**文档密级：绝密**

**Lugia Mega**

**标准、高效、开箱即用的前端可视化开发工具**

**建设规划及设计实现说明书**

**（v1.x）**

目录

[一、 前言 1](#_Toc18680_WPSOffice_Level1)

[1． 背景 1](#_Toc5028_WPSOffice_Level2)

[2． 前端工程化 1](#_Toc26689_WPSOffice_Level2)

[3． 为什么要做lugia-mega？ 1](#_Toc26765_WPSOffice_Level2)

[4． 特性 4](#_Toc26676_WPSOffice_Level2)

[5． 愿景 4](#_Toc14032_WPSOffice_Level2)

[二、 功能设计 5](#_Toc5028_WPSOffice_Level1)

[1． 功能组成 5](#_Toc13493_WPSOffice_Level2)

[（1） 项目管理 5](#_Toc5028_WPSOffice_Level3)

[（2） 页面设计器 6](#_Toc26689_WPSOffice_Level3)

[（3） 物料体系 6](#_Toc26765_WPSOffice_Level3)

[（4） 开发工具链 7](#_Toc26676_WPSOffice_Level3)

[（5） 模型规范 8](#_Toc14032_WPSOffice_Level3)

[2． 优点 9](#_Toc18055_WPSOffice_Level2)

[3． 如何使用 9](#_Toc21632_WPSOffice_Level2)

[4． 下载 9](#_Toc28283_WPSOffice_Level2)

[5． 安装 10](#_Toc27206_WPSOffice_Level2)

[（1） Windows 64系统版本的客户端 10](#_Toc13493_WPSOffice_Level3)

[（2） macOS系统的客户端 10](#_Toc18055_WPSOffice_Level3)

[6． 服务手册 10](#_Toc6496_WPSOffice_Level2)

[7． 浏览器兼容性 10](#_Toc25999_WPSOffice_Level2)

[8． 名词解释 11](#_Toc29114_WPSOffice_Level2)

[三、 原型设计 13](#_Toc26689_WPSOffice_Level1)

[1． 启动页 13](#_Toc17282_WPSOffice_Level2)

[2． 主界面 13](#_Toc20989_WPSOffice_Level2)

[（1） 主界面-项目 13](#_Toc21632_WPSOffice_Level3)

[（2） 主界面-项目-新建页面 15](#_Toc28283_WPSOffice_Level3)

[（3） 主界面-模板 17](#_Toc27206_WPSOffice_Level3)

[（4） 主界面-模板-使用 18](#_Toc6496_WPSOffice_Level3)

[（5） 主界面-区块 19](#_Toc25999_WPSOffice_Level3)

[（6） 主界面-区块-自定义区块 1 20](#_Toc29114_WPSOffice_Level3)

[（7） 主界面-区块-自定义区块 2 21](#_Toc17282_WPSOffice_Level3)

[（8） 主界面-区块-自定义区块 3 22](#_Toc20989_WPSOffice_Level3)

[3． 更多 23](#_Toc9991_WPSOffice_Level2)

[（1） 物料源设置 23](#_Toc9991_WPSOffice_Level3)

[（2） 项目默认配置 24](#_Toc17068_WPSOffice_Level3)

[（3） 通用设置 24](#_Toc2784_WPSOffice_Level3)

[（4） 关于 25](#_Toc24201_WPSOffice_Level3)

[四、 交互与详细设计 26](#_Toc26765_WPSOffice_Level1)

[1． 主页 26](#_Toc17068_WPSOffice_Level2)

[2． 创建 26](#_Toc2784_WPSOffice_Level2)

[3． 项目页 26](#_Toc24201_WPSOffice_Level2)

[4． 工具栏 27](#_Toc9647_WPSOffice_Level2)

[5． 左边栏 28](#_Toc29963_WPSOffice_Level2)

[（2） 区块面板 29](#_Toc9647_WPSOffice_Level3)

[（3） 组件面板 29](#_Toc29963_WPSOffice_Level3)

[（4） 图标面板 29](#_Toc5889_WPSOffice_Level3)

[6． 右边栏 30](#_Toc5889_WPSOffice_Level2)

[（1） 检查器面板 30](#_Toc17842_WPSOffice_Level3)

[（2） 页面 30](#_Toc28116_WPSOffice_Level3)

[（3） 属性 31](#_Toc17152_WPSOffice_Level3)

[（4） 事件与模型 32](#_Toc23605_WPSOffice_Level3)

[7． 组件操作 32](#_Toc17842_WPSOffice_Level2)

[（1） 选择组件 33](#_Toc533_WPSOffice_Level3)

[（3） 同时选择多个组件 33](#_Toc18667_WPSOffice_Level3)

[（4） 移动与复制组件 34](#_Toc25660_WPSOffice_Level3)

[（6） 删除与撤销 35](#_Toc14145_WPSOffice_Level3)

[（7） 编辑组件 35](#_Toc12790_WPSOffice_Level3)

[8． 检查器 36](#_Toc28116_WPSOffice_Level2)

[（2） 自定义 37](#_Toc32375_WPSOffice_Level3)

[（3） 外观 37](#_Toc15777_WPSOffice_Level3)

[（4） 锁定宽高比 38](#_Toc11785_WPSOffice_Level3)

[（5） 描边方向 38](#_Toc18060_WPSOffice_Level3)

[（6） 关闭属性 38](#_Toc4222_WPSOffice_Level3)

[（7） 颜色编辑器 39](#_Toc17959_WPSOffice_Level3)

[（9） 下拉选择 41](#_Toc16606_WPSOffice_Level3)

[（10） 文字移动 41](#_Toc15127_WPSOffice_Level3)

[（11） 数据录入 41](#_Toc5376_WPSOffice_Level3)

[（12） 图标 42](#_Toc22135_WPSOffice_Level3)

[9． 画布 44](#_Toc15777_WPSOffice_Level2)

[（1） 参考线与对齐简述 45](#_Toc27058_WPSOffice_Level3)

[（2） 参考线 45](#_Toc6540_WPSOffice_Level3)

[（3） 对齐 48](#_Toc15370_WPSOffice_Level3)

[（4） 窗体缩放 52](#_Toc12893_WPSOffice_Level3)

[（5） 窗体移动 52](#_Toc6706_WPSOffice_Level3)

[10． 上边栏 53](#_Toc11785_WPSOffice_Level2)

[（1） 返回项目 53](#_Toc5559_WPSOffice_Level3)

[（2） 新建项目 54](#_Toc4041_WPSOffice_Level3)

[（3） 保存项目 55](#_Toc2725_WPSOffice_Level3)

[11． Model管理 56](#_Toc18060_WPSOffice_Level2)

[12． 项目管理 56](#_Toc4222_WPSOffice_Level2)

[13． 设置 57](#_Toc17959_WPSOffice_Level2)

[（1） 物料源 57](#_Toc4615_WPSOffice_Level3)

[（2） 项目默认配置 58](#_Toc14958_WPSOffice_Level3)

[（3） 通用设置 58](#_Toc21062_WPSOffice_Level3)

[（4） 关于 59](#_Toc12676_WPSOffice_Level3)

[14． 导出 60](#_Toc16606_WPSOffice_Level2)

[（1） mega提供开发现代单页 React 应用需要的所有环境： 60](#_Toc20904_WPSOffice_Level3)

[（2） 代码拆分（Code Splitting） 61](#_Toc24857_WPSOffice_Level3)

[15． 父容器（编组） 61](#_Toc15127_WPSOffice_Level2)

[（2） 编组父容器 62](#_Toc9951_WPSOffice_Level3)

[16． 模型绑定 63](#_Toc5376_WPSOffice_Level2)

[（1） 操作 63](#_Toc28211_WPSOffice_Level3)

[17． 快捷键 65](#_Toc22135_WPSOffice_Level2)

[（1） 通用快捷键 65](#_Toc12146_WPSOffice_Level3)

[（2） 移动图层 & 更改尺寸 65](#_Toc17009_WPSOffice_Level3)

[五、 实现方案 66](#_Toc15777_WPSOffice_Level1)

[1． 整体方案 66](#_Toc27058_WPSOffice_Level2)

[2． 用户使用流程 67](#_Toc6540_WPSOffice_Level2)

[（1） 设计师/产品经理 67](#_Toc28660_WPSOffice_Level3)

[（2） 开发工程师 67](#_Toc13277_WPSOffice_Level3)

[（3） 功能划分 68](#_Toc17146_WPSOffice_Level3)

[3． 实现技术 68](#_Toc15370_WPSOffice_Level2)

[（1） 底层依赖 68](#_Toc1159_WPSOffice_Level3)

[（2） 内部模块 69](#_Toc13538_WPSOffice_Level3)

[（3） 相关 69](#_Toc16315_WPSOffice_Level3)

[4． 项目规范 70](#_Toc12893_WPSOffice_Level2)

[（1） 目录结构 70](#_Toc6818_WPSOffice_Level3)

[（2） 工作区 71](#_Toc27144_WPSOffice_Level3)

[（3） 安装依赖 72](#_Toc9269_WPSOffice_Level3)

[（4） 文件用途 74](#_Toc25214_WPSOffice_Level3)

[（5） .lugiad页面文件规范 75](#_Toc28522_WPSOffice_Level3)

[5． 约定式路由 76](#_Toc6706_WPSOffice_Level2)

[（1） 基础路由 76](#_Toc1728_WPSOffice_Level3)

[（2） 动态路由 77](#_Toc1561_WPSOffice_Level3)

[（3） 可选的动态路由 78](#_Toc30957_WPSOffice_Level3)

[6． 自定义组件 78](#_Toc5559_WPSOffice_Level2)

[（1） 目录结构 78](#_Toc19209_WPSOffice_Level3)

[（2） 文件规范 79](#_Toc7518_WPSOffice_Level3)

[（3） 元信息规范 79](#_Toc13074_WPSOffice_Level3)

[7． 接入到mega 81](#_Toc4041_WPSOffice_Level2)

[（1） 页面接入 82](#_Toc22479_WPSOffice_Level3)

[（2） 官方组件库接入 82](#_Toc21565_WPSOffice_Level3)

[（3） 自定义组价接入 83](#_Toc2679_WPSOffice_Level3)

[（4） 模型接入 84](#_Toc25325_WPSOffice_Level3)

[六、 模块设计 86](#_Toc11785_WPSOffice_Level1)

[1． @lugia/mega-utils 86](#_Toc2725_WPSOffice_Level2)

[（1） 包含 86](#_Toc12043_WPSOffice_Level3)

[（2） 模块依赖 87](#_Toc13663_WPSOffice_Level3)

[（3） 使用 87](#_Toc24781_WPSOffice_Level3)

[（4） 相关模块 87](#_Toc26615_WPSOffice_Level3)

[2． @lugia/mega-webpack 88](#_Toc4615_WPSOffice_Level2)

[（1） 模块依赖 88](#_Toc10254_WPSOffice_Level3)

[（2） API 88](#_Toc20794_WPSOffice_Level3)

[（3） 相关模块 90](#_Toc6864_WPSOffice_Level3)

[3． @lugia/mega-jest 90](#_Toc14958_WPSOffice_Level2)

[（1） 在命令行中使用 90](#_Toc10860_WPSOffice_Level3)

[（2） 使用API 90](#_Toc17261_WPSOffice_Level3)

[4． @lugia/babel-preset-mega 91](#_Toc21062_WPSOffice_Level2)

[（1） 使用 91](#_Toc14877_WPSOffice_Level3)

[5． @lugia/devtools-material 92](#_Toc12676_WPSOffice_Level2)

[（1） 功能说明 92](#_Toc1381_WPSOffice_Level3)

[（2） 结构划分 92](#_Toc30770_WPSOffice_Level3)

[（3） 获取有效文件夹 93](#_Toc14795_WPSOffice_Level3)

[（4） 读取文件夹源信息 94](#_Toc7983_WPSOffice_Level3)

[（5） 输出索引信息 94](#_Toc18407_WPSOffice_Level3)

[（6） 在命令行中使用 94](#_Toc31943_WPSOffice_Level3)

[（7） 使用API 95](#_Toc25598_WPSOffice_Level3)

[6． @lugia/devtools-conversion 96](#_Toc20904_WPSOffice_Level2)

[（1） 功能说明 96](#_Toc9219_WPSOffice_Level3)

[（2） 结构划分 96](#_Toc11627_WPSOffice_Level3)

[（3） 解压LugiaD文件 97](#_Toc30073_WPSOffice_Level3)

[（4） 生成组件信息 97](#_Toc28269_WPSOffice_Level3)

[（5） 使用 100](#_Toc22124_WPSOffice_Level3)

[7． @lugia/mega-scripts 101](#_Toc24857_WPSOffice_Level2)

[（1） 功能说明 101](#_Toc4937_WPSOffice_Level3)

[（2） 模块依赖 101](#_Toc30165_WPSOffice_Level3)

[（3） 在命令行中使用 103](#_Toc23484_WPSOffice_Level3)

[（4） mock 服务 103](#_Toc13971_WPSOffice_Level3)

[（5） 性能优化 104](#_Toc8408_WPSOffice_Level3)

[（6） 相关模块 106](#_Toc20577_WPSOffice_Level3)

[8． @lugia/mega 106](#_Toc9951_WPSOffice_Level2)

[（1） 模块依赖 107](#_Toc6828_WPSOffice_Level3)

[（2） 在命令行中使用 107](#_Toc1350_WPSOffice_Level3)

[（3） 脚手架来源 107](#_Toc11384_WPSOffice_Level3)

[（4） 相关模块 108](#_Toc30606_WPSOffice_Level3)

[9． @lugia/lugiax 108](#_Toc28211_WPSOffice_Level2)

[（1） 设计思想 108](#_Toc13116_WPSOffice_Level3)

[（2） Action 109](#_Toc19603_WPSOffice_Level3)

[（3） API设计 111](#_Toc29142_WPSOffice_Level3)

[（4） lugiax-router路由 112](#_Toc27368_WPSOffice_Level3)

[10． @lugia/mega-desktop 114](#_Toc12146_WPSOffice_Level2)

[（1） 模块依赖 114](#_Toc20245_WPSOffice_Level3)

[七、 物料体系 115](#_Toc18060_WPSOffice_Level1)

[1． 元信息生成存储 115](#_Toc17009_WPSOffice_Level2)

[2． 物料库元信息规范 116](#_Toc28660_WPSOffice_Level2)

[3． 区块元信息规范说明 117](#_Toc13277_WPSOffice_Level2)

[4． 布局规范说明 121](#_Toc17146_WPSOffice_Level2)

[5． 脚手架规范说明 121](#_Toc1159_WPSOffice_Level2)

[6． 关于发布/下载 123](#_Toc13538_WPSOffice_Level2)

[（1） 发布 123](#_Toc22166_WPSOffice_Level3)

[（2） 下载 123](#_Toc32107_WPSOffice_Level3)

[7． 缓存机制 124](#_Toc16315_WPSOffice_Level2)

[（1） 缓存使用 124](#_Toc1168_WPSOffice_Level3)

[八、 附录 125](#_Toc4222_WPSOffice_Level1)

[1． Flow类型 125](#_Toc6818_WPSOffice_Level2)

# 前言

### 背景

* 前端技术更新迭代速度快，业务开发人员由于需要花费大量时间在业务实现上，对前端的技术选型、实现方案、用户体验无法有效把控。
* 前端开发人员不足、技术水平不高，无法保证效率和质量。
* 前端发展到现在，在技术实现和开发方式上已经有了成熟丰富、多种多样的方案；需要对这些方案进行整合优化，以提高开发效率、代码质量和用户体验。
* 一个前端项目需要产品、交互、设计、前端共同完成，沟通成本大，开发流程长。需要把产品、交互、设计、前端合为一个整体，节省项目开发时间，简化开发流程。

### 前端工程化

在前端工程化方向上，可视化、屏蔽底层、提供元信息式的开发方式已经成为主流

### 为什么要做 lugia-mega？

* 给低水平的前端开发人员赋能，降低开发成本，提升前端开发质量。

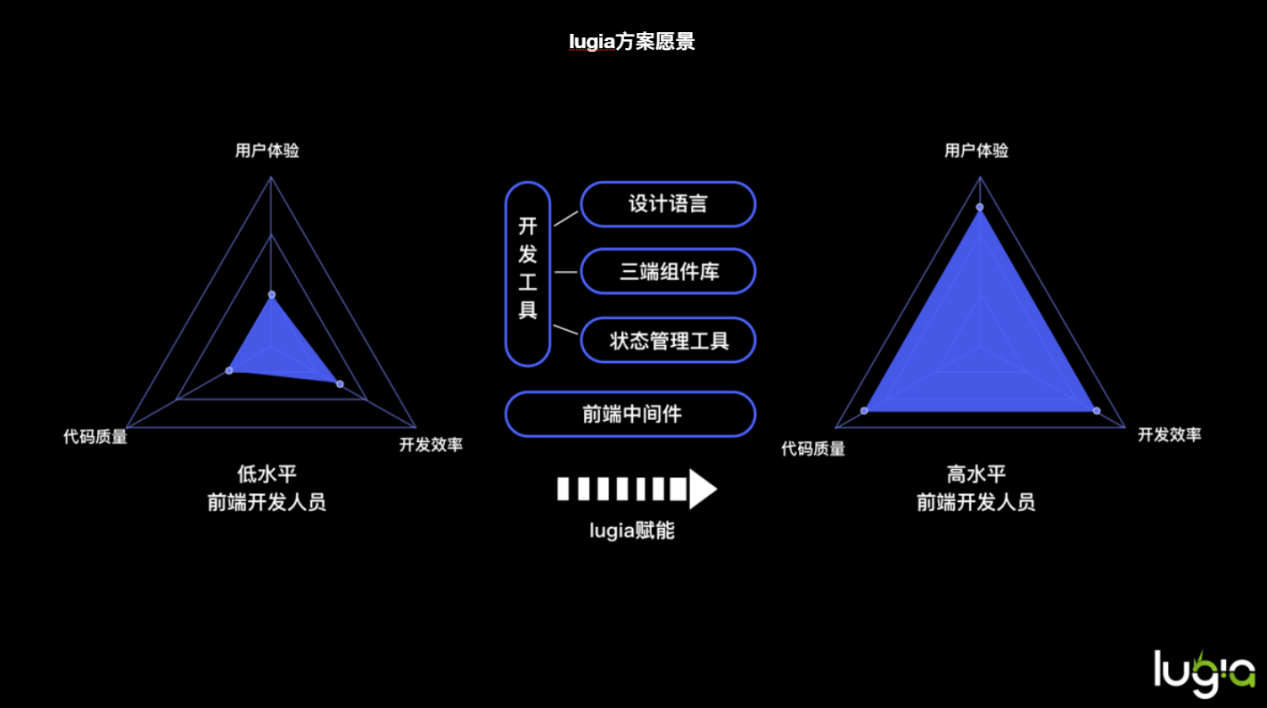


图1.1 lugia方案愿景

* 帮助设计师、产品经理快速设计原型、产品交互，成果可以直接让开发人员使用。
* 它也是“多端一体化”研发成果对外输出的主要承载体。

lugia-mega的终极目的是解决当前行业产品线上人员的杂糅。基于这款产品，产品交互，可以在应用上用简单的拖动就可以拼凑出一套完整的界面原型图。UI设计师可以通过交互模型上的原型图直接调整页面颜色样式字体。而对于研发，仅需要根据设计图作出相应的捆绑调试，就可以直接生成一套完整的项目流程。这样以来，不仅减少了项目的人力成本，还节约了项目时间。

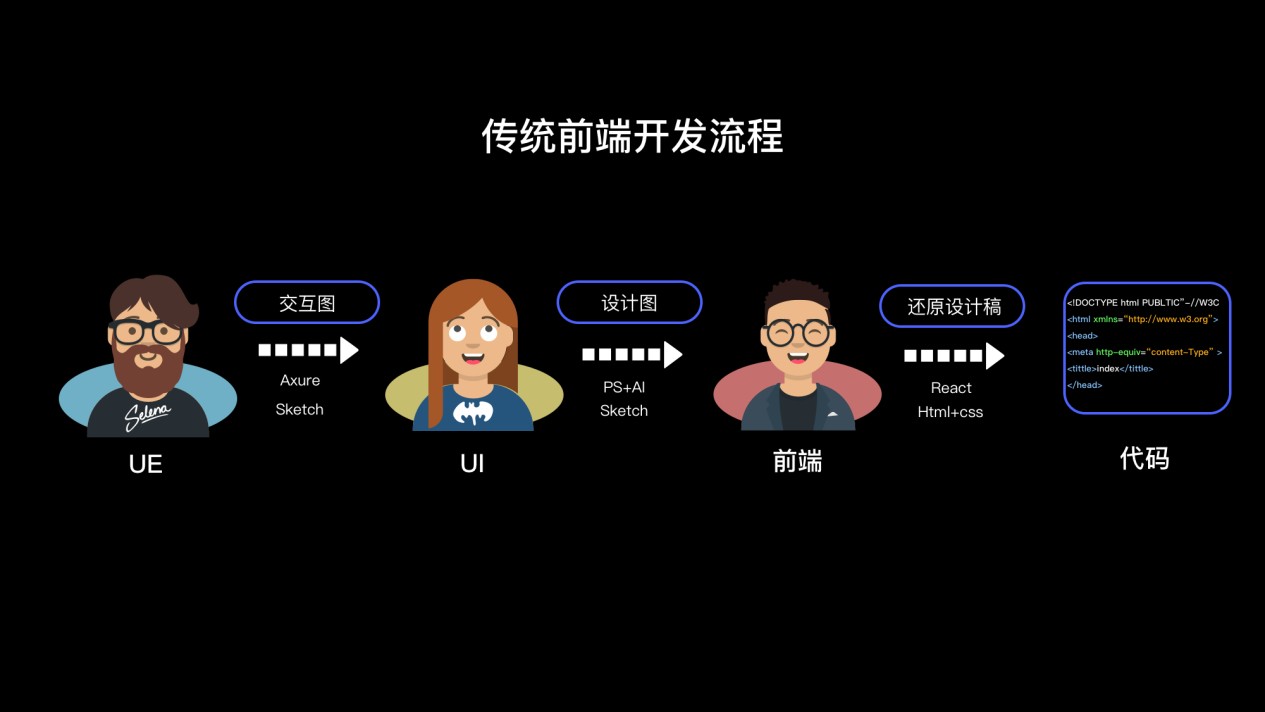


图1.2 传统前端项目开发流程

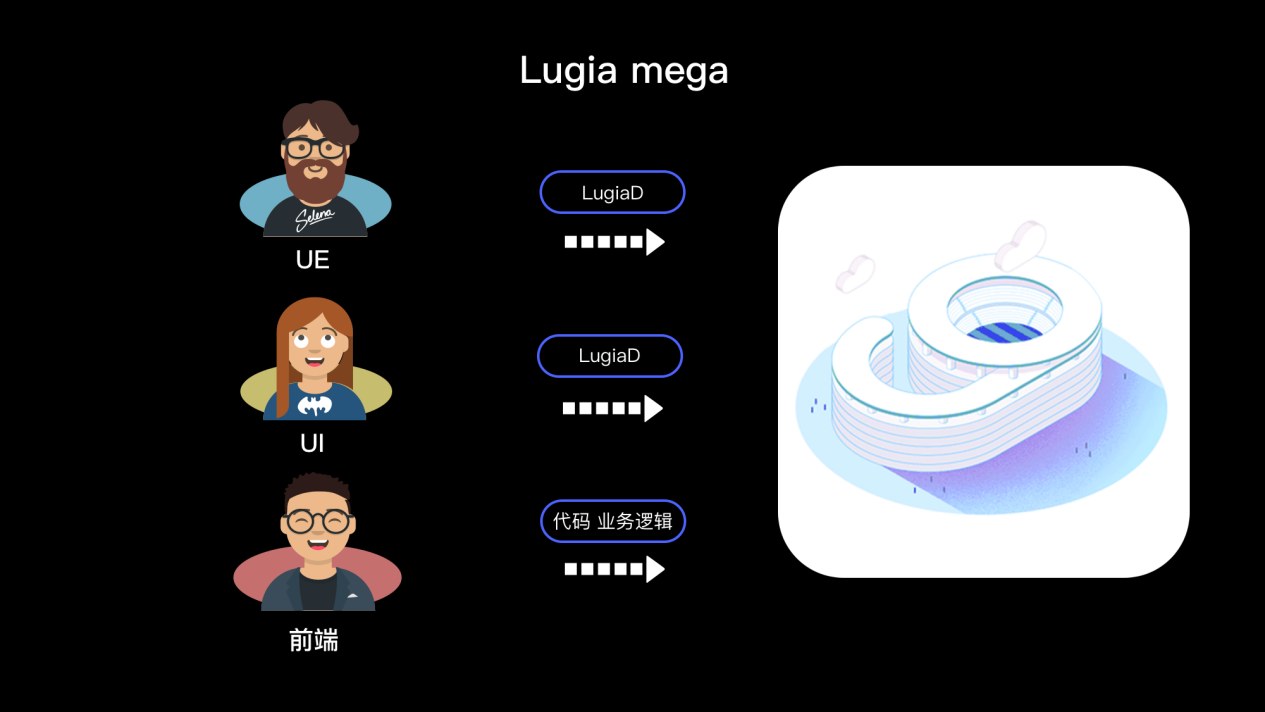


图1.3 lugia-mega 开发流程

总体来说Lugia产品包括一套标准的前端组件库、一套统一的模型管理框架、一套完整的工程构建工具链，以及一套高效并且开箱即用的前端可视化开发工具。本产品具有无需环境搭建、快速上手、跨平台支持等优点，其贯穿了整个项目的原型设计、效果设计、软件开发测试阶段，为用户极速构建与轻松管理高质量前端项目赋能。支持交互设计师快速迭代原型界面，支持UI设计师灵活迭代效果界面，支持软件开发设计师简单迭代最终界面。最终完成前端项目管理模式重构，使得三者产出成果可以互相传递复用，进而满足云原生大前端的快速迭代开发需求。

### 特性

* **标准**：约定、配置、初始化、原型设计、效果设计、模型绑定、开发调试、构建、依赖管理、项目管理、Mock 服务、代理配置...... lugia-mega贯穿了整个前端项目的生命周期并采用业内最优秀的解决方案。
* **高效**：高品质的物料体系，丰富易用的主题样式，高效快捷的模型管理，让您极速构建前端应用、轻松管理所有项目。
* **开箱即用**：lugia-mega是一个无需环境搭建、快速上手的跨平台桌面应用（Mac 和 Windows）。通过简单的拖动操作帮助产品经理、设计师快速设计原型、交互稿，成果可以直接让开发人员使用。

### 愿景

降低开发成本，提高前端开发质量，提升用户体验。

# 功能设计

### 功能组成

Lugia-mega主要由项目管理、页面设计器、物料体系、开发工具链、模型规范等部分组成。

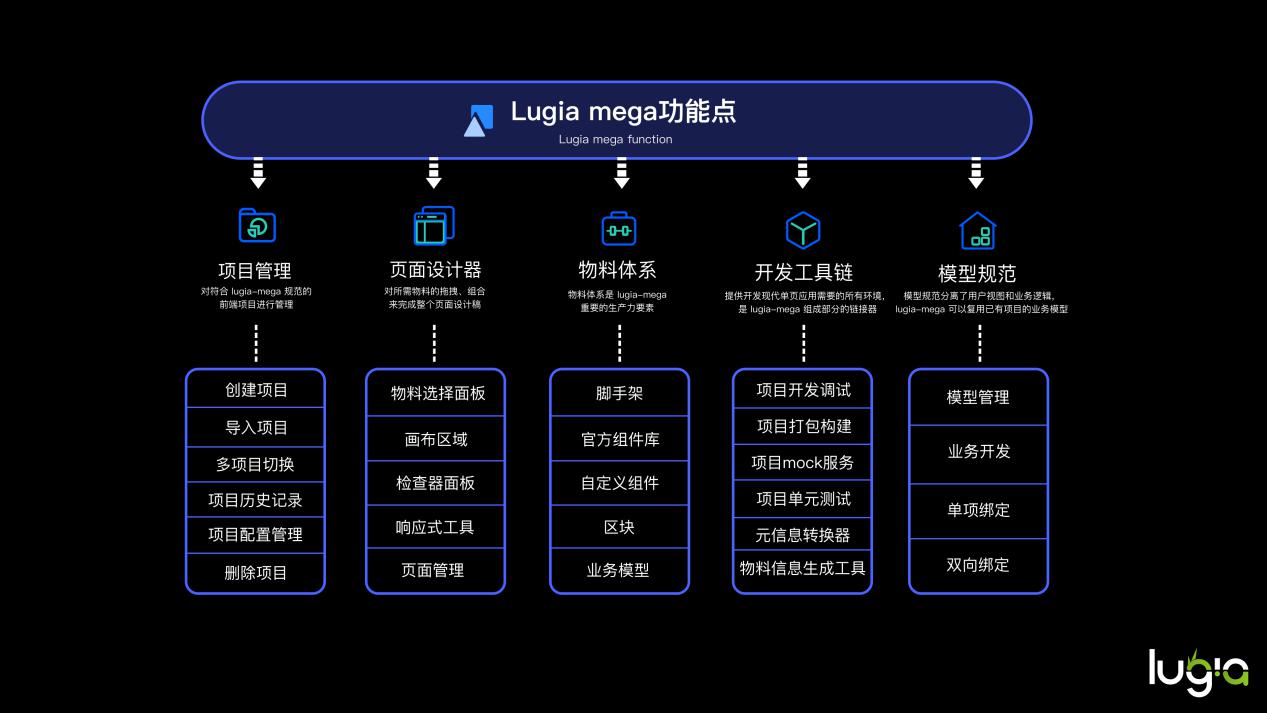


图2.1 lugia-mega 功能点

#### 项目管理

对符合 lugia-mega 规范的前端项目进行管理

* 创建项目：创建新项目时，提供推荐脚手架和布局等功能，并且有高级配置项可以自定义创建项目。
* 导入项目：导入一个符合lugia-mega规范的项目，进行多人协同开发。
* 多个项目间切换：可以在已经创建好的项目间快速进行切换，方便用户同时开发多个项目。
* 项目历史记录：会记录项目的操作历史，以方便用户在下次使用时快速进行开发。
* 项目配置管理：对项目进行主题、布局、依赖、mock数据等的配置管理，还有一些默认配置项的设置。
* 删除项目：删除一个项目。可选保留项目源文件，依旧可以打包部署，也可以再次导入到lugia-mega中继续开发。

#### 页面设计器

通过对所需物料的拖拽、组合等方式来完成整个页面设计稿

* 物料选择面板：在设计页面时选项所需的物料（组件库、自定义组件、区块、布局等），通过拖拽等方式移至画布区域。
* 画布区域：当前设计的页面，通过对所选的物料进行移动、删除、编辑、组合等操作完成设计稿。
* 检查器面板：检查器可以显示编辑当前选择的组件属性，进行参数的调整。用户可以基于自己的需求在检查器面板进行属性样式的调试。
* 响应式工具：提供设计响应式页面的能力。
* 页面管理：对当前项目的所有页面进行新建、编辑、删除等管理操作。

#### 物料体系

物料体系是 lugia-mega 重要的生产力要素

* 脚手架：创建一个新项目时所需的基础工程项目，一般包含菜单布局、基础页面、工具函数、自定义配置项等。脚手架要符合物料体系规范，并通过lugia-mega的对接机制对项目进行配置。
* 官方组件库：基于 Lugia Design 的高品质React组件库 Lugia Web，满足金融行业高性能组件的需求。
* 自定义组件：根据业务需求，通过编码的形式，对组件进行高度定制，并通过lugia-mega的对接机制接入使用。
* 区块：为了更加快捷的构建一个页面，将部分组件拼合成一个功能区块，只需简单组合便可以形成一个页面。
* 业务模型：通过 lugia-mega的对接机制把现有项目的业务模型复用到其他项目中。

#### 开发工具链

提供开发现代单页应用需要的所有环境，是 lugia-mega 组成部分的链接器

* 项目开发调试：提供一个简单方便的开发调试环境，并具有 live reloading (实时重新加载) 功能。支持browser-sync，保持多个浏览器和设备同步。
* 项目打包构建：把项目部署所需的资源构建在一起，并优化整个应用以提供最佳性能，自动按需加载，加快页面访问速度。
* 项目mock服务：mock数据是前端开发过程中必不可少的一环，是分离前后端开发的关键链路。通过预先跟服务器端约定好的接口，模拟请求数据甚至逻辑，能够让前端开发独立自主，不会被服务端的开发所阻塞。支持引入 json、excel、csv 格式的文件作为数据来源。
* 项目单元测试：交互式的单元测试，内置支持覆盖率报告功能，包括 UI 测试。
* 元信息转换器：把页面设计器生成的元信息转换为可执行代码。
* 物料信息生成工具：生成可供 lugia-mega 使用的物料信息，以便把页面设计器需要的物料对接到 lugia-mega 中。

#### 模型规范

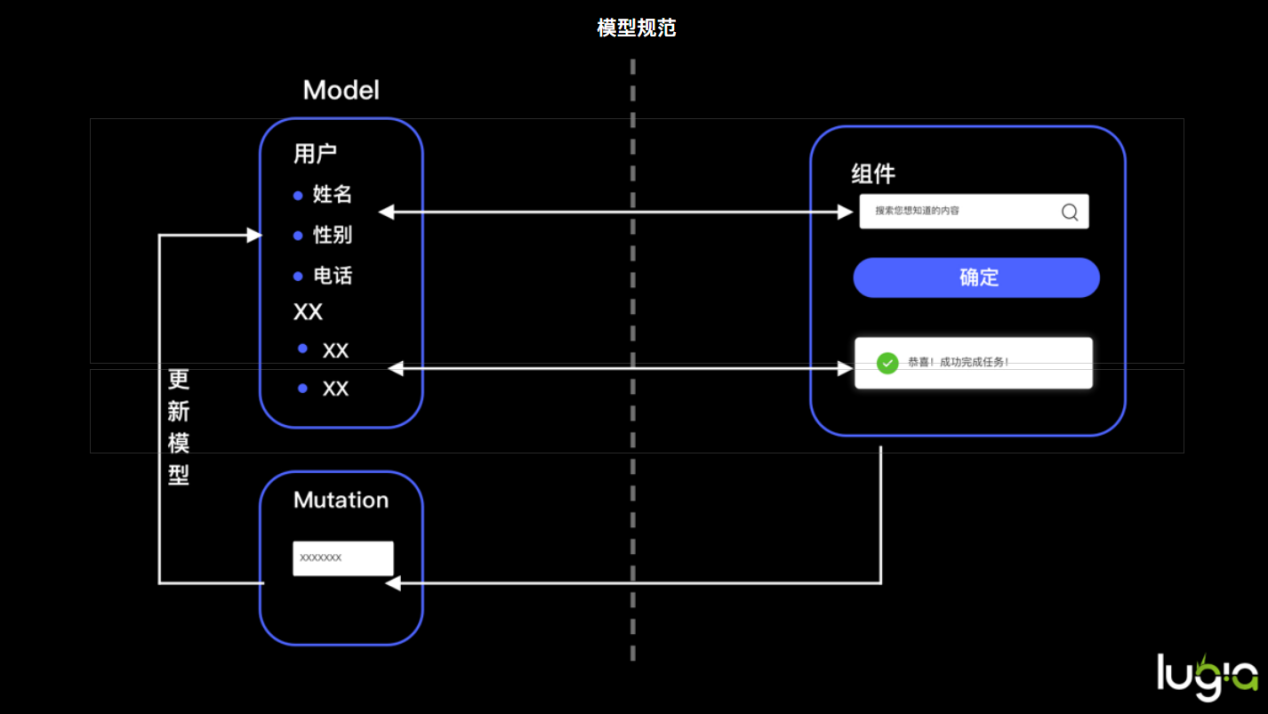


图2.2 模型规范



图2.3 业务模型复用

模型规范分离了用户视图和业务逻辑，lugia-mega 可以复用已有项目的业务模型

* 模型管理： 对项目中的业务模型进行管理，包括创建、查看、删除等，并接入到 lugia-mega 中进行使用。
* 业务开发：使用前端状态管理工具lugiax进行业务模型的逻辑编码。lugiax提供简单高效的全局状态管理方案、 基于async/await 的异步操作、快捷的双向绑定。
* 单项绑定：在lugia-mega中以可视化的方式快捷的把业务模型中的状态绑定到选中的组件上。
* 双向绑定：在 lugia-mega中以可视化的方式把组件中抛出的事件和业务模型的状态绑定到一起。

### 优点

现在我们开发一个前端应用，通常使用 [Create React App](https://github.com/facebook/create-react-app)、[vue-cli](https://github.com/vuejs/vue-cli) 这些命令行工具的形式，而 lugia-mega 是一个可视化的桌面应用（以下简称 mega），这会带来以下方面的优点：

* 无需再配置复杂的系统环境、环境变量等。
* 无需再安装复杂的 Node.js 环境。
* 有效的管理起每个项目下体积大、版本混乱的依赖项（node\_moduels）。

通过依赖 mega 内置固定版本的 Node.js 环境以及依赖管理功能，因而隔离外部环境，不用再安装配置各种各样的系统环境以及依赖。

### 如何使用

我们提供 macOS、Windows 64 两个主流系统的桌面应用，若需要其他系统的应用，我们也会根据实际情况进行支持。

只需简单的下载、解压，即可开始愉快地设计开发各种前端应用。

### 下载

在 [Lugia](http://192.168.102.79/) 网站上进行下载。

### 安装

#### Windows 64系统版本的客户端

仅需要解压下载的压缩包，双击运行 lugia\_mega.exe。

#### macOS系统的客户端

下载完成后打开 dmg 文件。

若出现 “不明开发者...” 的提示无法打开的话，请打开 系统设置>安全性与隐私>通用 允许打开应用。

成功打开 dmg 后，拖动 app 到 应用程序 文件夹后，双击打开即可运行。

### 服务手册

因系统的差异性，macOS 与 Windows 64 两个系统版本的应用界面上会有所差别，但功能上相差无几。

在 [Lugia](http://192.168.102.79/) 网站上下载 lugia-mega 服务手册。

### 浏览器兼容性

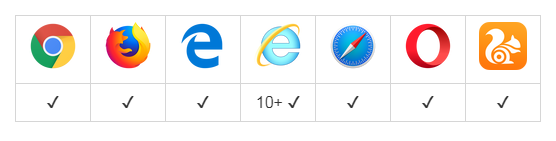


图2.4 浏览器兼容性

### 名词解释

| **名词** | **解释** |
| --- | --- |
| 设计语言 | lugiad 设计语言（一种规范），提供专业的视觉指导，提高用户体验。 |
| 物料库 | 在 lugia 中，组件、区块、布局、模板等统称为物料，由 lugia 团队维护，在内部有一套完整的开发规范和工具；基于此，你可以参与共建，也可以自建私有的物料库。 |
| 官方物料库 | 由 lugia 团队开发维护的物料库，质量保证，提供技术支持。 |
| 私有物料库 | 由业务团队开发维护的私有物料库；在实际项目中，官方提供的物料库和设计风格可能不能满足某些业务场景，这时可以自定义内部的私有物料库。 |
| 物料库开发者工具 | 我们提供了物料库开发者工具 lugia-devtool来满足自定义私有物料库的能力，支持自定义接入 React 的区块、布局、模板等功能；官方物料库也是使用这个工具进行开发维护的，通过这个工具你可以参与官方物料库的共建。 |
| 组件 | 最基础的物料，具有高度可复用性；包含三端：lugia-web（web 端）、lugia-mobile（移动端）、lugia-desktop（桌面端）。 |
| 区块 | 通过对大量的系统常用的场景进行分类、对比和抽象，基于基础组件组合而成，可以通过 lugia-mega 进行快速组合搭建应用，减少重复的开发，提升效率。 |
| 布局 | 我们提供了一些常见的页面布局，并且有一个更加灵活的布局系统，支持自定义布局。 |
| 模板 | 基于已有的区块搭建而成，我们提供了一些特定领域的模板；可以从零开始搭建应用，也可以选择特定类型的模板快速开始。 |
| 状态管理工具 | [现代 js 框架主要在解决 UI 与状态同步的难题](https://github.com/dt-fe/weekly/blob/master/57.%E7%B2%BE%E8%AF%BB%E3%80%8A%E7%8E%B0%E4%BB%A3%20js%20%E6%A1%86%E6%9E%B6%E5%AD%98%E5%9C%A8%E7%9A%84%E6%A0%B9%E6%9C%AC%E5%8E%9F%E5%9B%A0%E3%80%8B.md)，lugiax。 |
| 开发工具 | lugia-mega，一个基于前端物料库体系的可视化开发工具，提供高效、开箱即用、标准的前端工程化开发体验。 |

# 原型设计

### 启动页

loading，进行页面渲染

检查更新... 弹框提示更新

检查离线文档更新，自动更新离线文档

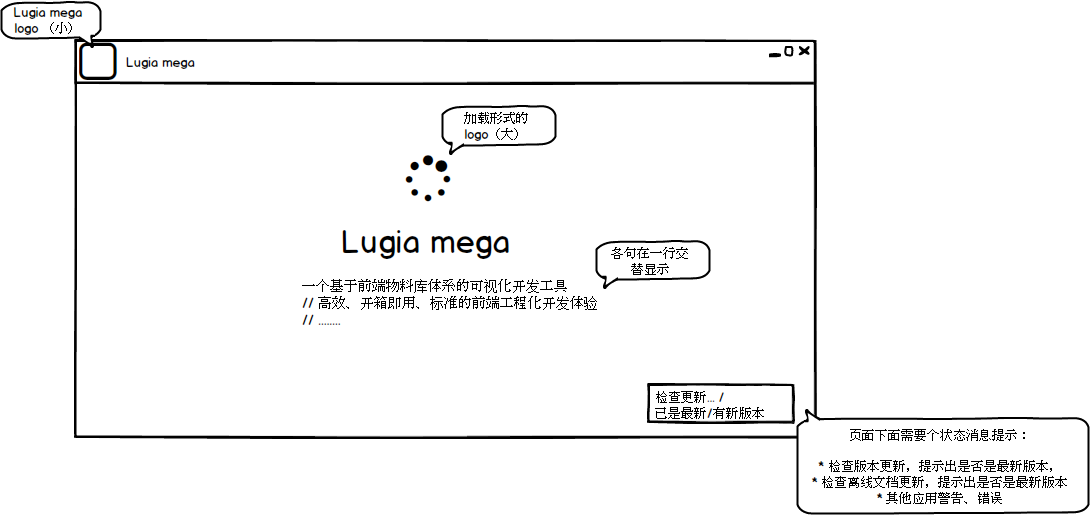


图3.1 启动页原型

### 主界面

页面渲染完毕后，从启动页进入主界面。

#### 主界面-项目

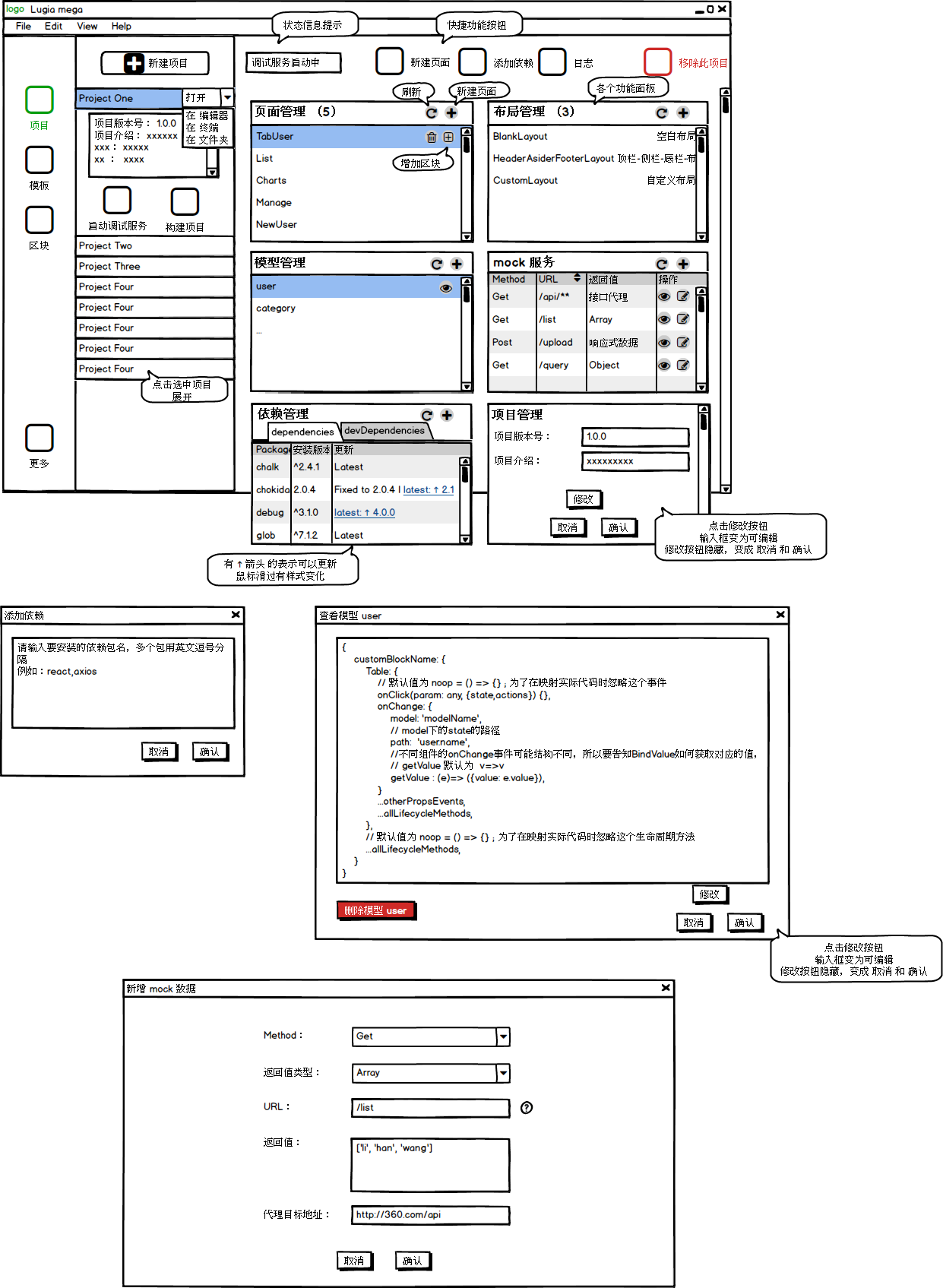


图3.2 主界面-项目原型

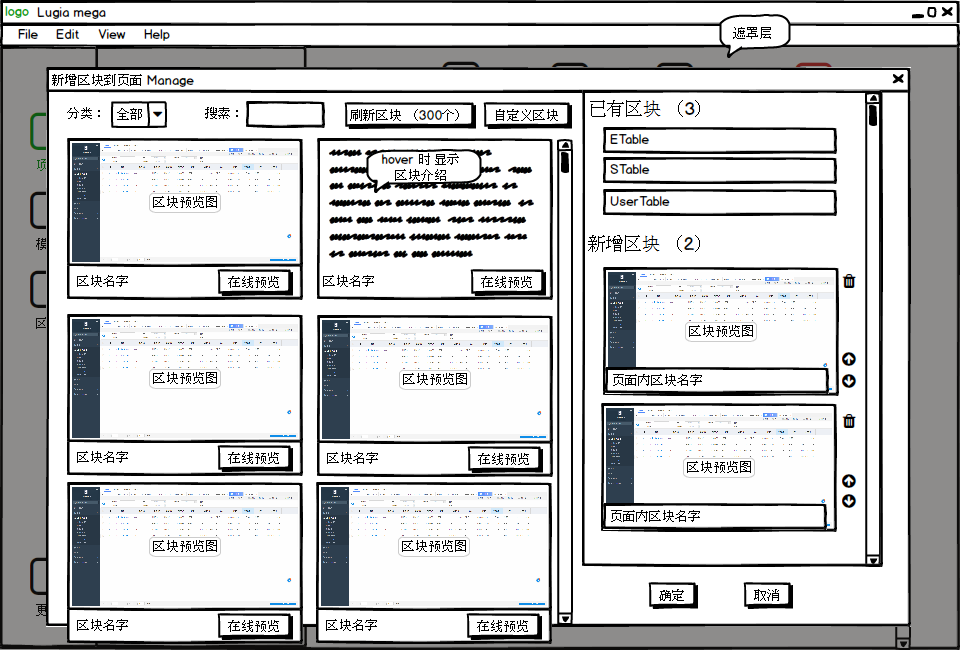


图3.3 区块添加到页面原型

#### 主界面-项目-新建页面

选择的区块需要有自定义名字。



图3.4 拼接页面原型

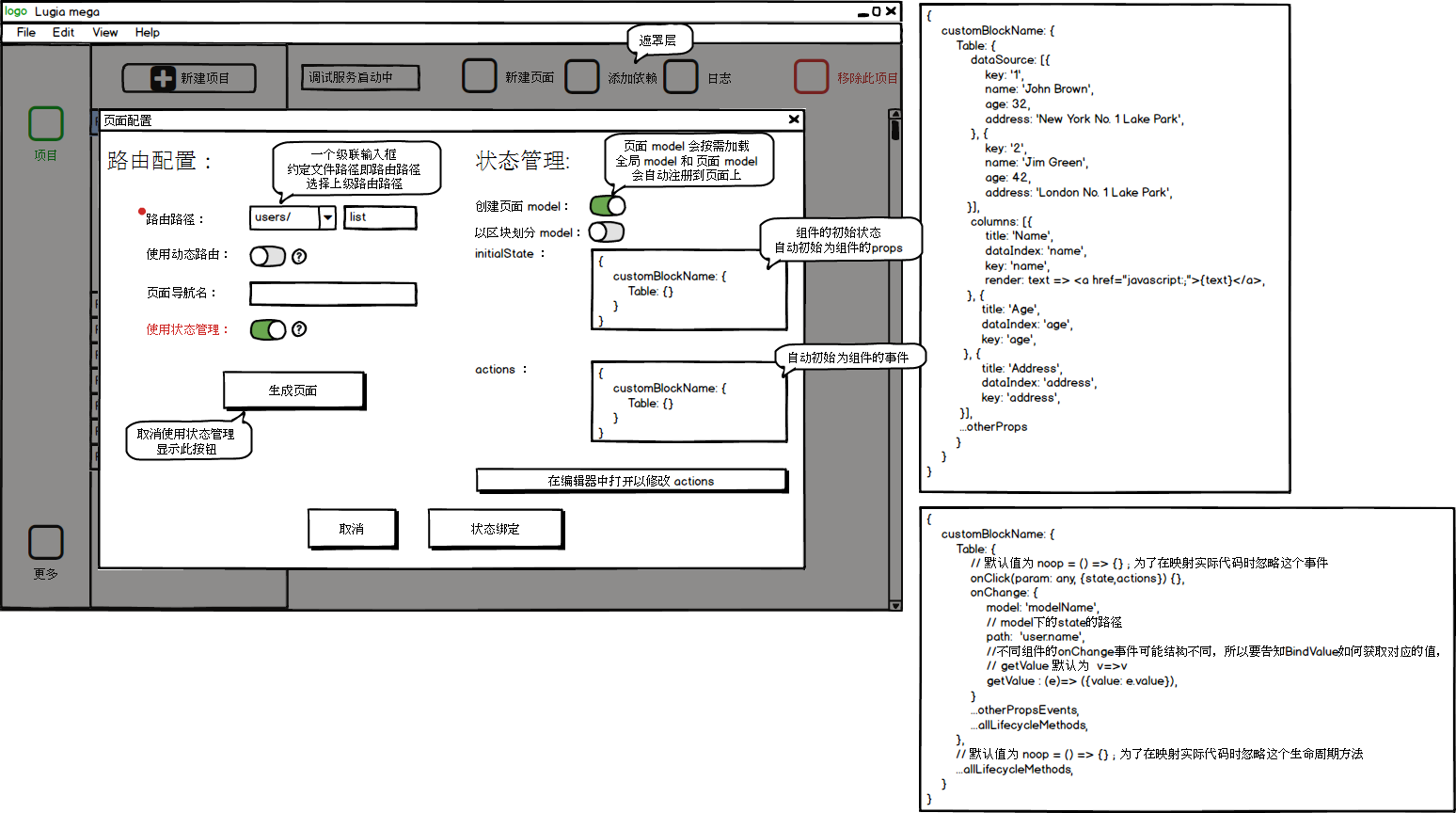


图3.5 新建页面原型

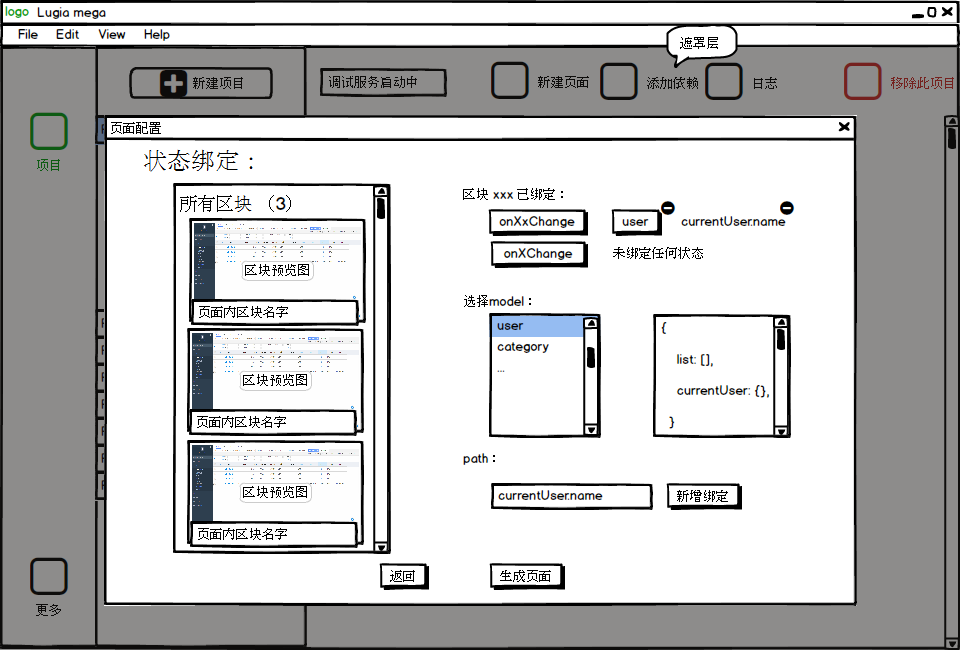


图3.6 组件绑定原型

#### 主界面-模板

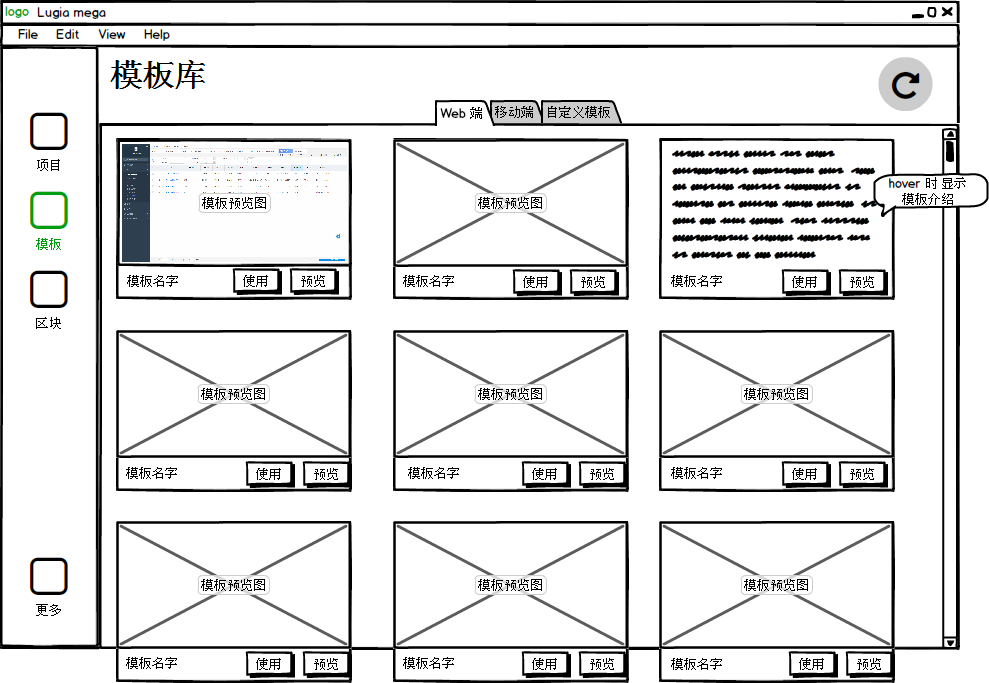


图3.7 模板查看原型

#### 主界面-模板-使用

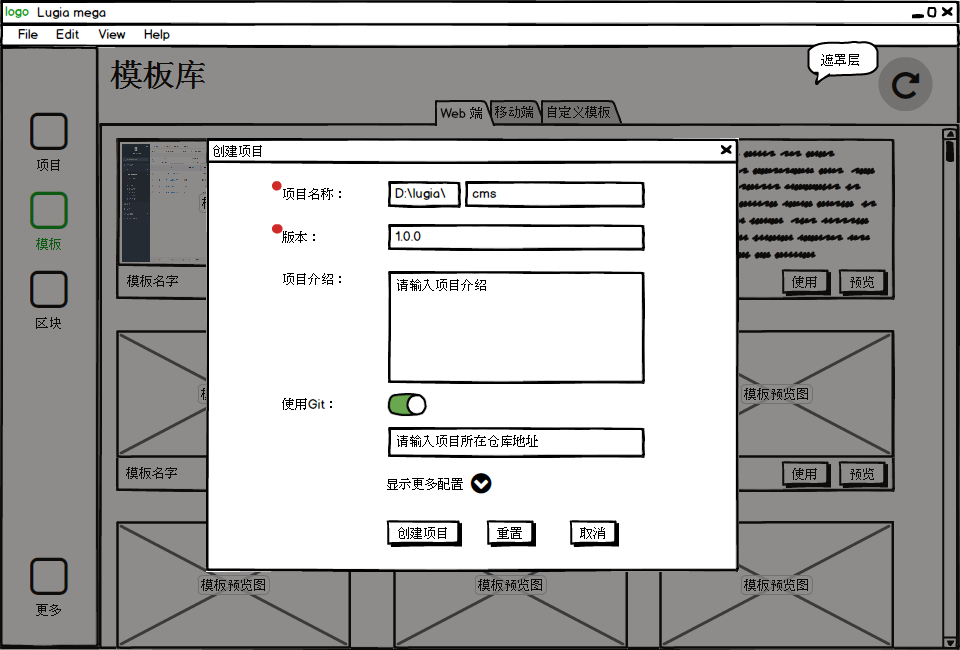


图3.8 使用模板原型

#### 主界面-区块



图3.9 区块查看原型

#### 主界面-区块-自定义区块1

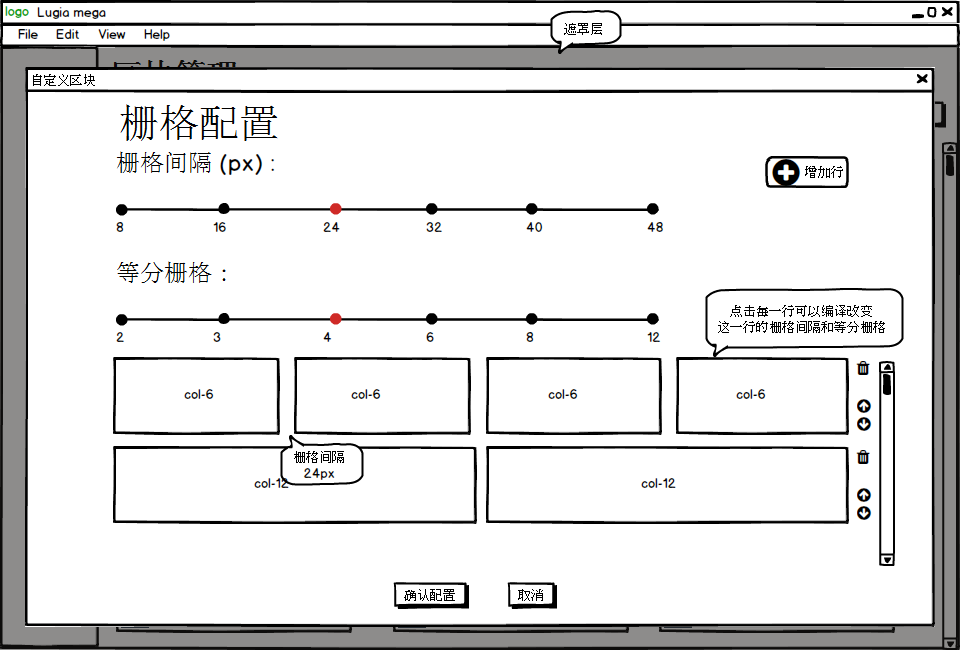


图3.10 自定义区块原型1

#### 主界面-区块-自定义区块2

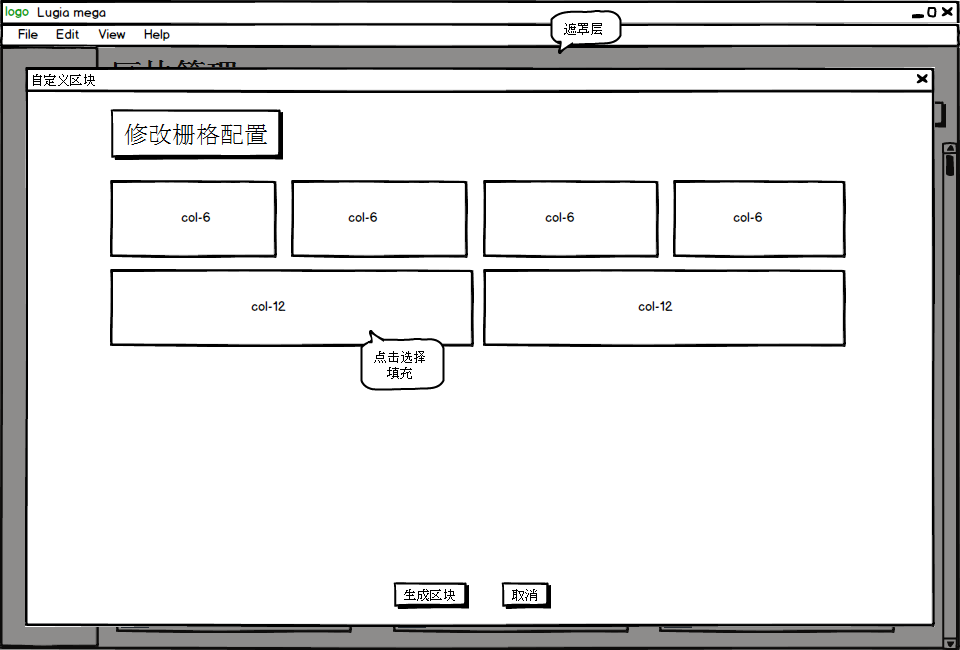


图3.11 自定义区块原型2

#### 主界面-区块-自定义区块3

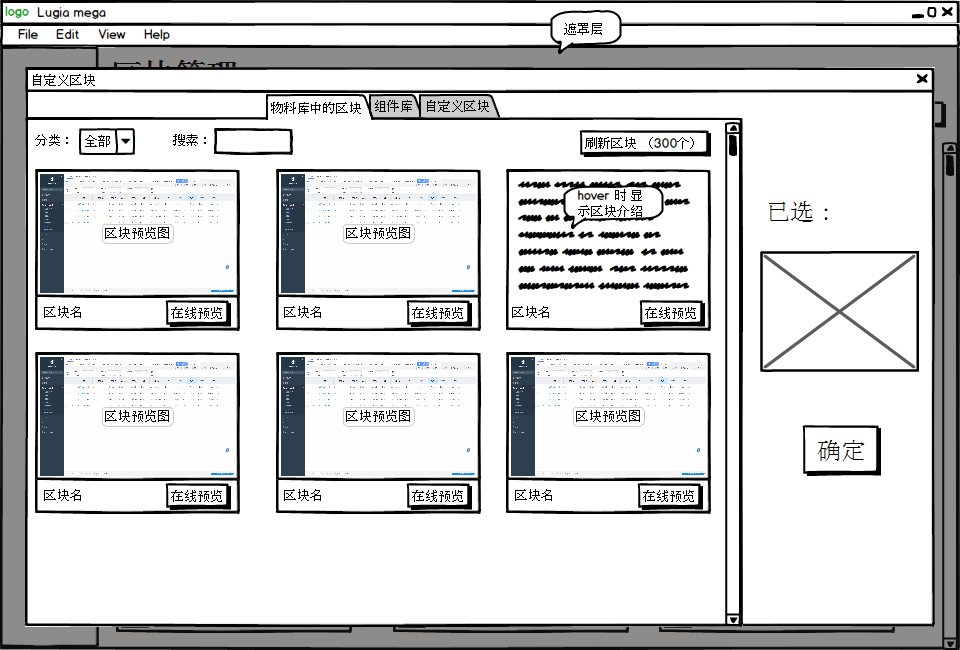


图3.12 自定义区块原型3

### 更多

#### 物料源设置

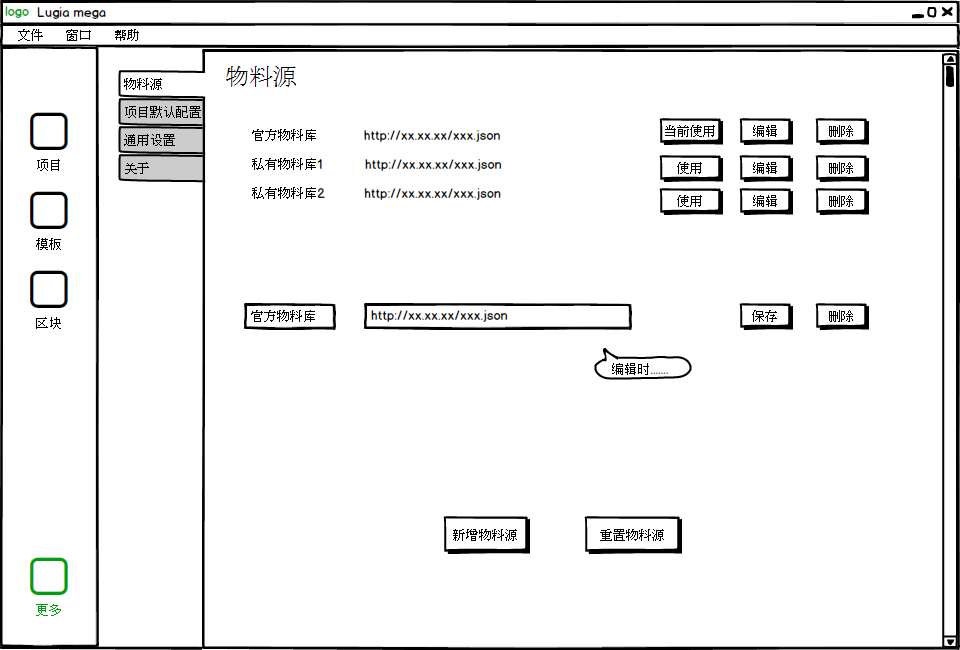


图3.13 物料源设置原型

#### 项目默认配置

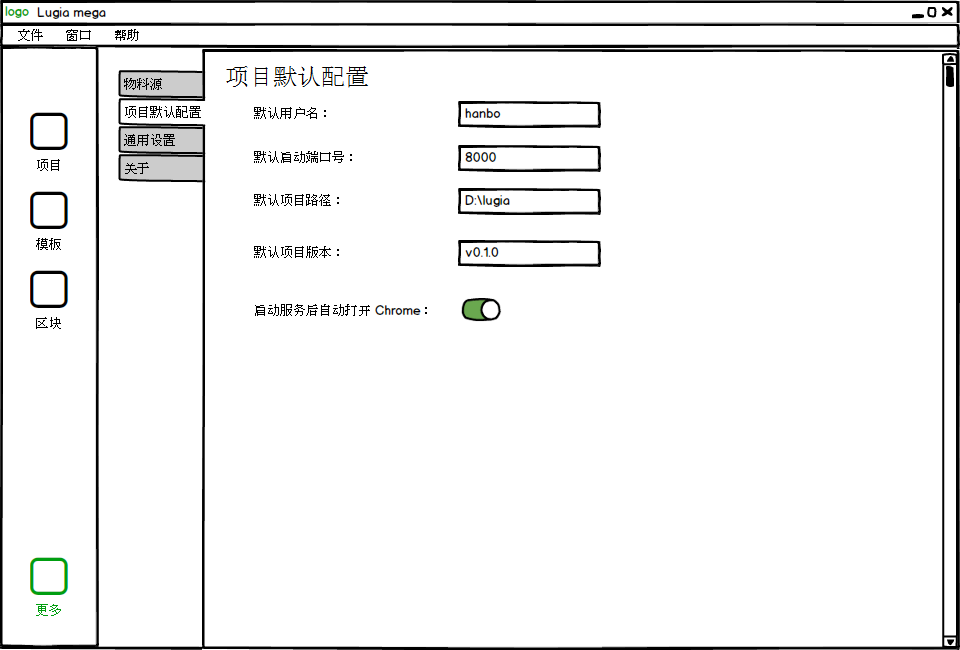


图3.14 项目默认设置原型

#### 通用设置

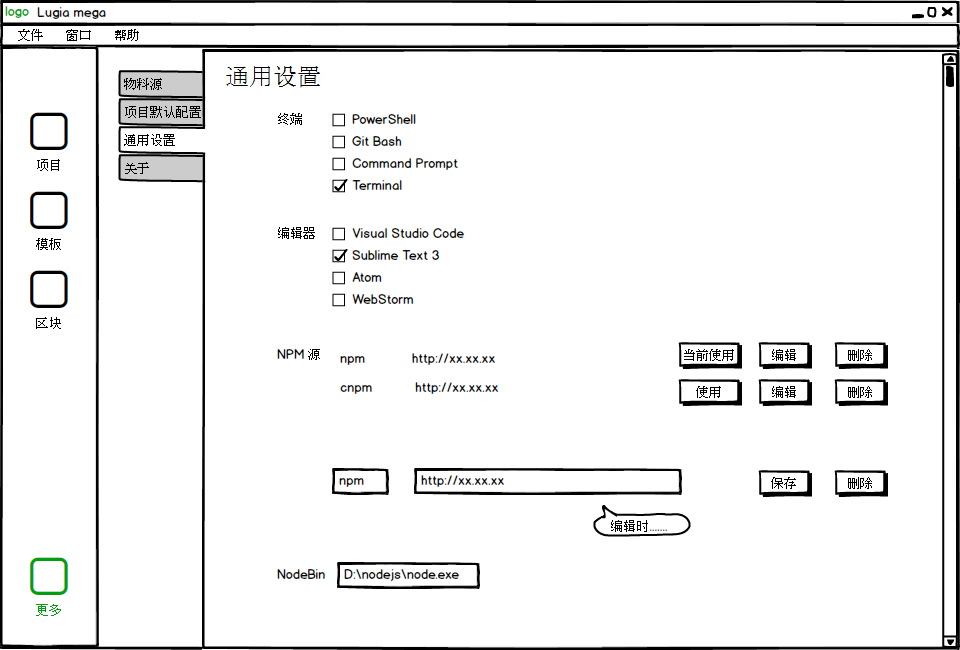


图3.14 通用设置原型

#### 关于

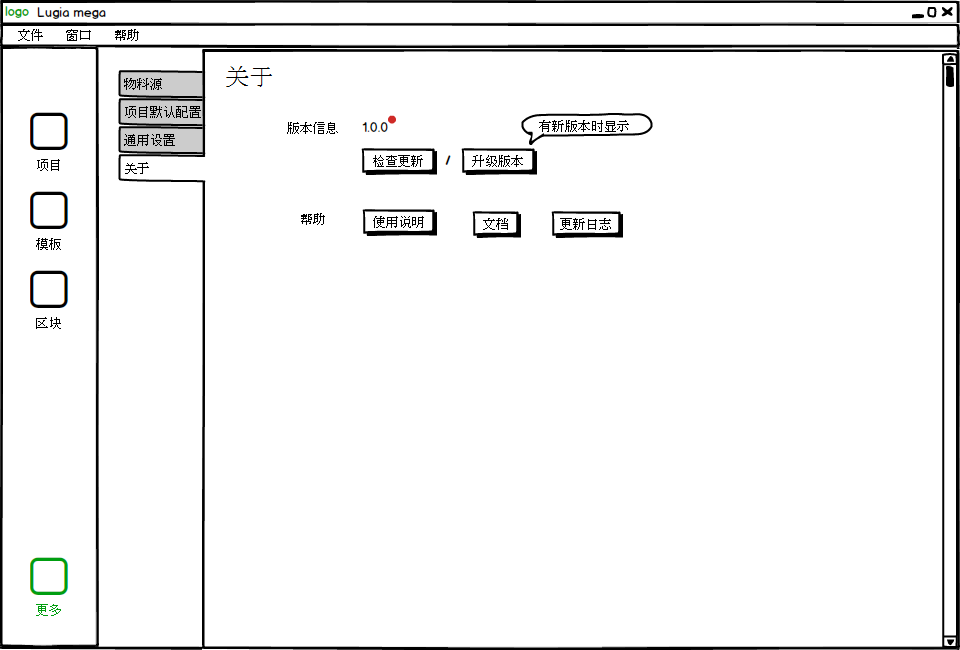


图3.14 关于原型

# 交互与详细设计

### 主页

主页采用最简化的展示，并记录项目历史文档。以方便用户在下次访问时可直接进行编辑。



图4.1 mega主页

### 创建

在页面构图之前，我们为使用者提供了推荐模版和布局等功能，使用者可以轻松的选取自己所创造的页面基本布局，以便于更快速的开始自己的灵感创作。

### 项目页

lugia的界面采用简洁化的设计样式。最顶端的工具箱包含了最重要的操作。左侧是应用组件部分，可以在该区域选择您所需要的组件形态在画布中进行编辑。右侧则是对于组件属性的动态参数调整，不仅满足于设计的静态页面颜色，还可以去修改一个组件的手势状态或者去增加一些双重绑定事件。最后中间的区域就是你正在创作的画布。



图4.2 mega画布

Lugia-mega 里没有浮动面板，检查器将会根据你选中的组件来显示所选单位，这样你能始终不受打扰的在画布上创作。

### 工具栏

lugia-mega的工具栏相当于整个项目的控制台，可以在上方对整体的项目进行管理、保存、导出等其他调试服务。



图4.3 mega工具栏

在工具栏中，第一组工具是用来对页面进行基础服务的。其中包含：新建页面、保存当前页面，在浏览器上预览。第二组是对页面的管理。包含：model管理、项目管理、全局设置等功能。第三组就是对完成编辑的页面打包构建，即：导出。

### 左边栏

我们认为一个页面是由4个元素拼合构成的：布局、区块、组件和图标。为了方便您更易上手，在左边栏中我们将4个元素按照从左到右、从大到小的方式分类排序。并全部用图像加文案的方式展示出来。



图4.4 mega左边栏

#### 屏幕快照 2019-03-15 下午2.52.42.png布局面板

布局面板是为了解决您中途需要修改布局的快捷操作。您可以通过鼠标点击选择一个当前更适合您的布局再进行页面的编辑。当然如果我们所提供的布局并不能满足您的需求，在布局的顶部您可以选择通栏的模式，这样就可以让页面不受布局的影响，让您自由发挥您的布局条件。

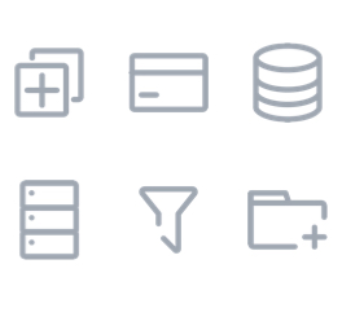
#### 区块面板

区块是为了方便您更快速的构建一个页面，将部分组件拼合成的一种功能。例如：图表、标题栏、说明信息等。您只需要通过鼠标点击想要的功能，拖拽到页面中想要的位置即可。当然您也可以对区块中不满意的组件进行属性调整。

#### 组件面板

Lugia基于对市场上产品的研究，将产品项目单元化。把一个页面所需要的所有组件编译成开源的控件，这样您在也不用去思考一些无意义的形状去构想页面，而是直接通过一些组件的按钮去摆放拼合成一个整体。

为了方便您查找，我们对组件做了两种分类查询您可以直接搜索想要的组件，也可以通过我们的分组整理进行筛选。同样，组件在画板上您也可以通过检查器自定义您想要的属性。



#### 图标面板

图标面板是为了方便您的完善页面元素。在此您可以通过查找选择您所需要的图标。

### 右边栏

#### 检查器面板

右侧的检查器能让您编辑当前使用的组件属性，进行参数的调整。为了让您更加方便编辑，当您选择一个组件时，您就会发现我们已经将检查器划分几个区域：“页面区、属性区和事件与模型编辑区域”

#### 页面

在页面区域您可以对您当前的画板进行一个整体的统筹布局。这里显示了您当前页面已应用的图层类型。您可以进行等级排列、组合区块、拆解区块等功能。



图4.5 mega页面面板

#### 属性



图4.6 mega属性面板

在属性区域您可以对当前选择的一个或多个组件进行属性的调整，该区域是一个初级用户需要基础学习才可灵活掌握的页面。在此区域您可以自定义您当前编辑组件的参数，而且lugia mega根据页面基础构成，在原调整静态页面的参数的维度上，增加了可以对该组件“未来的变化做相应调整”。

即：“滑动、点击和禁止”。



图4.7 mega组件状态属性

#### 事件与模型

Lugia-mega 为了满足开箱即用的市场需求，在事件与模型页面中可以针对不同的组件进行一系列捆绑。最显著的特点就是响应式和数据驱动，也就是将Model和View进行单向绑定或者双向绑定。



图4.8 mega模型绑定面板

### 组件操作

组件是lugia-mega中最基本的构成单位。它不同于其他设计产品的图层和对象。我们是将组件通过开源封装成一个个子元素。做为使用者，您在设计页面中思路模式需要以适应形状为基本元素去拼接来完整整个页面设计稿，转变成直接思考我需要的页面设计稿需要哪些组件。

例如：“当我设计一个简易图表查询功能。以前的做法需要自己通过矩形-转变成描边-设置宽高，在画一个矩形给它改变颜色，打上字查询。而通过lugia-mega仅需要思考一个图标查询，我需要一个图表和一个按钮就可以了”

这种思路上的转变给您带来的改变是不需在绞尽脑汁去做一些无意义的形状，而是直接去思考页面的全局应该有哪些变化。

#### 选择组件

上文已经讲到，在您使用lugia-mega时可以通过左边栏去选择您所需要的组件。您只需要鼠标点击选择你需要的组件，然后拖动鼠标到所需要的区域位置。松开鼠标，即可完成该组件，并可以编辑它。

#### 选中组件

当组件拖拽到页面后，您如果想对它进行调整。您只需要单机来选中组件，这时选择框的四角和边框上会同时出现6个小锚点。这些小锚点会给予您一个提醒：“该组件已经被选中”。shift+option（shift+alt）：组件会以组件中心为居中点进行等比例放大缩小。

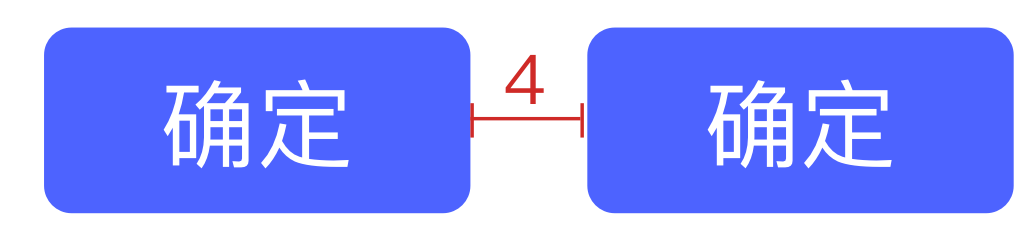
#### 同时选择多个组件

您可以通过鼠标在画板上框选住多个组件，如果您需要单独选择。可以按住键盘上的shift键来同时选择多个组件。同样如果您多选了不需要的组件，您也可以通过shift单机来取消该选择。

值得注意的是，在lugia-mega中涉及到了集合页面的概念（类似于打组）。当您选择一个集合页面的布局，该页面是以父容器为最高选择的。（简单的说，假如有一个盒子和一些组件，把其中的几个组件放到盒子里面。如果我选择了这个盒子和某一个在盒子外的组件，那么盒子内的组件也会被选中。因为它已经在盒子里面了）

#### 移动与复制组件

您可以选择任意组件，并通过拖动鼠标或者键盘上的方向键来移动它们。如果您觉得移动的不够舒适，则可以尝试按住键盘上的shift键来移动它们，这样就会让组件依照垂直或者水平方向移动。

Lugia mega为了满足移动时候的对齐问题。设置了移动状态下防治抖动的功能，当您移动某一个组件时候，lugia mega会自动帮您将它与相邻的组件根据四周变化做锚点吸附功能，并提供参考线作为吸附依据。

如果您想复制出一个相同的组件，您可以鼠标点击组件并按住键盘上的option（alt）键，拖拽到相应的位置即可。当然如果您仅想在原位复制一个组件。可以试试command+c(ctrl+c)进行复制，再按下command+v(ctrl+v)复制即可。

#### 放大缩小组件pasted-image.png

如果您想调试组件的尺寸，可以去使用组件的6个小锚点。直接将鼠标滑动到某一个锚点上，然后按住放大缩小就可以感受到该组件带来的变化，这样就省去了在检查器输数值才可得到的状态。而且对齐方式和移动组件一样。我们同样设置了对齐功能。当您放大一个组件与相邻组件某一个属性相邻时，会触发标尺来示意对齐，并带有吸附功能。具体标尺的对齐细节，等后面标尺章节在详细说明。

值得注意的是，如果您习惯了小锚点拖拽操作之后，可以通过一些快捷键，来让您的调整更为精准：

* Shift：如果您按住shift键放大缩小组件，会锁定组件的宽高进行等比例变化。
* option（alt）：组件会以组件中心为居中点，进行放大缩小。
* shift+option(shift+alt)：组件会以组件中心为居中点进行等比例放大缩小。

#### 删除与撤销

如果您对拖拽进来的组件感到不满意，您可以试试删除它。您只需要点击或框选一个或多个组件，然后按下键盘上的退格键，您就可以删除它了。

我们都知道在设计页面中会有很多的调试，我们当然知道如果您调试了很久的组件不小心被误删除了该怎么去找回来。lugia mega为您提供了强大的撤销功能。您只需要按键盘上的command+z（ctrl+z）即可复原刚才撤销的项。

#### 编辑组件

如果您需要对一个组件进行更准确的调整，您就需要借助右侧的检查器了，而具体变化成什么形状就取决于您想要的是什么类型风格的组件了。

和常规的设计软件不同的是，您使用lugia mega生成的页面不仅是平时设计师交付的设计稿，而是一个经过代码编译后的页面。所以您的工作不仅是完成一个静态页面的调整。您在编辑组件的时候，还要考虑到所编辑的组件鼠标滑过、点击、禁止时候的样式，以及我打开一个下拉菜单，究竟里面有多少字段，多少行选项提供给使用者。又或者，如果您需要该组件与其他组件产生一些关系，您还需要通过事件窗口为它们创造一些事件的绑定。

### 滑动输入条 copy.jpg检查器

#### 关于检查器

检查器会显示所选组件的一切属性调试。

用户可以基于自己的喜好在检查器面板进行属性样式的调试。我们对组件所调试的样式种类进行了整理。从自定义开始，包括外观、文本、图标和位移。当然在某些特定的组件上也有一些专属的修改项。

例如：当我们应用滑动滚动条时候，我们要去思考滑动条有多少个节点。

值得注意的是，每个组件是有不同的区域的，我们为了方便您操作，将区域进行细分整理。（以滑动滚动条为例：“滑动滚动条分为滑动条和提示框两个部分，当您编辑该组件时，需要手动切换自定义区域的滑动条和提示框切换选项卡。以便进行对修改）

#### 自定义

自定义相当于一个检查器面板的控制台。它会控制你修改的区域的中心位置是在哪。以及我要编辑的组件处于什么状态。

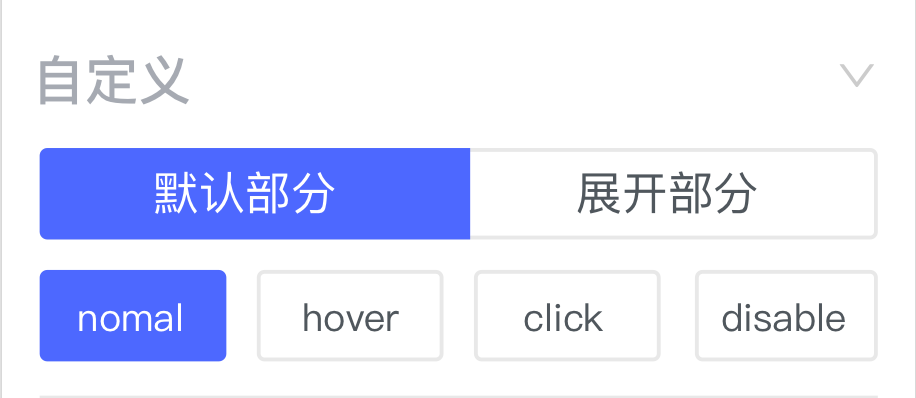


图4.9 mega自定义属性

在该区域主要分为两种功能：

1.对于组件区域的选择。（是默认的选择框还是展开的下拉列表）

2.对组件状态的选择。（是静止状态的样式，还是鼠标滑动的样式）

从级别顺序来讲：1的优先级永远大于2。

值得注意的是，因为组件的差异化，不是每个组件都会有这两种功能。有可能只有其中一种或全没有。



#### 外观

想改变一个组件的外貌？那您可以尝试一下外观界面参数调整。

外观界面包含了一个组件的外表可见的变化样式。例如：“宽度、高度、圆角、透明度、颜色填充、描边、描边的方向，阴影。

#### 锁定宽高比

在宽和高的中间，显而易见能看到一个锁的标示。这个功能是为了方便您锁定当前组件的长宽比例。当您手动将锁关闭时，无论是您输入宽高值，还是用组件的6个小锚点去拖动组件，组件都会依据当前的比例进行放大缩小。

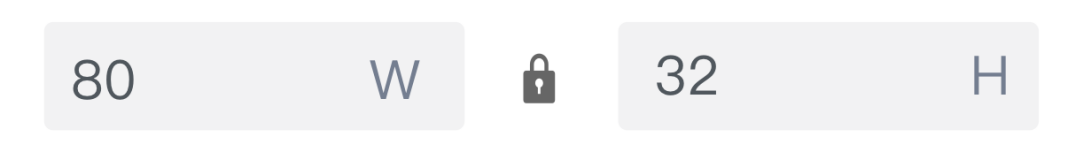


图4.10 mega锁定宽高比

#### 描边方向

我们看到在描边属性的下方会有一个醒目的描边方向调整。它可能会让您在应用过程中感到疑惑。



描边方向的主要功能是：“将一个组件的描边在组件的哪里显示。”



以一个简单的按钮为例：如果您选择的是全部，则在按钮上显示全部描边。若选择了“上”或其他的某一个方向，则显示的是当前选择的描边方向。

#### 关闭属性

在示意图中可以看到填充和描边有了不一样的指示。填充是带有丰富颜色的，而描边显示关闭。

这个的原始是在lugia mega里面是没有直接关闭属性。所有属性关闭与颜色相匹配。具体操作在颜色编辑器内部。

#### pasted-image.png颜色编辑器

颜色编辑器是在颜色窗口点击进入的。在颜色面板有几种调试参数。

目标源：修改目标源可以改变该组件颜色变化的区域。



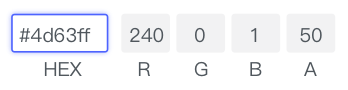
中间颜色识别区：可以直接选取颜色调整颜色属性。



滑动控件：可以选取颜色的范围和透明度。



面板控制区：最醒目的是左侧的吸管工具，使用吸管工具可以直接从画面上识取相似颜色。吸管右边是颜色的现实方式，其中包括：颜色禁止、全显、渐变。

数据输入区：在这里可直接对参数进行调整，

目前仅支持RGBA的颜色格式。



#### 文本

lugia mega采用公共字体系统。暂不支持自定义字体上传（我们在稍后的版本中会将自主上传字体功能进行开发）。

在文本区域和自定义区域类似，在面对不同组件时，会有多类字体展示，例如：“我们选择一个卡片样式，卡片样式会涉及到标题与副标题。那么在lugia mega里面修改主标题和副标题的方式需要在文本的选项卡切换。

在文本窗体页面，同样也有一些小的细节，为了方便您在使用过程中更加便捷。再次简单说明。

#### 下拉选择

可以看到在调整字号和字重属性右侧有一个三角箭头指示下拉操作，该功能是针对如果您不清楚具体数值的时，可以通过直接选择进行调试，免去了使用键盘输入的操作。



图4.11 mega属性下拉选择

#### 文字移动



图4.12 mega属性文字移动

在某些特定的组件，您可以自主调整文字的移动方向（padding），中间的锁定为锁定方向。

#### 数据录入

该功能是lugia针对组件独立开发的一种数据录入模式。说起缘由lugia mega是由组件为基点。所以针对组件我们设计了一套数据录入器。具体功能如下：

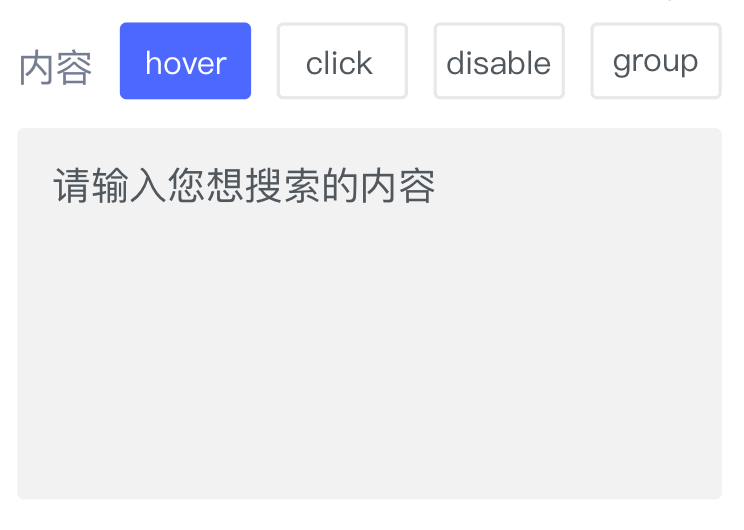


图4.13 mega数据录入

1.可以对组件上不同区域的文本初始值进行修改。

2.对于某些组件，可以用数据录入器增加内容。

（比如：下拉框里面有多少条选择数据，可以用文字编辑器输入）

3.点击内容区域的某一值，可以临时查看该字段的临时属性。

4.group（成组）这个属性是与树形选择器和表格相关的。举例来说：如果您想要将属性选择器里面的多个单一字段组合成一个整体。那么可以在文本编辑器中选择这些单一字段的组件，单击group就可以了。顺便别忘了成组后也给组取一个好的名称。

#### 图标

在我们设计组件中，会经常看到一些提醒图标。例如：当您想做一个搜索框的功能，您在组件中拖拽一个input框肯定不能表达这是搜索的意思。您需要在该input上某一个位置增加一个搜索的提醒图标。

那么功能性图标就可以让您完成这个需求。而且它很简单，您只需打开颜色面板就可让它生效。当然如果我们初始提供的图标并不让您满意，你可以通过后面的更改图标选择一个让您青睐有加的。



图4.14 mega属性图标

#### 移动

每个组件无论是在内部还是外部都会有移动参数的变化。对应为css的margin/padding值。

我们将两种属性转变为图形可视化工具。内部为padding/外部为margin，您只需要输入相对应的数值就可得出相对应的位移变化。

也许上面的描述您还是有一个疑问：“本身组件通过鼠标/方向键即可移动为何还要增加一个可输入的值”。



还是因为我们的基础元素是组件，举例来说：“当我们选择一个搜索框，选择器可能会有对应的描述文案（请输入您要查询的内容）那么该文字在组件的什么位置才是最好的呢？是距离左边框10px还是右对齐？”。而margin/padding正是为了解决这个问题。

图4.15 mega设置margin/padding

### 画布

Lugia mega画布是由两个部分组成，草稿层和主画布。两个部分是在无限画布中的一块固定的画框。当您在文件中创建一个新的画布时，画布之外的部分会将颜色变暗，这样您可以更清晰的看到画板草稿与背景之间的区别。

如果您要进行页面设计，您需要提前预设好您要设计的画面尺寸，注意只有在主画板中才是您最终生成的页面，而草稿层则是为了让您临时不需要的组件有一个可放置的区域。

我们的主画板会像一个特殊的组，它们永远是开放的状态，您不需要点击或双击来查看主画板上的内容。值得注意的是，画板的尺寸也不会因为您添加的内容的数量而自动伸缩。

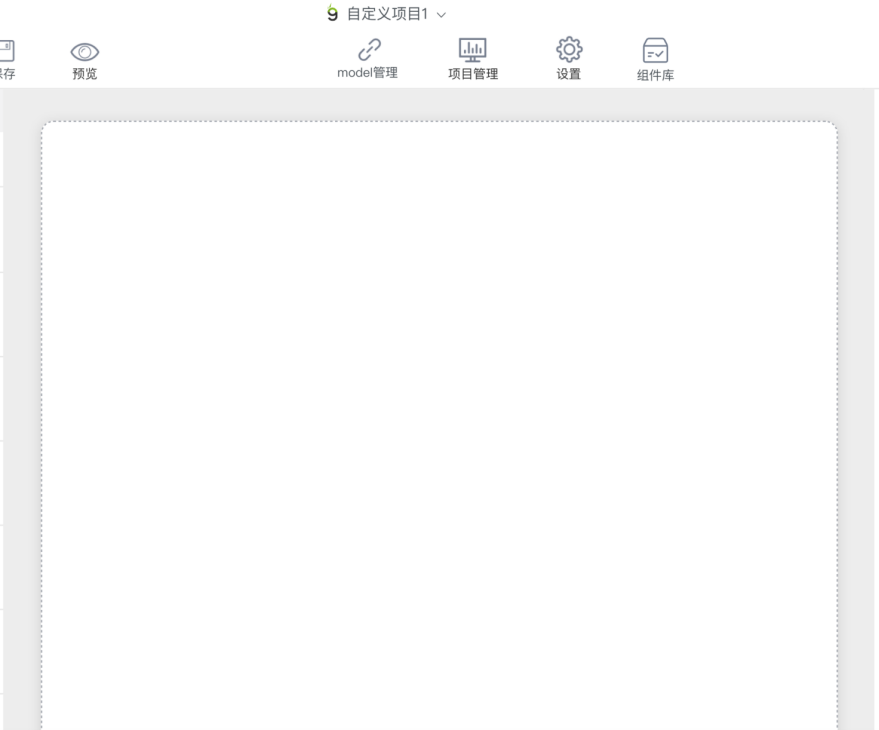


图4.16 mega画布

您在预设中设置的画面尺寸，这个尺寸会一直保留，除非您再次在设置中更改它。

#### 参考线与对齐简述

以上三个功能是为了让您做一些组件拖动位置摆放时变得更加清晰，准确。减少您拖动时候的负担。

#### 参考线

参考线在lugia mega中默认设置是被打开的。

当你在调节一个图层的大小或者移动一个图层的位置时，lugia mega标尺系统会用自带的吸附功能自动帮你把这个组件与其他临近组件的基点（6个锚点）进行吸附。

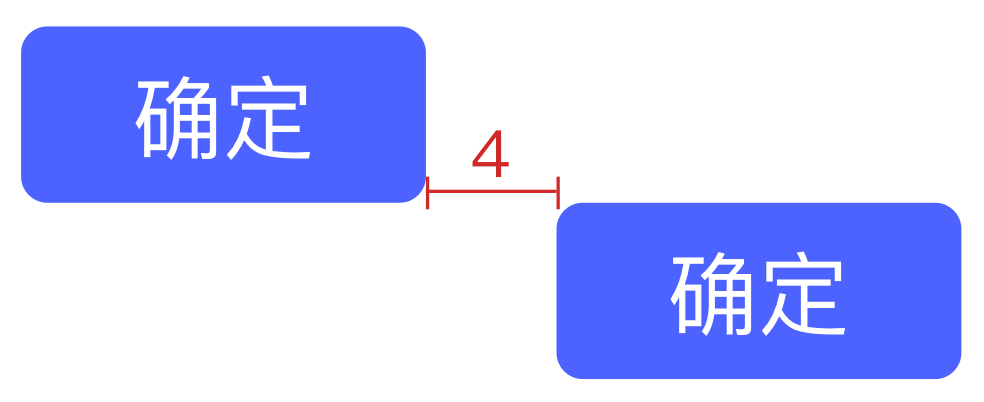
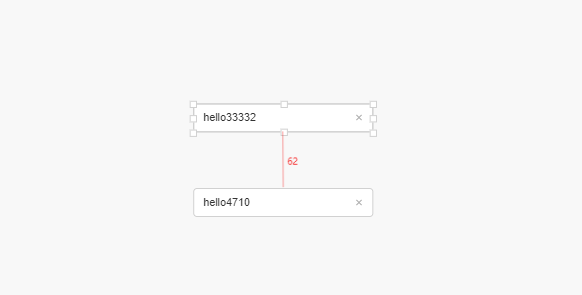


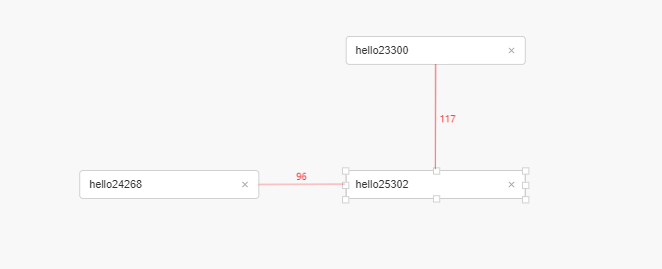
图4.17 mega画布参考线

如果lugia mega将某一组件自动与另一图层对齐，你会看见一条红线，两个图层便依据这条红线对齐。

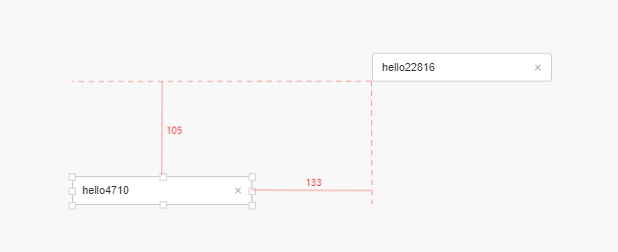
当参考线出现时，会出现一些特定的规则，例如：“若当您在画布上需要将两个相同宽高的组件进行对齐，那么则仅显示组件的中心对齐点，且中心对齐点的优先级永远高于其他锚点。”



还有一种更特殊的情况：“当画布上对3个组件进临近对齐时，若中线对齐则永远以中线为主，若不对齐则看其他锚点哪个为对齐点。



值得一说的是，lugia mega所附带的参考线不光可以作为您拖动组件时候的准确点，还可以作为一种测量工具。您可以尝试选中一个组件后，按住键盘上的option(alt)键，然后将鼠标滑动到其他的组件上。这时您会看到两个组件之间的距离的具体参数。

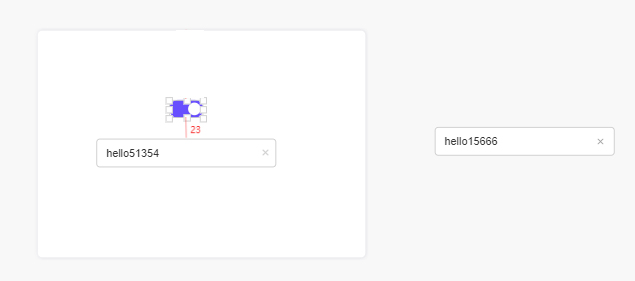


另一个值得注意的是，我们设计页面时候，经常会需要知道组件与页面边界的距离。lugia mega为您研发了组件边界参考线。您只需要拖动一个组件到临近的边界，您就会看到组件到达边界的距离。（该距离以边界的距离）



lugia mega对组件进行分组会有一个容器的概念。在稍后的章节里面会着重讲解父容器的使用。这里仅说父容器参考线与常规样式的区别。

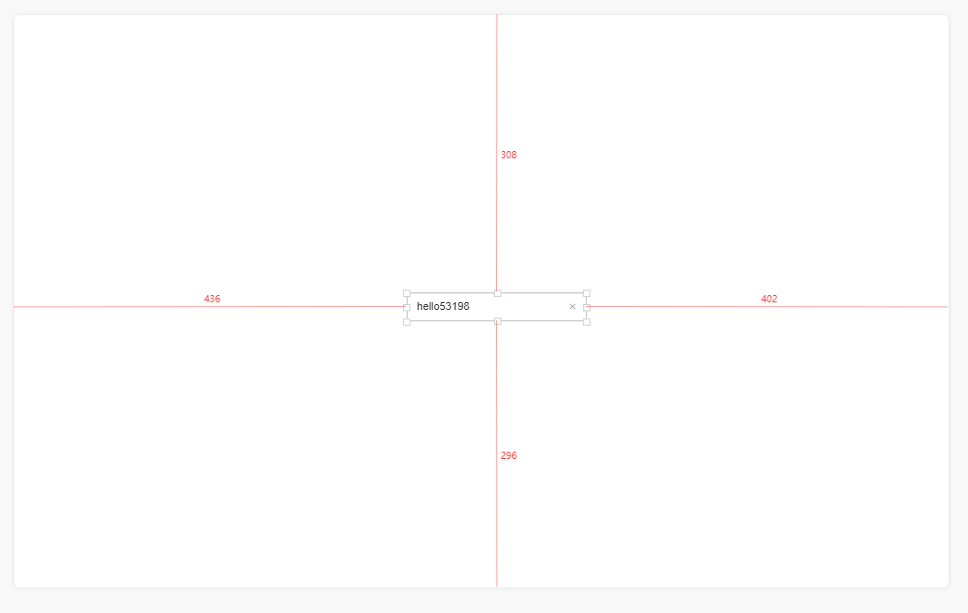
当您拖动几个组件到父容器时，父容器里面的组件仅会以父容器内部的组件之间产生标尺，不会影响到外部的组件。



当然父容器也是一个盒子，使用上面的option（alt）测量距离的功能，您也可以测量组件到父容器的距离。



这时就有一个麻烦的问题如果当您的组件已经在父容器内部，那么在此去测量父容器，那么则显示该组件在父容器的所有距离。



#### 对齐

我们知道如果构建一个组件对齐原则最快捷的办法并不是通过标尺去标记，也不是需要靠辅助线一点点的去测量。而是建立一套完整的对齐系统。



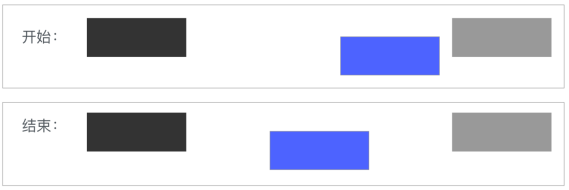
图4.18 mega组件对齐

我们为了让您对组件的把控更加灵活，在右侧检查器的上方会看到一排对齐图标。

横向居中分布（必须组件≥3块才可应用，若<3块则至灰）

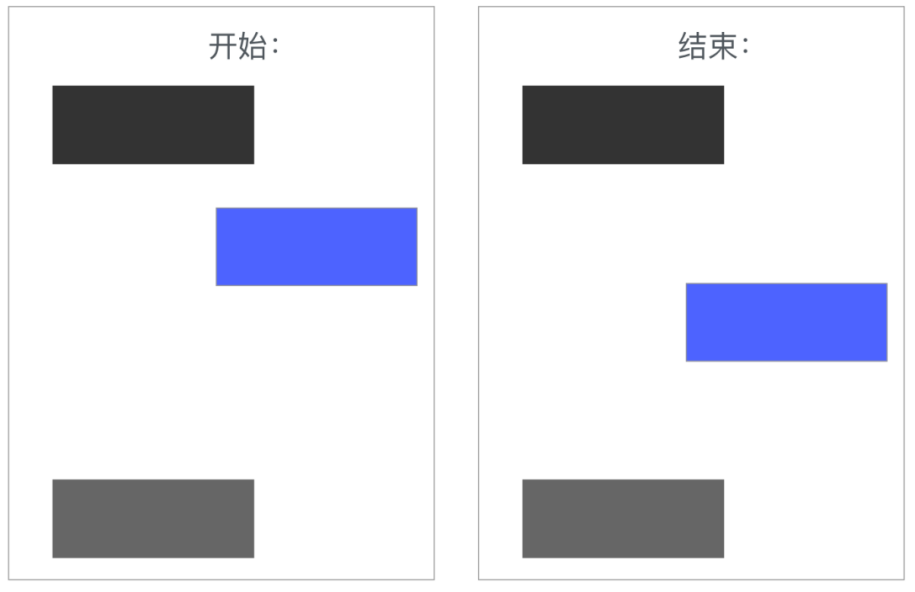
规则：两个组件以前后组件坐标为基准，中间的所有组件，横向居中分布。

示意图如下：



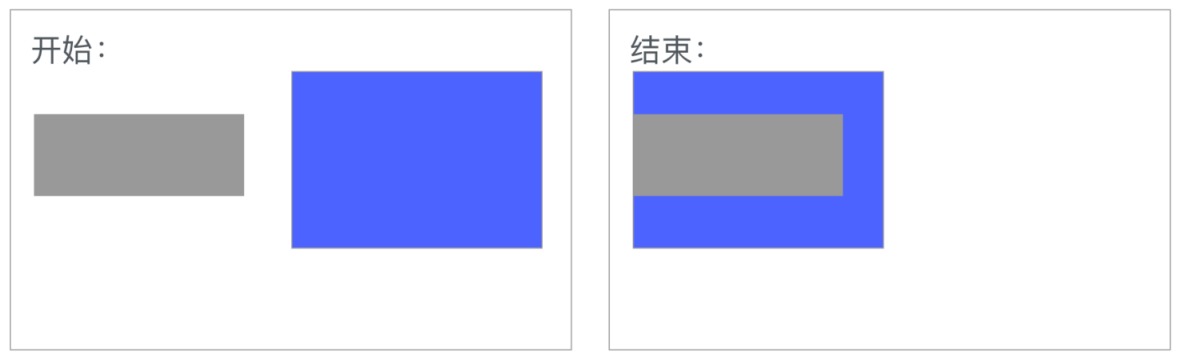
纵向居中分布（必须组件≥3块才可应用，若<3块则至灰）

规则：两个组件以前后组件坐标为基准，中间的所有组件，纵向向居中分布。

示意图如下：

左侧推进对齐（必须组件≥2块才可应用，若<2块则至灰）

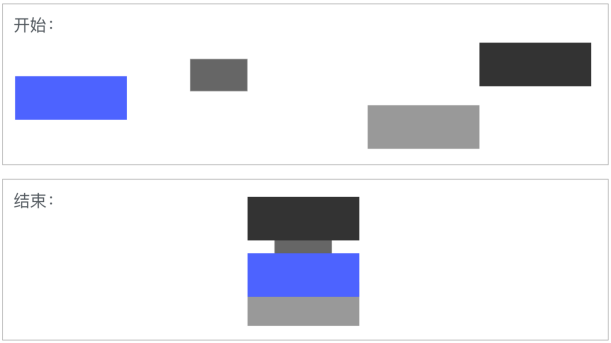
规则：以所选组件左侧位置为基准垂直平行移动。

示意图如下：

垂直居中对齐（必须组件≥2块才可应用，若<2块则至灰）

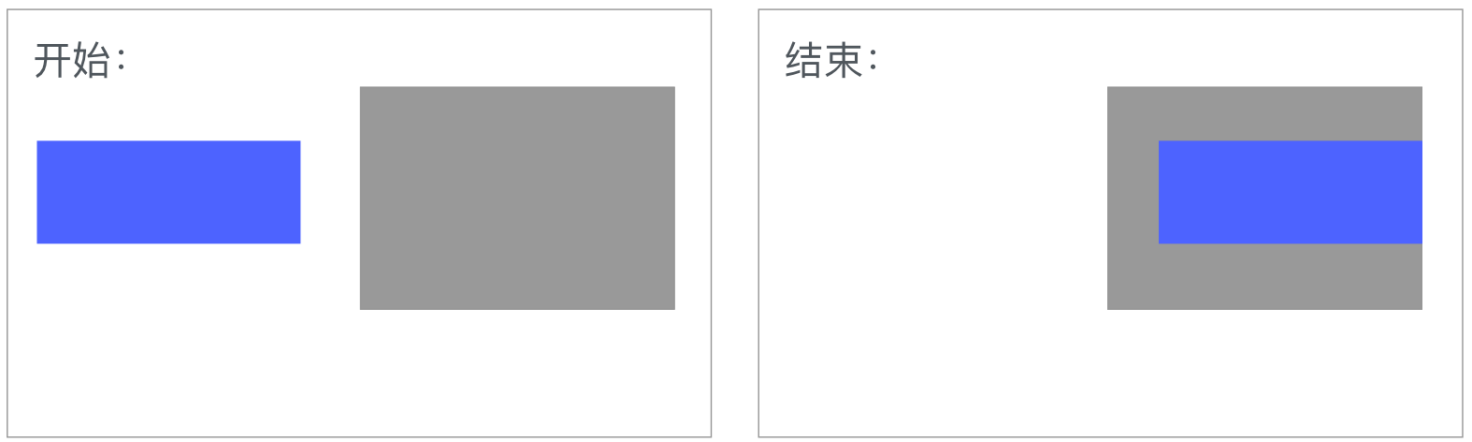
规则：以所选组件中垂线位置 做居中对齐。

示意图如下：



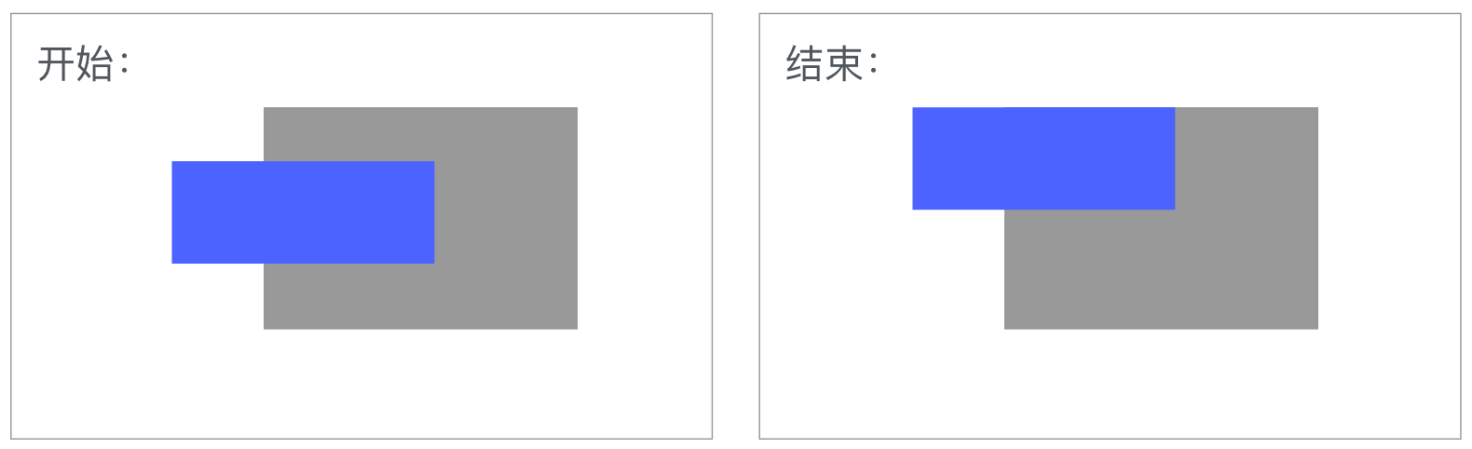
右侧推进对齐（必须组件≥2块才可应用，若<2块则至灰）

规则：以所选组件右侧位置为基准垂直平行移动。

示意图如下：

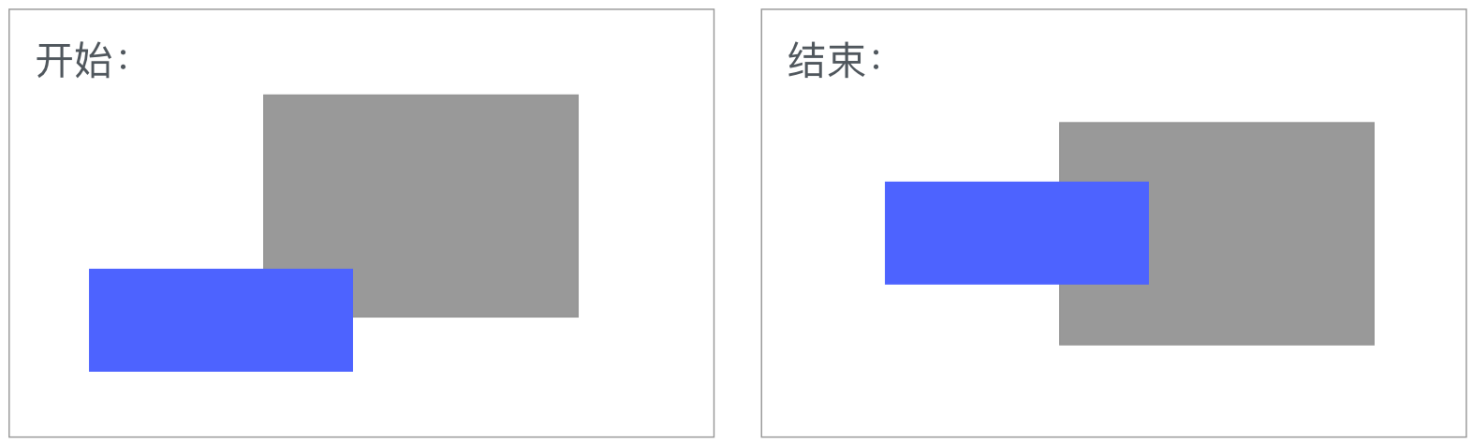
顶部推进对齐（必须组件≥2块才可应用，若<2块则至灰）

规则：以所选组件顶部位置为基准垂直移动。

示意图如下：

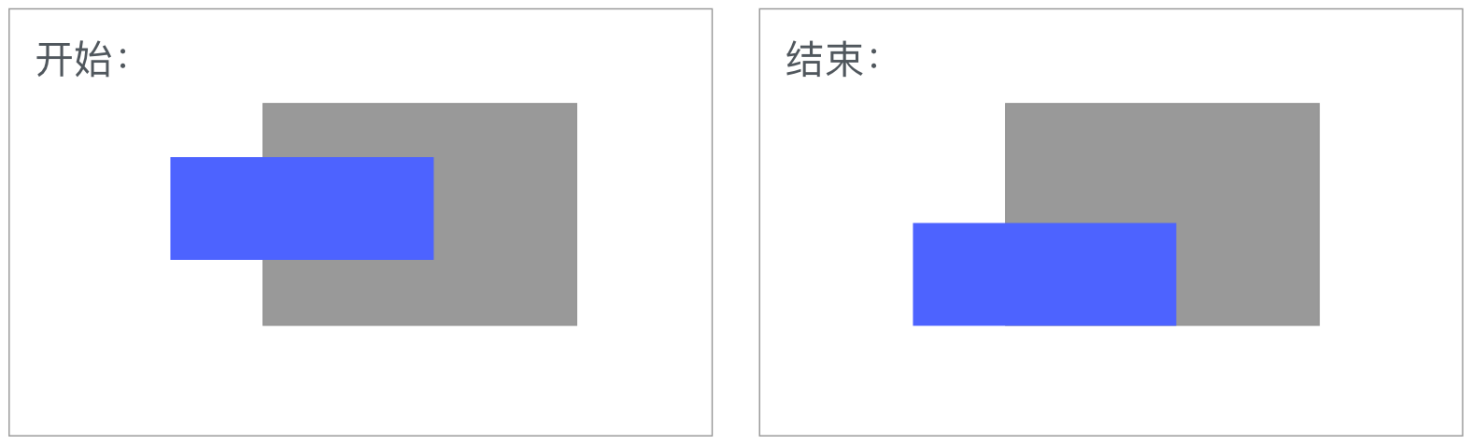
居中对齐（必须组件≥2块才可应用，若<2块则至灰）

规则：以所选组件中线为基准做居中对齐，所有组件根据此基准做向上或向下移动。

示意图如下：

底部推进对齐（必须组件≥2块才可应用，若<2块则至灰）

规则：以所选组件底部位置为基准垂直移动。

示意图如下：

在检查器顶部，我们可以看到除了以上描述的几种对齐按钮。在最左侧开始栏处还有一个类似的画面布局的图标（如左侧所示）。当您用鼠标点击它的时候，会看到一个间距的弹出窗口。



这个功能是为了让您更好的去对齐您的组件。当您选择两个或两个以上组件的时候。您可以使用间距调整功能去让他们保持同一数值排列。

比如说：我选择两个按钮组件。将水平间距设置为20像素，那么这两个组件就会按照20像素的间距水平排列。

#### 窗体缩放

因lugia mega里的画布尺寸是无限延伸的，以至于当页面放大时候也不会因为模糊而看不清具体画布的形态。当您想要放大缩小窗体进行更加合适的操作时候，您可以利用鼠标或键盘的配合。

若您想要放大缩小整个窗体，您可以点击屏幕中下方的浮动条“+或-”以改变当前窗体的位置。注：窗体改变的大小并不会影响实际的组件像素。

31.jpg

图4.19 mega画布窗体缩放

当然对于中级用户来说，每次通过点击才可方法缩小会影响您对页面构建的效率。lugia mega针对方法缩小独立研发几种关于方法缩小的快捷方式。当您用鼠标在屏幕中，您可以按住command（ctrl）+鼠标滚轮来帮助您放大查看画布或对象。您也可以使用键盘上的command+ “+或-”来放大缩小您的窗体。

#### 窗体移动

当您使用了放大缩小功能去移动您的画布，这时您会觉得画面虽然比例改变了，但是并没有屏幕适应到您想要的位置。这个时候就要借助另一项功能“窗体移动了”。

同样窗体移动最简单的办法就是通过方向键来控制所要移动的方向，但每次一像素的点击过于麻烦。因此我们开发了通过点击鼠标左键+键盘空格键的配合，当您按下空格键后您会发现您的鼠标会变成了一个手的形状，这个时候您只需要配合鼠标左键将画板移动到您想要的位置即可。

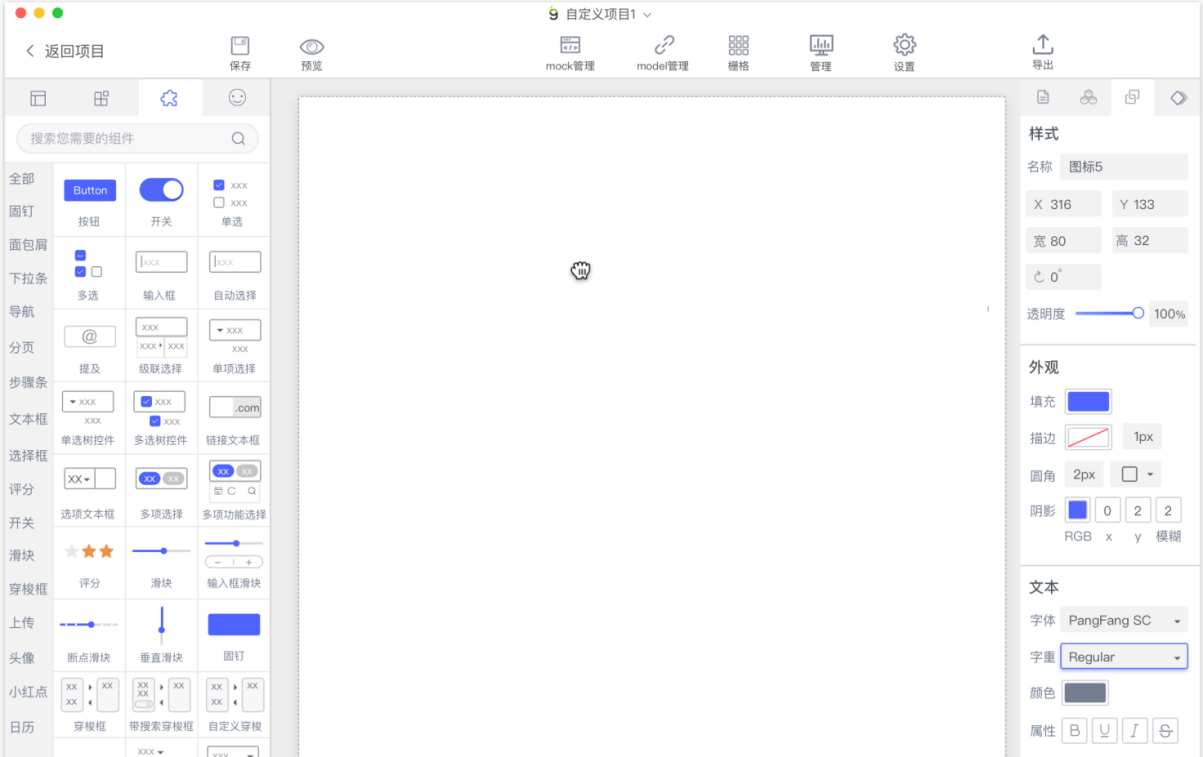


图4.20 mega画布窗体移动

### 上边栏

前面说到上边栏相当于页面的主控台。在这里您可以进行一系列的项目管理。



图4.21 mega上边栏

#### 返回项目

Lugia mega是以一个项目为一个窗体，所以如果在您操作多个项目时，可以通过返回项目去查看， 每一个项目为单独的一个lugia.D项目，我们会将所有您最近使用的项目都保存在启动页中。在返回启动页面中，您可以创建新项目或者打开其他的项目。



图4.22 mega查看所有项目

#### 新建项目

点击新建项目，只需输入页面所需尺寸，就可以开始您的页面设计了。



图4.23 mega新建项目

#### 保存项目

点击保存项目或运用快捷键command+s进行保存。文件格式为lugia专有格式名称xxx.lugiad。

**预览**

预览就是在开发模式下运行应用程序。

您也可以自动在浏览器中打开 http://localhost:3000 去查看它。

注意因为lugia mega不是一个静态页面的设计器而是一个完整的页面构建工具。所以您在操作过程中的任何操作在预览模式下都会实时的显示在浏览器中，页面会根据您的操作自动重新加载。而且您还可以看到构建错误和 lint 警告。

### Model管理

model管理顾名思义是对项目的model进行管理。

您只需点击上边栏的model管理项目就可看到

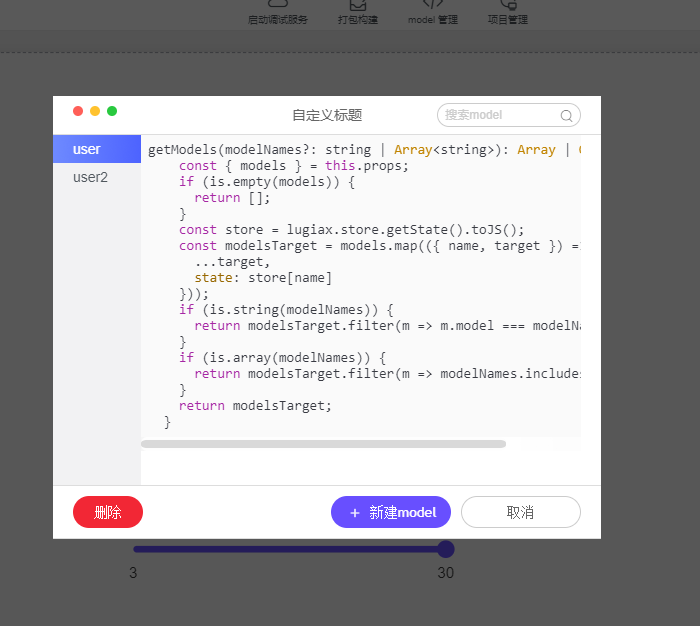


图4.24 mega model管理

### 项目管理

项目管理是用来管理本项目的配置文件。

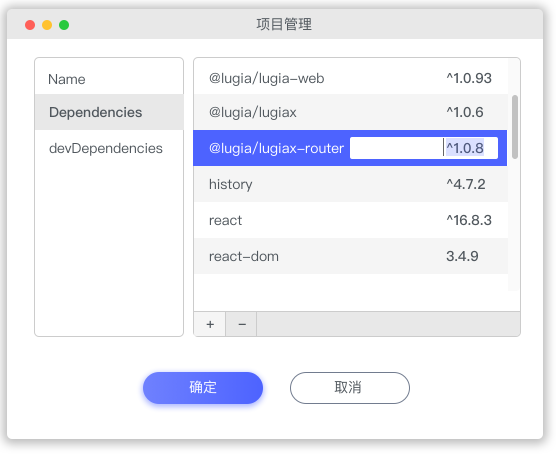


图4.25 mega 项目管理

### 设置

设置功能是对整个lugia mega页面的一些常规项的更改。您可以根据自己的喜好去配置相对应的属性。

#### 物料源

在 lugia 中，组件、区块、布局、模板等统称为物料，由 lugia 团队维护，在内部有一套完整的开发规范和工具；基于此，你可以参与共建，也可以自建私有的物料库。

在官方物料库中：Lugia 团队开发维护的物料库，质量保证，提供技术支持。

在私有物料库：在实际项目中，因为我们提供的物料库和设计风格可能不能满足您所需要的某些业务场景，这时您可以自定义内部的私有物料库。

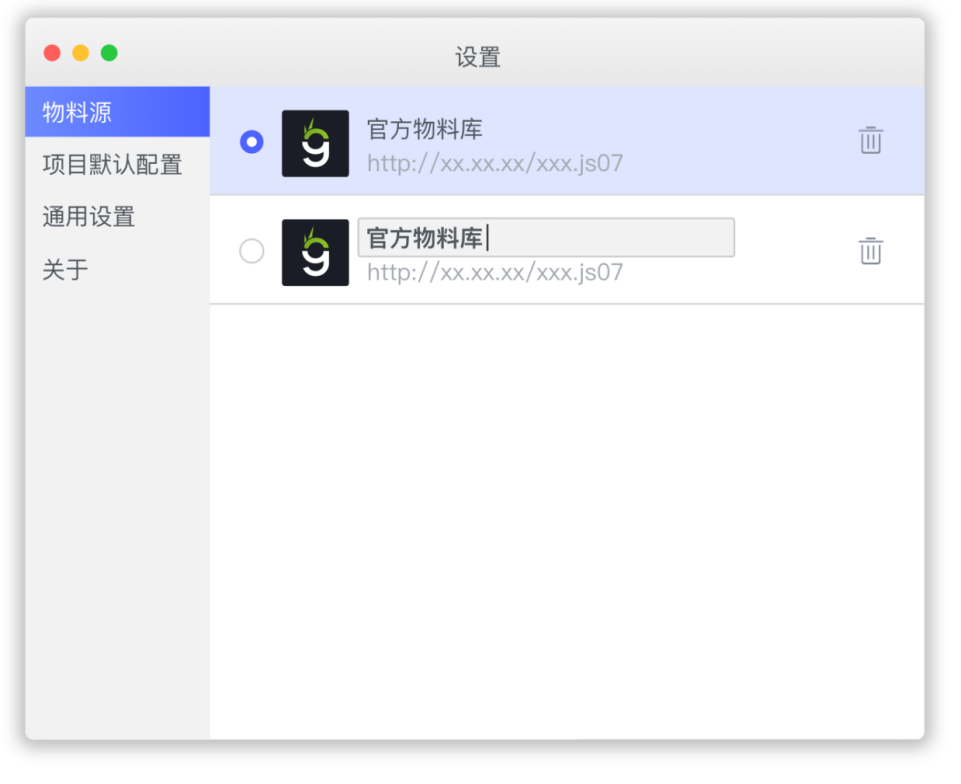


图4.26 mega 物料源设置

#### 项目默认配置

项目默认配置是对默认的用户名、端口号、路径和项目版本的管理。更改会影响您的项目初始值。



图4.27 mega 项目默认设置

您也可以在项目配置中去设置，是否在项目开启是打开浏览器。

#### 通用设置

在通用设置中，您可以选择您所需要的终端和编辑器，以及对应用的物料库进行使用选择或删除。



图4.28 mega 项目默认设置

#### 关于

关于页面为lugia mega当前的版本信息，您可以在此查看lugia mega的相关用户文档，也可以去检测我们最新的lugia mega版本。



图4.29 关于mega

如果您对lugia mega有更多的兴趣，我们欢迎您查看我们的文档和更新的日志。

### 导出

将生产环境需要的应用程序构建到项目 dist 目录。 mega会正确地打包构建并优化整个应用以提供最佳性能。dist下输出文件将会被压缩并包含哈希，以便更加有效的使用缓存策略。dist 下输出的文件可以部署到服务器上。

#### mega提供开发现代单页 React 应用需要的所有环境：

* 开箱即用，包含 start、build 和 test 命令。
* React，JSX，ES6，TypeScript 和 Flow 语法支持。
* 对 Less/Scss、css-modules、postcss、styled-components 的支持。
* CSS Autoprefixer 前缀自动补全。
* 交互式的单元测试，内置支持覆盖率报告功能；基于 jest，包括 UI 测试（基于 enzyme）
* mock 服务，支持引入 json、excel、csv 格式的文件作为数据来源。
* browser-sync 支持，保持多个浏览器和设备同步。
* 实时调试服务，包含错误警告。
* 打包构建脚本，把 js、css、图片及其他资源构建在一起，添加 hash 值和源码映射
* 通过一系列的内部优化，整体构建速度提升了 60% ~ 98%，默认开启缓存后速度会再次提升 300% ~ 500%。bebal 编译速度达到 8.58 runs/sec ±8%。
* 按需加载 Model 和 Router，加快访问速度；同时打包后每个文件体积可以有效控制在 200KB 以内。
* 灵活的配置支持。

#### 代码拆分（Code Splitting）

与用户使用前下载整个应用程序不同，代码分割允许你将代码分割成小块，然后按需加载。mega 会自动对每一个页面做代码切分，以保证更加快速的加载应用。这里使用了动态import() 进行代码拆分。它是在第3阶段的提案。import() 函数表单接受模块名作为参数，并返回一个Promise，该 Promise总是 resolves到模块的命名空间对象。

### 父容器（编组）

父容器，简单的说就是一个组合的功能。该功能是为了更好的将组件拼合成一个整体。您可以以拖拽的方式同时将多个图层放置到一个容器内，这样您就可以随意的移动和缩放，同时也可以在容器内修改每一个独立的组件。

Lugia mega中的父容器功能是非常强大的，因为对于逻辑层来讲，一个父容器就相当于一个简易的画布。您新创建画布上可实现的功能在父容器上都可以实现。

为了防止您的组件因放入父容器而变得不可控（例如sketch：放大缩小组内部图层也会跟随变化）。lugia mega在一定程度上减少了父容器对组件的占有权，也就是说父容器的形状改变并不会影响到内部组件的变化，且父容器内部组件在您的操控范围内也是可以随意拖拽、框选的。它并不需要在进入到一个相对应的父容器页面（例如:adobe illustrator成组后想修改其中内容，需要点击进入）。这样您就可以自由灵活的操控父容器内部的组件系统。

您可能会有疑问，将组件放入父容器，组件与父容器有什么影响呢？

还是回到之前描述的逻辑层关系，一个父容器与内部组件的关系与一个画布上新建一个组件的关系一致。可同时移动，可同享一个标尺系统而已。

#### 创建父容器

若您想要创建一个父容器，您只需要在左侧组件中找到一个命名为盒子的组件。将盒子组件拖拽到画板上，这时页面中就会自动生成一个父容器层。

值得一说的是，您也可以在右侧检查器中修改父容器的属性。将它变成您需要的样式。

#### 编组父容器

若您想将组件拖拽到容器中，您只需要直接将画布上的组件用鼠标拖拽到已经生成好的父容器中就可以了。当把组件放置在父容器中，父容器会有颜色的反馈。且鼠标的光标显示为可添加状态。

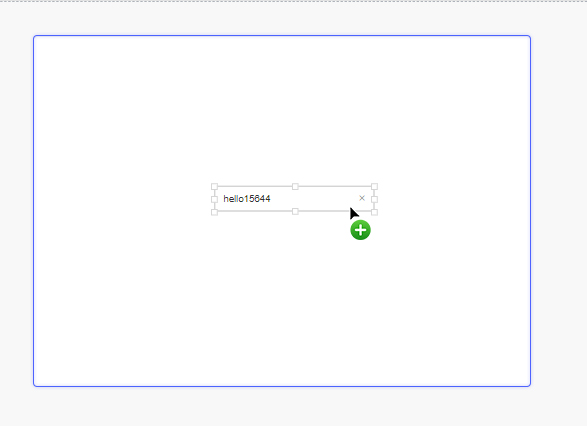


图4.30 编组父容器

### 模型绑定

我们针对lugia mega组件，给予组件一个mode来进行绑定，以便与管理组件的状态。

我们提供了3个步骤：“首先您要选择一个要绑定的组件，为组件指定绑定的模型。

其次，通过组件内部触发的事件来更新model mutation，实时更新model状态。

最后您可以通过模型绑定来将组件的部分属性与模型相关联从而达到模型更新组件的目的。我们通过实时读取lugia x的模型文件，来实时加载到lugia mega内部。



图4.31 mega模型绑定

#### 操作

当您选择一个组件的时候，您可以在检查器上选择绑定窗口。

具体操作步骤如图所示：

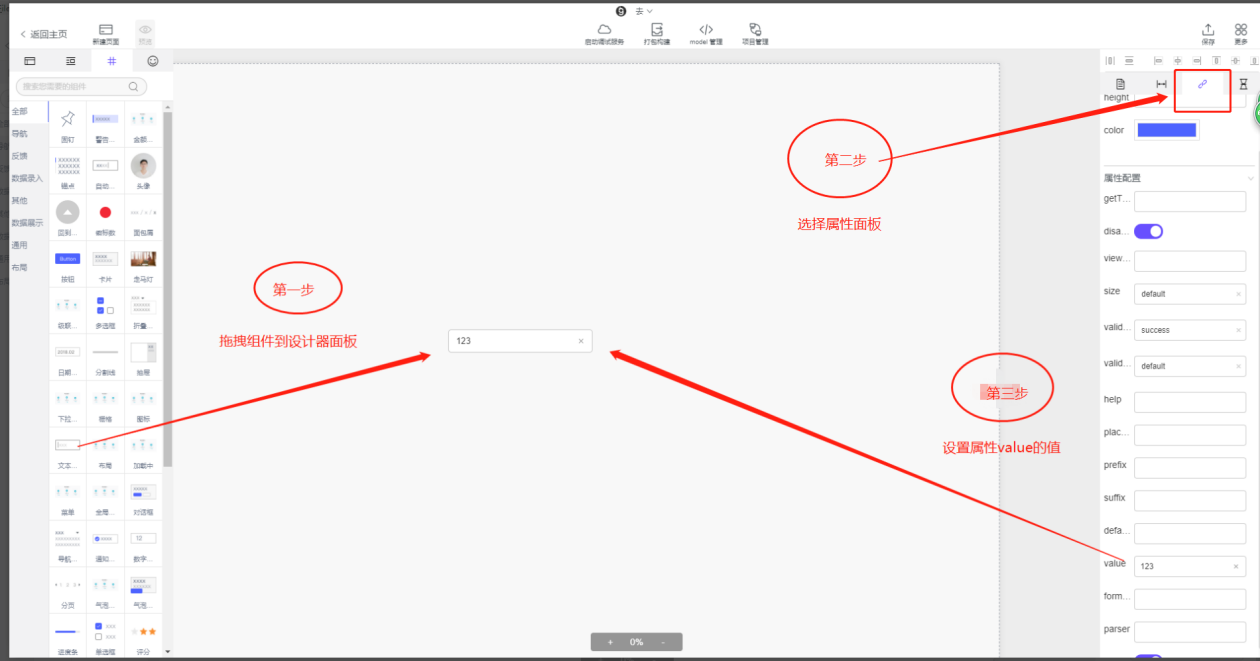


图4.32 mega组件单项绑定

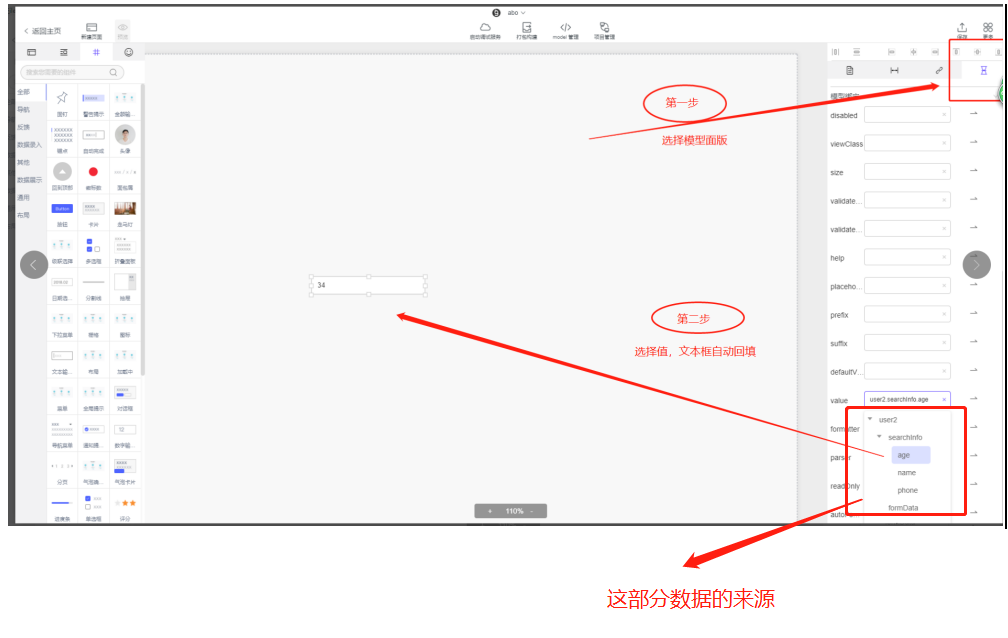


图4.33 mega组件双向绑定

这个文件所在的位置是新建项目时，根据设置的路径存放的；



图4.34 mega模型存储位置

### 快捷键

lugia mega有一部分为数不多的快捷键，来为您服务。当您掌握之后可以在一定时间内提高您的工作效率。

#### 通用快捷键

* Space：抓手工具
* command+c：复制
* command+x：剪切
* command+v：复制
* Option+鼠标左键：拖动复制
* option+shift+鼠标左键：拖拽平移复制
* command+z：撤销操作
* Backspace：删除

#### 移动图层 & 更改尺寸

* Shift+鼠标左键移动：基于拖动的方向锁定相对应的x\y轴坐标
* Shift+鼠标单击组件：可以进行多选
* 按住锚点+shift：等比例缩放
* 按住锚点+option：组件以中心为居中点进行缩放
* 按住锚点+shift+option：组件以中心为居中点进行等比例缩放

# 实现方案

### 整体方案

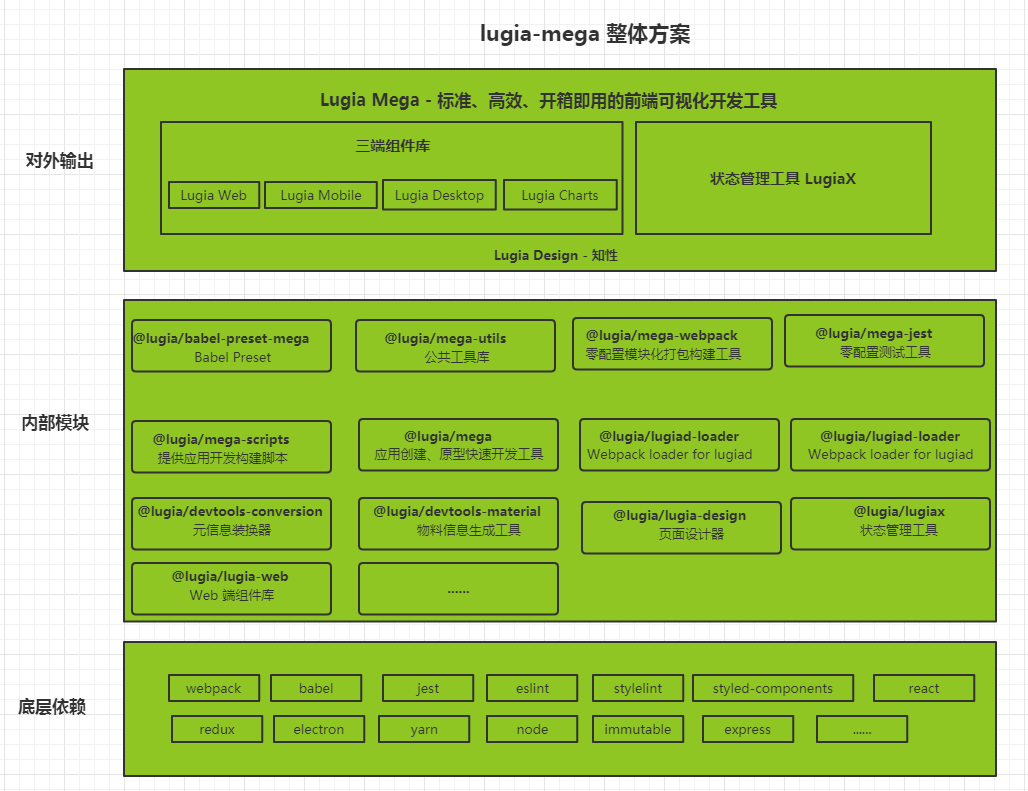


图5.1 lugia-mega 整体方案

lugia-mega 是 **多端一体化** 研发成果对外输出的主要承载体。设计语言、三端组件库、状态管理工具等研发成果会与 lugia-mega 整体打包向用户输出使用。

现代前端的技术体系深度依赖node和npm社区，所以Lugia-mega 各个功能点的实现是分散到多个单独模块的。这些功能模块会通过组合的形式来支撑 lugia-mega。

### 用户使用流程

因为 lugia-mega 的用户群体大概分为两类，一类是负责设计页面的设计师或者产品经理，一类是负责业务逻辑开发的前端工程师或者后端工程师，所以在设计实现方案时需要考虑到两类用户的不同使用习惯。

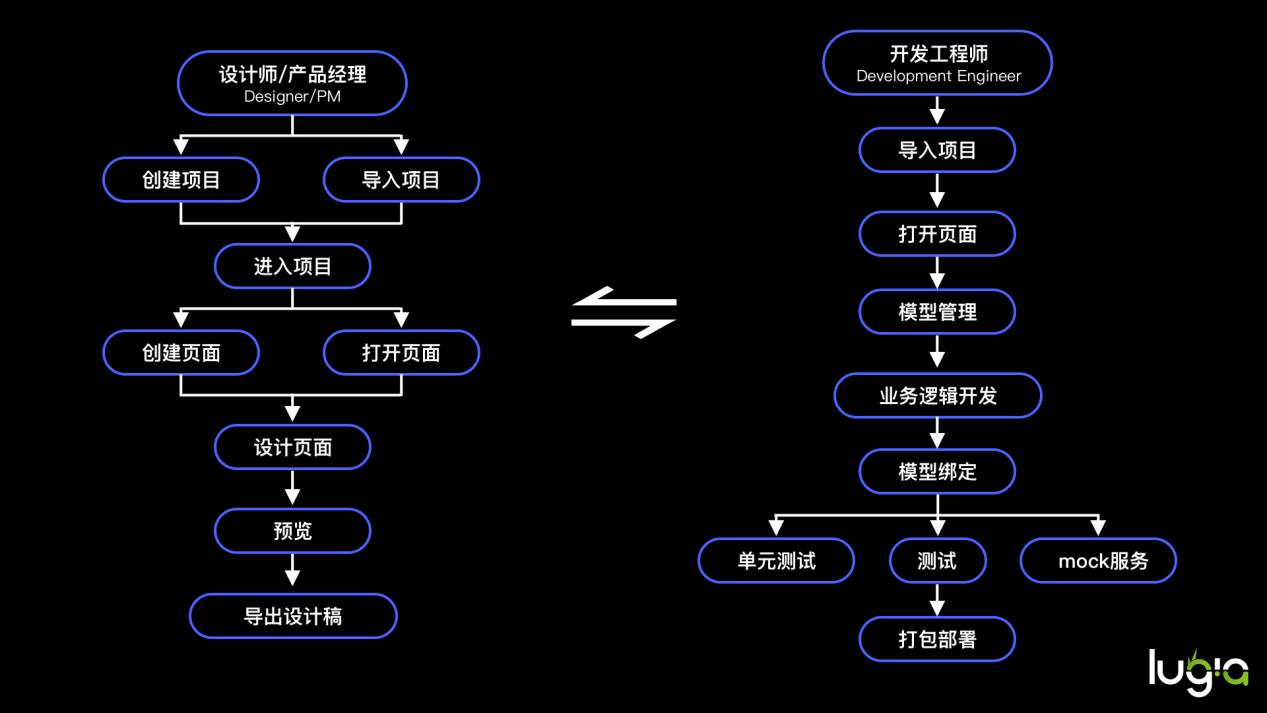


图5.2 lugia-mega用户群体使用流程

#### 设计师/产品经理

对于设计师/产品经理来说，关心的是产品的实现方案和交互设计，一般情况下使用最多的功能部分是项目管理、页面设计器、物料体系。

#### 开发工程师

对于开发工程师来说，关系的是业务逻辑的开发，重点在于项目模型的设计和实现，一般情况下使用最多的功能部分是项目管理、开发工具链、模型规范。

#### 功能划分

lugia-mega 会在本地分析不同用户群体的使用习惯，对于不常用的功能点做弱化处理。

用户使用习惯的分析数据（以下简称用户使用数据）会保存到本地，以便供 lugia-mega 使用。

当用户使用 lugia-mega 的时候可以选择使用习惯，配置会保存到用户使用数据中。

用户使用数据会影响到 lugia-mega 的交互方式，但是功能点并不会消失，只是在使用入口上会弱化。

### 实现技术

#### 底层依赖

lugia-mega 底层依赖 [node](https://nodejs.org)、[webpack](https://webpack.js.org/)、[bable](https://babeljs.io/)、[electron](https://electronjs.org/)、[react](https://reactjs.org/) 、[redux](https://redux.js.org/)等开源技术。

webpack专门处理各个资源模块化构建，而 gulp 会补充 webpack 一些功能较为薄弱的环节，例如多线程压缩 css、js，文件监听。

开发调试、打包构建部分基于 webpack 体系来完成。

以 electron 为基础构建框架开发桌面应用，兼容主流的 macOS 以及 Windows 64 系统。

视图层使用 React 技术栈。

模型规范基于 Redux 实现。

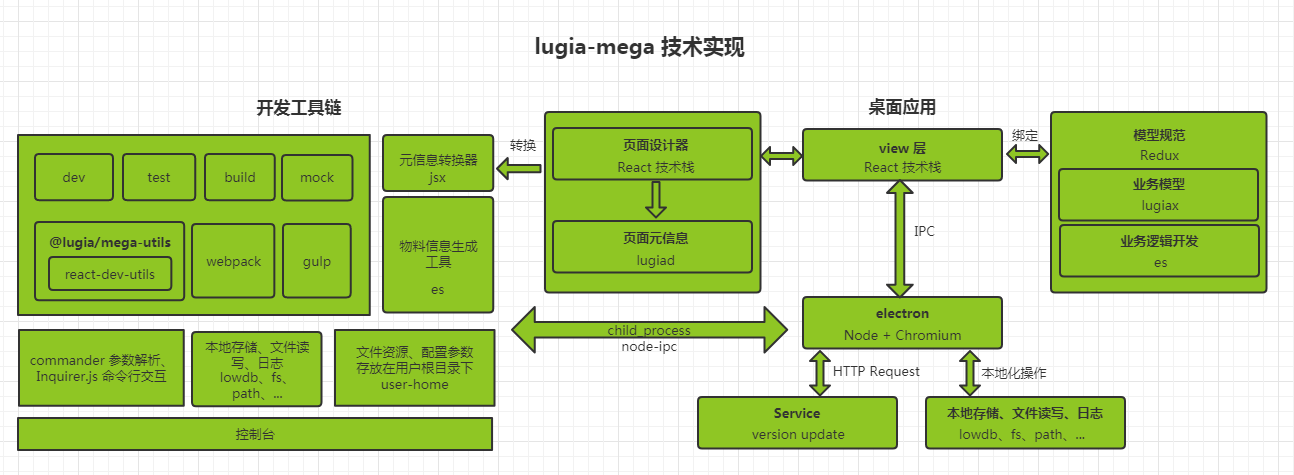


图5.3 lugia-mega 技术实现

#### 内部模块

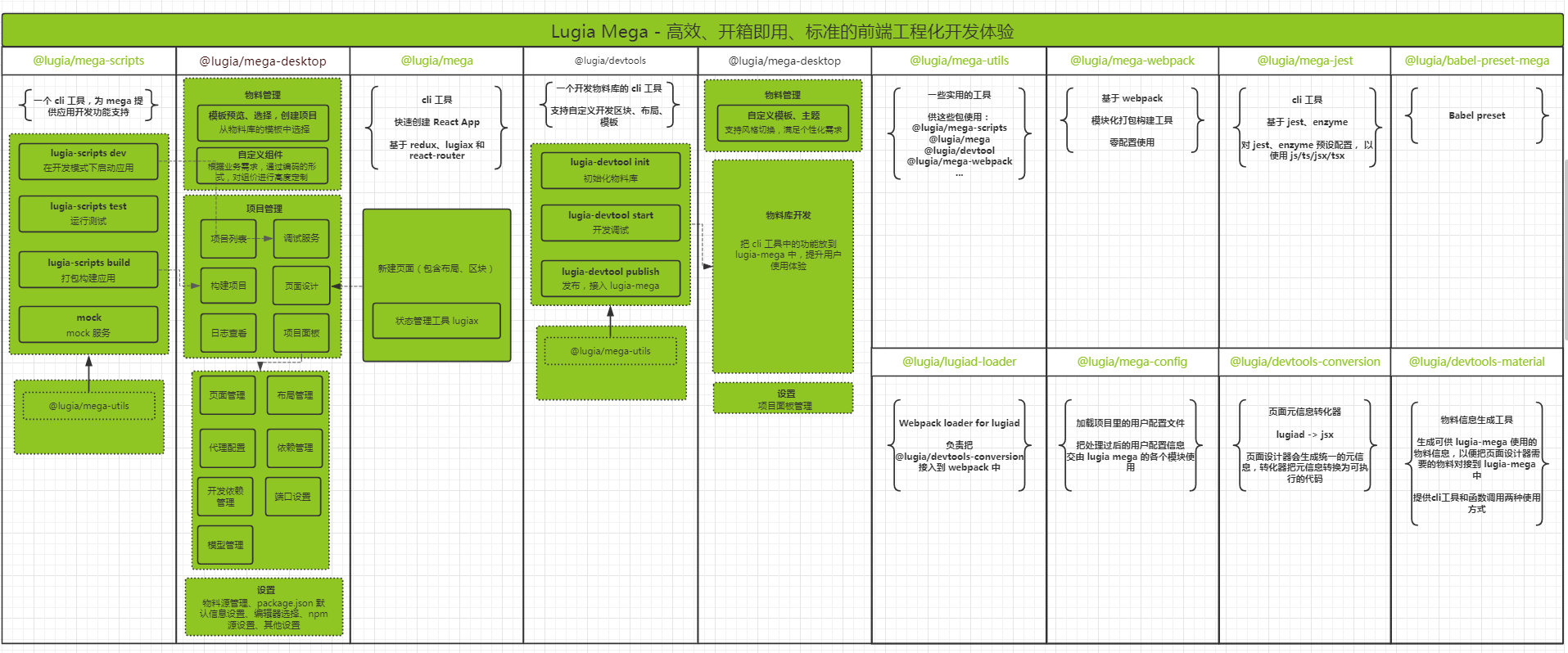


图5.4 lugia-mega内部模块

#### 相关

* [commander](https://github.com/tj/commander.js) 让命令行界面开发变得简单。
* [react-dev-utils](https://github.com/facebook/create-react-app/blob/next/packages/react-dev-utils/README.md) 一些打包构建相关的模块，[create-react-app](https://github.com/facebook/create-react-app) 的核心代码。
* [webpack-merge](https://github.com/survivejs/webpack-merge) 合并 webpack 配置。
* [node-ipc](https://github.com/RIAEvangelist/node-ipc) 进程间通信。
* [inquirer](https://github.com/SBoudrias/Inquirer.js) 更好的命令行交互。
* [user-home](https://github.com/sindresorhus/user-home) 跨平台获取用户根目录。
* [Nunjucks](https://github.com/mozilla/nunjucks) 功能强大的模板引擎。
* [download-git-repo](https://github.com/flipxfx/download-git-repo) 下载 git 上的资源。

### 项目规范

符合 lugia-mega 规范的项目（以下简称mega项目）才可以在 lugia-mega 中使用。

#### 目录结构

符合 lugia-mega 规范的项目目录结构如下：

* config/配置目录
  + mega.desktop.config.json mega-desktop配置信息
  + pages.config.json 页面配置信息
  + routing.config.js 路由配置信息
* dll/启动开发调试服务时生成的dll文件
  + DevDLL.dependencies.json 实际构建的dll依赖项
  + DevDLL.js 构建生成的dll文件
  + DevDLL.manifest.json dll的索引树
  + ...... 构建dll时生成的其他资源项
* mock/ mock服务数据文件
  + index.js mock 入口文件
  + ...... mock数据
* node\_modules/模块依赖
* portal/预置的项目框架
  + index.js 项目的入口文件
  + ...... 其他文件

# public/公共资源文件夹

* + index.html webpack构建时所需的html文件
  + favicon.ico 应用图标
  + manifest.json 应用资源清单
* src/项目源码目录
  + components/自定义组件目录
  + models/模型目录
  + pages/页面目录
* test/测试文件
* .editorconfig EditorConfig配置文件
* .gitignore git忽略配置文件
* lugia.config.js lugia配置文件
* mock.config.js mock配置文件
* package.json 项目配置文件
* README.md 项目说明文件
* yarn.lock yarn依赖锁定文件

#### 工作区

由lugia-mega管理的项目除了被用户看到的项目文件外，还会在系统用户目录下生成一个mega工作区。mega工作区中主要存储着下载的项目依赖包和项目在使用过程中产生的缓存文件。

**为什么需要工作区？**

* 工作区是设置软件包体系结构的一种新方式，默认情况下从 Yarn 1.0 开始使用。工作区允许你可以使用这种方式安装多个软件包，就是只需要运行一次”yarn install”便可将所有依赖包全部安装。
* 项目的依赖包可以链接在一起，这意味着工作区中的项目可以相互依赖，同时始终使用最新的可用代码。 这也是一个比 ”yarn link “更好的机制，因为它只影响你工作区的依赖树，而不会影响整个系统。
* 由mega管理的所有项目依赖将被安装在一起，这样可以让 Yarn 来更好地优化它们，至少工作区可以有效的减少node\_modules所占用的存储空间。
* Yarn 将使用一个单一的 lock 文件，而不是每个项目多有一个，这意味着更少的冲突和更容易进行代码检查。
* 隔离mega在编辑项目时所需的文件，不会对项目产生副作用。这样，由mega编辑过的项目也可以脱离mega独立运行、构建等。

#### 安装依赖

由于实际的依赖包存储在工作区，每次在安装时需要先同步项目和工作区。

1. 使用软链接工作区的node\_modules到项目中，以便项目运行时可以找到所需要依赖包。
2. 拷贝项目中的package.json和yarn.lock到工作区，在工作区安装依赖时必须的文件。
3. 在工作区中使用yarn安装依赖。

**为什么要使用软链接？**

在node里链接有link（硬链接）和symlink （软链接）两种：

**link：**

* 文件有相同的 inode 及 data block。
* 只能对已存在的文件进行创建。
* 不能交叉文件系统进行硬链接的创建。
* 不能对目录进行创建，只可对文件创建。
* 删除一个硬链接文件并不影响其他有相同 inode 号的文件。

**symlink：**

* 软链接有自己的文件属性及权限等。
* 可对不存在的文件或目录创建软链接。
* 软链接可交叉文件系统。
* 软链接可对文件或目录创建。
* 创建软链接时，链接计数 i\_nlink 不会增加。
* 删除软链接并不影响被指向的文件，但若被指向的原文件被删除，则相关软连接被称为死链接（即 dangling link，若被指向路径文件被重新创建，死链接可恢复为正常的软链接）。

综上所述，使用symlink（软链接），但在使用时需要注意：

* 在 windows 下，symlink 有权限限制，需要管理员权限。
* 参数type 设为 junction，不需要权限，不能链接文件，只能链接目录。
* 一些安全软件，比如 360 会阻止创建链接，用户需要允许操作。

#### 文件用途

1. 与 lugia mega 强依赖的项目数据、配置信息的文件：

* ./config/mega.desktop.config.json
* ./config/pages.config.json
* ./config/routing.config.js
* ./lugia.config.js
* ./mock/index.js

这些文件有以下特征：

* 需要存储在项目目录下。
* 均为相对路径。
* 由 mega 生成、管理。
* 不允许用户修改。
* 有这些文件的项目才会被认为是一个 mega 项目。

1. 存放用户的项目源码

* ./src/pages
* ./src/components
* ./src/models

这些文件有以下特征：

* 主要有页面、模型、自定义组件、资源等。
* 需要存储在项目目录下。
* 相对路径。

1. 项目缓存、临时文件

* ./cache
* customComponents.entry.js
* lugiaWeb.entry.js

这些文件有以下特征：

* 由 mega 生成、管理。
* 不允许用户修改。
* 不需要用户放到git、svn等上保存。
* 存储到工作区。
* 相对路径。

1. 其他文件

项目运行所需要的其他文件。

#### .lugiad页面文件规范

在mega中设计的页面会生成中间文件.lugiad。这是一个json格式的文件，存储着页面的所有信息。

**主要字段说明：**

* widgetId2ChildPad：叶子节点组件信息
* mainDependencies：组件依赖信息
* mainPad：根组件信息
* point：组件坐标信息
* layers：根组件所包含的叶子组件
* id2WidgetInfo：根组件所包含的叶子组件信息
* themes：组件主题信息
* lugiax：组件绑定信息
* widgetId2PropsName2BindInfo：组件绑定props信息
* widgetId2EventName2MutationInfo：组件绑定事件信息
* layoutInfos：响应式分辨率信息
* mode2Config：屏幕尺寸下对应的页面信息
* mainPadSize：页面大小
* widthRange：屏幕分辨率范围
* mode2LayoutData：屏幕尺寸下对应的组件信息
* point：组件定位
* zip：是否压缩过

### 约定式路由

mega项目根据 pages 目录自动生成路由配置。

#### 基础路由

假设 pages 目录结构如下：

+ pages/

+ users/

- list.js

- users.js

- index.js

那么会自动生成路由配置如下：

[

{ value: '/, path: './pages/index.js' },

{ value: '/users', path: './pages/users.js' },

{ value: '/users',

children: [{ value: '/users/list', path: './pages/users/list.js' }]

}

]

#### 动态路由

约定带 $ 前缀的目录或文件为动态路由。

比如以下目录结构：

+ pages/

+ $post/

- index.js

- comments.js

+ users/

- $id.js

- index.js

会生成路由配置如下：

[

{ value: '/, path: './pages/index.js' },

{ value: '/users',

children:[{ path: '/users/:id', path: './pages/users/$id.js' }]

},

{ value: '/:post', path: './pages/$post/index.js',

children:[{ value: '/:post/comments', path: './pages/$post/comments.js' }]

}

];

#### 可选的动态路由

约定动态路由如果带 $ 后缀，则为可选动态路由。

比如以下结构：

+ pages/

+ users/

- $id$.js

- index.js

会生成路由配置如下：

[

{ value: '/', path: './pages/index.js' },

{ value: '/users',

children:[{ value: '/users/:id?': path: './pages/users/$id$.js' }]

}

]

### 自定义组件

为了满足多种多样的业务需要，除了使用官方提供的组件库设计页面，还可以通过编程的方式添加自定义组件，并接入到mega中使用

#### 目录结构

* ./src/components/ 自定义组件目录
  + - btn/ 自定义组件Btn
      * index.js 入口文件
      * lugia.btn.zh-CN.json 组件元信息
      * btn.png 组件缩略图

#### 文件规范

* 组件文件夹命名单词之间用”-“连接
* 组件源信息命名格式为：lugia.{文件夹名}.zh-CN.json。如果有子组件需要导出，子组件源信息命名与父组件相同，需要把文件夹名换位父组件源信息childrenWidget字段中的命名。
* 组件缩略图命名为：{文件夹名}.png。我们仅支持png类型的图片，默认大小不超过10KB，可通过limit更改图片大小限制。子组件缩略图命名为 {父组件源信息childrenWidget字段命名}.png；如果没有配置缩略图，则展示默认图片。

#### 元信息规范

整个元信息存储在JSON文件中，以JSON对象的形式描述对应的组件信息。

例如：

**lugia.checkbox.zh-CN.json：**

{

"widgetName": "Checkbox", //组件名称

"title": "多选框", //组件展示标题

"desc": "多选框。", //组件详细说明

"props": { //组件属性说明

"checked": { //属性名

"type": "boolean", //该属性类型

"desc": "指定Checkbox是否选中", //该属性详细描述

"defaultValue": "undefined" //该属性的默认值，没有为 undefined

},

......

},

"events": { //组件事件说明

"onChange": { //事件名

"desc": "Checkbox改变时回调", //事件详细描述

"args": [ //事件参数

{

"name": "event", //参数名

"desc": "关闭时的DOM事件", //参数详细描述

"type": "Object" //参数类型

},

......

]

}

},

"type": { //类型抽取，如props 属性类型有多种，可抽取类型到type

"CheckboxType": ["default", "vertical"],

"CheckboxStyle": {

"width": {

"type": "number",

"desc": "Checkbox宽度"

},

"color": {

"type": "string",

"desc": "Checkbox颜色"

}

}

},

"childrenWidget": ["checkbox-group"], //子组件源信息

"category": ["数据录入"] //组件类型描述

}

**lugia.checkbox-group.zh-CN.json：**

{

"widgetName": "CheckboxGroup",

"title": "多选框组",

"desc": "多选框组。",

"props": {

......

},

"events": {

......

},

"type": {

......

},

"category": ["数据录入"],

"componentName": "Group", //导出组件组件名

"needExport": true //组件是否需要导出，用于需要单独导出的子组件

}

### 接入到mega

在mega项目中，页面、官方组件库、自定义组件、模型等需要接入到mega中使用，以便对整个项目的操作都可以在mega中完成。

#### 页面接入

项目里的所有页面都会接入到mega中进行管理，对于页面的创建、设计、删除等操作都可以在mega中完成。

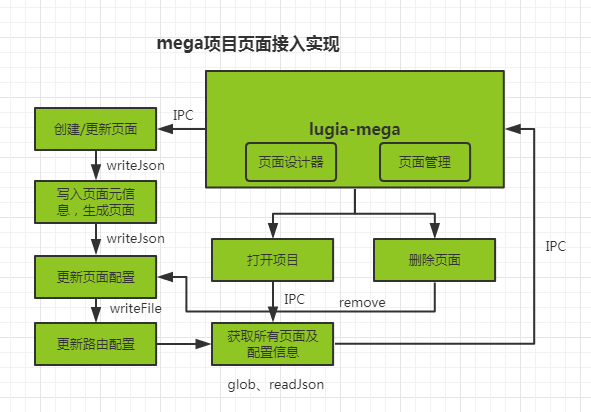


图5.5 lugia-mega 项目页面接入实现

#### 官方组件库接入

mega创建的项目中会默认使用 @lugia/lugia-web组件库。

项目中的@lugia/lugia-web组件库会接入到mega 中作为设计页面的物料使用。

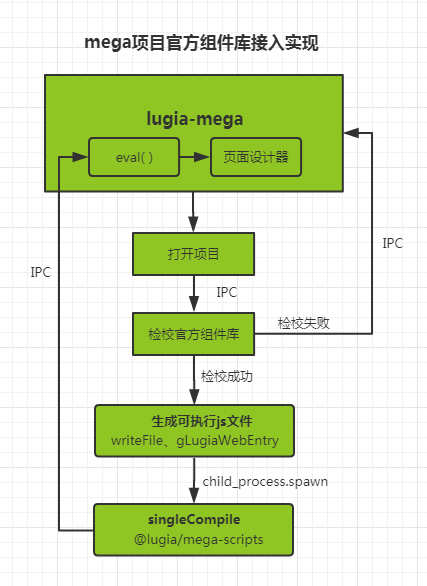


图5.6 lugia-mega 项目官方组件库接入实现

#### 自定义组件接入

在上一节中所述的自定义组件会接入到mega中作为设计页面的物料使用。

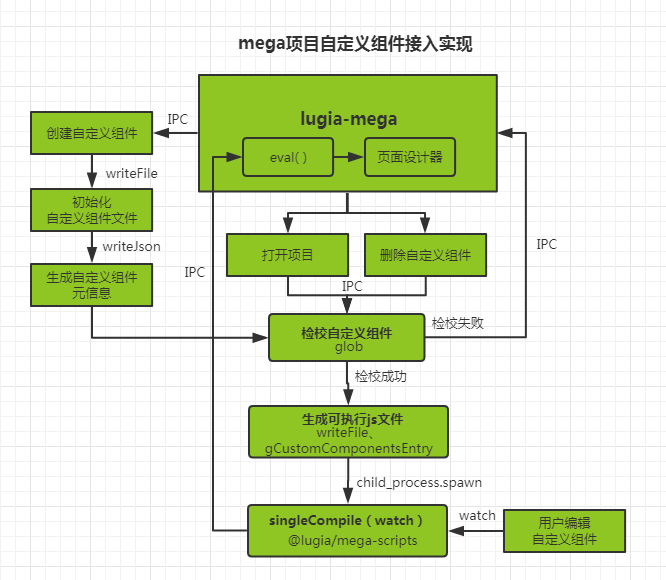


图5.7 lugia-mega 项目自定义组件接入实现

#### 模型接入

项目的业务逻辑需要通过编码业务模型的方式实现。

项目里的所有模型都会接入到mega中进行管理，对于模型的创建、查看、删除、单项绑定、双向绑定等操作都可以在mega中完成。

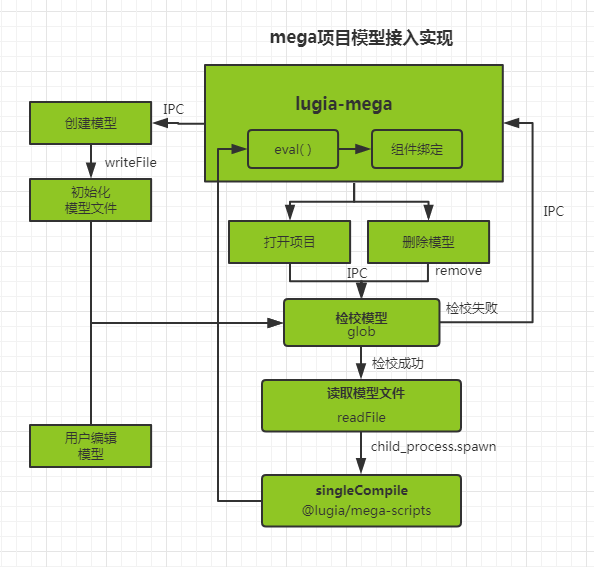


图5.8 lugia-mega项目模型接入实现

# 模块设计

### @lugia/mega-utils

公共工具库，主要为@ lugia/mega- scripts ,@ lugia/mega , @ lugia/mega- webpack 服务。

#### 包含

* 文件的读写、处理。
* 路径的处理。
* 命令行界面工具，颜色、清屏、状态、日志输出......
* 私有的 babel preset 和 plugin
* 私有的 webpack loader 和 plugin
* 私有的 gulp plugin

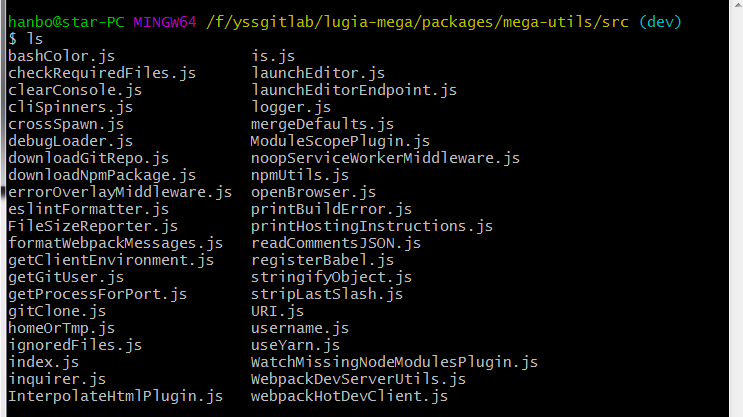


图6.1 @lugia/mega-utils模块现有工具

#### 模块依赖

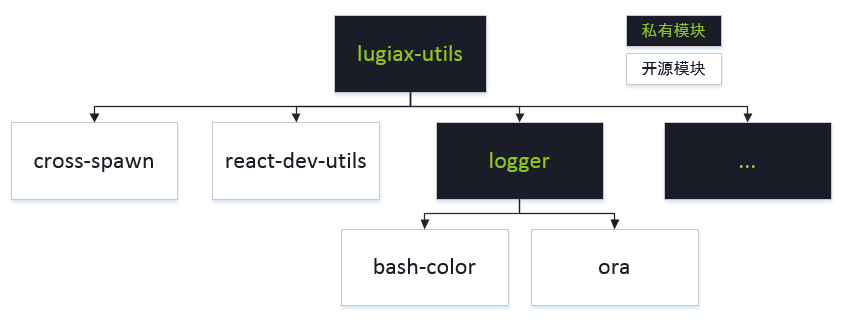


图6.2 @lugia/mega-utils模块依赖

#### 使用

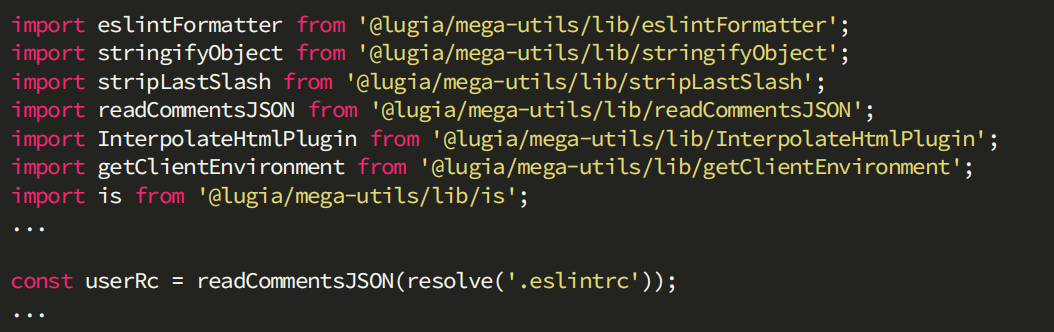


图6.3 @lugia/mega-utils使用

#### 相关模块

* [babel](https://github.com/babel/babel)
* [webpack](https://github.com/webpack/webpack)
* [gulp](https://github.com/gulpjs/gulp)
* [bash-color](https://github.com/mbilokonsky/bash-color)
* [react-dev-utils](https://github.com/facebook/create-react-app/blob/next/packages/react-dev-utils/README.md)
* [cross-spawn](https://github.com/moxystudio/node-cross-spawn)
* [ora](https://github.com/sindresorhus/ora)

### @lugia/mega-webpack

对 webpack 的封装，零配置使用。

#### 模块依赖

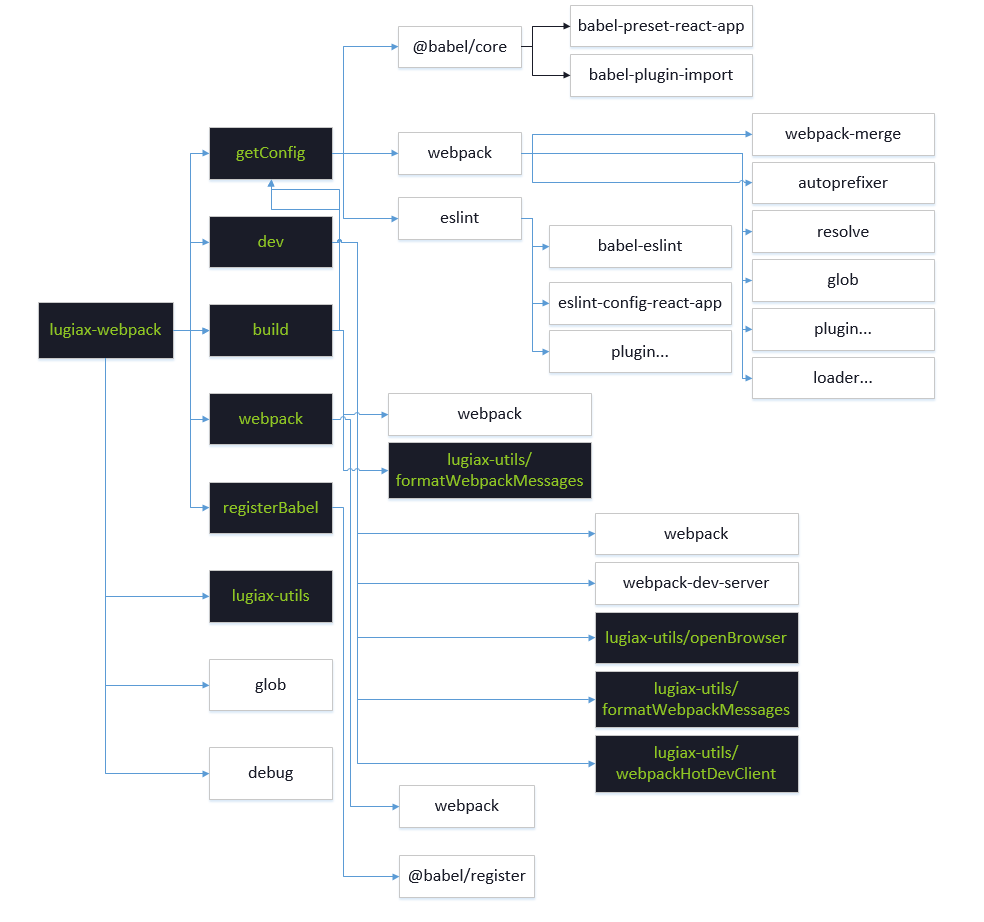


图6.4 @lugia/mega-webpack 模块依赖

#### API

* **getConfig**

通过参数获取 webpack 配置。

在开发或者构建时使用，会获取预设的配置，包含构建 react 技术栈应用的所有 loader 和 plugin



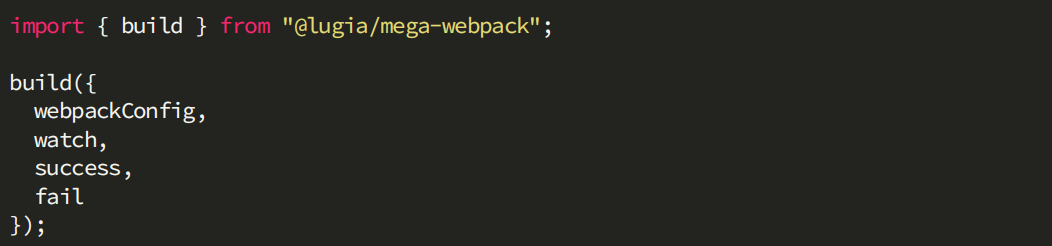
* **dev**

启动服务，开发。



* **build**

打包构建。



* **webpack**

**webpack导出，方便使用webpack的api**



* **registerBabel**

**对@babel/register的封装，零配置使用。**



#### 相关模块

* [debug](https://github.com/visionmedia/debug)
* [glob](https://github.com/isaacs/node-glob)
* [resolve](https://github.com/browserify/resolve)
* [webpack-merge](https://github.com/survivejs/webpack-merge)
* [babel-plugin-import](https://github.com/ant-design/babel-plugin-import)
* [autoprefixer](https://github.com/postcss/autoprefixer)
* [eslint](https://github.com/eslint/eslint)
* [babel-eslint](https://github.com/babel/babel-eslint)

### @lugia/mega-jest

前端测试框架。对 jest、enzyme 预设配置，适用于[Babel](https://babeljs.io/)、[TypeScript](https://www.typescriptlang.org/)、[Node](https://nodejs.org/en/)、[React](https://reactjs.org/)、[Angular](https://angular.io/)、[Vue](https://vuejs.org/)等项目。

#### 在命令行中使用

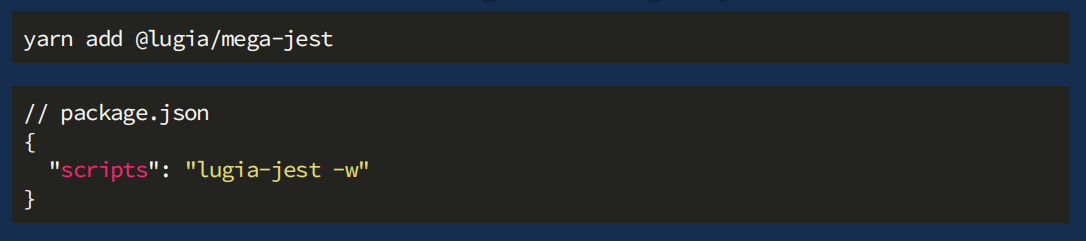


图6.5 @lugia/mega-jest 在命令行中使用

#### 使用API

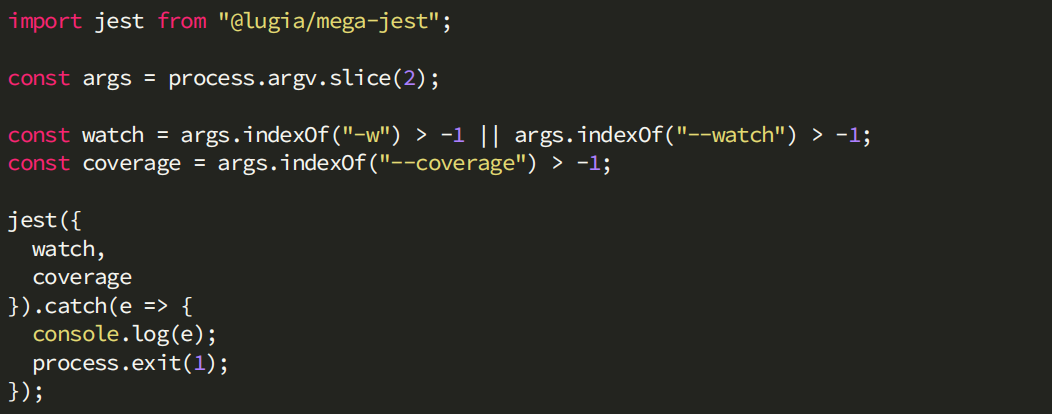


图6.6 @lugia/mega-jest 使用API

### @lugia/babel-preset-mega

babel@ 7 preset

#### 使用

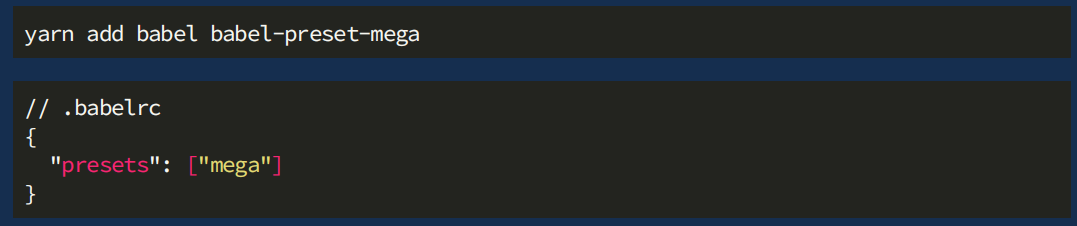


图6.7 @lugia/babel-preset-mega 使用

### @lugia/devtools-material

#### 功能说明

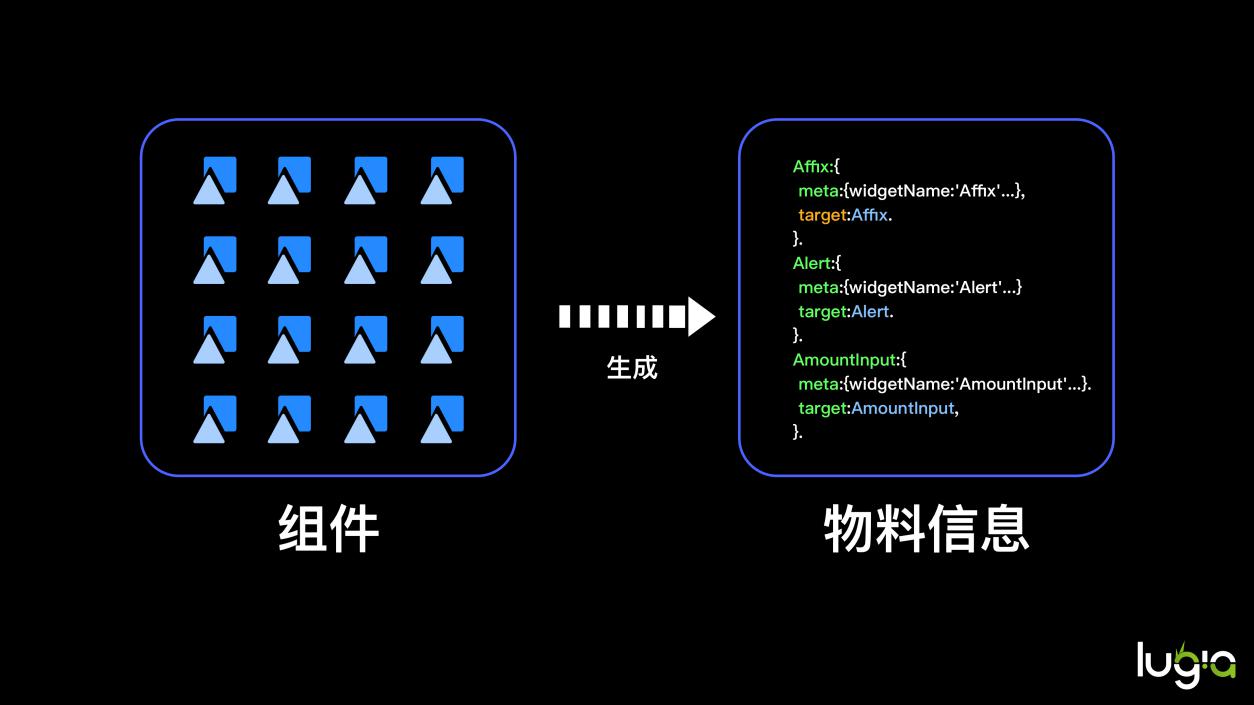


图6.8 @lugia/devtools-material功能说明

该工具是获取多个组件源信息，并整合成索引信息，以对象的形式导出来供mega使用。通过索引信息可以实现对组件的属性进行相应配置，从而展现组件更多状态或实现组件更多功能。

#### 结构划分

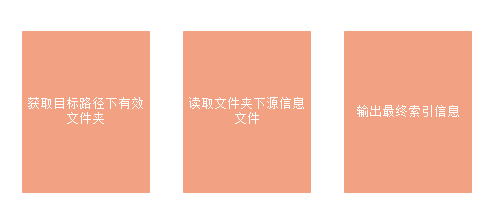


图6.9 @lugia/devtools-material结构划分

#### 获取有效文件夹

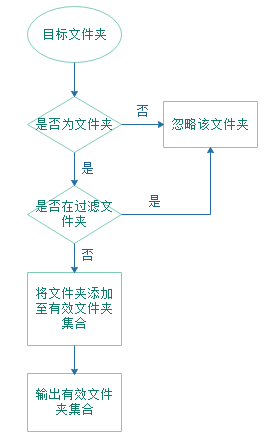


图6.10 @lugia/devtools-material获取有效文件夹

通过node对目标文件目录进行扫描，判断是否为有效文件夹，如果不是文件夹则忽略不管，并且不在过滤数组中的文件夹名将会被添加到有效文件夹的集合中，扫描完成后，输出该集合。

#### 读取文件夹源信息

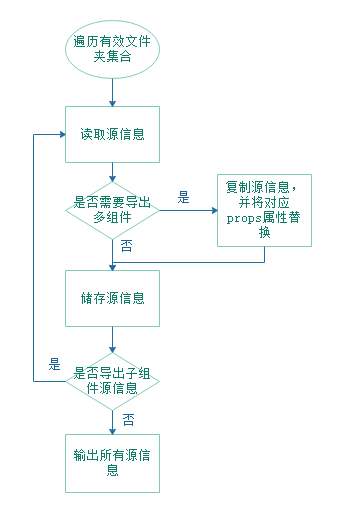


图6.11 @lugia/devtools-material读取文件夹源信息

遍历有效的文件夹名称集合，来读取对应的组件源信息，读取成功后储存该源信息。同时判断是否需要导出多种该组件，需要的话复制该源信息，并将源信息对应的props属性替换为需要导出组件的props，并存储修改后的源信息。判断是否有子组件源信息需要导出，如果需要则重复上述步骤。最后，将所有储存的源信息进行整理。

#### 输出索引信息

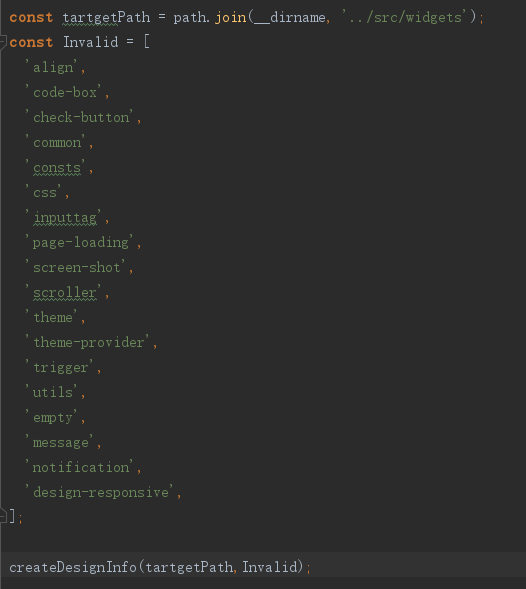
将整理后的源信息输出到指定目录。

#### 在命令行中使用

1557410947(1)

命令行执行，在需要被扫描的目录下运行lugia-create命令，就可以对该目录文件夹进行扫描，生成对应的源信息文件designInfo.js到该文件夹下。命令行可以通过 --filter=a b，--limit=10240,--outExtend=lugia的方式追加参数，filter指定不需要扫描的文件夹，limit指定组件缩略图最大占用空间,outExtend指定组件导出扩展路径。

#### 使用API



函数执行，引入方法，直接调用方法执行。必需参数为目标路径和不需要扫描的文件夹名字集合，目标路径需要指定一个有效路径信息来进行扫描。选填参数以对象的形式写在参数最后，包括 limit、outExtend和outFile。outFile===string可以指定函数输出结果为字符串，不填则输出到文件夹designInfo.js下。

### @lugia/devtools-conversion

#### 功能说明

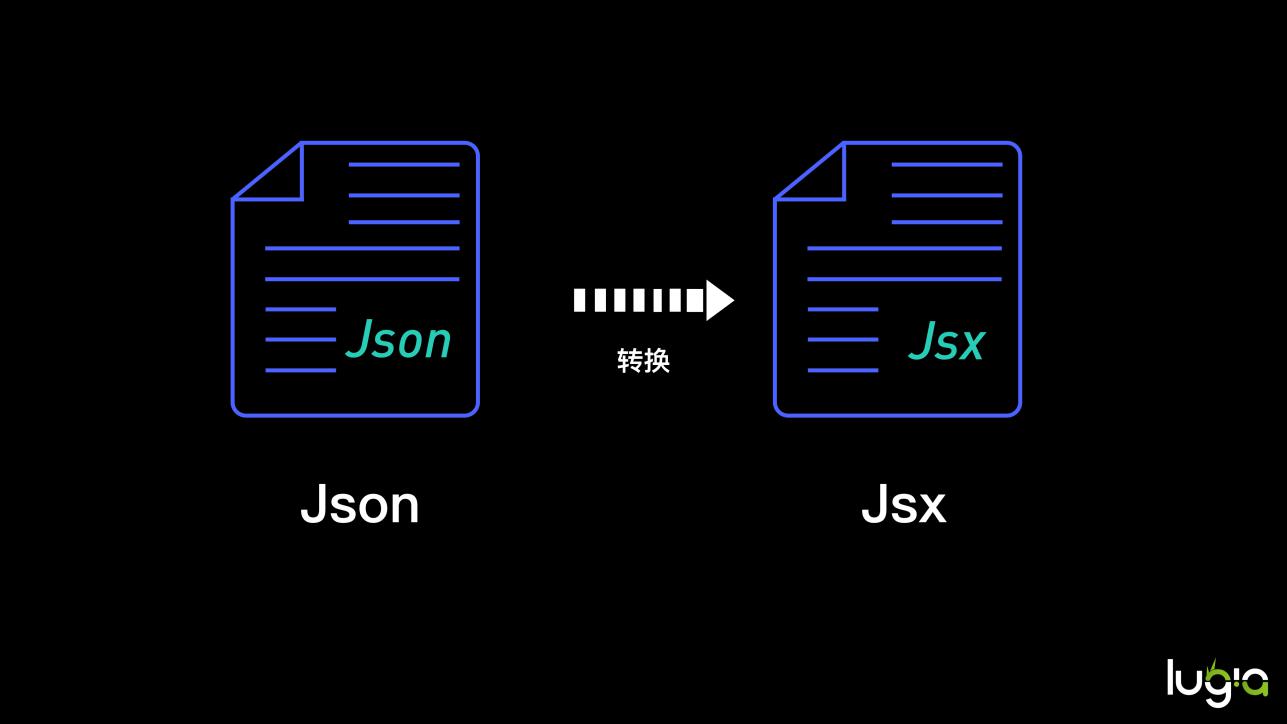


图6.12 @lugia/devtools-conversion功能说明

lugiaD文件是对mega设计界面的描述信息，但是不能直接使用，需要借助一个工具来对它进行转换。所以我们设计了lugia 物料体系的转换文件工具，主要用于将 lugiad 的代码转换成可执行的 jsx语法。

#### 结构划分



图6.13 @lugia/devtools-conversion结构划分

#### 解压LugiaD文件

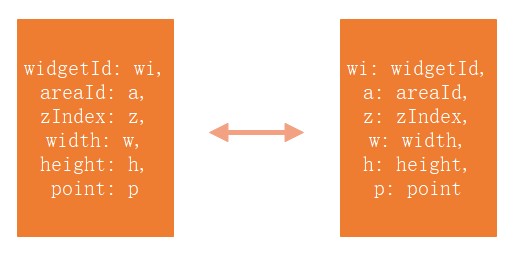


图6.14 @lugia/devtools-conversion解压LugiaD文件

lugiaD文件在生成的时候，为了减小占用空间，我们对lugiaD的部分字段进行了压缩处理。对每一个字段从压缩表进行扫描，压缩成更小的字节，被压缩过的字节，也可以通过解压缩表进行恢复。所以在转换之前需要对lugiaD文件进行解压缩处理。

#### 生成组件信息



图6.15 @lugia/devtools-conversion生成组件信息

1. 生成依赖信息

根据lugiaD信息中mainDependencies字段生成组件依赖信息。

根据lugiax中model字段生成模型依赖信息。

1. 生成叶节点组件

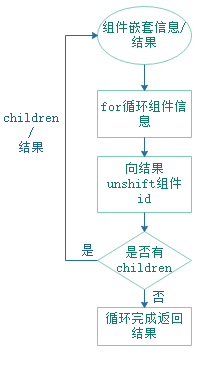
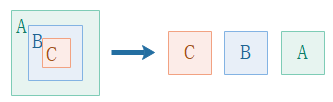


图6.16 @lugia/devtools-conversion生成叶节点组件

根据 mainPad 下 children 字段来递归出所有子组件，递归结果将所有的组件id添加到一个数组中，所有组件的children组件被添加到这个数组的最前面，从而保证内层组件在被外层组件使用之前进行声明。

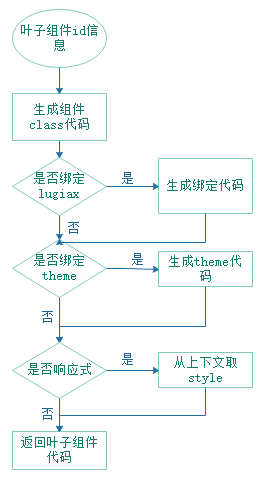


图6.17 @lugia/devtools-conversion生成叶节点组件

根据递归的结果来遍历数组，生成对应的组件class。所有组件的class名信息储存在一个对象中，id作为key，class作为value。在生成过程中，需要检查组件id是否在lugiax中，如果在的话，生成lugiax绑定信息，同时更新组件id所对应的class储存信息。还需要检查组件id是否在theme属性中，如果在，生成theme代码。在生成组件前，根据layoutInfos下面的mode2Config字段检查是否使用响应式组件，如果需要响应式，则每个组件width、height以及定位的top、left需要从上下文获取，否则直接使用id2WidgetInfo下对应的值。

1. 生成根节点组件

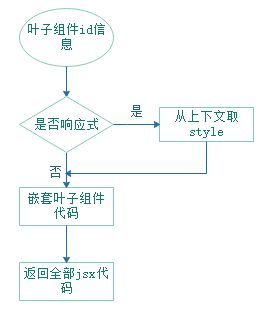
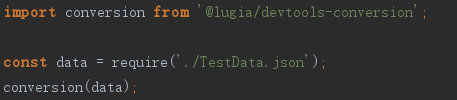


图6.18 @lugia/devtools-conversion生成根节点组件

根据根组件信息生成最外层组件，如果组件id在叶子组件class名对象中，则取该值，否则使用组件widgetName作为组件名生成组件，同时还要检查组件id是否在theme中，以及是否响应式，原理同上。

1. 返回生成代码

#### 使用



conversion接收一个对象参数，返回为一串字符串文本。

### @lugia/mega-scripts

#### 功能说明

react 应用开发工具，可配置版的 [create-react-app](https://github.com/facebook/create-react-app)，提供 dev, build, test, mock 等功能

开发现代单页 React 应用需要的所有环境：

* 开箱即用，包含 start、build 和 test 命令。
* React，JSX，ES6，TypeScript 和 Flow 语法支持。
* 对 Less/Scss、css-modules、postcss、styled-components 的支持。
* CSS Autoprefixer 前缀自动补全。
* 交互式的单元测试，内置支持覆盖率报告功能；基于jest，包括 UI 测试（基于 enzyme）。
* mock 服务，支持引入json、excel、csv 格式的文件作为数据来源。
* browser-sync 支持，保持多个浏览器和设备同步。
* 实时调试服务，包含错误警告。
* 打包构建脚本，把 js、css、图片及其他资源构建在一起，添加hash值和源码映射。
* 灵活的配置支持。

#### 模块依赖



图6.19 @lugia/mega-scripts 模块依赖

#### 在命令行中使用

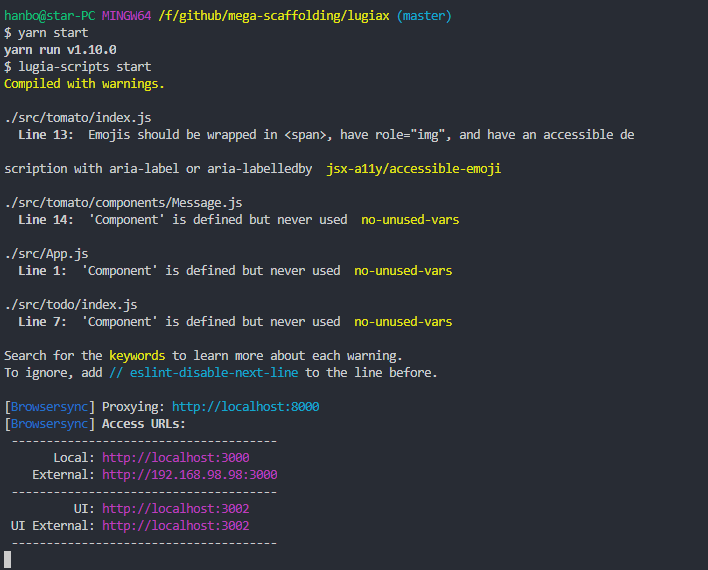
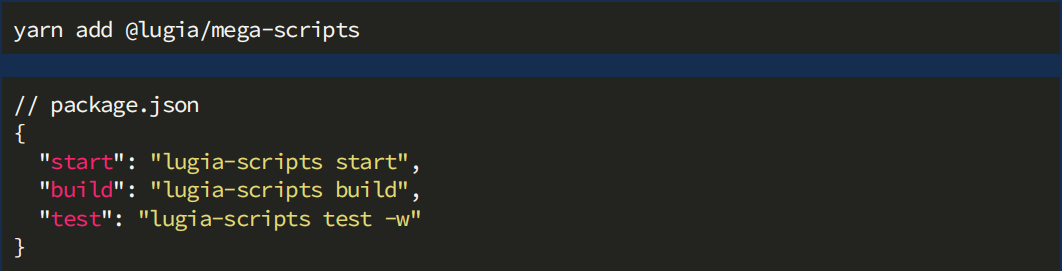


图6.20 @lugia/mega-scripts 在命令行中使用

#### mock 服务

* 可以结合 express 使用
* 可以和 webpack-dev-server 结合使用
* 在配置文件中灵活使用mock.config.js

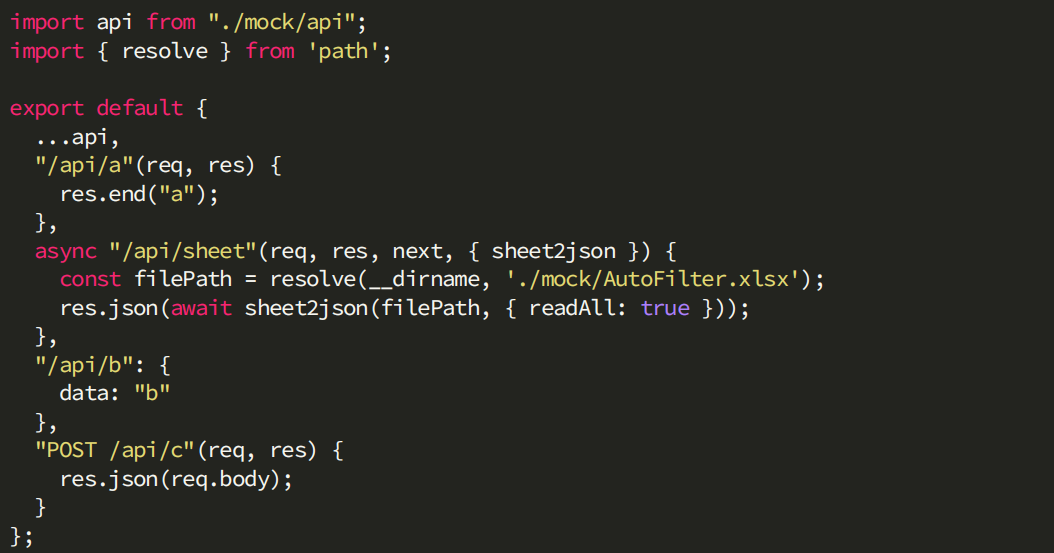


图6.21 @lugia/mega-scripts 配置mock

#### 性能优化

* 通过一系列的内部优化，整体构建速度提升了60% ~ 98%，默认开启缓存后速度会再次提升300% ~ 500%。bebal 编译速度达到8.58 runs/sec ±8%。
* 按需加载 Model 和 Router，加快访问速度；同时打包后每个文件体积可以有效控制在 200KB 以内。

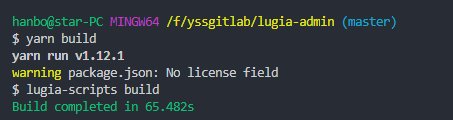
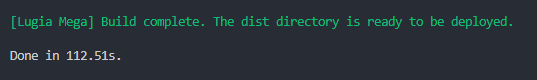
  


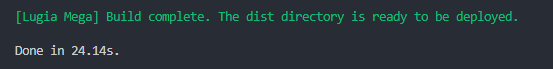
图6.22 @lugia/mega-scripts 构建速度  


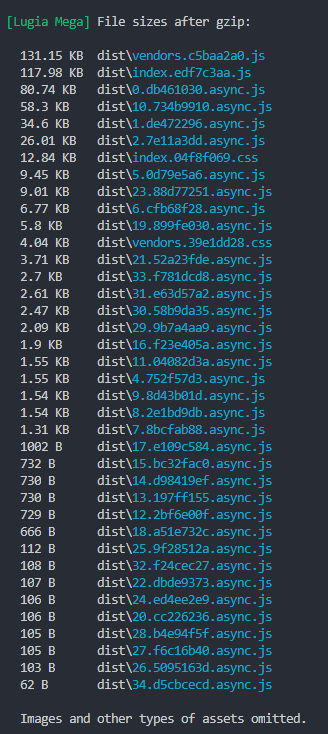
图6.23 @lugia/mega-scripts 开启缓存后构建速度  


图6.24 @lugia/mega-scripts 代码切分

#### **相关模块**

* [chokidar](https://github.com/paulmillr/chokidar)
* [commander](https://github.com/tj/commander.js) 让命令行界面开发变得简单
* [inquirer](https://github.com/SBoudrias/Inquirer.js) 更好的命令行交互
* [body-parser](https://github.com/expressjs/body-parser)
* [http-proxy](https://github.com/nodejitsu/node-http-proxy)

### @lugia/mega

快速创建 React App，基于 redux、lugiax 和 react-router。

* 零配置，包含 init 和 generate (别名g) 命令。
* 在 macOS、Windows、Linux 上使用。
* 轻量级，易学易用。
* 支持 mobile 和 react-native。
* 支持 HMR。
* 动态加载 Model 和 Router：按需加载加快访问速度。

#### 模块依赖



图6.25 @lugia/mega 模块依赖

#### 在命令行中使用

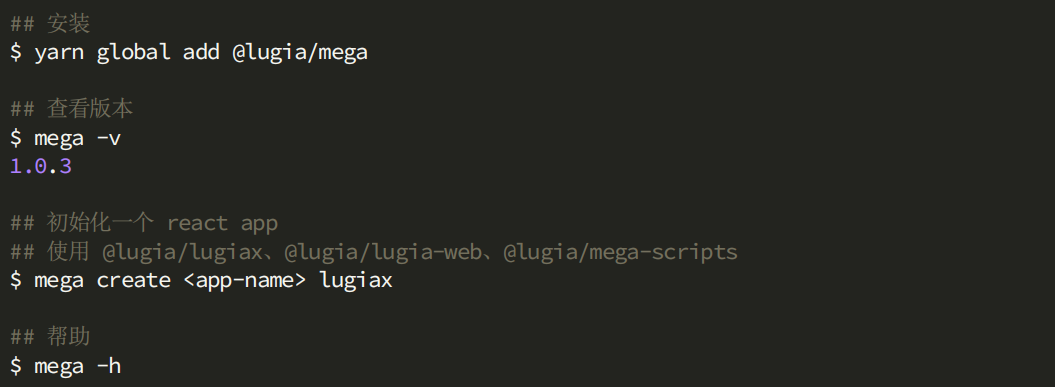


图6.26 @lugia/mega 在命令行使用

#### 脚手架来源

@lugia/mega在创建一个项目时，或从不同的地方拉取脚手架模板。

目前提供以下几种方式：

* local 来自本地。
* npm 来自npm，需要先发布到npm上。
* github 来自github，需要先上传到github上。
* gitlab 来自gitlab，需要先上传到gitlab上。

#### 相关模块

* [user-home](https://github.com/sindresorhus/user-home) 跨平台获取用户根目录
* [Nunjucks](https://github.com/mozilla/nunjucks) 功能强大的模板引擎
* [download-git-repo](https://github.com/flipxfx/download-git-repo)

### @lugia/lugiax

一个基于 Redux 的前端状态管理工具。提供简单高效的全局状态管理方案、 基于 async/await 的异步操作、快捷的双向绑定。LugiaX 内置路由库，对 react-router 做了轻量封装，使用起来更加简单明了。

#### 设计思想

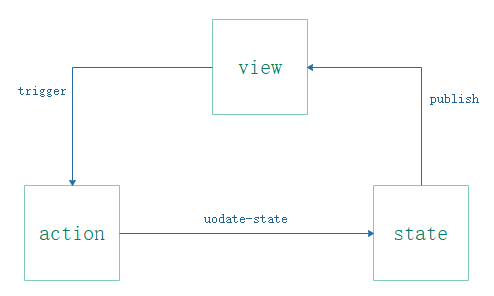


图6.27 @lugia/lugiax 设计思想

基于redux + redux-saga 封装出更加简单、更易上手的状态管理工具。只需掌握action和state的概念即可。我们引入了 mutation 的概念（mutation + state）， 简化了 redux。lugiax 的 state 是不可变类型的数据，可参看 Immutable ；Immutable数据一旦创建，就不能更改。而 mutation 就是修改 state 的唯一途径。state被修改后，并不会通知全局来进行更新，而是通知所绑定的对应的Component来进行更新。

#### Action

本着将action的方法设计成最小业务处理单元，并且是一个纯函数。本框架中的action与redux、redux-saga中的action区别如下：

|  |  |  |
| --- | --- | --- |
| 区别 | redux & redux-saga | flybird |
| action影响的范围 | 任意一个action是可以通过触发所有的reducer，来修改所有的store的值。 | action被固定在model的范围内，在它内部只能修改自己model范围内的状态。 |
| action的dispatch分发 | 需要借助dispatch进行分发，reudx-saga中是通过put进行分发。 | action注册完成后自动封装上dispatch |
| action的结构 | 一个返回类型+参数的对象。如：{type: 'login', user: 'ligx'}. | 一个带业务行为的纯函数，这段处理方法内部可以去执行其它action，包括外部的action方法。 |
| action的监听获取 | redux-saga通过take(‘actionType’) | take(action)，需要引入对应模块的action函数的引用。 |
| action的触发引用 | 可以通过字符串常量的方式来制定。 | 在本框架中，action是由注册器导出的，只有注册器中有这个方法，action才可使用。从开发时就确定了业务之间的逻辑关系。 |

* Action定义：async actionName(param: any, {state,actions})

Action本身是一个异步函数。

param的参数：允许只有一个，调用时传入多函数将会打印警告信息。如果参数为空，param将拿到一个空对象:{}。

* Action的使用

sv.register注册完成后会返回actions对象，里面将按action名称存入一个个封装过的aciton，Action的定义如下：

[actionName](param)

actionName.\_\_ID\_\_ : 对应的ActionID，当需要进行take、takeEvery操作的时候会用到。

React组件中直接调用即可，如：actions.[actionName](入参)

PS：在Component中，调用两个Action，而这两个Action中如果包含了异步操作的话。那是无法保证其先后顺序。

import {actions} from '../model/Login';

export default ()=>{

const onClick = ()=>{

actions.login(); //包含AJAX的后台异步验证

actions.sayHello(); //不能保证aciton.login执行完毕执行该action。

};

return <button onClick={onClick} />

}

#### API设计

* {actions} = sv.register(param)

param：

{

model: '', // 模型名称必填属性，必须唯一否则将会报错。

state: {}, //组件的初始状态 类型为非 null & 非 undefined即可

verify(){ // 数据变更检验，具体说明见下文

},

actions:{ // 本模型对外提供的一系列业务操作

async login(){ // 一个业务操作 具体说明见下文

}

}

}  
returned:

{

actions: { login} //供React组件或其它Model的Action进行调用的触发action的方法。 具体说明见下文

}

* Connect高阶组件

@connect(

(state)=>{

return { name: state.name };

}

)

* Bind高阶组件

lugiax.bind(userModel,

model=>{

const result = { value: model.get('name'), pwd: model.get('pwd'), };

return result;   
}, {

onChange: (mutations, e) => { return mutations.changeName({ name: e.target.value, });   
},

onClick: (mutations, e) => { return mutations.changePwd({ pwd: newPwd, });  
 }, } )(InputTask //Component 组件)

* BindTo高阶组件

lugiax.bindTo(

tomato, // 模块名称（必填）

{ taskName: 'value', // string 绑定属性值（必填） },

{ onChange: { taskName(v) { return v; }, }, } )  
(InputTask //Component 组件)

lugiax.getState()  
lugiax.getState(modelName)

#### lugiax-router 路由

lugiax 对 react-router 做了轻量的封装，createApp 和 createRoute 供你创建路由使用。

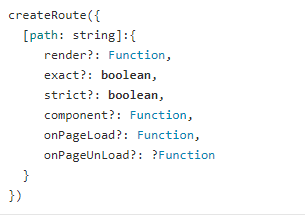
* createRoute  
  

图6.28 @lugia/lugiax createRoute

除了 component 的静态打包外，还提供了 render 动态打包，用于代码分割。

* 页面生命周期函数

onPageLoad 页面加载完成后执行。

onPageUnLoad 页面卸载时执行。

* createApp

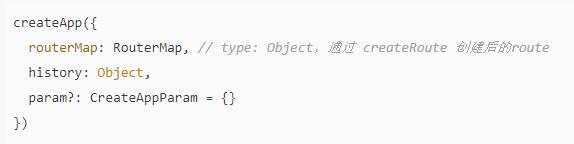


图6.29 @lugia/lugiax createApp

param 提供 loading 和 onBeforeGo 两个api；

loading: 路由切换的加载页面，可配置 Component 组件；

onBeforeGo 跳转之前的回调，可做权限处理。

### @lugia/mega-desktop

#### 模块依赖



图6.30 @lugia/mega-desktop 模块依赖

# 物料体系

一套通用的描述物料的元数据的标准格式，规范约定了物料的类型、名称、版本、数据源、存储位置等信息。

### 元信息生成存储

从每个区块、布局、脚手架的目录下提取相应文件，自动生成文件

使用物料库开发者工具 lugia-devtool 生成

* 区块

package.json、src/ ==> .lugia/mega.js

* 布局

package.json、src/ ==> .lugia/mega.js

* 脚手架

package.json、prompts.js ==> .lugia/mega.js

* 物料库

这是一个 [lerna](https://github.com/lerna/lerna) 项目

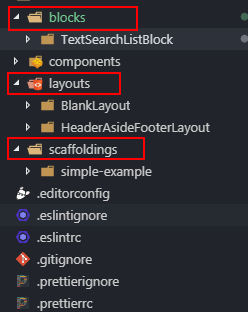


图7.1 物料库项目结构

// lerna.json..."packages": [

"blocks/\*",

"layouts/\*",

"scaffoldings/\*"],...

+ package.json ==> .lugia/mega.js

### 物料库元信息规范

{

// 元信息类型

"metaType": "materials",

// 名称

"name": "react-materials",

// 类型(react、vue、mobile、RN...）

"type": "react",

// 是否官方物料库

"official": true,

// 区块元数据

"blocks": [BlockDesc],

// 布局元数据

"layouts": [LayoutDesc],

// 脚手架元数据

"scaffoldings": [scaffoldingDesc],

// 公共下载配置

"download": {

"type": "npm",

"registry": "http://192.168.102.79:5001/", // (可)

}}

### 区块元信息规范说明

{

// 元信息类型

"metaType": "block",

// (必)标识名 英文

"name": "text-search-list-block",

// (必)标题

"title": "文本搜索列表",

// （必）文件名

"dirName": "TextSearchListBlock",

// (可)区块详细说明

"description": "",

// (必) download 字段描述区块下载方式

"download": {

"type": "npm", // 或者 github、gitlab、local

// npm

"packageName": "text-search-list-block",

"registry": "http://192.168.102.79:5001/", // (可)

"version": "1.0.0",

// npm end

// github

"repository": "owner/name",

"clone": false,

"branch": "master",

// github end

// gitlab

"repository": "owner/name",

"clone": false,

"branch": "master",

"customOrigin": "http://192.168.102.73:8081/",

// gitlab end

// local

"path": "/f/materials/TextSearchListBlock",

// local end

"sourceCodeDirectory": "src"

},

// (必) 分类

"categories": ["信息展示"],

// (必) 截图

"screenshot": "https://xxx.png",

// (必) 发布时间

"publishTime": "1535979136796",

// (必) 最后修改时间

"updateTime": "1535979136796",

// (可) 额外依赖项

"dependencies": {

"uri": "1.0.1"

},

// (必) 外部使用的依据

"useConfig": {

"propTypes": {

title: PropTypes.string,

dataSource: PropTypes.array,

onChange: PropTypes.func,

Icon: PropTypes.element,

},

"defaultProps": {

title: '',

dataSource: [{}],

onChange: noop = () => {},

Icon: (

<Icon

type="warning"

style={{

color: '#FFA003',

}}

/>

),

},

},

// (必) 最佳宽度比例

"proportion": 0.5,

// (可) 保留字段, 依赖组件

"components": {

"@lugia/web/table": {}

},

// (可) 保留字段

"extra": {

// 分词, 用于搜索

"participle": {

/\* ... \*/

}

}}

### 布局规范说明

布局规范与区块类似

增加一个 thumbnail 用来指定抽象缩略图，用在新建页面流程中，用来选择布局，没有则使用默认的。

删除 proportion

{

// (必) 抽象缩略图

"thumbnail": "https://xxx.png"

// 保留字段

"mapFilesToUseConfig": {}}

### 脚手架规范说明

{

// (必)标识名 英文

"name": "@lugia/pro",

// (必)标题

"title": "门户脚手架",

// (可)详细说明

"description": "",

// (可)脚手架预览地址

"homepage": "https://xxx.html",

// (必) source 字段描述下载方式，同上

"source": {},

// (可) 分类

"categories": [],

// (必) 截图

"screenshot": "https://xxx.png",

// (必) 发布时间

"publishTime": "1535979136796",

// （可） meta 文件地址

"metaFile": "./meta.js",

// (必) 最后修改时间

"updateTime": "1535979136796",

// (可) 保留字段

"extra": {

// 分词, 用于搜索

"participle": {

/\* ... \*/

}

}}

### 关于发布/下载

#### 发布

**单独发布**

为了使在 desktop 中创建的 区块、布局 可以公用，在 desktop 中可以单独发布 区块、布局。

默认发布在项目根目录下:

* .lugia/local-materials/blocks
* .lugia/local-materials/layouts

单独下载的 区块、布局会默认存储到项目根目录下，以便下次使用。

**整体发布**

使用物料库开发者工具 lugia-devtool 自动发布。

提供 强制本地发布 选项以便测试。

#### 下载

目前提供以下几种方式：

* local 来自本地
* npm 来自npm，需要先发布到npm上
* github 来自github，需要先上传到github上
* gitlab 来自gitlab，需要先上传到gitlab上

### 缓存机制

除本地方式外，其实三种方式下载的文件都会存储在~user-home/.lugia/mega/cache/ 下。

命名方式：物料库名字 / blocks/layouts/scaffoldings/ / 区块名字 / 版本

例子：C:\Users\hanbo\.lugia\tmp\official-materials\blocks\TextSearchListBlock\1.0.0

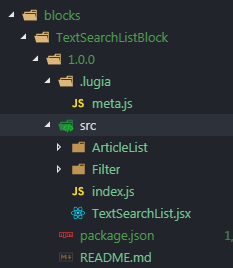


图7.2 物料缓存格式

#### 缓存使用

* 在无法连接网络时使用
* 在下载失败时，会查找是否有缓存文件可以使用

# 附录

### Flow类型

// @flow

import type {

Dispatch as ReduxDispatch,

Store as ReduxStore,

ActionCreator as ReduxActionCreator

} from 'redux';

import \* as React from 'react';

import { CUSTOM\_COMPONENTS, LUGIA\_WEB } from './constants';

export type InitialStateType = {

+historyProjects: HistoryProjectsType

};

export type Action = {

+type: string

};

export type AsyncAction = (dispatch?: Dispatch, getState?: GetState) => any;

export type ActionCreator = ReduxActionCreator<Action | AsyncAction>;

export type ActionCreators = { [string]: ActionCreator };

export type bindActionCreators = (

actionCreators: ActionCreators,

dispatch: Dispatch

) => ActionCreators;

export type GetState = () => AllStateType;

export type Dispatch = ReduxDispatch<Action>;

export type Store = ReduxStore<GetState, Action>;

export type AllStateType = {

+[string]: any

};

export type HistoryProjectType = {

id: string,

name: string,

title?: string,

path: string,

workspacePath: string,

favorite?: boolean,

createDate: number,

openDate: number

};

export type HistoryProjectsType = Array<HistoryProjectType>;

export type CreateType = {

choosedScaffolding: null | ScaffoldingDataType,

choosedLayout: null | LayoutDataType,

choosing: boolean,

lastProjectDir: null | string,

createSuccess: boolean,

initMaterials: boolean,

logs: Array<React.Node | HistoryProjectType>

};

export type CreateProjectType = {

choosedScaffolding: null | ScaffoldingDataType,

choosedLayout: null | LayoutDataType,

projectName: string,

projectDir: string,

projectPath: string,

projectVersion: string,

projectDescription: string,

useGit: boolean,

initCommitInfo: string

};

export type ProjectPageType = {

id: string,

name: string,

title: string,

icon: string,

meta: Object,

width: number,

height: number,

backgroundColor: string

};

export type ProjectPagesType = Array<ProjectPageType>;

export type ProjectManageInfoType = {

isDev: boolean,

waiting: boolean, // 在做dev、build等操作时会有等待状态

pages: ProjectPagesType,

mid?: string,

pid?: number,

killed?: boolean,

pkgCache?: Object

};

export type ProjectManageHistorysType = {

[key: string]: ProjectManageInfoType

};

export type ProjectManageType = {

// HistoryProjectType.id: null | string,

projectID: null | string,

switching: boolean,

buildSuccess: boolean,

building: boolean,

currentPage: null | string,

generatePageSuccess: boolean,

historys: ProjectManageHistorysType,

models: Array<{

asset: {

content: string,

folder: string,

name: string,

path: string

},

entry: string,

name: string,

path: string,

target?: Object,

targetName: string,

error?: string

}>

};

export type DownloadType =

| {

type: 'npm',

name: string,

// 源，默认官方 npm 源

registry?: string,

// 默认 latest

range?: string,

// 模块入口，默认 src

main?: string

}

| {

type: 'github',

// "owner/name"

repository: string,

// 使用 git clone

clone?: boolean,

// 默认 master

checkout?: string,

main?: string

}

| {

type: 'gitlab',

repository: string,

// custom gitlab url

origin?: string,

clone?: boolean,

checkout?: string,

main?: string

}

| {

type: 'local',

path: string,

main?: string

};

export type ComponentPropsTypes = Array<{

name: string,

type: any,

desc: string,

defaultValue: any,

designOnly?: true

}>;

export type ComponentEventsTypes = Array<{

name: string,

desc: string,

args?: Array<{

name: string,

type: any,

desc: string

}>

}>;

export type ComponentModuleNameType = LUGIA\_WEB | CUSTOM\_COMPONENTS;

export type ComponentMetaType = {

metaType: 'component',

id: string,

widgetName: string,

moduleName: ComponentModuleNameType,

title: string,

hidden?: true,

categories: Array<string>,

description: string,

screenshot?: string,

events?: ComponentEventsTypes,

props?: ComponentPropsTypes,

childrenWidget?: Array<string>,

type?: {

[key: string]: any

}

};

export type ComponentTargetType = Function | Object | null;

export type ComponentMetasType = Array<ComponentMetaType>;

export type ComponentTargetsType = {

[key: string]: ComponentTargetType

};

export type ComponentsLibType = {

moduleName: ComponentModuleNameType,

metas: ComponentMetasType,

targets: ComponentTargetsType,

version?: string

};

export type BlockDataType = {

// 元信息类型

metaType: 'block',

// (必)标识名 英文

name: string,

// (必)标题

title: string,

// （必）存储到项目中的文件（夹）名

dirName: string,

// (可)区块详细说明

description: string,

// (必) download 字段描述区块下载方式

download: DownloadType,

// (必) 分类

categories: Array<string>,

// (必) 截图

screenshot: string,

// (必) 发布时间

publishTime: number,

// (必) 最后修改时间

updateTime: number,

// (可) 额外依赖项

dependencies: {

[key: string]: string

},

// (必) 外部使用的依据

useConfig: {

propTypes: {

[key: string]: string

},

defaultProps: Object

},

// (必) 最佳宽度比例，默认 1

proportion: number,

// (可) 保留字段, 依赖组件

components?: {

[key: string]: BlockDataType

}

};

export type LayoutDataType = {

// 元信息类型

metaType: 'layout',

// (必)标识名 英文

name: string,

// (必)标题

title: string,

// （必）文件名

dirName: string,

description: string,

download: DownloadType,

// (必) 分类

categories: Array<string>,

// (必) 截图

screenshot: string,

// (必) 发布时间

publishTime: number,

// (必) 最后修改时间

updateTime: number,

// (可) 额外依赖项

dependencies: {

[key: string]: string

},

// (必) 外部使用的依据

useConfig: {

propTypes: {

[key: string]: string

},

defaultProps: Object

},

// (可) 保留字段, 依赖组件

components?: {

[key: string]: BlockDataType

},

// 保留字段

mapFilesToUseConfig?: Object

};

export type ScaffoldingDataType = {

// 元信息类型

metaType: 'scaffolding',

// (必)标识名 英文

name: string,

// (必)标题

title: string,

description: string,

// (可)脚手架预览地址

homepage: string,

download: DownloadType,

// (必) 分类

categories: Array<string>,

// (必) 截图

screenshot: string,

// (必) 发布时间

publishTime: number,

// (必) 最后修改时间

updateTime: number,

// （可） meta 文件地址

metaFile: string

};

export type BlocksDataType = [] | Array<BlockDataType>;

export type LayoutsDataType = Array<LayoutDataType>;

export type ScaffoldingsDataType = Array<ScaffoldingDataType>;

export type MaterialsDataType = {

// 元信息类型

metaType: 'materials',

// 标识名，唯一 英文

name: string,

// 类型(react、vue、mobile、RN...）

type: 'react' | 'vue' | 'mobile' | 'react-native',

// 是否官方物料库

official: boolean,

// 显示的标题

title: string,

// 描述信息

description: string,

// 关键字

keywords: Array<string>,

// 物料库 logo url

logo: string,

// 区块元数据

blocks: BlocksDataType,

// 布局元数据

layouts: LayoutsDataType,

// 脚手架元数据

scaffoldings: ScaffoldingsDataType,

// 公共下载配置

download: {

type: 'npm' | 'github' | 'gitlab' | 'local'

}

};

export type MaterialsItemType = {

name: string,

type: string,

official: boolean,

title: string,

downloadUrl: string,

description: string,

keywords: Array<string>,

logo: string,

enable: boolean,

cachePath?: string,

cacheData?: MaterialsDataType

};

// 存储在用户目录下

export type MaterialsType = Array<MaterialsItemType>;

export type EditorsType =

| 'VisualStudioCode'

| 'WebStorm'

| 'SublimeText'

| 'Atom';

export type MacOSTerminalsType = 'Terminal' | 'iTerm2';

export type SettingsType = {

projectAuthor?: string,

projectPort?: number,

projectDir?: string,

projectVersion?: string,

autoOpenBroswer?: boolean,

terminals?: Array<{

type: string,

enable: boolean,

program: string,

exist: boolean

}>,

editors?: Array<{

type: string,

enable: boolean,

program: string,

exist: boolean

}>

};

export type ResultResolve<D> = {

error: null | string,

data: null | D

};

export type Result<D> = Promise<ResultResolve<D>>;