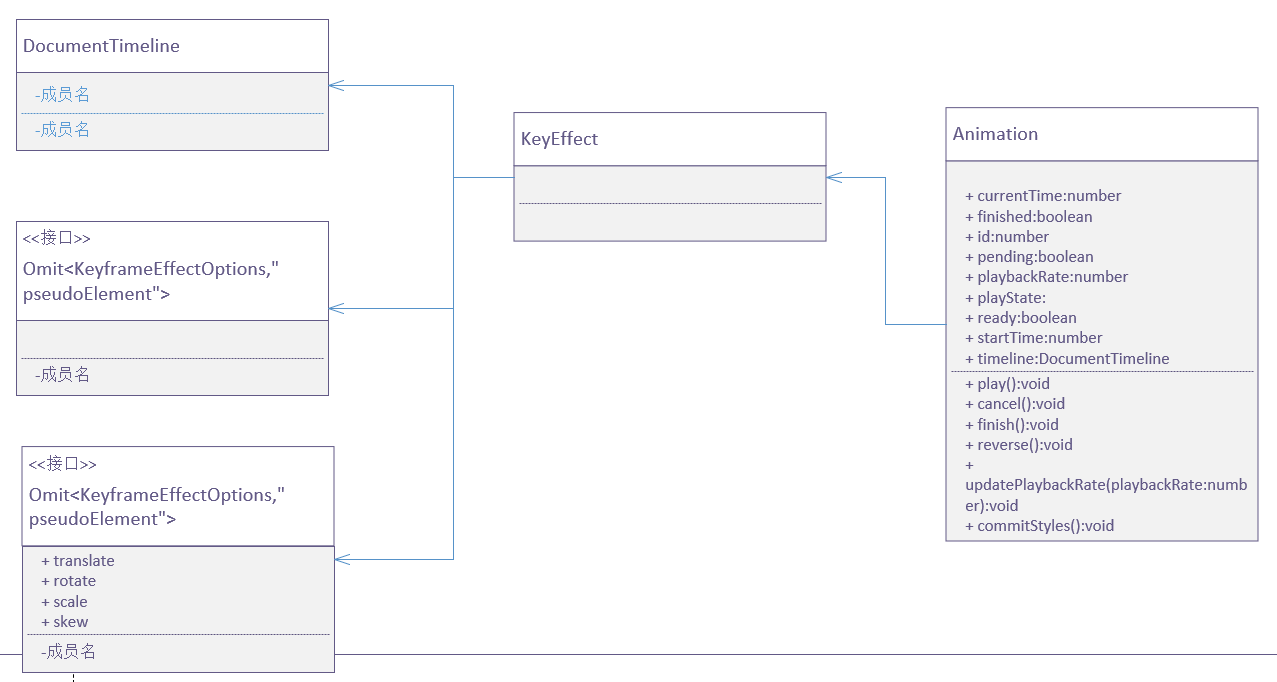
、

图4.2.5.1 动画类类图

## 4.3本章小结

这个段落主要介绍了一个类库的体系结构和功能模块。此类库是基于JS原生2D绘图API的一个面向对象的封装，为了提供更高层次的抽象和交互功能。分为以下几个模块：

总体模块：定义了一个命名空间Mano，用于封装类库的所有全局属性和方法。其中包括一个Canvas类，用于表示画布元素，维护图元的状态和渲染。

图元模块：定义了一个Graphic类，用于表示所有的图元对象，继承了EventTarget类，实现了事件交互的功能。Graphic类有多个子类，分别对应不同的图形类型，如Rect，Arc，Text等。Graphic类还有一些属性和方法，用于控制图元的样式，如颜色，阴影，字体，变换等。

填充类模块：定义了一个Fillable类，用于表示可以填充图元的类型，如颜色，渐变，图案等。Fillable类有多个子类，分别对应不同的填充方式，如Color，Gradient，Pattern等。Color类又有多个子类，分别对应不同的颜色表示方式，如Rgb，Hsl等。

像素图像处理模块：定义了一个类，用于操作画布上的像素数据，提供了读取和写入像素颜色的方法，以及一些其他的属性，如颜色格式，高度，宽度等。

动画模块：定义了一个类，用于实现图元的动画效果，提供了一些控制动画播放的方法，如开始，暂停，取消等。还定义了两个接口，用于规范用户输入的动画状态和配置信息。

# 参考文献

[1] Edward Angel,Dave Shreiner.Interactive Computer Graphics A Top-Down Approach with WebGL,Seventh Edition[M].PEARSON,2016,(03).

[2] Donald Hearn,M.Pauline Baker,Warren R.Carithers.Computer Graphics with OpenGL fourth Edition[M].PEARSON,2014, (03).

[3] John E.Hughes,Andries van Dam,Morgan Meguire,David F.Sklar,James D.Foley, Steven K.Feiner.Computer Graphics Principles and Practice Third Edition[M]. PEARSON,2018.

[4] Matt Frisbie.Professional JavaScript for Web Developers,4th Edition[M]. TURING,2020, (02).

[5] J.David Eisenberg,Amelia Bellamy-Royds.SVG Essentials[M].O`REILLY,2015, (01).

[6] David Flanagan.JavaScript: The Definitive Guide[M].O’REILLY,2012, (01).

[7] Allen Wirfs-Brock,Brendan Eich. JavaScript 20 years[M]. Broadview,2021,(01).

[8] Anders Hejlsberg. TypeScript Documentation[Z].

[9] ???. MDN[Z].

[10] ???. Google Developer[Z].