# 目录

[摘 要 I](#_Toc22869)

[Abstract II](#_Toc6240)

[目 录 1](#_Toc3951)

[第1章 概 述 4](#_Toc11035)

[1.1 Web开发背景 4](#_Toc26730)

[1.1.1 Web开发的主要方向 6](#_Toc21259)

[1.1.2 2d绘图在Web开发中的地位 6](#_Toc21259)

[1.2 本类库的优势 5](#_Toc5498)

[1.2.1 现有的2d绘图类库 6](#_Toc21259)

[1.2.2 本类库的设计主旨 6](#_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 16](#_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 18](#_Toc6908)

[3.2.1 总体模块 18](#_Toc22642)

[3.2.2 图元 18](#_Toc22642)

[3.2.2.1 图元封装 18](#_Toc27583)

[3.2.2.2 属性封装 18](#_Toc27583)

[3.2.2.3 事件交互 18](#_Toc27583)

[3.2.3 填充类封装 18](#_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 22](#_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 35](#_Toc14857)

[5.1.1 类库使用场景 35](#_Toc22642)

[5.1.2 面对各种问题的使用方法 35](#_Toc515)

[5.1.2.1 建立场景 35](#_Toc27583)

[5.1.2.2 绘制对象 35](#_Toc27583)

[5.1.2.3 设置属性 35](#_Toc27583)

[5.1.2.4 添加交互 35](#_Toc27583)

[5.1.2.5 添加动画 35](#_Toc27583)

[5.1.3 使用的效果 35](#_Toc22642)

[5.2 类库的扩展插件写法 39](#_Toc19144)

[5.2.1 自定义图元插件的编写 39](#_Toc17464)

[5.2.2 自定义颜色的编写 39](#_Toc15956)

[5.3 本章小结 42](#_Toc2233)

[结 论 43](#_Toc5015)

[参考文献 44](#_Toc28629)

[致 谢 45](#_Toc28629)

# 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

在当今这个日益发展的年代中，大多数的产业向高新科技靠拢，其中高新科技向互联网领域靠拢。在网络的发展中，网络开发以及网络前端开发占据到了举足轻重的地位。众所周知，2015年是网络开发的转折点，因为2015年前后，技术上有着重大的改革及发展。前端由过去的单调的，局限性的开发变为全领域的，复杂的，随着底层技术的突破，诸多技术应运而出，产生了不同种新兴的编程思想，企业在开发过程中也与时俱进，摒弃了曾经的部分软件编程观念，转而用新的观念进行代替，其中最大的特点就是前端由客户端向服务器端的转化，随着本地JavaScript运行时的发展和优化，以及大量优秀的第三方包的加入，使得前端曾经的以HTML和CSS为首的架构逐渐转化为后端以Note.js和其他框架组合为主的架构。与此同时底层技术的发展也在不断的迭代，2015年之后JavaScript的es2015版本发行，将曾经的复杂的 ，难以融合于软件工程项目的JavaScript的脚本语言转化成为了一个面向对象的具有良好的模块化系统的支持异步多线程的可以更好利用客户端资源的编程语言。这致使web的整体发展由过去的静态页面为辅，服务器控制动态效果为主的传统架构改变为时事和服务器通信，且可以将前端内容从后端完全分离的新型架构。大量的框架的引进，其拓展性让更多的第三方插件得到更好的传播和应用，使代码的可复用性提高一个层次，甚至少量代码即可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日益发展，前端对于客户端资源的占用以及其安全性也不断增强。最新一批的JavaScript API主要从几个方面对客户机进行控制：以内部或外部硬件管理为主的控制，例如对于外部硬件及其接口的控制，以及对于GPU的控制，对于音频输出的控制，虚拟储存及虚拟运行汇编语言，对于物理感应器的控制。也通过了诸多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。

在此之中，控制外部硬件为主的API的代表有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。对于GPU的控制前端的权限目前并不多，主要权限集中于WebGPU API，次要的权限集中于WebGL和WebGL2 API。后两者则是人们熟知的音频节点和WebAssembly。

传统的前端通常以制作UI界面为主，直到目前前端的UI界面依然是非常重要的组成部分之一，追逐美观大气的界面以及合乎用户需求的图形系统，也是软件开发的过程中至关重要的一环，对于优秀的页面和效果的追求，致使人们在前端计算机图形学的道路上不断的发展，也使上述诸多图形学API得到重视。直到目前为止，市面上较为常见的制作动画以及优化页面图形效果的方法主要有以下几种：利用CSS的动画和渐变模块进行优化，利用JS诸多渐变和动画API进行优化，例如Animation API，利用2D绘图和SVG等绘图API来对网页的2D动画效果进行优化，之后则是上述的三种3D绘图API。

倘若仔细分析这几种动画的制作方式不难发现一个问题，CSS的动画和渐变实际上是设置属性，其动画效果在内部运行。原生和框架中封装好的动画API也是设置属性在内部运行。对于SVG而言，有专用的SMIL动画语言以及浏览器较好的支持和接口帮助。只有对于2D绘图API，3D绘图API没有直接支持动画运行，在开发过程中通常是手写动画效果。还有另一个问题就是这两种API是基于面向过程的，在管理以及开发大型应用上表现复杂且无法直接接入DOM与用户交互。正是因为开发过程中的需要与实际所拥有的代码工具不接轨，所以矛盾便应运而生，那就是如何将这两种API转化为面向对象的，可与用户交互的，支持动画的。

## 1.2 本类库的优势

### 1.2.1 现有的2d绘图类库

对于将面向对象语言中的部分面向过程的底层API面向对象化是一个比较老生常谈的话题，因为语言设计之初需要考虑语言的功能设定是否单一，是否可复用，而且诸多API之间不能出现太复杂的交集，才能降低耦合性。但是代价就是使用起来往往不方便。以下列举了市面上的部分已有的2D绘图类库。

Two.js：Two.js 是一个用于绘制现代的平面动画的类库，它支持多种渲染方式，包括 SVG、Canvas 和 WebGL。Two.js 的特点是提供了一个简单而强大的对象模型，可以让开发者方便地创建和管理图形对象，以及对它们进行变换、分组、嵌套、动画等操作。Two.js 还提供了一个 SVG 解释器，可以将 SVG 文件导入到 Two.js 的场景中。

Konva：Konva 是一个用于桌面和移动应用的 Canvas 2D 类库，它提供了一个面向对象的 API，可以让开发者创建和操作各种图形、文本、图片、视频等。Konva 的优点是提供了高性能的渲染、事件处理、动画、滤镜、裁剪、拖放等功能，以及对多个 Canvas 层的支持。Konva 还提供了一些扩展模块，如 Konva React、Konva Vue 和 Konva Angular，可以让开发者在不同的前端框架中使用 Konva。

LeaferJS：LeaferJS 是一个绚丽多彩的 Canvas 2D 类库，它具备瞬间创建 100 万个图形的超强能力，可以结合 AI 进行绘图，生成界面。LeaferJS 的特点是提供了一个灵活而强大的图形系统，可以让开发者自由地定义图形的属性、样式、动画、事件等。LeaferJS 还提供了一些高级功能，如渐变、阴影、混合、裁剪、剪贴板、导出等。

Paper.js：Paper.js 是一个用于矢量图形的类库，它基于 HTML5 的 Canvas 元素，但提供了一个类似于 SVG 的 API，可以让开发者以对象的方式操作图形。Paper.js 的特点是提供了一个强大的数学引擎，可以让开发者轻松地实现各种图形学操作，如路径、曲线、变换、布尔运算、几何运算等。Paper.js 还提供了一个交互式的脚本环境，可以让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

上述诸多类库较为成熟完整，且功能丰富，但是也有部分问题。

其一，他们并没有最大程度的实现与原生语法的对接。设计类库应尽量与原生语法最大程度的对接，才能让用户更快速度的学习应用。类库是一种封装和复用代码的方式，可以提高开发效率和代码质量。类库的设计者需要考虑很多因素，如功能、性能、可读性、可维护性、可扩展性、兼容性等。其中一个重要的因素是与原生语法的对接，即类库的 API 应该尽可能地遵循和利用原生语言的语法、风格和习惯，以提供一致和自然的编程体验。

其二，这些类库在设计之时更多在乎如何抽象出一个更高层次的代码层级，而不致力于保持每一个操作的原子性和可复用性。类库的 API 应该尽可能地提供一些简单、完整、独立、通用的操作，以提供一致和高效的编程体验。保持原子性可以避免一些潜在的错误和异常，如数据不一致、资源泄漏、状态不同步等，还增强类库的可组合性，因为可以将多个操作组合起来实现更复杂的功能，而不需要担心操作之间的干扰和冲突，一个类负责一个功能，即使出现新的扩展功能也可以通过已有的概念快速组装。保持可复用性可以扩展类库的功能和适用范围，因为可以将一个操作应用到不同的对象和数据上，以实现不同的目的和效果。

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

本类库是一个用于在浏览器中绘制和操作 2D 图形的 JavaScript 类库，其核心功能一定是绘图功能以及绘图相关的其他功能，还有必须的数学运算以及其他工具类，由于本类库功能按照设计主旨设计，所以只会封装最低程度的代码以保持器扩展性。所需功能还需要与用户交互，所以需要依赖于事件类进行封装。其余功能仅仅封装成面向对象，不做过多修改，同时需要动画功能。详细模块划分如下：

图形学转换和数学实现：

负责处理图形的各种变换，例如平移、旋转、缩放等。使用数学算法确保图形在屏幕上正确显示。例如，用户想要将一个矩形旋转90度，这个模块会计算新的顶点坐标，以正确绘制旋转后的矩形。

图形绘制面向对象化：

使用面向对象的方法绘制图形，使代码更易于维护和扩展。创建图形对象，设置属性（例如颜色、线宽），并在屏幕上绘制它们。例如，创建一个蓝色的矩形对象并绘制在屏幕上。此模块进一步分为三个小模块：图形样式：定义图形的样式，例如线宽、填充颜色、边框颜色。文字样式：管理图形中的文本样式，例如字体、大小、颜色。图形内操作：处理图形内部的元素，例如组合、裁剪。用户可以选择将多个图形组合成一个复杂的图形，或者裁剪一个图形的一部分。

颜色封装：

管理图形的颜色属性，使其更具可定制性。用户可以设置图形的填充颜色、边框颜色等。例如，将矩形的填充颜色设置为红色，边框颜色设置为黑色。

像素和图像处理：

处理像素级别的图像操作，例如滤镜、亮度调整等。用户可以模糊图像、改变色调或增加饱和度。例如，应用黑白滤镜或增加图像的对比度。

动画：

实现图形的平滑过渡和动态效果。创建平移、缩放、旋转等动画。例如，创建一个渐变动画，使矩形从左到右平移。

DOM事件：

处理与图形相关的DOM事件，例如点击、拖动等。用户可以点击图形选择它，或者拖动图形改变位置。例如，点击矩形以更改其颜色。

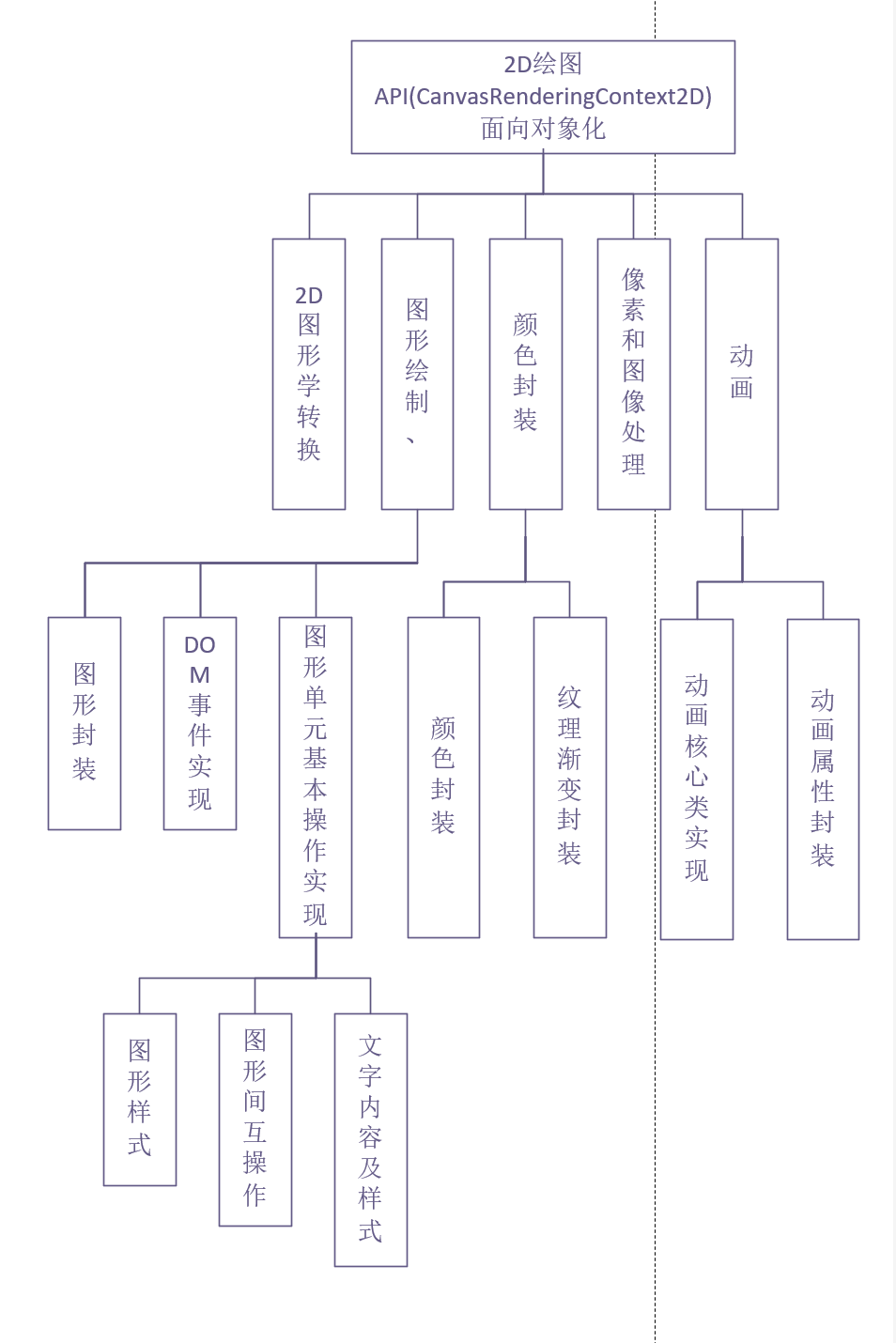
具体功能模块图如图1.3.1。

图1.3.1 功能模块图

# 第2章 2d计算机图形学相关技术

## 2.1 语言介绍

### 2.1.1 JavaScript语言介绍

参考，JavaScript权威指南6th

红宝书4th

js语言精粹

mdn

### 2.1.2 TypeScript语言介绍（再参考点ts其他书籍）

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 NodeJS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。它允许我们通过 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。

## 2.3 计算机图形学2d部分的原理介绍

### 2.3.1 二维坐标系统

参考：计算机图形学，基于webgl的交互式计算机图形学，3D图形系统设计与实现

### 2.3.2 二维图形学变换

### 2.3.3 线性插值算法（点及颜色）

### 2.3.4 贝塞尔曲线原理

### 2.3.5 填充算法

### 2.3.6 剪裁算法及锯齿直线渲染

## 2.4 js对2d计算机图形学的实现

### 2.4.1 图元的实现

参考：MDN，JavaScript权威指南6，红宝书4

### 2.4.2 图形学转换的实现

### 2.4.3 颜色的实现

### 2.4.4 图像的引入

## 2.5 本章小结

# 第3章 类库总体设计

## 3.1 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。

### 3.1.1 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多个属性和50多个方法，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。

3、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。

4、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。

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。

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。

动画实现可以在后期迭代更新和维护中将图元类进行拓展，将类库注册为一个单独的XML命名空间后封装成标签，这样就可以令图元类继承于Node和Element，后续可以通过拓展的方式衔接到Animation API。如果在没实现封装标签的情况下，则可以定义一个动画类利用requestAnimationFrame()绘制动画。

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。具体的功能封装设计图如图3-1所示。

### 3.1.2 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。

转化为面向对象，那就一定要继承JS所有对象的基类Object类。同时我们形有交互的需求，既然有用户交互的需求，就一定要继承事件类，即为EventTarget类。每一个图源都有公用的属性和方法，并且每一个图元都图元类，所以说我们要将其继承于图元基类，在这里我们定其为Graphic类。到这里我们每一个图元类的大致继承链就可以确定了，即为某个具体图元类继承于Graphic，Graphic继承于EventTarget，EventTarget继承于Object。例如，一个矩形图元Rect类，因为它是一个图源，拥有所有图源共用的属性和方法，所以它要继承于Graphic类，当用户想要为他注册点击事件或者是其他的交互事件的时候，根据用户的操作，要触发相应的函数，所以他要继承于EventTarget，从根，他属于一个对象，所以他要继承于Object基类，所以Rect类实例对象继承链为：Rect实例对象->Rect->Graphic->EventTarget->Object。

一个图源对象中可以设置诸多的属性，2D绘图原生API中，属性大约分之为以下几类：设置阴影，设置字体，设置文字格式，设置边框绘制。这四类属性是通用的，所以我们可以将这四个属性设置于图元基类之中。就四个属性的类型，都是相应的类，比如阴影属性对应的阴影类，字体属性对应字体类。所以四个属性的类和图形基类产生了直接依赖关系。

属性中还有一个比较需要注意的点是图形变换相关功能。图形学变换是在绘图过程中较为常用的一种算法功能。在于将一个图形进行刚体的或非刚体的变换。其本质在于图形由点所组成，利用变换矩阵作用于图形的点，从而致使整个图形的变化。在此之中，关键在于两点，一个是组成图形的点，还有一个是致使点变换的矩阵如何表示。对于点的表示，一个是利用坐标数字进行表示，另一个是利用JS API中的DOMPoint或DOMReadonlyPoint类来表示。对于矩阵的表示可以使用二维数组或者用JS API中的DOMMatrix类来表示。关于图形学变换的矩阵计算操作，可以利用JS API的DOMMatrix类中的关于矩阵的转换运算操作直接进行实现。是一种性价比比较高，且与原生语法契合比较密的一种实现方式，所以在此我们将选择这一种实现方式。所以关于图源基类的转换属性，转换属性的类型是DOMMatrix类，于是图元基类与DOMMatrix形成了直接依赖关系。

图元相关模块的类及其关系介绍到此几乎完毕。

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。可以细分为纯颜色静态属性、Rgb和Hsl，十六进制色等。就这一特点我们可以大致设计出颜色中的类和诸多子类之间的关系，例如我们在这里设颜色基类为Color类，颜色语法我们将其设置为其子类，具体关系请见下文。

纯颜色静态属性是指一些常见的颜色，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，所以说颜色名称为Color类的属性而存在。

Rgb子类是基于RGB（红绿蓝）颜色模型的一种颜色子类。我们将Rgb单独设计成一个子类，继承于Color，为了与其他颜色类区分，我们可以设置其类型为rgb。

Hsl子类则是基于HSL（色相、饱和度、亮度）颜色模型的另一种颜色子类。通过调整色相、饱和度和亮度这三个属性的值，创建颜色。我们将Hsl单独设计成一个子类，继承于Color，为了与其他颜色类区分，我们可以设置其类型为hsl。

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。

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，例如用户自定义图像作为图案的话，则此类便与HTMLImageElement引起关联，如果用户引入视频的话，则此类便与HTMLVideoElement引起关联，如果用户引入其他画布的话，则此类便与HTMLCanvasElement引起关联，如果是离屏画布，则是OffscreenCanvas类，如果用户引入位图的话，则此类便与ImageBitmap引起关联，如此诸多，不一一列举。

3、像素操模块分析

关于像素操作，其内部的颜色数据我们通常用类型化数组（Typed Array）来表示，原因是像素的颜色由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，所以需要类型化数组。类型化数组单个元素内存占用大小根据计算机基础数据类型分为几类：8位，16位，32位，64位，数字类型分为整，非负整数，浮点数。像素数据类型通常为Uint8Array和Uint8ClampedArray两者的区别主要在于如果给Uint8ClampedArray设置一个0-255范围之外的值，它将简单地默认为0或255，而Uint8Array只会取数值二进制值的前8位。所以我们可以断定像素操作和类型化数组类有依赖关系。

4、动画模块分析

动画模块则是基于实现的不同而不同，如果后期迭代开发将图元封装成一个XML标签（特征是继承了Element类）的话，则可以推断，图元继承链改为：Graphic->Element->Node->EventTarget->Object，则与Element和Node两个类产生了继承关系。同时依赖了Animation API的诸多类。如果不封装为标签模式的话需要自行实现动画过渡效果，则需要依赖部分原生语法。

## 3.2 类库的功能模块设计

在上文，我们已经对模块进行了划分，大致进行了分类，并且理清了其中的依赖和继承关系。承接上文，我们对每一个模块的结构做大致设计

### 3.2.1 总体模块

首先我们需要指定一个命名空间，用于为我们所书写的诸多类腾出空位，其根本原因在于JS对于全局变量的控制。JS本身的机制运行于全局对象之上，所以盲目的在全局对象上进行声明和覆盖会修改预定义类，从而导致意料之外的问题，随着版本迭代，更多的新的类出现后，容易出现后期问题。于是我们定义一个对象或一个类，可以暂时称之为Lynette，也是本框架的名称和XML命名空间。里面封装了与此类库有关的所有全局属性方法，例如下一个要介绍的一个类，Canvas类。此框架在设计之初写用例的时候就已经设计了这一种代码风格，例如Graphic类，在JS全局对象中应该先引用Lynette后书写Lynette.Graphic即可。

Canvas类是画布类型，2d绘图之中，所有操作固然在画布上进行。画布需要维护一个属性就是目前已绘制的所有图元，再就是其他的全局状态，例如是否开启全局抗锯齿，以及图元融合的算法指定。方法比较重要的是渲染图元，用于渲染目前已有的所有图元。再比较重要的就是添加删除图元方法。再就是清空画布和画布默认重绘颜色，类型可以是Color类­

### 3.2.2 图元

根据以上描述，我们可以对图元模块进行一个大致设计。首先分析是JS诸多已有的语法，EventTarget类已经继承于Object类，DOMMatrix和DOMPoint在高版本JS中属于内置语法， 这几个类的定义和引用被分到JS原生语法包中。

以下为图元包的其他内容。

#### 3.2.2.1 图元封装

图元包首先是基类，基类可以创建一个Graphic类继承于EventTarget。其次是定义每一个图元子类，子类均继承于Graphic基类，我们可以根据已有的图形算法定义图元子类，比如rect()可以封装为Rect类，参数可以化之为属性，每一个图形都有不同的参数，所以可以将这些与众不同的参数定义为图元子类的属性。于此相似的子类有：Arc，Ellipse，Line，BezierCurve，Rect，RoundRect，Text。同时我们需要让用户可以自定义图形，我们添加一个CustomGraphic子类。这些类和子类构成了图元类和子类的基本结构。

#### 3.2.2.2 属性封装

根据已有条件我们可以大致确定图元基类的属性和方法。首先需要一个set get content属性，用于设置一个图元内内容的文本。其次需要textFormat属性，类型为TextFormat类，shadow属性类型为Shadow类，border属性类型为Border类，font属性类型为Font类，transform属性类型为DOMMatrix类，这些均为控制样式的属性。还需要fillType属性，设置为FILLL\_TYPE枚举类型，描述图元绘制方式。backgroundColor属性和color为Fillable类，用于设置文字和背景的填充。后续迭代中，我们会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，属性可以大概设置为控制 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，属性大概可以分值为阴影模糊度，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。

同时我们需要编写Border类，用于控制框样式。属性大概分为，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。

我们需要编写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枚举类型主要用制绘制图源的两种不同的方法。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，后者是仅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。

#### 3.2.2.3 事件交互

我们需要与事件交互，前文中，我们已经断定了图源基类要继承于EventTarget类，EventTarget类是DOM2规范中的一个类，是用于注册事件，触发事件以及维护事件的一个类，内部封装了addEventListener()，dispatchEvent()，removeEventListener()等DOM2事件核心方法，我们给用户的一个注册可选项就是利用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，所以实现DOM0事件注册方法的方法之一就是监测所有以on开头的属性的赋值和访问，并且同名事件注册至DOM2事件中。事件触发DOM2的事件注册触发。

由于图源本身并不是一个对象，所以他也不能直接的注册事件，是人为的把他封装成了对象，并且给他赋予可注册事件对象的特征，所以事件的触发要手动触发，但是绘图API中只有画布可以直接的进行事件注册，因为它属于HTML元素对象。我们的实现方法是将所有的事件转而注册至画布对象中，利用路径检测方法检测当前点击坐标是否属于此图元，从而触发事件。但本身，其不需要多余的类来支持，仅需要在画布上注册事件并绑定住默认触发事件，在每一次触发事件的时候自动执行一次默认触发事件即可，每一次触发事件均遍历所有图元和事件，如果在图元数量比较大的时候可以通过大概轮廓进行排除，从而优化时间复杂度。

### 3.2.3 填充类封装

可填充类可以设置一个基类，名称直接命名为Fillable即可。后写三个衍生类，第一个用于表示颜色，即为Color类，第二个用于表示渐变，即为Gradient类，第三个用于表示图像，即为Parttern类。

#### 3.2.3.1 颜色类

第一个是Color类，Color有多种方法表示，所以Color类有多个子类，首先，Color类应该有所有关键字颜色所有常量属性，并为其设置初始值，其次，要有一个type属性用于区分子类类型。Color类应该有一个子类为Rgb，Rgb有四个属性用于表示三个分量和透明度，类型可以设置为rgb，若果不设置透明度则自动设置为1。Hsl同理，四个属性表示三个分量和透明度，类型可以设置为hsl，如果需要封装其他颜色标识方法可以后续添加。当然，在封装的过程中如果需要统一管理类型值的话，我们需要额外添加一个枚举类型。

#### 3.2.3.2 图案类

第二个是图案类，需要一个属性来记录用户选择的图像，这个属性的类型要包括所有种类用户可能选择的类型，还有一个方法负责添加图像。还有一个属性负责管理图像的重复方法，例如双轴循环，x轴循环，不循环。

#### 3.2.3.3 渐变类

第三个是渐变类。一个渐变通常需要两个以上的颜色和对应颜色的偏移量，所以我们需要定义一个添加颜色和偏移量的方法于渐变类。还需要一个数组储存传入颜色和偏移量的值。渐变类有三个衍生类，第一个线性渐变，需要两个属性，即为开始和结束点的坐标，可以用DOMPoint和数字表示。第二个是辐射渐变，可以有四个属性，四个属性是顶点坐标，类型与上同，还有两个是辐射园半径，一个是内圆一个是外圆，均可以用数字表示。第三个是角度渐变，需要两个属性，两个是中心点坐标，一个是开始转角，可以用数字表示。

### 3.2.4 像素图像处理

像素图像处理主要工作是记录像素数据，并提供需改渠道。所以需要一个属性来记录底层颜色数据，为保证数据完好，可以设置属性类型为Uint8ClampedArray，以方便与底层对接，我们同时需要提供两个方法，一个是用于获取像素颜色，一个是用于设置像素颜色，这两个方法需要提供一个共同的参数，像素坐标点。如果写入像素颜色的话则需要一个新的颜色作为参数，这个参数类型可以是Color或一组数字。需要一个字符串用于表示底层颜色格式，此颜色格式需要在画布创建时被写入。还有一个高度值，和一个宽度值，这是两个数字类型属性。

### 3.2.5 动画封装

在封装动画模块的时候我们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，所以我们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。所以动画封装的实质是一个插值参数方程计算器。基于这一思想，我们创建一个大概得动画架构。动画封装可以由一个类和两个接口组成。一个类为动画模块的核心，封装了所有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。

动画类属性大致为动画状态属性，动画信息属性，其他属性。状态属性例如：是否已经结束，是否在等待，是否准备好，均为布尔值，播放状态为枚举值。信息属性为当前时间，动画id，播放速率等，均为数字。还需要一个时间线，它是运用了DocumentTimeline类型。方法中需要控制动画播放的方法，例如开始，暂停，取消，反方向播放，提前结束。还有更新播放速率的方法。

两个接口其中用于记录不同时刻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，我们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。

## 3.3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Lynette，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。