插件系统简介

Johnny Wu

主要内容

- Electron 简介
- 创建插件的基本流程
- 如何编写一份简单的插件面板
- Cocos Creator 中提供了哪些内置控件
- 使用 Vue.js 来更快捷的编写界面
- 如何为 cc.Component 自定义属性检查器

Electron 简介

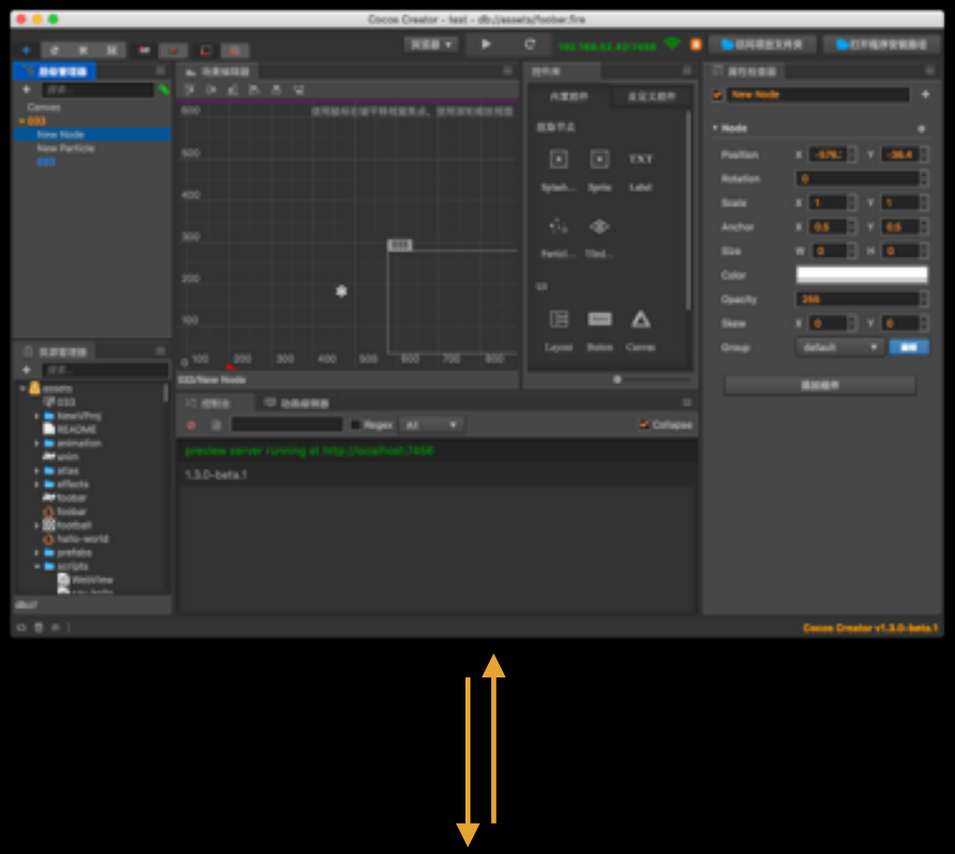
- Electron 是什么?
- 如何理解 Electron 中的 “主进程” 和 “渲染进程”?
- Electron 中如何进行进程间通讯（IPC）?
- Cocos Creator 对 Electron 做了哪些改进?

Electron 是什么？

- 前身: Atom Shell
- 开发团队: Github
- 目标: Chromium + Node.js = 跨平台桌面应用框架
- 初衷: 为 Atom 编辑器量身定制

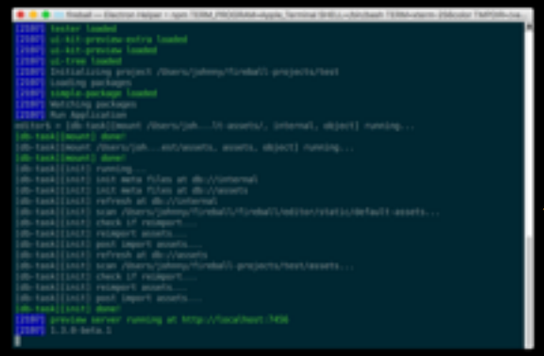
Electron 中的 “主进程” 和 “渲染进程”

渲染进程 1



渲染进程 2

渲染进程 3



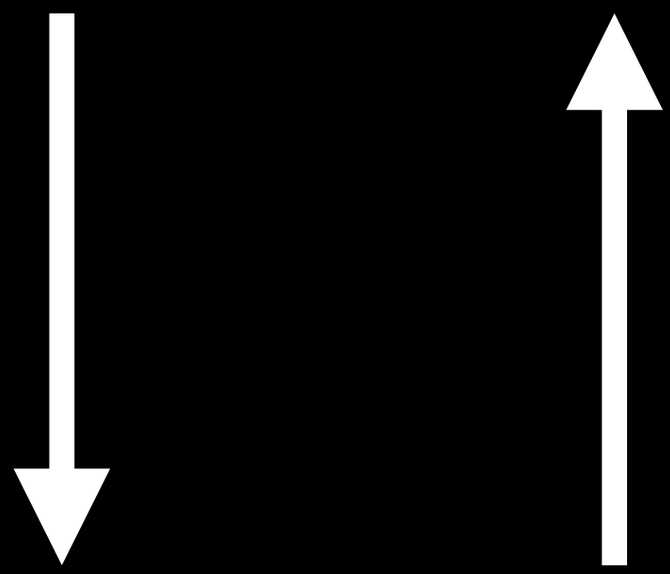
主进程

Electron 中如何进行进程间通讯（IPC）？

```
const {ipcRenderer} = require(‘electron’);
ipcRenderer.send (‘foo-bar’, {
  foo: ‘hello foo’,
  bar: 123,
});
```

渲染进程

```
const {ipcRenderer} = require(‘electron’);
ipcRenderer.on (‘foo-bar’, (event, data) => {
  console.log(data.foo);
  console.log(data.bar);
});
```



```
const {ipcMain} = require(‘electron’);
ipcMain.on (‘foo-bar’, (event, data) => {
  console.log(data.foo);
  console.log(data.bar);
});
```

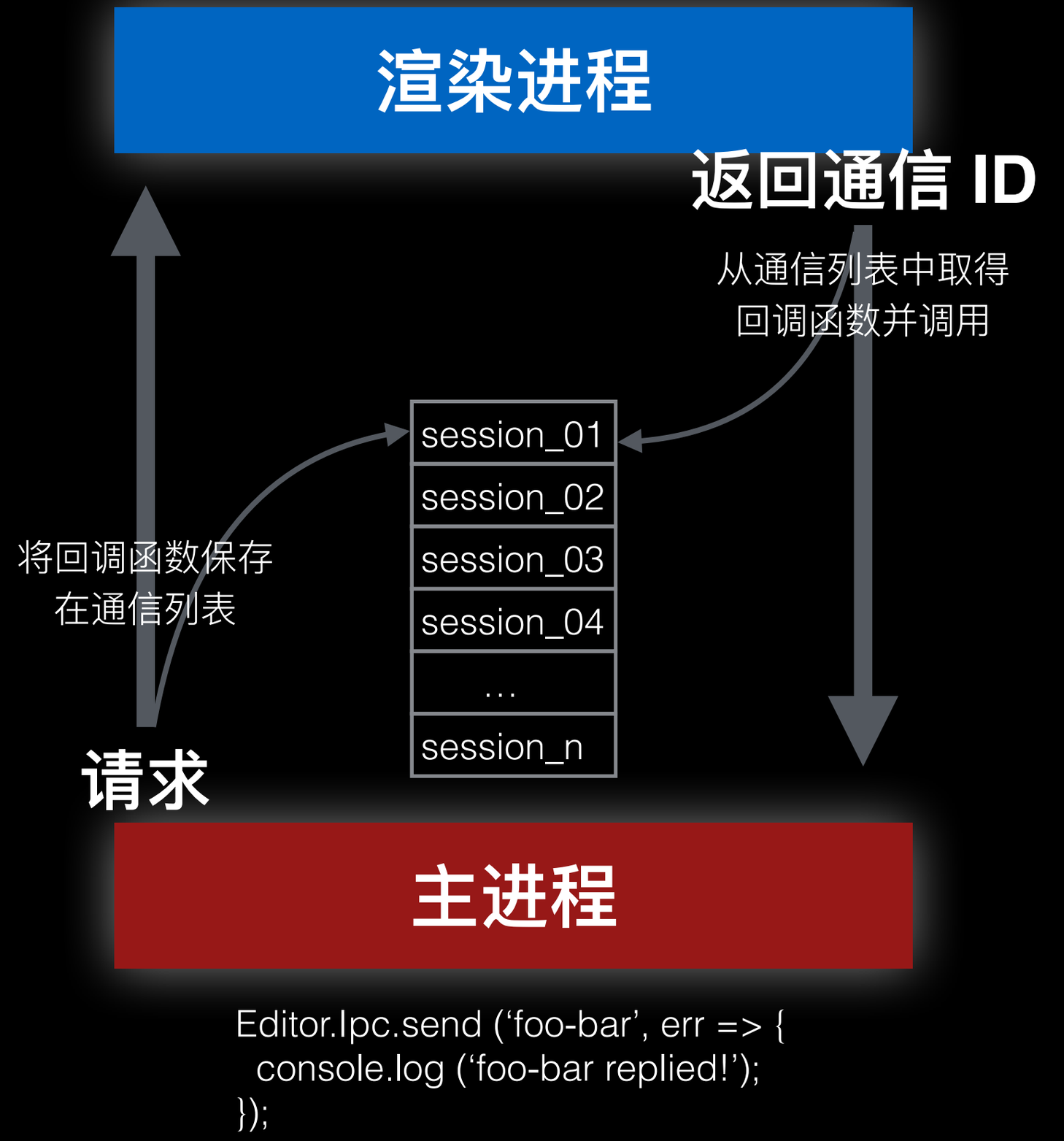
主进程

```
const {BrwoserWindow} = require(‘electron’);
let bw = BrwoserWindow.fromId (0);
bw.webContents.send ( ‘foo-bar’, {
  foo: ‘hello foo’,
  bar: 123,
});
```

Cocos Creator 对 Electron 做了哪些改进?

- 改进 IPC 消息接口
- 加入自定义协议
- 改进 Electron 的 BrowserWindow 接口
- 改进 Electron 的 MenuItem 接口

改进 IPC 消息接口

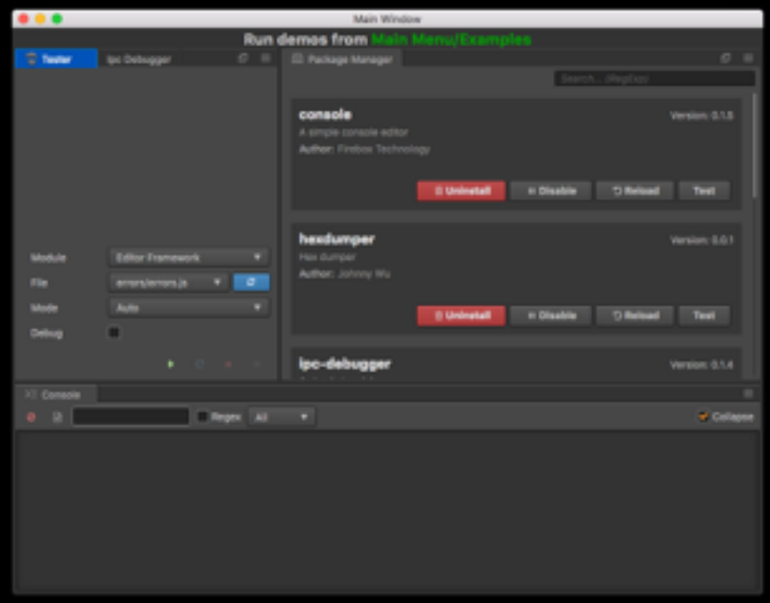



```
Editor.Ipc.send (‘foo-bar’, err => {
  console.log (‘foo-bar replied!’);
});
```

自定义协议

- app://
- packages://{package-name}
- theme://
- 等等...

BrowserWindow vs Editor.Window



BrowserWindow

- 没有窗口列表，无法管理多个窗口
- 查找窗口困难
- 无法保存窗口状态

Editor.Window

- 维护了窗口列表
- 窗口内保存了面板 (panel) 信息
- 查找方便
- IPC 收发更简单

Menu vs Editor.Menu

Menu

- 创建方法: Menu.buildFromTemplate (tmpl)
- 添加/插入新菜单: 复杂
- 修改原有菜单: 复杂
- 删除原有菜单: 及其复杂

Editor.Menu

- 创建方法: new Editor.Menu (tmpl)
- 添加/插入新菜单: menu.add ('foo/bar', { ... })
- 修改原有菜单: menu.set ('foo/bar', { ... })
- 删除原有菜单: menu.remove('foo/bar')

为什么要设计 Editor.Menu ?

1. Electron (哦不, 实际上是 Chromium) 的 Menu 设计槽点太多
2. 我们希望加入 menu path 这个重要的功能点: 'foo/bar/foobar'
3. 我们希望基于 menu path 这个想法设计一套 API

创建插件的基本流程

- 新建你的插件文件夹
- 在插件文件夹中写入 package.json 和其他重要文件
- 将插件放入 Creator 中的指定位置
- 运行 Creator 并打开你的项目
- 继续调整插件...

插件包的基本结构

不可缺少

- **package.json** — 插件包的配置文件
- **main.js** — 主进程的入口函数
- **panel/index.js** — 面板（渲染进程）的入口函数

package.json 中的重要字段

```
{
  "name": “my-package”,
  "main": “main.js”,
  "version": “1.0.0”,
  "main-menu": {
    "Panel/Simple Panel": {
      "message": “my-package:open”
    }
  },
  "panel": {
    "main": "panel/index.js",
    "type": "dockable",
    "title": "Simple Panel",
    "width": 400,
    "height": 300
  }
}
```

- **name**: 插件包的名字（全局唯一）
- **main**: 入口函数文件
- **version**: 版本信息 (采用 semver 管理)
- …

main.js 主进程入口函数

```
'use strict';

module.exports = {
  load () {
  },

  unload () {
  },

  messages: {
    open() {
      Editor.Panel.open('simple-package');
    },
  },
}
```

- **load**: 插件加载时被调用
- **unload**: 插件卸载时被调用
- **messages**: 在主进程中注册 IPC 消息

插件的安装地址

全局 (插件将被应用于所有项目中)

- ~/.CocosCreator/packages (Mac)
- C:\Users\{你的用户名}\.CocosCreator\packages (Windows)

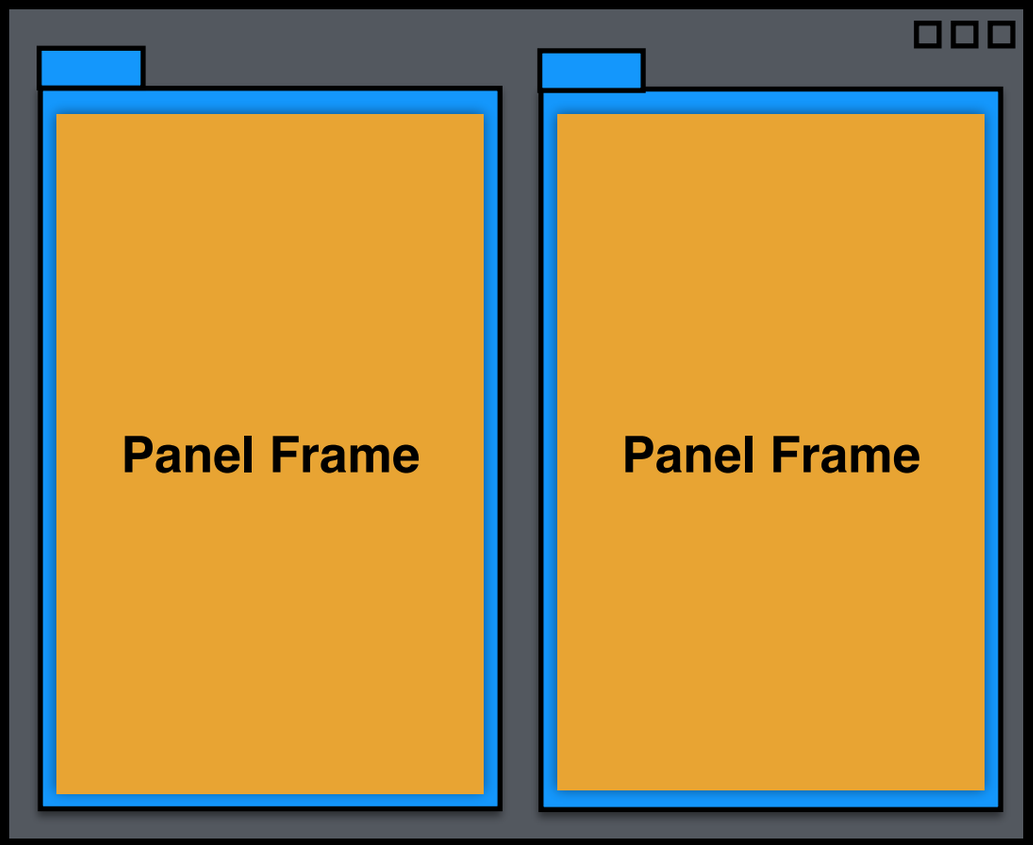
项目 (仅对应的项目会加载插件)

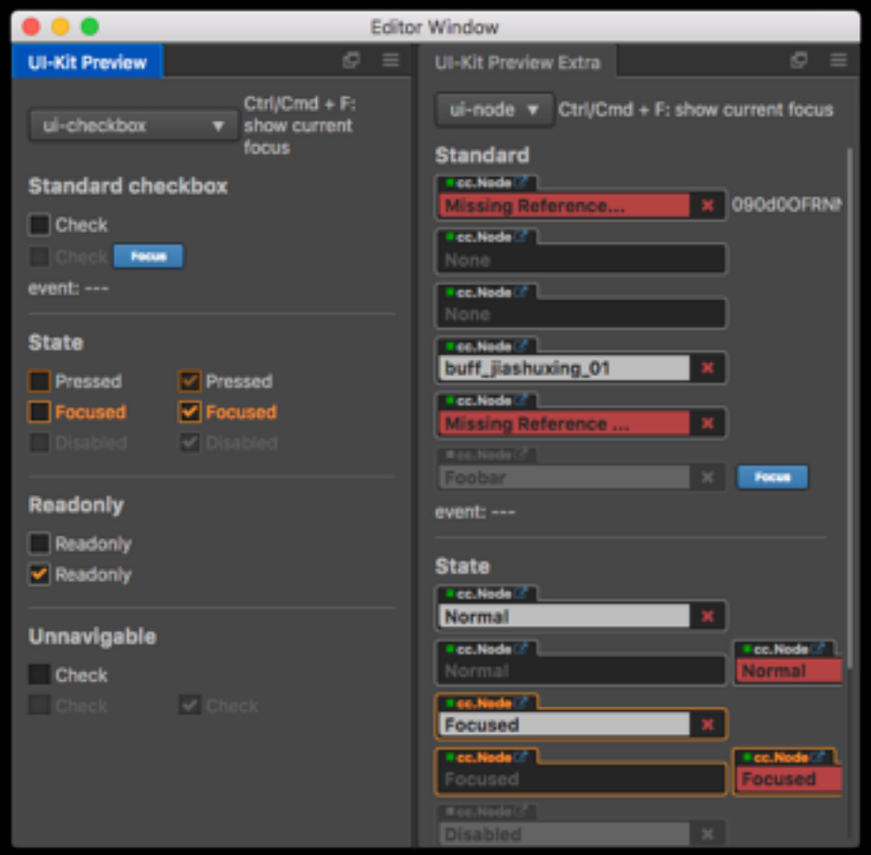
- {你的项目路径}/packages

创建面板

- 什么是面板 (panel)
- 如何在 package.json 中注册面板
- 编写 Panel 的入口文件
- 为面板加入 template — 界面模板
- 为面板加入 style — CSS 样式
- 为面板加入 messages — IPC 消息
- 为面板加入 listeners — DOM 消息
- 面板间通讯

什么是面板 (panel)


Panel Frame
Panel Frame


Editor Window
UI-Kit Preview
UI-Kit Preview Extra
ui-checkbox
Ctrl/Cmd + F: show current focus
Standard checkbox
Check
Check
Focus
event: ---
State
Pressed
Pressed
Focused
Focused
Disabled
Disabled
Readonly
Readonly
Readonly
Unnavigable
Check
Check
Check
ui-node
Ctrl/Cmd + F: show current focus
Standard
cc.Node
Missing Reference...
090d0OFRN
cc.Node
None
cc.Node
None
cc.Node
buff_jiashuxing_01
cc.Node
Missing Reference ...
cc.Node
Foobar
Focus
event: ---
State
cc.Node
Normal
cc.Node
Normal
cc.Node
Normal
cc.Node
Focused
cc.Node
Focused
cc.Node
Focused
cc.Node
Disabled

如何在 package.json 中注册面板

```
{
  "name": “my-package”,
  "main": “main.js”,
  "version": “1.0.0”,
  "main-menu": {
    "Panel/Simple Panel": {
      "message": “my-package:open”
    }
  },
  "panel": {
    "main": "panel/index.js",
    "type": "dockable",
    "title": "Simple Panel"
  },
  “panel-02": {
    "main": “panel/index-02.js",
    "type": "dockable",
    "title": "Simple Panel 02"
  }
}
```

Panel ID = package名 + panel后缀名

Panel ID = my-package

Panel ID = my-package-02

编写 Panel 的入口文件

```
// panel/index.js
Editor.Panel.extend({
  style: `
    :host {
      margin: 5px;
    }

    h1 {
      color: #09f;
    }
  `,

  template: `
    <h1>Hello World</h1>
  `,

  ready () {
    console.log('Panel is ready');
  }
});
```

关键字段:

- style
- template
- messages
- listeners
- $
- behaviors
- dependencies
- ready ()
- run (argv)
- close ()

编写 template 和 style

- CSS 和 Shadow DOM
- 使用内置的排版 class 提高排版效率
- 其他技巧...

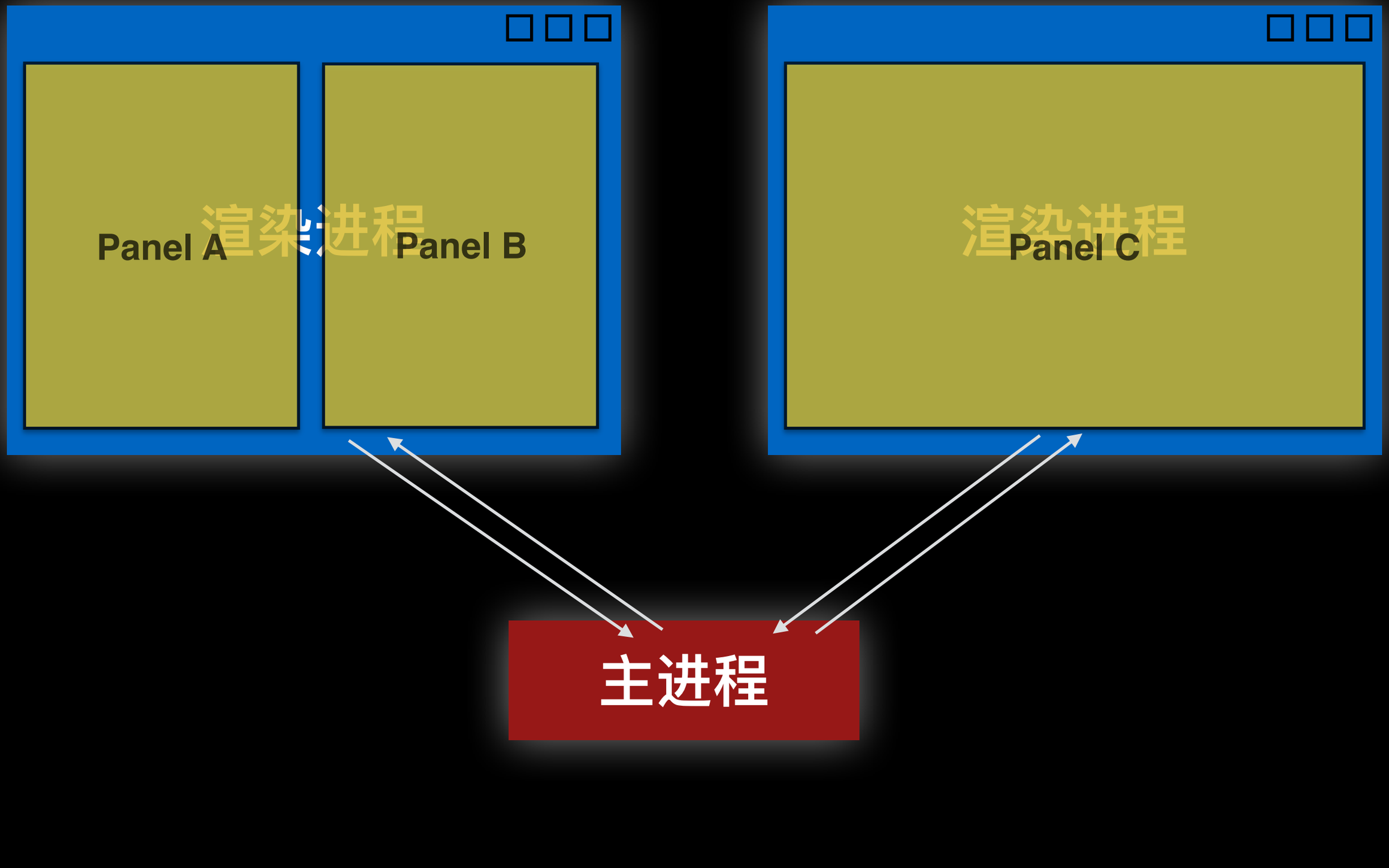
注册 messages 和 listeners

```
// panel/index.js
Editor.Panel.extend({
  ...
  ...
  messages: {
    'foo:bar' () {
      console.log('hello foo:bar!’);
    }
  },

  listeners: {
    'foo-bar' () {
      console.log('on foo-bar’)
    }
  },
});
```

messages => 处理发送到面板/窗口的 IPC 消息

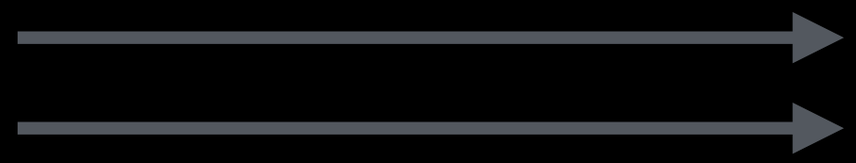
listeners => 处理发送到面板的 DOM 消息

面板间通讯



其他字段简介

```
// panel/index.js
Editor.Panel.extend({
  dependencies: [
    'packages://my-package/foo.js',
    'packages://my-package/bar.js',
  ],

  template: `
    <div id="foo"></div>
    <div id="bar"></div>
  `,

  $: {
    foo: '#foo',
    bar: '#bar'
  }
});
```

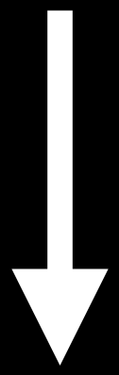


Creator 中提供的内置控件

- 通过 开发者/UI-Kit Preview 预览内置控件
- 控件消息: confirm, cancel 和 change
- 控件属性: value, …
- <ui-prop> 简介

使用 Vue.js 来更快捷的编写界面

+



AWESOME !!!

Vue.js 简介

- 前端框架
- 提供强大的模板数据绑定功能
- 提供可靠的 Web Component 定义方法
- 等等...

在面板中使用 Vue.js

```
// panel/index.js
Editor.Panel.extend ({
  template: `
    <h1>{{ message }}</h1>
    <ui-button v-on:confirm="onApply">Apply</ui-button>
  `,

  ready () {
    new Vue ({
      el: this.root,
      data: {
        message: 'Hello World',
      },
      method: {
        onApply () { console.log('foobar'); }
      },
    });
  }
});
```

使用 Vue.component

```
// panel/index.js
Editor.Panel.extend ({
  dependencies: [
    ‘packages://my-package/panel/foo-bar.js’
  ],

  template: `
    <foo-bar msg=“hello”></foo-bar>
  `,

  ready () {
    new Vue ({
      el: this.root,
    });
  }
});
```

```
// panel/foo-bar.js
Vue.component(‘foo-bar', {
  template: '<span>{{ msg }}</span>’,
  props: ['msg'],
})
```

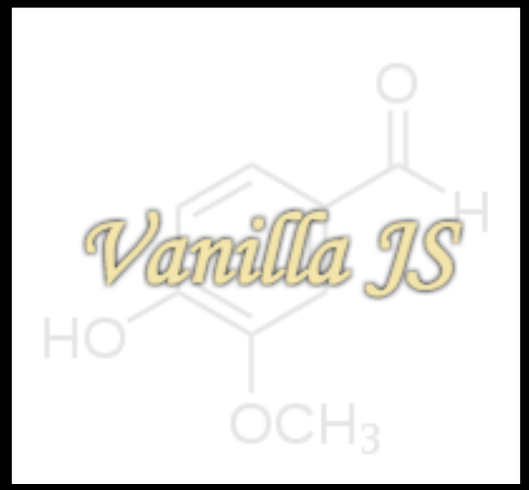
和内置控件一起使用

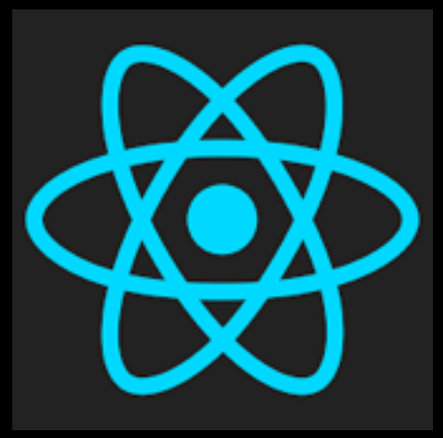
```
// panel/index.js
Editor.Panel.extend ({
  template: `
    <ui-num-input v-value="{{foo}}"></ui-num-input>
    <ui-input v-value="{{bar}}"></ui-input>
    <ui-checkbox v-readonly="{{isReadonly}}"></ui-checkbox>
    <ui-button v-disabled="{{btnDisabled}}"></ui-button>
  `,

  ready () {
    new Vue ({
      el: this.root,
      data: {
        foo: 100,
        bar: 'Hello World',
        btnDisabled: true,
        isReadonly: true,
      }
    });
  }
});
```

其他一些 Vue.js 的特性

- v-if
- v-for
- directive
- style binding
- ...

除了 Vue.js，我们还支持...





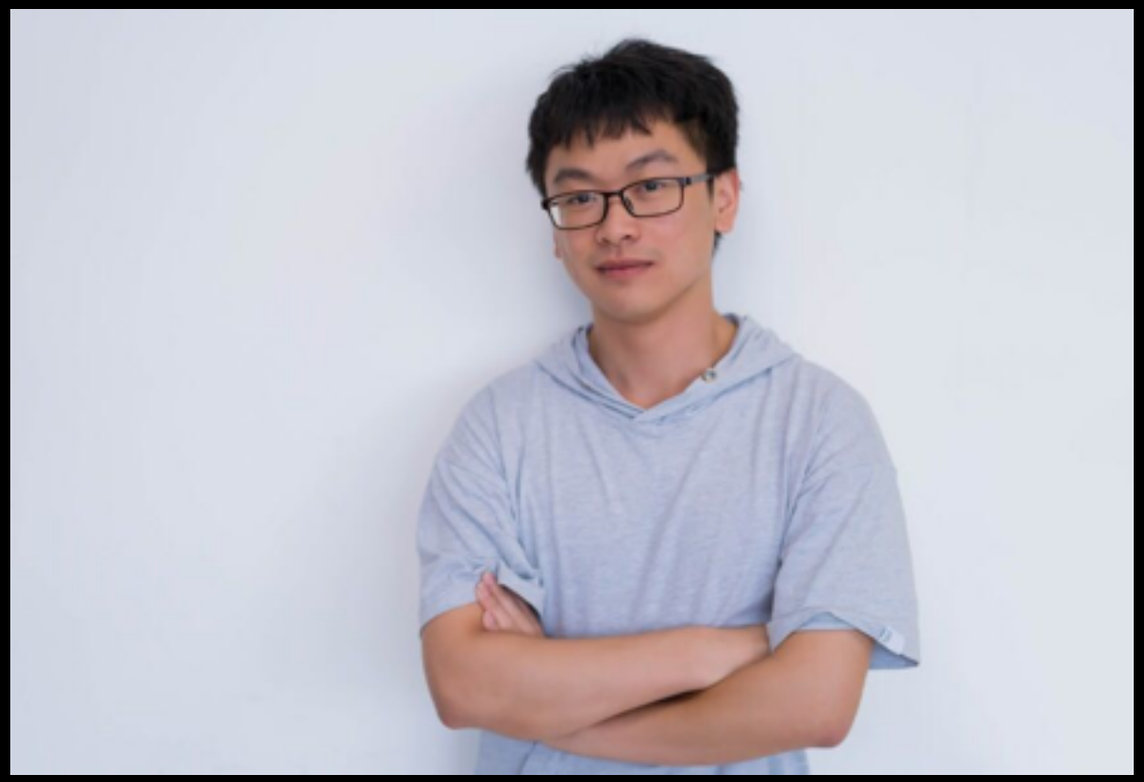
如何为 Component 自定义属性检查器

```
cc.Class({
  name: ‘custom-comp',
  extends: cc.Component,
  editor: {
    inspector: 'packages://foobar/inspector.js',
  },
  properties: {
    foo: 'Foo',
    bar: 'Bar'
  },
});
```

```
Vue.component('custom-comp-inspector', {
  template: `
    <cc-prop :target.sync="target.foo"></cc-prop>
    <cc-prop :target.sync="target.bar"></cc-prop>
  `,

  props: {
    target: {
      twoWay: true,
      type: Object,
    },
  },
});
```

下周直播预告

- **直播主题**：打开脑洞，用 cc.graphics 动态生成奇妙图形
- **主播**：youyou
- **时间**：9.13（周二）晚8点



谢谢