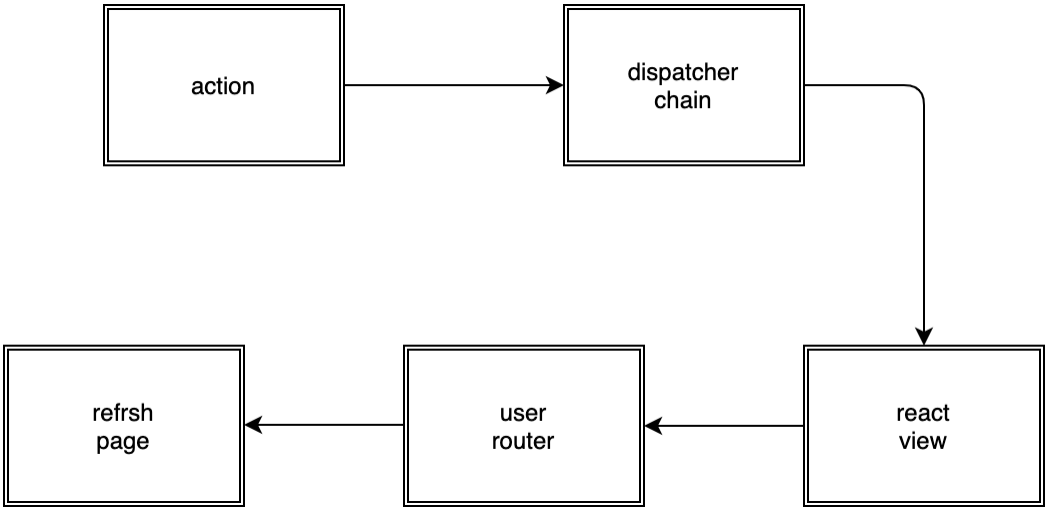
SyriusApplet架构实现

架构不变式：

view = Observe(action)

单向数据示意图：



变化侦测篇：

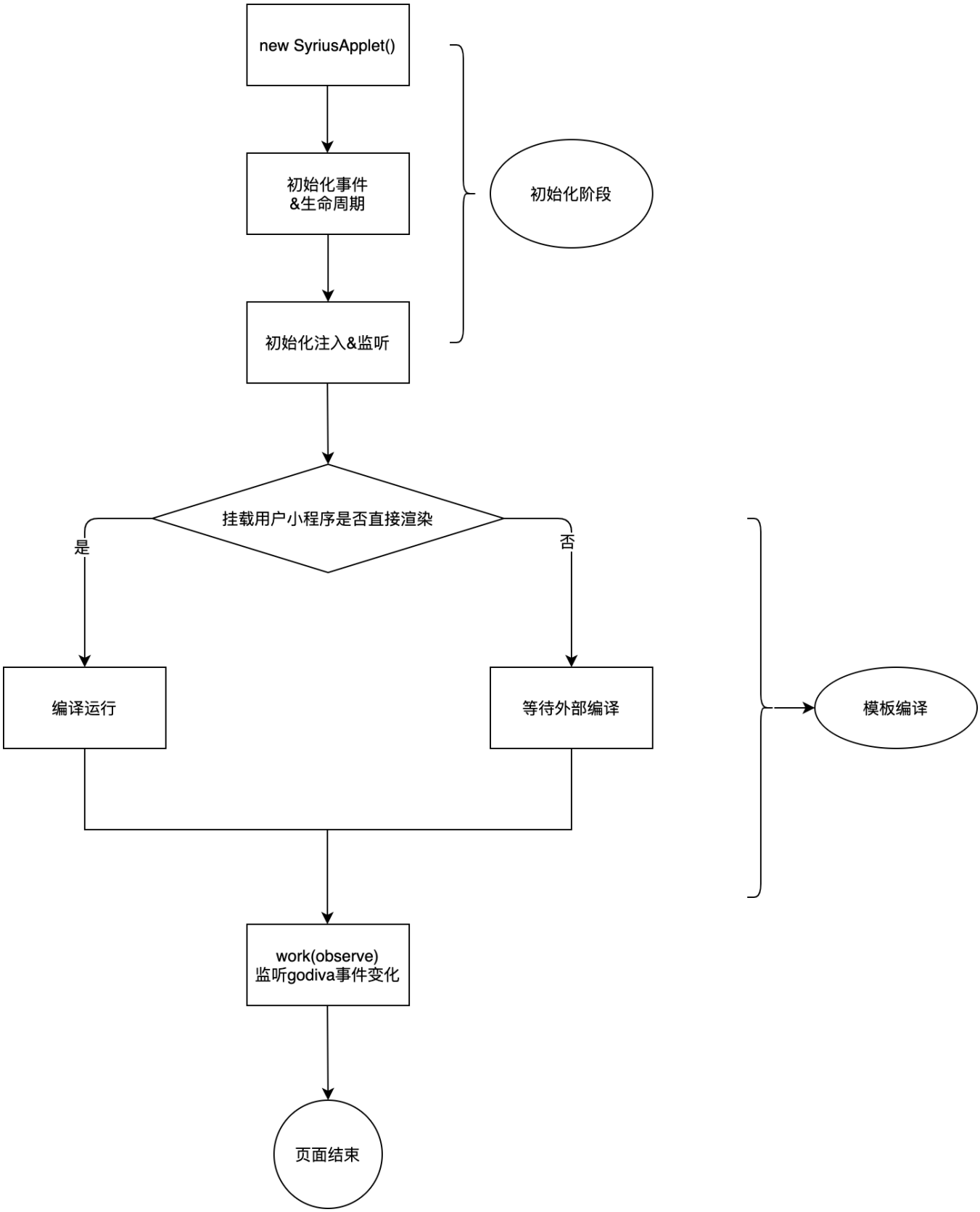
```
function observe(state, $setCurrentActionInfo,$getLastActionInfo) {
    let $_NEW_CURRENT_ACTION_INFO = {};
    //  Listen to GODIVA events
    window.actionRequestCallback = (actionRequestJson) => {
        //  Read the last action info
        let $_LAST_ACTION_INFO = $getLastActionInfo();
        //  Save the new action info of GODIVA
        $_NEW_CURRENT_ACTION_INFO = $setCurrentActionInfo(evalActionReqToObj(actionRequestJson));

        // OBSERVE CHAIN
        let chain = moveToPackBindRegion
            .after(bindPack)
            .after(moveToPickingProduct)
            .after(pickingProduct)
            .after(packageRegion)
            .after(moveToHoldRegion)
            .after(moveToPackageRegion)
            .after(moveToException)
            .after(moveToBindCarrier)
            .after(bindNewCarrier)
            .after(waitNewTask)
            .after(exceptionRegion)
            .after(moveToShelfStorageRegion)
            .after(onShelf)
            .after(onShelfOver)
            .after(moveToWormholeRegion)
            .after(waitingForWormhole);

        try {
            let nextActionInfo = window.GodivaJsInterface.getNextAction();
            let actionReq = JSON.parse(nextActionInfo);
            let nextAction = evalActionReqToObj(actionReq);
            console.log(" nextAction", nextAction);
            chain(evalActionReqToObj(actionRequestJson), state, nextAction, $_LAST_ACTION_INFO);
        } catch (error) {
            chain(evalActionReqToObj(actionRequestJson), state, {}, $_LAST_ACTION_INFO);
        }

    }
}
```

模板编译篇：

```
let syriusApplet = new SyriusApplet({
    applet: <YOUR_APP>,
    state: {
        MOVE_TO_PACK_BIND_REGION: function () {

        },
        WAIT_NEW_TASK: function () {

        },
        MOVE_TO_BIND_CARRIER_REGION: function () {

        },
        MOVE_TO_SHELF_BIND_REGION: function () {

        },
        MOVE_TO_HOLD_REGION: function () {

        },
        MOVE_TO_EXCEPTION_REGION: function () {

        },
        DEAL_EXCEPTION: function () {

        },
        BIND_PAYLOAD: function () {

        },
        BIND_SHELF_PAYLOAD: function () {

        },

        TEAR_DOWN_OLD_PAYLOAD: function () {

        },
        BIND_NEW_CARRIER: function () {

        },
        MOVE_TO_PICK_PRODUCT: function (nextAction) {

        },
        PICKING_PRODUCT: function () {

        },
        EMERGENCY_STOP: function () {

        },
        MOVE_TO_SHELF_STORAGE_REGION: function (nextAction) {

        },
        ON_SHELF: function () {

        },
        ON_SHELF_OVER: function () {

        },
        MOVE_TO_WORMHOLE_REGION: function () {

        },
        WAITING_FOR_WORMHOLE: function () {

        }
    },
    work: function (observe) {
        return observe(this.$state, this.$setCurrentActionInfo,this.$getLastActionInfo);
    },
});
```

生命周期篇：

综述：



初始化阶段（new SyriusApplet）

```
function SyriusApplet(options) {
    this._init(options);
}


initMixin(SyriusApplet);
```

初始化阶段（initMixin）

```
function initMixin(SyriusApplet) {
    SyriusApplet.prototype._init = function (options) {
        const vm = this;
        vm.$options = options;
        vm._self = vm;
        this.$applet = vm.$options.applet;
        this.$state = vm.$options.state;
        this.$work = vm.$options.work || {};
        this.$setCurrentActionInfo = (actionJson) => {
            vm.$currentActionInfo = actionJson;
            return vm.$currentActionInfo;
        }
        this.$getLastActionInfo = () => {
            return vm.$currentActionInfo
        }
        /**
         *  Function
         *  init function.prototype
         */
        init();
        /**
         *
         * init event
         */
        initEvent(vm);
        /**
         *
         * init state(observe)
         */
        if (this.$applet && typeof this.$work === "function") {
            this.$work(observe);
        }
    }
}
```

初始化阶段（init）

```
function init(){
    Function.prototype.after = function(fn) {
        let self = this;
        return async function() {
            let ret = await self.apply(this, arguments);
            if (ret === "nextSuccessor") {
                return fn.apply(this,arguments);
            }
            return ret;
        }
    };
}
```

初始化阶段（initEvent）

```
let eventListenersInterface = window.EventListenersInterface;

initGodivaEvent(vm);
initVoiceLightEvent(vm, eventListenersInterface);
initScannerEvent(vm, eventListenersInterface);
initMoveManager(vm, eventListenersInterface);
initPoseEvent(vm, eventListenersInterface);
initEquipmentStatusEvent(vm, eventListenersInterface);
initKnowledgeBaseEvent(vm);
initCommonEvent(vm);
initLaunchEvent(vm);
initBaroEvent(vm, eventListenersInterface);
```

实例方法篇：

数据相关

```
    vm.$options :options

    vm.$applet  :$appletvuesyriusApplet
```

```
    vm.$currentActionInfo :actionDetailInfo<Object>
```

指令篇：

通用事件（initCommonEvent）
设备状态事件（initEquipmentStatusEvent）

```
function initEquipmentStatusEvent(vm, eventListenersInterface){
    let equipmentStatus = new EquipmentStatus(eventListenersInterface);
    vm.onceCheckCameraStatus = async function(){
        let status = await equipmentStatus.onceCheckCameraStatus();
        console.log(status);
        return status
    }
}
```

godiva消息事件（initGodivaEvent）

```
function initGodivaEvent(vm) {
    let godivaReportEvent = new GodivaReportEvent();
    let godivaTeleportEvent = new GodivaTeleportEvent();
    let godivaTaskEvent = new GodivaTaskEvent();
    let godivaInitEvent = new GodivaInitEvent()

    /**
     * godiva
     * report data to godiva
     * @param currentAction
     */
    vm.reportEvent = function(currentAction){
        godivaReportEvent.reportEvent(currentAction);
    }
    vm.onNotifyListen = function(resCallbackObj){
        godivaReportEvent.onNotifyListen(resCallbackObj);
    }

    /**
     * godiva calculate witch floor can arrive result
     * godiva
     * @param startNode
     * @returns {*}
     */
    vm.getOutGoingNodesForTeleport = function(startNode){
        return godivaTeleportEvent.getOutGoingNodesForTeleport(startNode);
    };
    vm.onCancelTask = function(){
        return godivaTaskEvent.onCancelTask();
    };
    vm.onPauseTask = function(){
        console.log("pauseTask");
        return godivaTaskEvent.onPauseTask();
    };
    vm.onResumeTask = function(){
        return godivaTaskEvent.onResumeTask();
    };
    vm.onGetAllTask = function(){
        return godivaTaskEvent.onGetAllTask();
    }
    vm.onGetAllTaskInPool = function(){
        return godivaTaskEvent.onGetAllTaskInPool();
    }
    vm.godivaConfig = function(){
        return godivaInitEvent.godivaConfig();
    }

}
```

知识库事件（initKnowledgeBaseEvent）
启动事件（initLaunchEvent）
移动事件（initMoveManager）

```
function initMoveManager(vm, eventListenersInterface) {

    let move = new Move(eventListenersInterface);
    vm.safeInitNavigation = function (callback) {
        return move.safeInitNavigation(callback);
    }
    vm.safePauseMove = function (callback) {
        return move.safePauseMove(callback);
    }
    vm.safeResumeMove = function (callback) {
        return move.safeResumeMove(callback)
    }
    vm.isPauseMove = async function(){
        return await move.isPauseMove();
    }
    vm.getLastMoveStatus = function(){
        return move.lastMoveStatus;
    }
}
```

位姿事件（initPoseEvent）

对外暴露的API部分

```
function initPoseEvent(vm, eventListenersInterface) {
    let pose = new Pose(eventListenersInterface);
    vm.safeSwitchMapByChangeLayer = function(mapLayerName, loadingCallback){
        return pose.safeSwitchMapByChangeLayer(mapLayerName, loadingCallback);
    };
    vm.listenRobotPose = function(callback){
        pose.listenRobotPose(callback);
    }
    vm.unbindListenRobotPose = function(){
        pose.unbindListenRobotPose();
    };
    vm.listenLayerPose = function(callback){
        pose.listenLayerPose(callback);
    };
    vm.unbindListenLayerPose = function(){
        pose.unbindListenLayerPose();
    };
    vm.safeManualPose = function(wormhole, loadingCallback){
        return pose.safeManualPose(wormhole, loadingCallback);
    };
}
```

扫码枪事件（initScannerEvent）

对外暴露的API部分

```
function initScannerEvent(vm, eventListenersInterface) {
    let scanner = new Scanner(eventListenersInterface);

    vm.unbindScan = function () {
        scanner.unbindScan();
    }
    vm.scanCode = function (resCallbackObj = {
        success: function () {
        }, fail: function () {
        }
    }) {
        scanner.onScan(resCallbackObj);
    }
}
initVoiceLightEvent
safeRetry
```

全局API篇：

init.js

```
function init(){
    Function.prototype.after = function(fn) {
        let self = this;
        return async function() {
            let ret = await self.apply(this, arguments);
            if (ret === "nextSuccessor") {
                return fn.apply(this,arguments);
            }
            return ret;
        }
    };
}
```