实验报告

—— 日历的开发

陈广庆

2016年10月28日

目录

1 前言

这次实验作业，我选择了做日历

通过Electron，只用一份源代码，就能同时生成Windows、Linux、maxOS三个系统的应用，经过简单的修改，还能生成网页版(goushi.me/yun/lunar-calendar/lunar-calendar.html)

同时实现了以下功能：

- 按周、月的方式显示日历
- 显示农历和节假日
- 记事本功能，可记录每日的活动
- 点击跳转到上一年/月
- 点击跳转到下一年/月
- 通过输入，跳转到指定年/月份
- 返回今天
- 退出日历
- 总在最上

开发备注

```
# 安装依赖包（在此之前，需要先安装Node V6）
npm install
# 打包并测试
npm start
# 生成Windows 64位的应用
npm run build-win-64
# 生成Windows 32位的应用
npm run build-win-32
# 生成Linux 64位的应用
npm run build-linux-64
# 生成Linux 32位的应用
npm run build-linux-32
# 生成Mac 64位的应用
npm run build-darwin-64
# 生成Mac 32位的应用
npm run build-darwin-32
```

2 设计思路

Electron可以看做一个由JavaScript控制的Chromium浏览器，可以把在其之上开发的网站封装成桌面应用。

Electron可以使用绝大多数的npm包，里面有个Menubar可以帮你处理好工具栏应用的系统底层细节。

所以接下来只需要写一个农历网页。

网页主要由React&Flux实现

前端界面元素由'material-ui'提供

后端农历数据由'lunar-calendar-zh'提供

在Electron里，需要把React写的js文件转码，因此还用了browserify把它打包成SPA(Single Page Application)

手动打包再构建的话，不利于开发，因此还用了gulp实现一键自动化打包、构建

3 设计过程

3.1 index.html

首先用个简单的HTML文件写个网页框架，该网页只有一个ID为'app'的div元素，由React渲染

index.html

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="
       fonts.css">
    <title>Lunar-Calendar</title>
</head>
<body style="margin: 0">
<div id="app"></div>
</body>
<!-- reason: https://github.com/electron/electron/
    issues/1611 -->
<script>var electronRequire = require;</script>
<script src="build/app.js"></script>
</html>
```

3.2 main.js

然后是Electron的main process，主要是创建个Menubar的对象，并在其中加载index.html
同时绑定了“退出”，“重启”，“隐藏”，“显示”等事件

main.js

```js
var mb = menubar({
    alwaysOnTop: conf.get('alwaysOnTop') || false,
    preloadWindow: true,
    dir: __dirname,
    icon: __dirname + '/icon.png',
    tooltip: '新世界的大門',
    width: 800,
    height: 436
});
```

```

mb.on('ready', function () {
    console.log('app is ready');

    ipc.on('app-quit', function () {
        mb.app.quit();
    });

    ipc.on('app-restart', function () {
        mb.app.relaunch();
        mb.app.quit();
    });

    mb.on('hide', function () {
        mb.window.webContents.send('app-hide')
    });

    mb.on('show', function () {
        mb.window.webContents.send('app-refresh')
    });
});
```

3.3 React&Flux

3.3.1 View

3.3.1.1 app.js

react部分的出口，主要是用MainPanel组件渲染ID为'app'的div元素

react/app.js

```
var React = require('react');
var ReactDom = require('react-dom');
var MainPanel = require('./components/main-panel');
import InjectTapEventPlugin from '
    react-tap-event-plugin';

// Needed for onTouchTap
// http://stackoverflow.com/a/34015469/988941
InjectTapEventPlugin();

ReactDom.render(<MainPanel/>,
    document.getElementById('app'));
```

3.3.1.2 main-panel.js

MainPanel组件由LunarPanel组件与DatePanel组件组成，通过⟨GridList/⟩组件布局

同时注意要监听CalendarStore的事件'changeDay'，CalendarStore的属性'activeDay'一改变，MainPanel就要重新渲染组件

react/components/main-panel.js

```javascript
var React = require('react');
var ipc =  electronRequire('electron').ipcRenderer;
var DatePanel = require('./date-panel');
var LunarPanel = require('./lunar-panel');
var CalendarStore = require('../stores/calendar-store');
var CalendarAction = require('../actions/calendar-actions');
var COLOR = require('../constants/calendar-color');

import MuiThemeProvider from 'material-ui/styles/MuiThemeProvider';
import {GridList, GridTile} from 'material-ui/GridList';

var MainPanel = React.createClass({
    getInitialState: function () {
        return {
            today: CalendarStore.getToday(),
            activeDay: CalendarStore.getActiveDay(),
            activeMonth:
                CalendarStore.getActiveMonth()
        };
    },

    componentDidMount: function () {
        CalendarStore.addChangeDayListener(
            this._onChange);
        ipc.on('app-refresh', function () {
            CalendarAction.refresh();
        })
    },

    _onChange: function () {
        this.setState({
            today: CalendarStore.getToday(),
            activeDay: CalendarStore.getActiveDay(),
            activeMonth:
                CalendarStore.getActiveMonth()
        });
    },

    render: function () {
        var styles = {
            mainPanel: {
                border: '2px solid ' +
                    COLOR.mainPanel.border
            }
        };

        return (
            <MuiThemeProvider>
            <GridList cols={10} cellHeight={432}
                padding={0} style={styles.mainPanel}
                >
                <GridTile cols={4}>
                    <LunarPanel activeDay={
                        this.state.activeDay}/>
                </GridTile>
                <GridTile cols={6}>
                    <DatePanel
                        today={this.state.today}
                        activeDay={
                            this.state.activeDay}
                        activeMonth={
                            this.state.activeMonth}
                    />
                </GridTile>
            </GridList>
            </MuiThemeProvider>
        );
    }
});

module.exports = MainPanel;
```

3.3.1.3 lunar-panel.js

LunarPanel即左边的显示详细信息的面板
这里主要是前端的设计，通过⟨Transitive/⟩组件实现动画效果

react/components/lunar-panel.js

```
var React = require('react');
var Transitive = require('react-transitive-number');
var COLOR = require('../constants/calendar-color');

import {GridList, GridTile} from 'material-ui/GridList';
import Paper from 'material-ui/Paper';

var LunarPanel = React.createClass({
    render: function () {
        var activeDay = this.props.activeDay;
        var festivalElements = [];
        var key_index = 0;

        if (activeDay.term) {
            festivalElements.push(<br key={'festival' + (key_index++)}/>);
            festivalElements.push(
                <Transitive key={'festival' + (key_index++)}>{activeDay.term}</Transitive>
            );
        }
        if (activeDay.lunarFestival) {
            festivalElements.push(<br key={'festival' + (key_index++)}/>);
            festivalElements.push(
                <Transitive key={'festival' + (key_index++)}>{activeDay.lunarFestival}</Transitive>
            );
        }

        var styles = {
            ganZhiPanel: {
                background: COLOR.ganZhiPanel.background,
                color: COLOR.ganZhiPanel.color,
                lineHeight: '60px',
                textAlign: 'center'
            },
            ...
            ...
            ...
            hlPanel: {
                background: COLOR.hlPanel.background,
                color: COLOR.hlPanel.color,
                textOverflow: 'ellipsis',
                overflow: 'hidden',
                whiteSpace: 'nowrap'
            }
        };

        return (
            <GridList cols={10} cellHeight={31} padding={0}>
                <GridTile cols={10} rows={2} style={styles.ganZhiPanel}>
                    <Transitive>{activeDay.GanZhiYear + '年'}</Transitive>
                    <Transitive>{'    ' + activeDay.GanZhiMonth + '月'}</Transitive>
                    <Transitive>{'    ' + activeDay.GanZhiDay + '日'}</Transitive>
                </GridTile>
                <GridTile cols={10} rows={1} style={styles.dayPanel}>
                </GridTile>
                <GridTile cols={3} rows={6} style={styles.yearPanel}>
                    <br/><Transitive>{activeDay.year}</Transitive><br/>
                    <Transitive>{activeDay.month + '月'}</Transitive>
                </GridTile>
                <GridTile cols={4} rows={6} style={styles.dayPanel}>
                    <Paper zDepth={2} style={styles.dayDetail}>
                        <Transitive>{activeDay.day}</Transitive>
                    </Paper>
                </GridTile>
                <GridTile cols={3} rows={6} style={styles.festivalPanel}>
                    <br/>{festivalElements}
                </GridTile>
                <GridTile cols={10} rows={1} style={styles.monthPanel}>
                </GridTile>
                <GridTile cols={10} rows={2} style={styles.monthPanel}>
                    <Paper zDepth={1} style={styles.monthDetail}>
                        <Transitive>{activeDay.lunarMonthName + '  '}</Transitive>
                        <Transitive>{activeDay.lunarDayName}</Transitive>
                    </Paper>
                </GridTile>
                <GridTile cols={10} rows={2} style={styles.hlPanel}>
                    '宜' :{activeDay.hl_y}<br/>
                    '忌' :{activeDay.hl_j}
                </GridTile>
            </GridList>
        )
    }
});

module.exports = LunarPanel;
```

3.3.1.4 date-panel.js

DatePanel组件即右边的面板，由DateController组件与DateChoosePanel组件组成

react/components/date-panel.js

```
var React = require('react');
var DateController = require('./date-controller');
var DateChoosePanel = require('./date-choose-panel')
    ;

var DatePanel = React.createClass({
    render: function () {
        return (
            <div>
                <DateController activeDay={
                    this.props.activeDay} />
                <DateChoosePanel
                    today={this.props.today}
                    activeDay={this.props.activeDay}
                    activeMonth={
                        this.props.activeMonth}
                />
            </div>
        )
    }
});

module.exports = DatePanel;
```

3.3.1.5 date-choose-panel.js

DateChoosePanel组件即右下方的选择日子的面板，通过⟨Table/⟩组件实现表格布局

这里每格都是一个按钮，按下后，会触发'changeDay'这个action

react/components/date-choose-panel.js

```
var React = require('react');
var CalendarAction = require('../actions/
    calendar-actions');
var COLOR = require('../constants/calendar-color');

import {Table, TableBody, TableHeader,
    TableHeaderColumn, TableRow, TableRowColumn}
    from 'material-ui/Table';
import {FlatButton} from 'material-ui'

var DateChoosePanel = React.createClass({
    render: function () {
        var today = this.props.today;
        var activeDay = this.props.activeDay;
        var activeMonth = this.props.activeMonth;

        var table = [];
        var tableBody = [];
        var tableRow = [];

        var styles = {
            tableHeader: {
                background:
                    COLOR.tableHeader.background
            },
            tableBody: {
                background:
                    COLOR.tableBody.background
            },
            orangeLabel: {
                lineHeight: 0,
                fontSize: 10,
                color: COLOR.orangeLabel.color
            },
            greyLabel: {
                lineHeight: 0,
                fontSize: 10,
                color: COLOR.greyLabel.color
            },
            tableButton: {
                minWidth: 66,
                height: 50
            }
        };

        table.push(
            <TableHeader key='tableHeader'
                adjustForCheckbox={false}
                displaySelectAll={false}
                enableSelectAll={false}
                style={styles.tableHeader}
            >
                <TableRow>
                    <TableHeaderColumn>日</
                        TableHeaderColumn>
                    <TableHeaderColumn>一</
                        TableHeaderColumn>
                    <TableHeaderColumn>二</
                        TableHeaderColumn>
                    <TableHeaderColumn>三</
                        TableHeaderColumn>
                    <TableHeaderColumn>四</
                        TableHeaderColumn>
                    <TableHeaderColumn>五</
                        TableHeaderColumn>
                    <TableHeaderColumn>六</
                        TableHeaderColumn>
                </TableRow>
            </TableHeader>
        );

        for (var i = 0; i < activeMonth.length; i++)
             {
            var date = activeMonth[i];

            var label = '';
            if (date.lunarFestival != undefined)
                label = <div style={styles.orangeLabel}>{date.lunarFestival}</div>;
            else if (date.term != undefined)
                label = <div style={styles.orangeLabel}>{date.term}</div>;
            else if (date.lunarDay == 1)
                label = <div style={styles.orangeLabel}>{date.lunarMonthName}</div>;
            else
                label = <div style={styles.greyLabel}>{date.lunarDayName}</div>;

            var styles_tableCell = {
                background: COLOR.tableCell.background.normal,
                paddingLeft: 0
            };

            ...
            ...
            ...

            tableRow.push(
                <TableRowColumn style={styles_tableCell} key={'tableRowColumn' + i}>
                    <FlatButton
                        label={date.day}
                        labelStyle={styles_numberLabel}
                        labelPosition={'before'}
                        rippleColor={COLOR.numberLabel.color.ripple}
                        onTouchTap={this._onClick.bind(this, date)}
                        style={styles.tableButton}
```

```
                        >
                            {label}
                        </FlatButton>
                    </TableRowColumn>
                );

                if ((i+1) % 7  == 0) {
                    tableBody.push(
                        <TableRow key={'tableRow' + i}>
                            {tableRow}
                        </TableRow>
                    );
                    tableRow = [];
                }
            }

            table.push(
                <TableBody
                    displayRowCheckbox={false}
                    style={styles.tableBody}
                    key='tableBody'
                >
                    {tableBody}
                </TableBody>
            );

            return (
                <Table selectable={false}>
                    {table}
                </Table>
            )
        },

        _onClick: function (date) {
            CalendarAction.changeDay(date);
        }
});

module.exports = DateChoosePanel;
```

3.3.1.6 date-controller.js

DateController组件即右上方的toolbar

只有这里需要从DialogStore与ConfigStore中获取数据（对话框、设置选项的状态），因此也要监听DialogStore的'closeAllDialog'事件

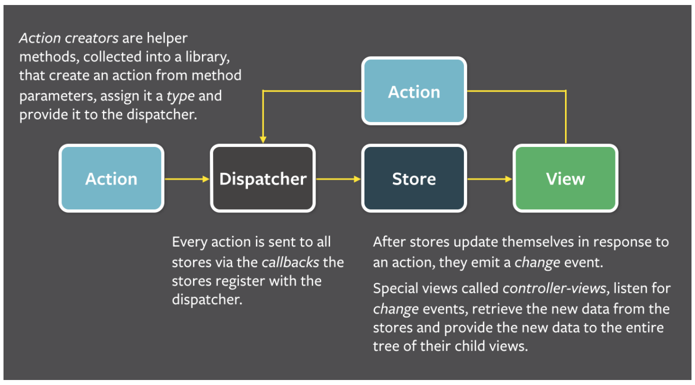
每一个按钮都会触发相应的action

react/components/date-controller.js

```
var DateController = React.createClass({
    getInitialState: function () {
        return {
            noteDialog: DialogStore.getNote(),
            optionDialog: DialogStore.getOption(),
            aboutDialog: DialogStore.getAbout(),
            registerDialog: DialogStore.getRegister
                (),
            note: this.props.activeDay.note,
            alwaysOnTop: ConfigStore.getAlwaysOnTop
                ()
        };
    },

    componentDidMount: function () {
        DialogStore.addCloseDialogListener(
            this.closeAllDialog);
        ipc.on('app-hide', function () {
            DialogAction.closeDialog();
        })
    },

    openDialog: function (name) {
        if (name == 'note')
            this.setState({
                note: this.props.activeDay.note,
                [name + 'Dialog']: true
            });
        else if (name == 'option')
            this.setState({
                alwaysOnTop:
                    ConfigStore.getAlwaysOnTop(),
                [name + 'Dialog']: true
            });
        else
            this.setState({
                [name + 'Dialog']: true
            });
    },

    closeDialog: function (name) {
        ...
        ...
        ...
    },

    closeAllDialog: function () {
        this.setState({
            noteDialog: false,
            optionDialog: false,
            aboutDialog: false,
            registerDialog: false
        });
    },

    handleToggle: function (event, toggled) {
        this.setState({
            [event.target.name]: toggled
        });
    },

    changeNote: function (event) {
        this.setState({
            note: event.target.value
        });
    },

    render: function () {
        var activeDay = this.props.activeDay;

        var noteDialogActions = [
            <RaisedButton
                label='保存'
                primary={true}
                icon={<Save/>}
                onTouchTap={this._saveNote}
            />,
            <FlatButton
                label='關閉'
                primary={true}
                onTouchTap={this.closeDialog.bind(
                    this, 'note')}
            />
        ];

        ...
        ...
        ...

        var styles = {
            dateController: {
                background:
                    COLOR.dateController.background
            },
            yearField: {
                width: 40
            },
            monthField: {
                width: 18
            }
        };

        return (
            <Toolbar style={styles.dateController}>
            ...
            ...
            ...
            </Toolbar>
        );
    },

    _minusYear: function () {
        CalendarAction.minusYear();
    },

    _addYear: function () {
        CalendarAction.addYear();
    },

    _changeYear: function (event) {
        var value = parseInt(event.target.value ||
            0);
        if (value < RANGE.MIN_YEAR || value >
            RANGE.MAX_YEAR)
             return;
        var activeDay = this.props.activeDay;
        activeDay.year = value;
        CalendarAction.changeDay(activeDay);
    },

    _minusMonth: function () {
        CalendarAction.minusMonth();
    },

    _addMonth: function () {
        CalendarAction.addMonth();
    },

    _refresh: function(){
        CalendarAction.refresh();
    },

    _saveNote: function () {
        CalendarAction.changeNote(
            this.props.activeDay,
            this.state.note.substr(0,140));
    },

    _saveOption: function () {
        ConfigAction.setAlwaysOnTop(
            this.state.alwaysOnTop);
    },

    _quitApp: function(){
        CalendarAction.quitApp();
    }
});

module.exports = DateController;
```

3.3.2 Dispatcher

3.3.2.1 dispatcher.js


Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a *type* and provide it to the dispatcher.
Action
Action
Dispatcher
Store
View
Every action is sent to all stores via the *callbacks* the stores register with the dispatcher.
After stores update themselves in response to an action, they emit a *change* event.
Special views called *controller-views*, listen for *change* events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

Dispatcher是Flux模型里的核心，主要是以下两点：

- View分发actions给Dispatcher，即“触发事件”
- Store在Dispatcher上注册相应action，即“响应事件”

这两者的实现一般放在相应View或者Store的文件里，所以dispatcher.js本身很简单：

react/dispatchers/dispatcher.js

```
/**
 * Created by gou4shi1 on 16-8-16.
 */

var Dispatcher = require('flux').Dispatcher;

module.exports = new Dispatcher();
```

3.3.3 Action

3.3.3.1 calendar-actions.js

这里定义了日历相关的一系列操作的函数，函数执行后会分发相应的action到Dispatcher上

react/actions/calendar-actions.js

```
var Dispatcher = require('../dispatchers/dispatcher');
var TYPE = require('../constants/action-type');
var ipc =  electronRequire('electron').ipcRenderer;
var CalendarAction = {
    changeDay: function (activeDay) {
        Dispatcher.dispatch({
            actionType: TYPE.CHANGE_DAY,
            activeDay: activeDay
        });
    },
    changeNote: function (activeDay, note) {
        Dispatcher.dispatch({
            actionType: TYPE.CHANGE_NOTE,
            note: note,
            activeDay: activeDay
        });
    },
    addMonth: function () {
        Dispatcher.dispatch({
            actionType: TYPE.ADD_MONTH
        });
    },
    ...
    ...
    refresh: function () {
        Dispatcher.dispatch({
            actionType: TYPE.REFRESH
        });
    },
    quitApp: function () {
        ipc.send('app-quit');
    }
};
module.exports = CalendarAction;
```

3.3.3.2 config-actions.js

目前，设置选项里只有“总在最上”一项可供设置：

react/actions/config-actions.js

```
var Dispatcher = require('../dispatchers/dispatcher'
    );
var TYPE = require('../constants/action-type');

var ConfigAction = {
    setAlwaysOnTop: function (value) {
        Dispatcher.dispatch({
            actionType: TYPE.ALWAYS_ON_TOP,
            value: value
        });
    },
};

module.exports = ConfigAction;
```

3.3.3.3 dialog-actions.js

根据需求，只有“关闭所有对话框“这一操作需要与Store交互

react/actions/dialog-actions.js

```
var Dispatcher = require('../dispatchers/dispatcher'
    );
var TYPE = require('../constants/action-type');

var DialogAction = {
    closeDialog: function () {
        Dispatcher.dispatch({
            actionType: TYPE.CLOSE_DIALOG
        });
    },
};

module.exports = DialogAction;
```

3.3.4 Store

3.3.4.1 calendar-database.js

农历相关的后端数据由NPM里的'lunar-calendar-zh'包提供，calendar-database把'Lunar-calendar-zh'包、黄历、记事本封装在一起

react/stores/calendar-database.js

```
var RANGE = require('../constants/calendar-range');
var LC = require('lunar-calendar-zh');
var HL = require('./huangli/huangli');
var Assign = require('object-assign');
var _find = require('lodash').find;

var CalendarDatabase = {
    noteData : localStorage,

    isInRange: function (year, month, day) {
        if (year < RANGE.MIN_YEAR || year >
            RANGE.MAX_YEAR)
            return false;
        if (month < RANGE.MIN_MONTH || month >
            RANGE.MAX_MONTH)
            return false;
        if (day < 1 || day > LC.calendar(year, month
            , false).monthDays)
            return false;
        return true;
    },

    changeNote: function (year, month, day, note) {
        if (!this.isInRange(year, month, day))
            return false;
        year = year.toString();
        month = (month < 10 ? '0' : '') + month;
        day = (day < 10 ? '0' : '') + day;
        this.noteData.setItem(year + month + day,
            note);
    },

    getLunarByDay: function (year, month, day) {
        if (!this.isInRange(year, month, day))
            return {};
        var monthData = LC.calendar(year, month,
            false).monthData;
        var dayData =  _find(monthData, { 'day': day
            });
        year = year.toString();
        month = (month < 10 ? '0' : '') + month;
        day = (day < 10 ? '0' : '') + day;
        var hl = HL['hl' + year]['d'+month+day];
        //encapsulation:
        return Assign(dayData, {
            hl_y: hl.y || '',
            hl_j: hl.j || '',
            note: this.noteData.getItem(year + month
                 + day) || ''
        });
    },

    getLunarByMonth: function (year, month) {
        if (!this.isInRange(year, month, 1))
            return {};
        var monthData = LC.calendar(year, month,
            true).monthData;
        //encapsulation:
        monthData.forEach(function (dayData, index)
            {
            monthData[index] = this.getLunarByDay(
                dayData.year, dayData.month,
                dayData.day);
        }.bind(this));
        return monthData;
    },

    getMonthDays: function (year, month) {
        if (!this.isInRange(year, month, 1))
            return {};
        return LC.calendar(year, month, false).
            monthDays;
    }
};

module.exports = CalendarDatabase;
```

3.3.4.2 calendar-store.js

有了calendar-database的封装，calendar-store的实现就很简洁了

它本身只有一个属性'activeDay'，即当前选中的日子，默认是今天

然后是一系列get操作与change操作

因为要监听'changeDay' 这一事件，所以calendar-store以EventEmitter为原型

同时也要在Dispatcher上注册相应action

react/stores/calendar-store.js

```
var CalendarDatabase = require('./calendar-database');
var Dispatcher = require('../dispatchers/dispatcher');
var TYPE = require('../constants/action-type');
var RANGE = require('../constants/calendar-range');
var EventEmitter = require('events').EventEmitter;
var Assign = require('object-assign');

var CalendarStore = Assign({},
    EventEmitter.prototype, {
    activeDay: false,

    getToday: function () {
        var date = new Date();
        return CalendarDatabase.getLunarByDay(
            date.getFullYear(), date.getMonth() + 1,
             date.getDate());
    },

    getActiveDay: function () {
        return this.activeDay || this.getToday();
    },

    getActiveMonth: function () {
        var activeDay = this.getActiveDay();
        return CalendarDatabase.getLunarByMonth(
            activeDay.year, activeDay.month);
    },

    changeDay: function(year, month, day) {
        //over year
        if (month == 13) {
            year++;
            month = 1;
        }
        if (month == 0) {
            year--;
            month = 12;
        }
        if (year > RANGE.MAX_YEAR || year < RANGE.MIN_YEAR)
            return false;
        if (day > CalendarDatabase.getMonthDays(year, month))
            day = 1;
        this.activeDay = CalendarDatabase.getLunarByDay(year, month, day);
        this.emit(TYPE.CHANGE_DAY);
        return true;
    },

    changeNote: function (activeDay, note) {
        CalendarDatabase.changeNote(activeDay.year, activeDay.month, activeDay.day, note);
        this.activeDay = CalendarDatabase.getLunarByDay(activeDay.year,activeDay.month, activeDay.day);
        this.emit(TYPE.CHANGE_DAY);
    },

    addChangeDayListener: function (callback) {
        this.on(TYPE.CHANGE_DAY, callback);
    }
});

Dispatcher.register(function (action) {
    var activeDay = CalendarStore.getActiveDay();

    switch (action.actionType) {
        case TYPE.CHANGE_DAY:
            activeDay = action.activeDay;
```

```
                CalendarStore.changeDay(activeDay.year,
                    activeDay.month, activeDay.day);
                break;
            case TYPE.ADD_MONTH:
                CalendarStore.changeDay(activeDay.year,
                    activeDay.month + 1, activeDay.day);
                break;

            ...
            ...
            ...

            case TYPE.REFRESH:
                activeDay = CalendarStore.getToday();
                CalendarStore.changeDay(activeDay.year,
                    activeDay.month, activeDay.day);
                break;
            case TYPE.CHANGE_NOTE:
                CalendarStore.changeNote(
                    action.activeDay, action.note);
                break;
            default:
                break;
        }
});

module.exports = CalendarStore;
```

3.3.4.3 dialog-store.js

dialog-store只需要维护对话框的状态，套路与calendar-store类似

react/stores/dialog-store.js

```
var Dispatcher = require('../dispatchers/dispatcher');
var TYPE = require('../constants/action-type');
var EventEmitter = require('events').EventEmitter;
var Assign = require('object-assign');

var DialogStore = Assign({}, EventEmitter.prototype, {
    note: false,
    option: false,
    about: false,
    register: false,

    getNote: function () {
        return this.note;
    },
    ...
    ...
    ...
    closeAllDialog: function () {
        this.note = false;
        this.option = false;
        this.about = false;
        this.register = false;
        this.emit(TYPE.CLOSE_DIALOG);
    },
    addCloseDialogListener: function (callback) {
        this.on(TYPE.CLOSE_DIALOG, callback);
    }
});

Dispatcher.register(function (action) {
    if (action.actionType == TYPE.CLOSE_DIALOG)
        DialogStore.closeAllDialog();
});

module.exports = DialogStore;
```

3.3.4.4 config-store.js

通过NPM里的'ConfigStore'包把设置选项保存在本地 注意，设置为“总在最上”后，要向ipc传输信号，使应用重启

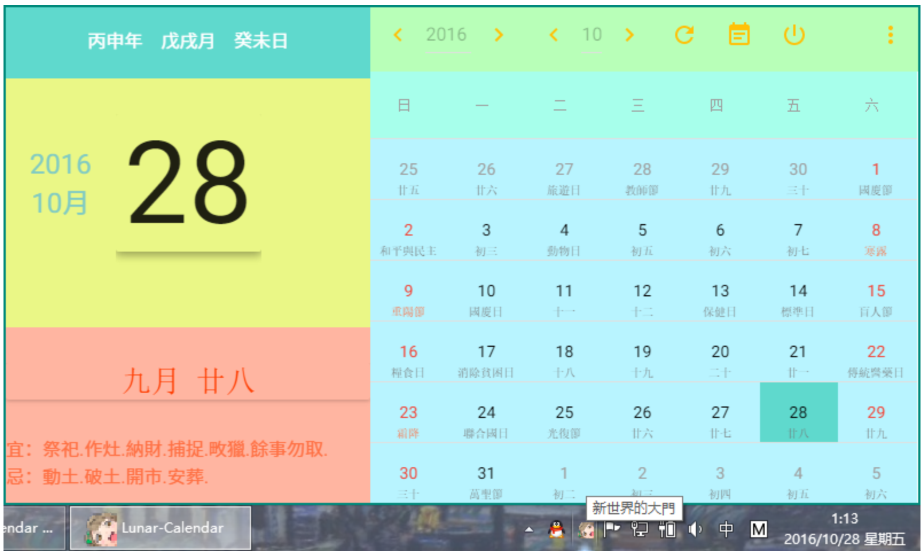
react/stores/config-store.js

```
var Configstore = electronRequire('configstore');
var pkg = require('../../package.json');
var ipc =  electronRequire('electron').ipcRenderer;
var Dispatcher = require('../dispatchers/dispatcher'
    );
var TYPE = require('../constants/action-type');

var conf = new Configstore(pkg.name);

var ConfigStore = {
    alwaysOnTop: conf.get('alwaysOnTop'),

    getAlwaysOnTop: function () {
        return this.alwaysOnTop;
    },

    setAlwaysOnTop: function (value) {
        var restart = (value != conf.get('
            alwaysOnTop'));
        conf.set('alwaysOnTop', value);
        if (restart)
            ipc.send('app-restart');
    }
};

Dispatcher.register(function (action) {
    if (action.actionType == TYPE.ALWAYS_ON_TOP)
        ConfigStore.setAlwaysOnTop(action.value);
});

module.exports = ConfigStore;
```

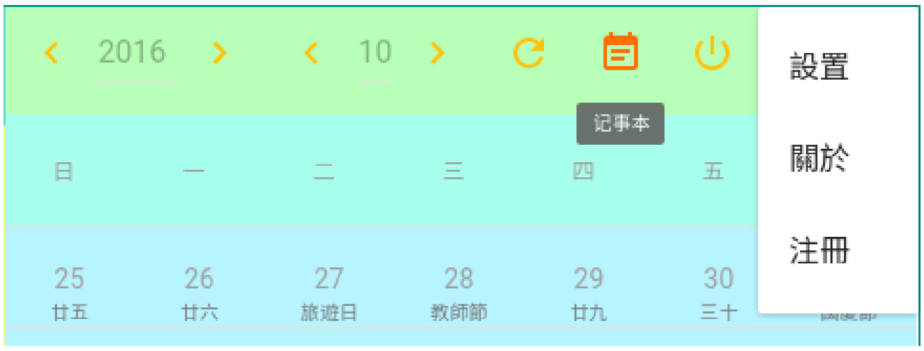
4 测试情况

4.1 启动界面

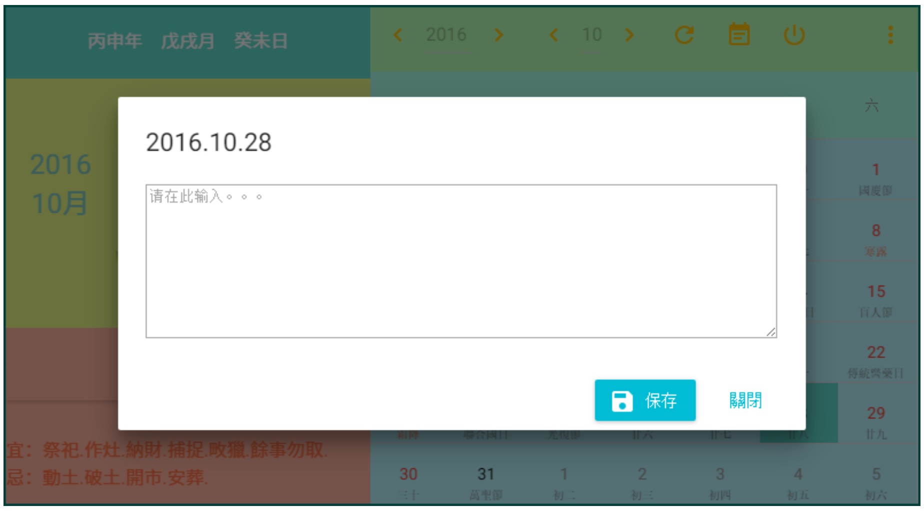
在Windows右下角的托盘栏里单击图标即可启动



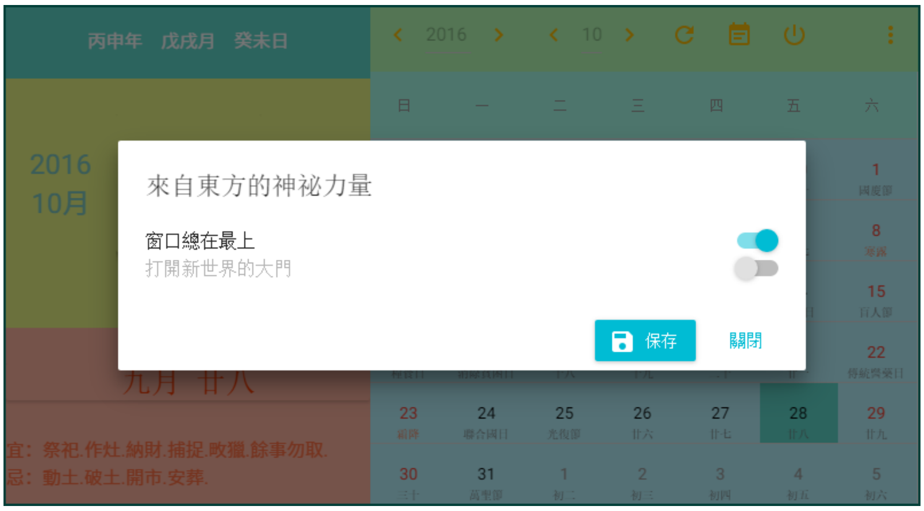
4.2 工具栏界面



4.3 记事本界面



4.4 设置菜单

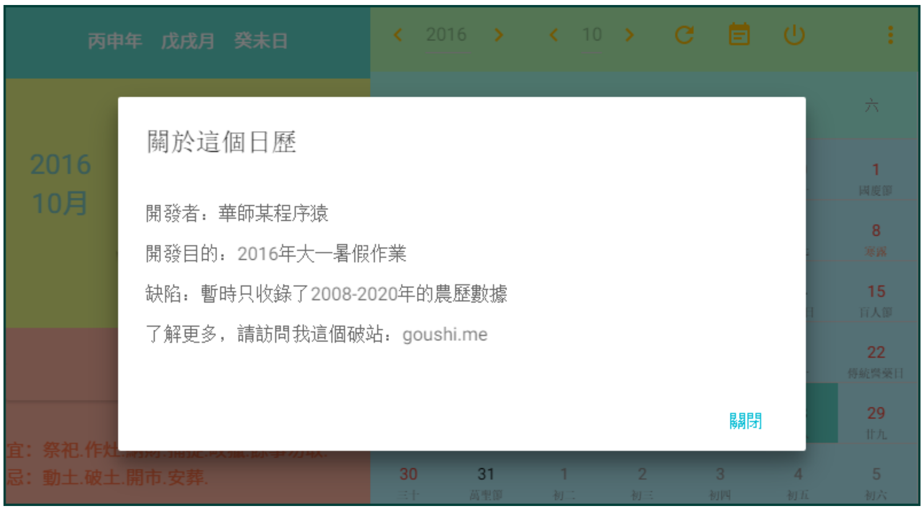


來自東方的神祕力量

窗口總在最上 [✓ 開]

打開新世界的大門 [關]

4.5 关于菜单



關於這個日歷

開發者：華師某程序猿

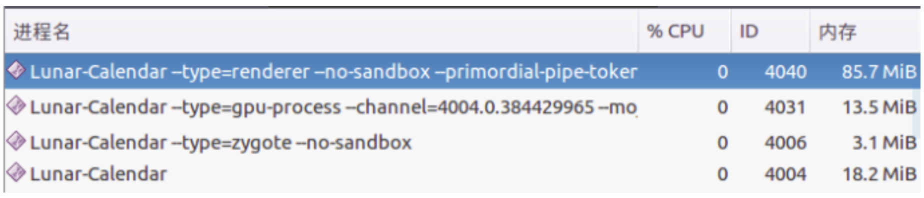
開發目的：2016年大一暑假作業

缺陷：暫時只收錄了2008-2020年的農歷數據

了解更多，請訪問我這個破站：goushi.me

4.6 占用资源

总共约占用100MB的内存，但是CPU消耗很小



进程名	% CPU	ID	内存
Lunar-Calendar –type=renderer –no-sandbox –primordial-pipe-toker	0	4040	85.7 MiB
Lunar-Calendar –type=gpu-process –channel=4004.0.384429965 –mo	0	4031	13.5 MiB
Lunar-Calendar –type=zygote –no-sandbox	0	4006	3.1 MiB
Lunar-Calendar	0	4004	18.2 MiB

5 优点与改进

5.1 优点

- 功能强大，实现了一般农历需要实现的所有功能
- 界面优美，采用了Google的Material设计标准

5.2 缺点

- 程序占用硬盘空间过大，多达一百多MB
- 程序占用内存空间较大，约一百MB

6 参考文献

6.1 Q&A

https://stackoverflow.com

6.2 Electron

http://electron.atom.io/

6.3 Menubar

https://github.com/maxogden/menubar

6.4 React

https://facebook.github.io/react/docs/getting-started.html

6.5 Flux

https://facebook.github.io/flux/docs/overview.html

6.6 browserify

http://browserify.org/

6.7 gulp

http://gulpjs.com/

6.8 Material Design

https://material.google.com/

6.9 Material-ui

http://www.material-ui.com/