小程序运行时开发文档

1952890 顾文涛 数据科学与大数据技术

一 、开发需求分析

开发任务的目标是完成一个小程序进行时，一个可以在浏览器（chrome）端运行小程序的小程序框架。

按照项目要求，小程序框架需要实现双线程，即利用 webworker 实现大多数小程序所走的逻辑视图层分离。

除了双线程外，小程序运行时还需要做到一个基于 vdom 的渲染框架。按照要求，即利用 babel，首先通过 jsx 来描述页面，然后编译成 render function，执行后产生 vdom。

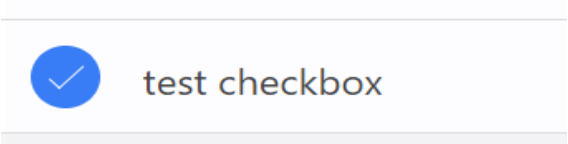
最后，是对 DSL 渲染的要求，主要 包含条件、循环以及引用。

二、实现成果简介

通过四周的时间，完成的成果如下图所示：

随着时间推移，右边泡泡中的数字会变大，代表有更新的消息：

中间有 checkbox 按钮用来测试 jsx 是否能够正常编译渲染：



三、运行步骤

小程序以及小程序运行时主要有两个主要步骤：

A　npm install

目的：安装写于 package-lock 中的各种依赖。这里面直接使用到的依赖主要包含 superstatic、babel、jsx、worker-load 等。

B npm start

目的：启动项目。在 package.json 文件中写了 start 脚本的具体操作，如下图所示：

```
▷ 调试
"scripts": {
  "dev": "cross-env NODE_ENV=development webpack-dev-server --inline --hot --progress",
  "start": "superstatic build -p 8080 --host 127.0.0.1 --gzip -c '{\"rewrites\": [{\"source\":\"**\",
  "prestart": "npm run build",
  "build": "cross-env NODE_ENV=production webpack -p --progress",
  "prebuild": "mkdirp build && ncp src/assets build/assets",
  "test": "eslint {src,test}"
}
```

Npm start 主要做了两件事，一个是 build 操作，另一个是 superstatic 的 server 热启动操作。把安装和启动流程走完后得到的结果如下（node11.15.0）

```
Superstatic started.
Visit http://127.0.0.1:8080 to view your app.
```

在 localhost：8080 里就可以得到小程序运行效果。

四、代码开发分析

4.1 webworker 双线程通信

按照文档的指导，应当把渲染层与逻辑层分离，因此在项目中，包含 dom.js 以及 worker.js。两者之间的通信直接通过 postmessage 完成，如下图所示：

Dom.js

```
worker.postMessage({
    type: 'event',
    event
});
```

Worker.js

```
function send(message) {
    postMessage(JSON.parse(JSON.stringify(message)));
}
```

4.2 jsx 生成 fuction 与 vdom

在 app.js 中，包含了 jsx 如下：

```
render(props, { items, message }) {
    return (
        <div class="app">
            <div class="bar bar-header bar-dark">
                <button class="button icon ion-information-circled" onClick={this.explainWebWorker}
                <h1 class="title">
                    实习作业
                </h1>
                <progress-spinner dark style="margin-left: 10px;" />
            </div>

            <div class="content has-header">
                <div class="list">
                    <a class="item item-avatar item-icon-right" target="_blank">
                        <h2>running in a Web Worker</h2>
```

```
(property) JSX.IntrinsicElements.h1: JSX.HTMLAttributes
<h1 class="title">
```

通过 render 可以把 jsx 转化为 render function。这其中涉及到的核心递归，在 lib/util.js 之中：

```
export function splice(arr, item, add, byValueOnly) {
    let i = arr ? findWhere(arr, item, true, byValueOnly) : -1;
    if (~i) add ? arr.splice(i, 0, add) : arr.splice(i, 1);
    return i;
}
```

除了核心递归以外，还包括了其他函数：

```
appendChild(child) {
    child.remove();
    child.parentNode = this;
    this.childNodes.push(child);
    if (this.children && child.nodeType===1) this.children.push(child);
    mutation(this, 'childList', { addedNodes:[child], previousSibling:this.childNode
}
insertBefore(child, ref) {
    child.remove();
    let i = splice(this.childNodes, ref, child), ref2;
    if (!ref) {
        this.appendChild(child);
    }
    else {
        if (~i && child.nodeType===1) {
            while (i<this.childNodes.length && (ref2 = this.childNodes[i]).nodeType
            if (ref2) splice(this.children, ref, child);
        }
        mutation(this, 'childList', { addedNodes:[child], nextSibling:ref });
    }
}
replaceChild(child, ref) {
    if (ref.parentNode===this) {
        this.insertBefore(child, ref);
        ref.remove();
```

4.3 DSL

由于时间的限制，在 DSL 上并没有实现 if 与 for 的编译，但是做到了数据传递，在 app.js 中包含数据传递的方式如下：

```
render(props, { items, message }) {
```

```
    <div class="item item-divider">{items.length} data_pass_example</div>
    { items.map( item => (
        <Item {...item} />
    )) }
</div>
```

其中 render 是对 jsx 的 render，而 items 在示例 app 中如下：

```
let items = [];
for (let i=0; i<10; i++) {
    items.push({
        id: i,
        name: `${getNames()} ${getNames()}`,
        unread: Math.max(0, Math.random()*20|0-10)
    });
}
return items;
```

五、心得与总结

通过为期四周的字节培训，我了解到了更多关于前端小程序的运行底层逻辑，特别是编译渲染的知识。这些知识都是学校学不到的，因此感激字节提供的培训机会，希望以后能够在前端开发的方向上学到更多，产出更多。