Financial Technology
腾讯支付基础平台与金融应用线

React+Reflux实践及性能调优

—— linchuang（黄志鹏）

个人简介

黄志鹏
LincHuang

腾讯CDG通讯充值与彩票业务部前端开发组成员。13年加入腾讯，主要负责手Q/微信渠道QQ彩票前端开发工作。15年业务转型，目前致力于竞猜平台搭建。

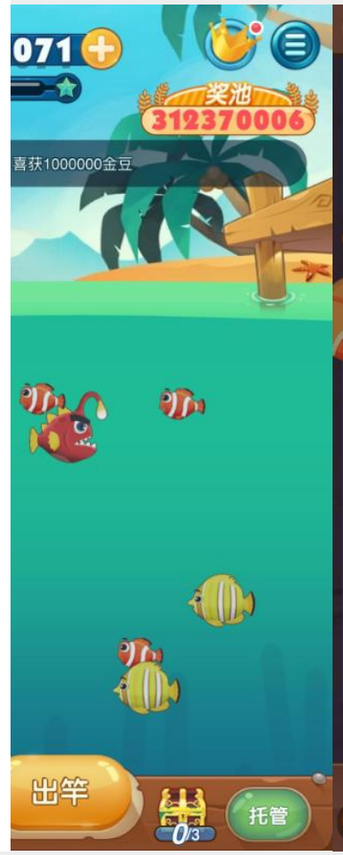
QQ彩票

竞猜平台

竞猜平台应用


本期加奖8%
打赏
上期：
梅小9
往期
投注时间还剩
15S
梅兰竹菊 1赔2.6
梅
兰
竹
菊
4-10 1赔0.8
三同花通选 1赔24
11-17 1赔0.8
小
三同花通吃
大
三同花通吃
三同花单选 1赔150
x3F天勤x...
龙
LincWong
博染
再爷
我的金豆
98457
100
1000
10000
0/3
撤销


虚拟赛事
热门
全部
法国
未开始
--:--
00:00
欧洲足球锦标赛
国际
vs
21:00
星期日，2016/06/26
法国
德尚，迪迪埃
主教练
里佐利，尼古拉
(ITA)
裁判
胜 1.31
平 4.25
81027人竞猜
直播中
1:0
80'
欧洲杯
猜·享
FC横滨
胜1.32
平
平3.30
收起
直播中
天天送豆
玩就送200金豆

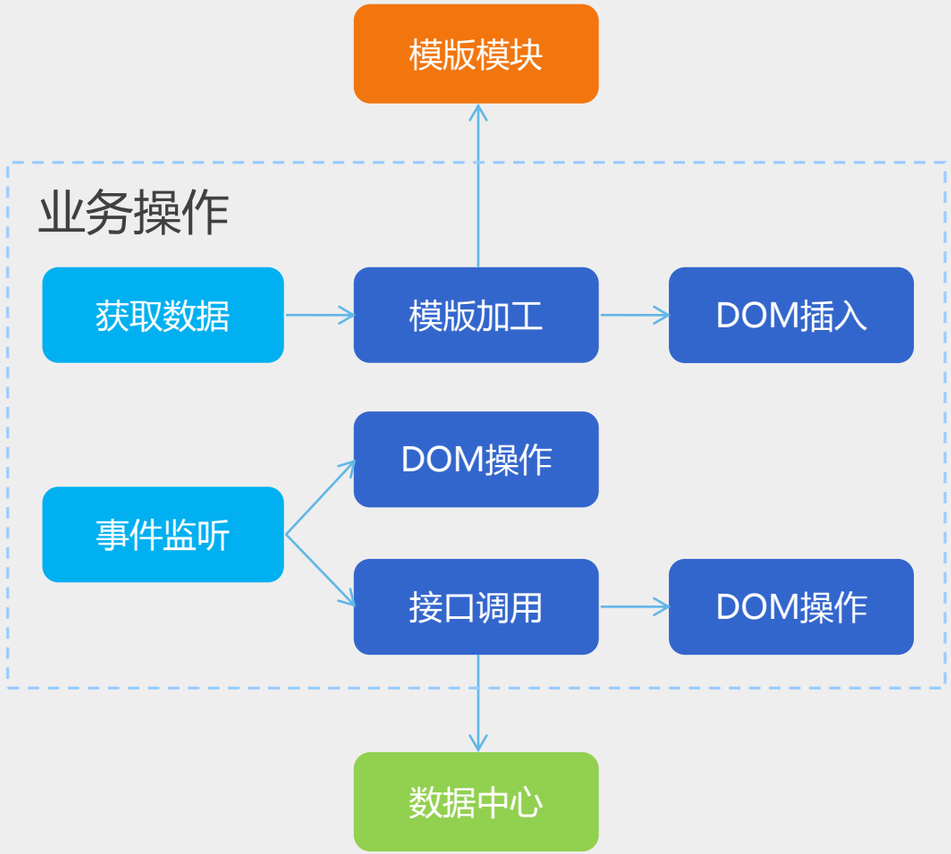

071
奖池
312370006
喜获1000000金豆
出竿
0/3
托管

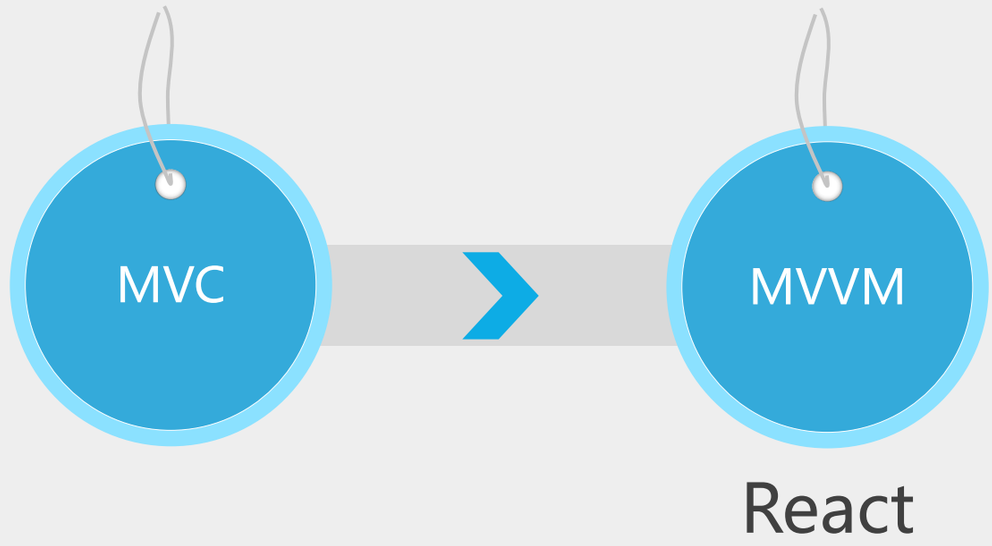

真实场
虚拟场
3501.80
3501.92
23:56
23:57
第1437期
赏
赏
股神争霸·虚拟场
截止投注
猜虚拟指数23:57是否超过
3499.92
3501.92
3503.92
超过
奖励 x1.98
VS
不超过
奖励 x1.72
0 0 : 1 1 猜虚拟指数23:57最后一个数字
猜单双
猜范围
猜尾数
单数
奖励 x1.86
VS
双数
奖励 x1.86
我的金豆
73071
100
1000
1万

CONTENTS

Financial Technology
腾讯支付基础平台与金融应用线

开发模式的转变





React

React开发实践

开发实例

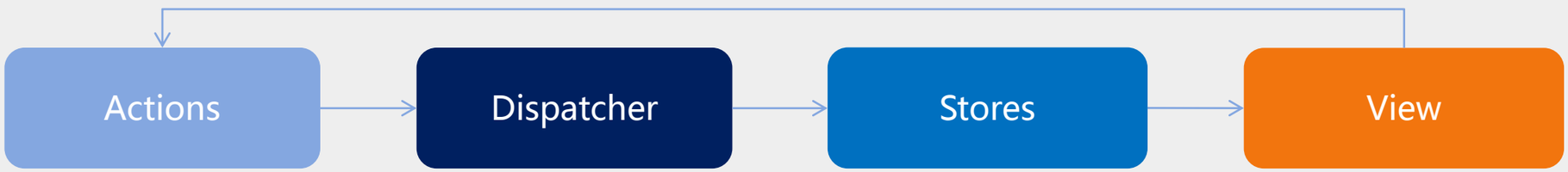
```
exports.body = React.createClass({
    componentWillMount:function(){

        et.on("lg.react.touchItem",function(e,t){
            var target = e.target;
            if($(target).parents('li').length>0){
                $(target).parents('li').toggleClass('fold-list');
            }else if($(target).parents('.guess-mod').length>0){
                $(target).parents('.mod').toggleClass('list');
            }

        }.bind(this));

        et.on("lg.react.getMatchInfo",function(e,t){
            return this.state.info;
        }.bind(this));

        if(!this.windowHeight){
            this.windowHeight = $(window).height();
        }
        et.on("lg.react.parentMinHeight", function(){
            this.setMiniHeight();
        }.bind(this));

    },
    getInitialState:function(){
        return {odds:[],odds_copy_str:"",info:{},selList:[],is_open
    },
    setInfo:function(info){
        this.setState({info:info});
        et.emit("lg.react.matchinfo",info);
        if(info.state*1 >= 2){
            et.emit("lg.react.clean"); //如果比赛结束，要清除选中态
        }
    },
    setGuessUserInfo:function(guessUserInfo){
        this.setState({guessUserInfo:guessUserInfo});
    },
```

暴露

存在哪些问题

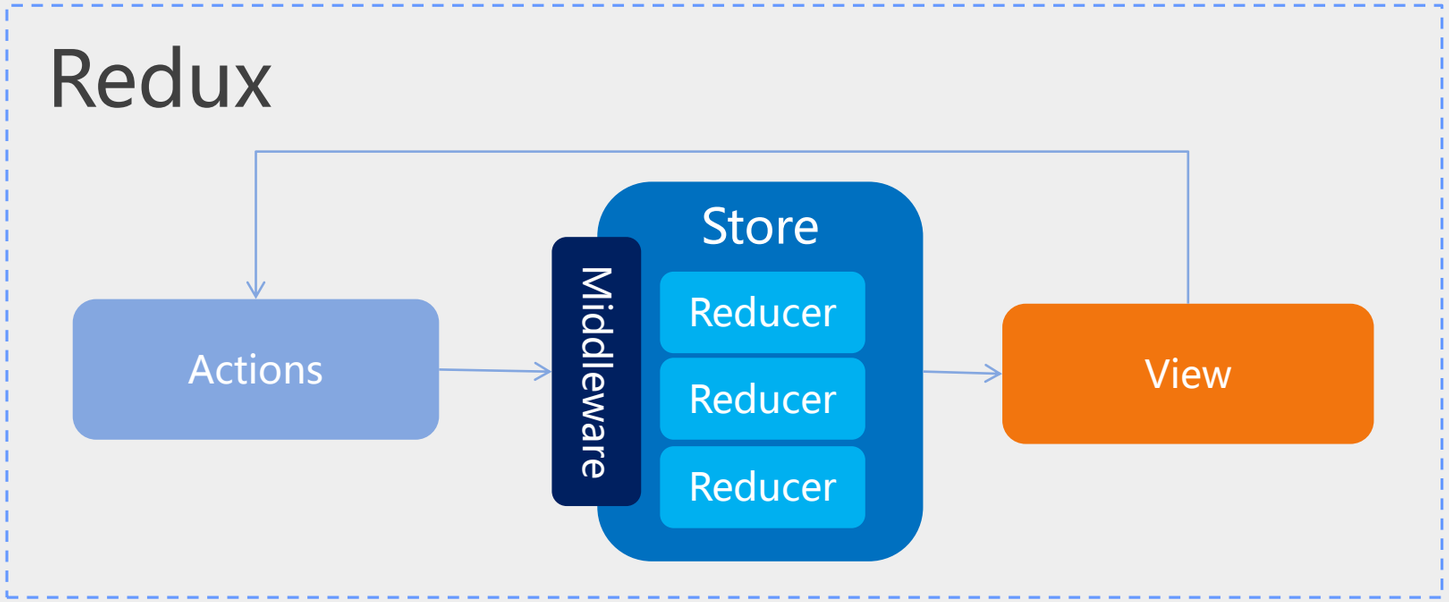
- 组件抽象程度不够
- 组件间数据流混乱

flux架构



- ✓ 单向数据流，行为可预测
- ✓ 命令-查询职责分离
- ✓ view层实现真正的组件化

Redux or Reflux 选型



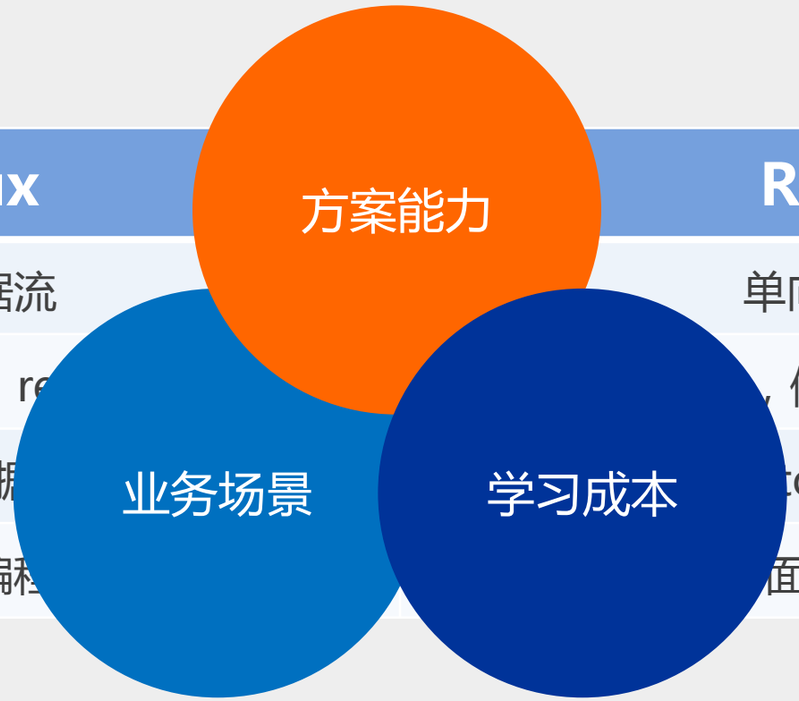
- ✓ Reducer函数
- ✓ middleWare中间件
- ✓ 单一状态树

Redux or Reflux 选型

- ✓ 移除了单例的dispatcher
- ✓ stores直接监听actions
- ✓ Action具备Promise、hooks

怎么选？

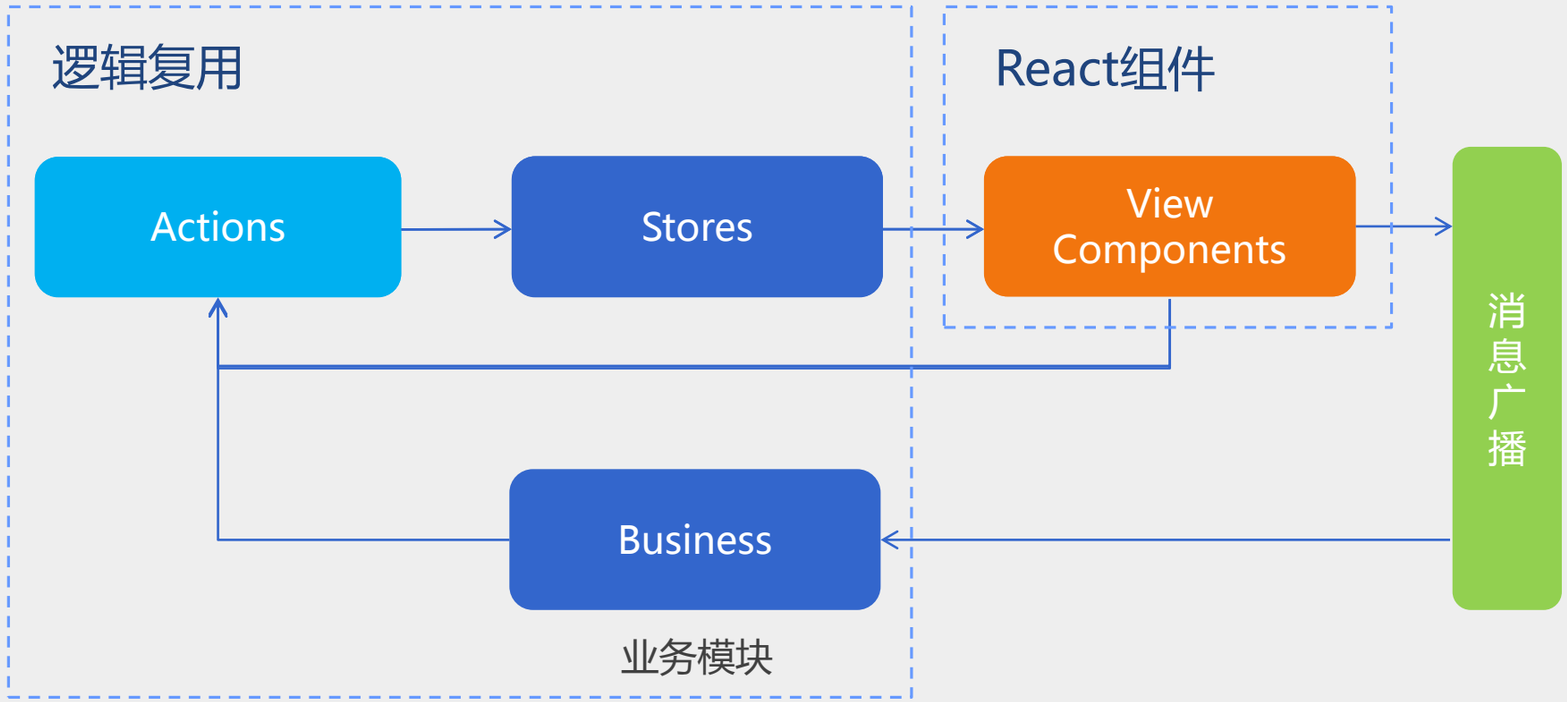
Redux		Reflux
单向数据流		单向数据流
middleWare、re		，保留store/action
单一状态树、数据		tore多状态
函数式编程		面向对象



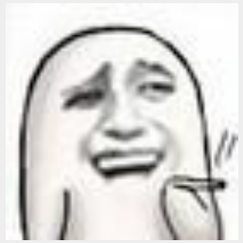
Reflux

React+Reflux实践

Reflux设计模式



- ✓ 开发模式固化
- ✓ 业务/视图职责分离
- ✓ 数据流向清晰
- ✓ 应用代码量减少



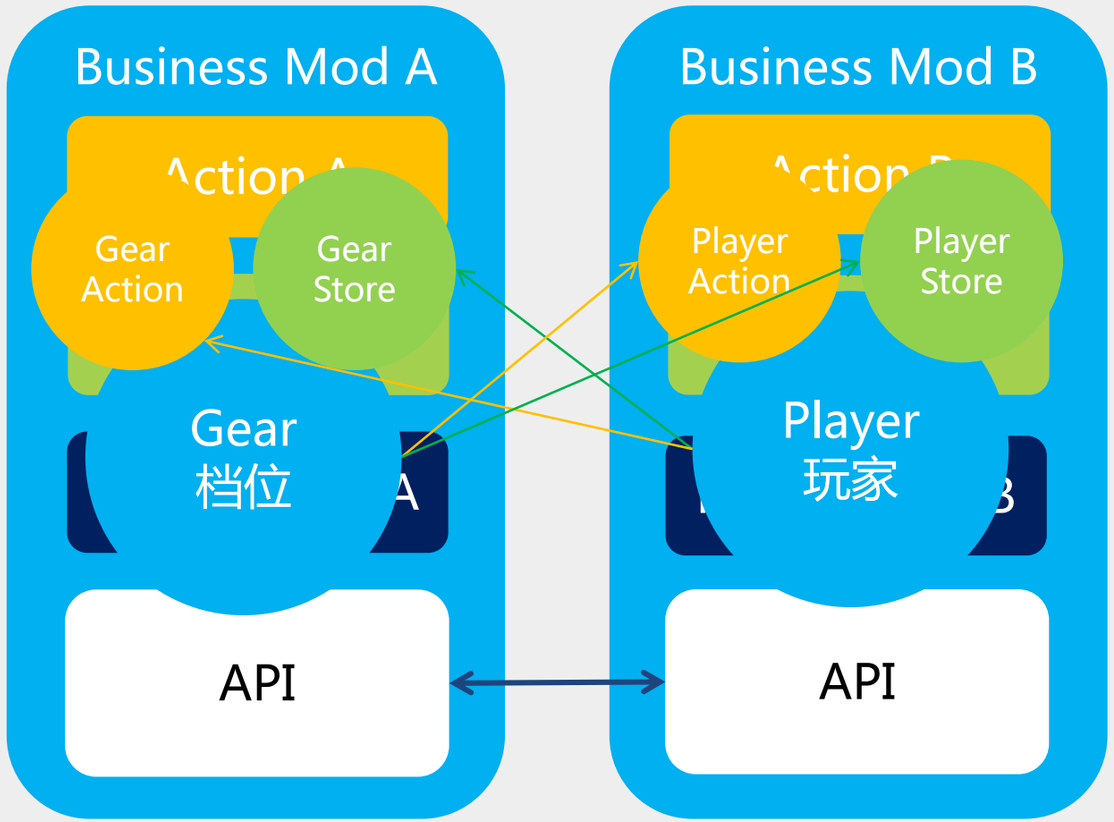
React+Reflux = 很棒的代码？

多Store/Action开发痛点

```
function Player(actions, stores) {
    var farmStore = stores.farmStore,
        playerStore = stores.playerStore;
    this.addBetChip = function() {

    }
    this.betGame = function(nameId) {
        var curCoin = farmStore.getMySelectChip().val * 1,
            myInfo = playerStore.getUserInfoByIndex(2);
        if(!myInfo.uid) {
            return;
        }
        var timeLine = farmStore.getTimeLine();

        if(timeLine.tag != 'allowBet') {
            return;
        }
        if(myInfo.money - curCoin < 0) {
            farmAction.showWarning(true,'余额不足',1000);
            et.emit('view.mallIndex.dialog');
            return;
        }
        playerAction.setInfo.trigger({
            money: myInfo.money - curCoin
        },2);
        var chipsType = farmStore.getFootData().curChips + 1;
        var chipsInfo = {
            playerIndex: 2,
            type: chipsType,
            uid: myInfo.uid,
            refName: refName,
            money: curCoin
        }
        // tableAction.addBetChips([chipsInfo], true);
    }
```

最少知识原则

减少对象之间的联系：对象之间难免产生联系，当一个对象必须引用另外一个对象的时候，我们可以让对象只暴露必要的接口（API），让对象之间的联系限制在最小的范围内。



Business收敛Store/Action

- ✓ 业务模块收敛管理，降低外部访问权限
- ✓ 模块之间通过封装API通讯
- ✓ 不暴露Store/Action给外部模块

✓抹除Store/Action的存在，不暴露过多的成员

举个例子

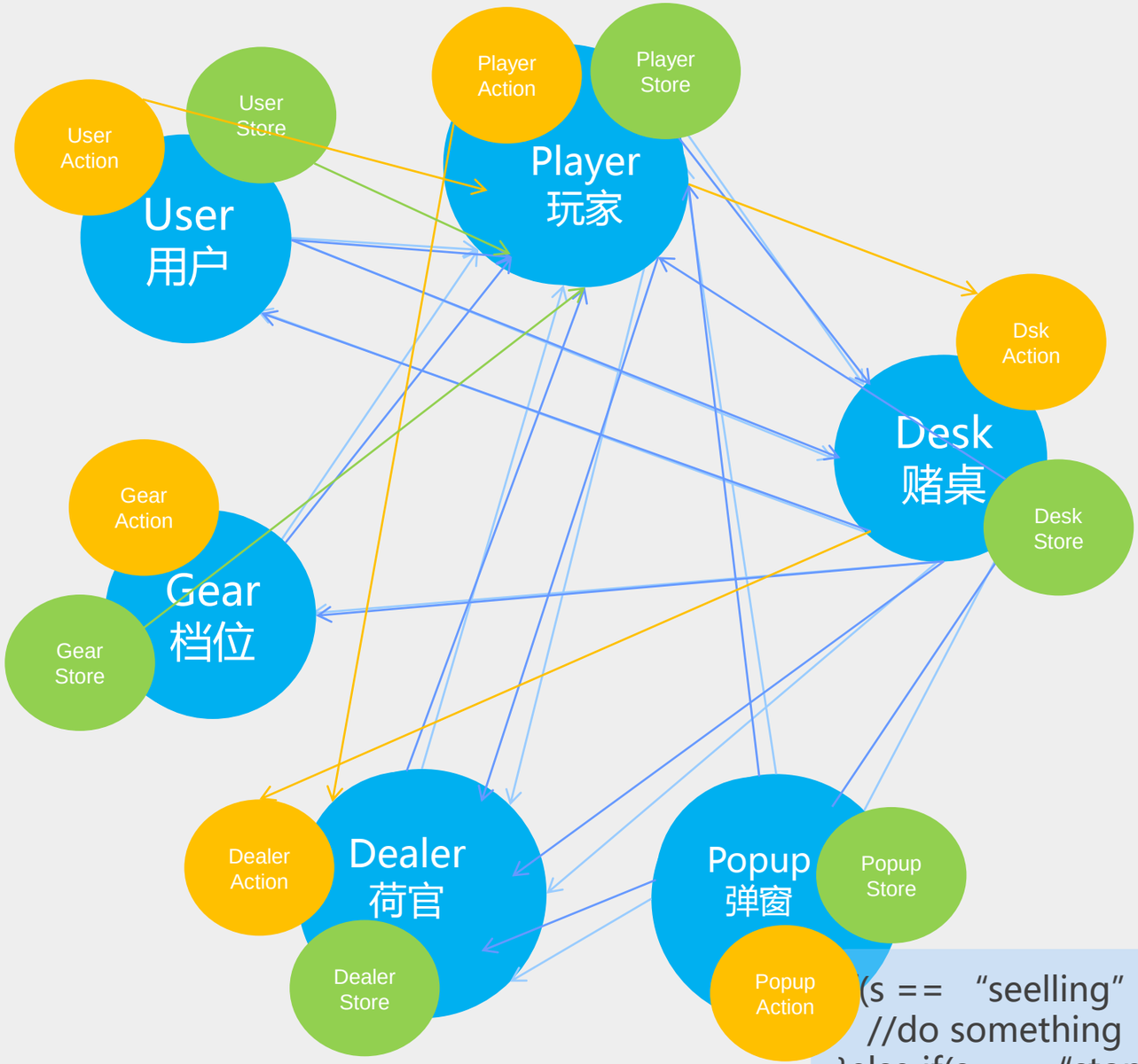
gs.user模块

```
define("gs.common.user", function(require, exports, module) {
    var _cacheThisModule_;
    var React = require("react"),
        et = require("event"),
        UI = require("lot.lotUI"),
        lotTool = require("lot.tool"),
        mdata = require('gs.common.data'),
        gstool = require('gs.common.tool'),
        userAction = require("gs.common.user.action")(),
        userStoreMod = require("gs.common.user.store");

    /**
     * 初始化
     * @param  {[type]} view     [description]
     * @param  {[type]} tplModelInitFn [具体的模板]
     * @param  {[type]} uiModel  [lot.lotUI或者你业务自定义的u
     * @return {[type]}          [description]
     */
    exports.init = function(options) {
        //............

        var userObj = {
            getUserJindou: userStore.getUserJindou,
            updateJindou: function(jindou) {
                userAction.updateJindou(jindou);
            }
        };

        return userObj;
    };
```

gs.pay模块

```
/**
 * 下单
 */
function pay(view, player, betArea, gearObj, userObj, payObj) {
    var playerAction = player.playerAction;
    payObj.isRealPay = true; //是真实投注
    payObj.payPoint = gearObj.getCurrentChipVal();
    payObj.chipMoveStartOffset = gearObj.getSelectedChipOffset(); //master投注时,
    betArea.pay(payObj, userObj.getUserJindou(), function(res, newUserJindou) {
        //下单成功回调
        if (newUserJindou && newUserJindou !== "" && !isNaN(newUserJindou)) {
            userObj.updateJindou(newUserJindou);
        }
        playerAction.addBetChip(payObj.payPoint, payObj.odds, payObj.selectOption
    });
}
```

- ✓ 业务模块封装愈加合理
- ✓ 对于调用方来说，只需要知道这个业务模块提供的API，不用care 什么store和action。知道的越少越好！
- ✓ 面向接口（抽象）编程




```
(s ==  “seelling” ){
  //do something
}else if(s ==  “stop_deal” ){
  //do something
}
```

感觉好了一点点...

待解决的主要问题

- 模块间互相依赖
- 通讯过度依赖消息
- 业务模块中混杂场景状态/业务判断
  - 状态散乱在各模块，调试困难
  - 增加状态，需要改动很多对应的业务模块

中介者Mediator模式

用一个中介对象来封装一系列的对象交互。中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互。

中介者模式——进一步解耦

中介者例子


```
//股指核心调度模块，中介者
define("gs.stock.entra", function                    rts, module) {
    exports.init = function (vie
        //lo        urce
        //                是扭转模块
                    Proces
                    模块
        //                yer.init();
        //                ser.init();
        //
        bindEvent(args)
    };
    function bindEvent(view, processObj, isVirtualStock, objs) {
        var stockMain = objs.stockMain,
            areaMain = objs.betAreaMain,
            aFyx = objs.betAreaFyx,
            objs.chart,
            objs.dealers,
            = objs.roberts,
            = objs.history,
            obj = objs.userObj,
        player = objs.player,
        popup = objs.popup,
        taskObj = objs.taskObj;

        //########                ########以下为核心
        //监听
        proces                    ockData.upd
        //监听
        processOb                 showPrizeAnimatio
        //########################以下为全局的一些事件响应########
        //盘口切换
        et.on("gs.stock.switchPankou", function (index) {});
        //打赏荷官
        et.on("stock.dealer.rewardNpc", function (pos) {});
    }
```

调度过程

```
//开奖处理
processObj.on("gs.stock.showPrizeAnimation", function(kjInfo) {
    var currentTopic = processObj.getCurrentTopic();
    var kjResult = kjInfo.drawInfo[stockMain.getCurrentBets()];
    kjResult.profit = kjInfo.profit; //将profit字段复制过去
    kjResult.fund = kjInfo.fund; //将fund字段复制过去

    //更新防沉迷信息
    fcm.update('gsstock', kjInfo.fund, kjResult.profit);

    //展示开奖结果
    stockMain.setKjResult(kjResult.detail, kjResult.result, kjResult.fund, kjResult.profit);

    //荷官播报开奖点位信息
    dealerSay(dealers, "", "result", "", { time: currentTopic.fdrawTime.substring(11, 16), po

    //开左边还是右边
    var kjPos = kjResult.result === "A" ? "left" : "right";

    //获取该发奖的荷官实例
    var prizeDealer = dealers[kjPos];

    //荷官收筹码
    prizeDealer.collectChips(function() {
        //荷官发筹码
        prizeDealer.pushChips(kjResult, function() {
            if (!isNaN(kjInfo.jindou)) {
                userObj.updateJindou(kjInfo.jindou);
            }

            if (profit > 0) {
                //荷官提示盈利
                dealerSay(dealers, "", "draw", kjPos);
            } else if (profit < 0) {
                //荷官提示亏损
                dealerSay(dealers, "", "nodraw", kjPos);
            }

            //通知开奖动画结束
            view.setTimeout(function() {
                processObj.emit(currentTopic.topicId + ".prizeAnimationEnd");
            }, 2000);
        });
    });

});
```

React+Reflux 实践总结

React

1. React只负责view，不应该有业务逻辑
2. Store只负责数据，不应该有业务逻辑
3. Store提供getApi给外部调用查询数据
4. Store最小化更新
5. 需要业务模块承载业务逻辑

✓不被React/Reflux绑架代码

Business

1. 中介者模式减少业务模块间的耦合
2. 业务模块收敛

   Store、Action只在业务模块内访问

   面向接口编程，行为通俗易懂

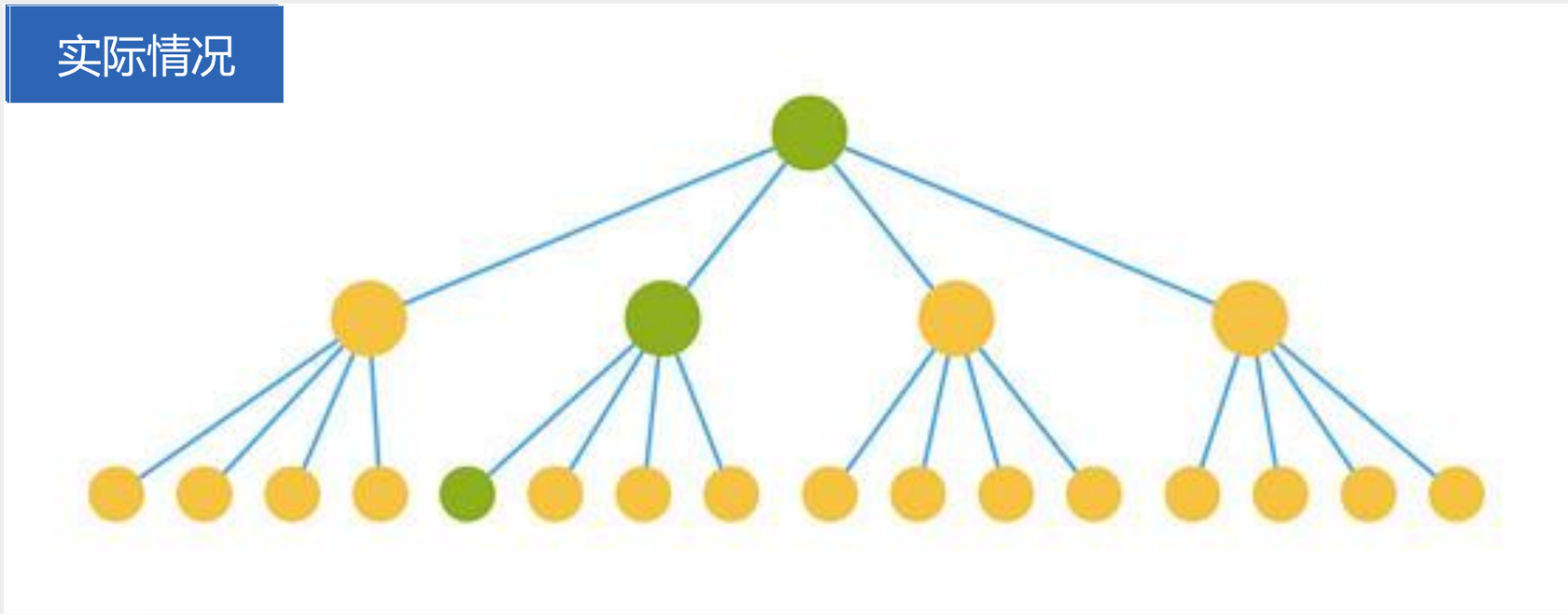
   解决了全局消息的滥用

✓引入设计模式，解耦提优

CONTENTS

React性能调优

实际情况

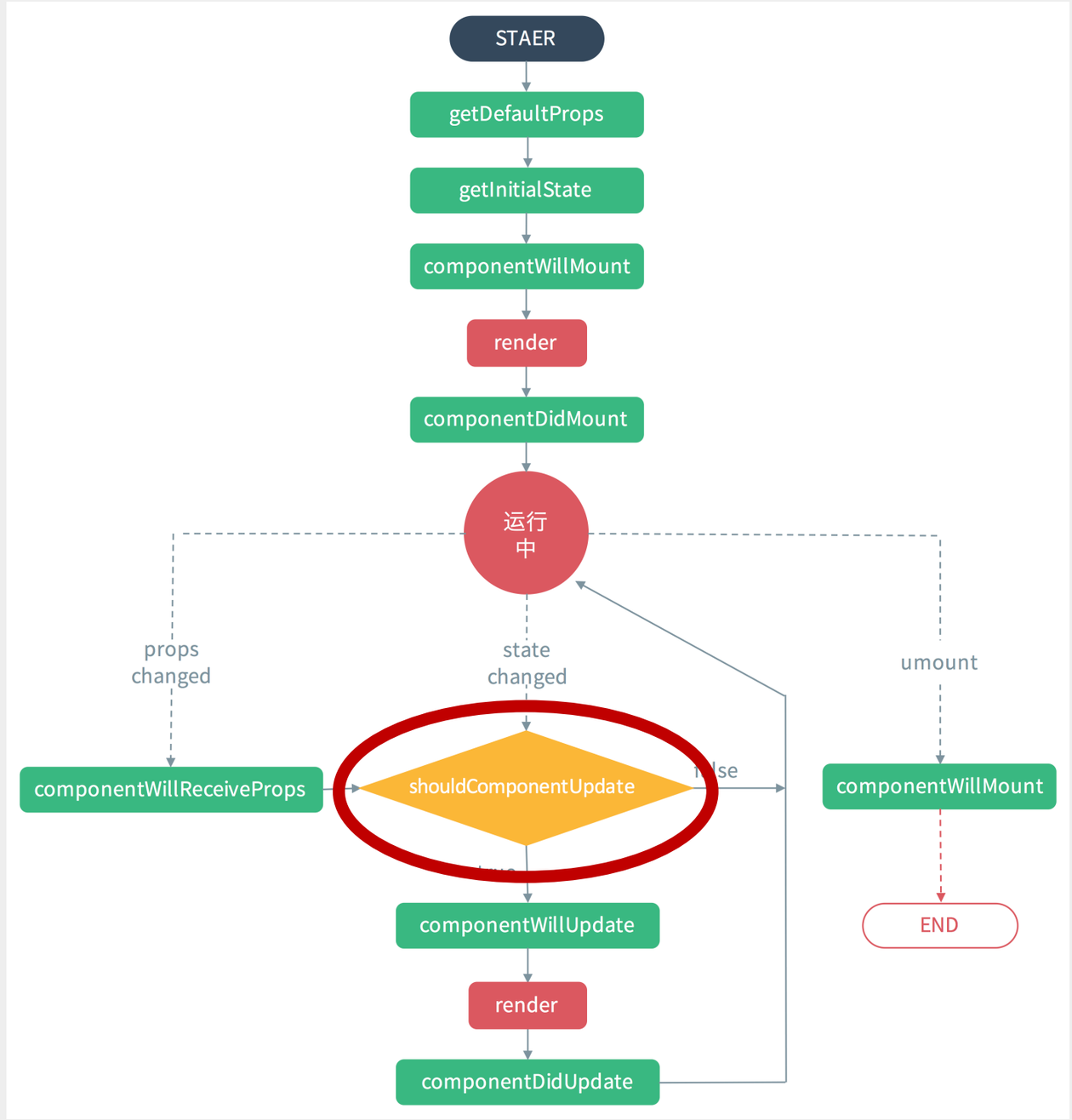


✓如何减少多余的Diff？

- ✓ 利用SCU来减少不必要的diff
  - ✓ 无需更新的组件直接return false

```
var Header = React.createClass({
    shouldComponentUpdate: function(nextProps, nextState) {
        return false;
    },
    render: function() {
        return (
            <header className="container-hd">
                <nav className="tab-mod">
                  <ul className="tab-list">
                    <li className="tab-item">
                      <button>全部</button>
                    </li>
                    <li className="tab-item">
                      <button>待开奖</button>
                    </li>
                    <li className="tab-item">
                      <button>中奖</button>
                    </li>
                  </ul>
                </nav>
            </header>
        )
    }
})
```

- ✓ 会发生状态更新的组件进行数据比较决定是否需要更新



数据比较data Compare

- ✓ 对组件的新老数据做比较
  - ✓ 原始数据类型
  - ✓ 引用数据类型

原始数据类型	引用数据类型
JSON.stringify(a) == JSON.stringify(b)	Immutability Helpers
React.PureRenderMixin	Immutable.js
Lodash.isEqual	代码规范来规避

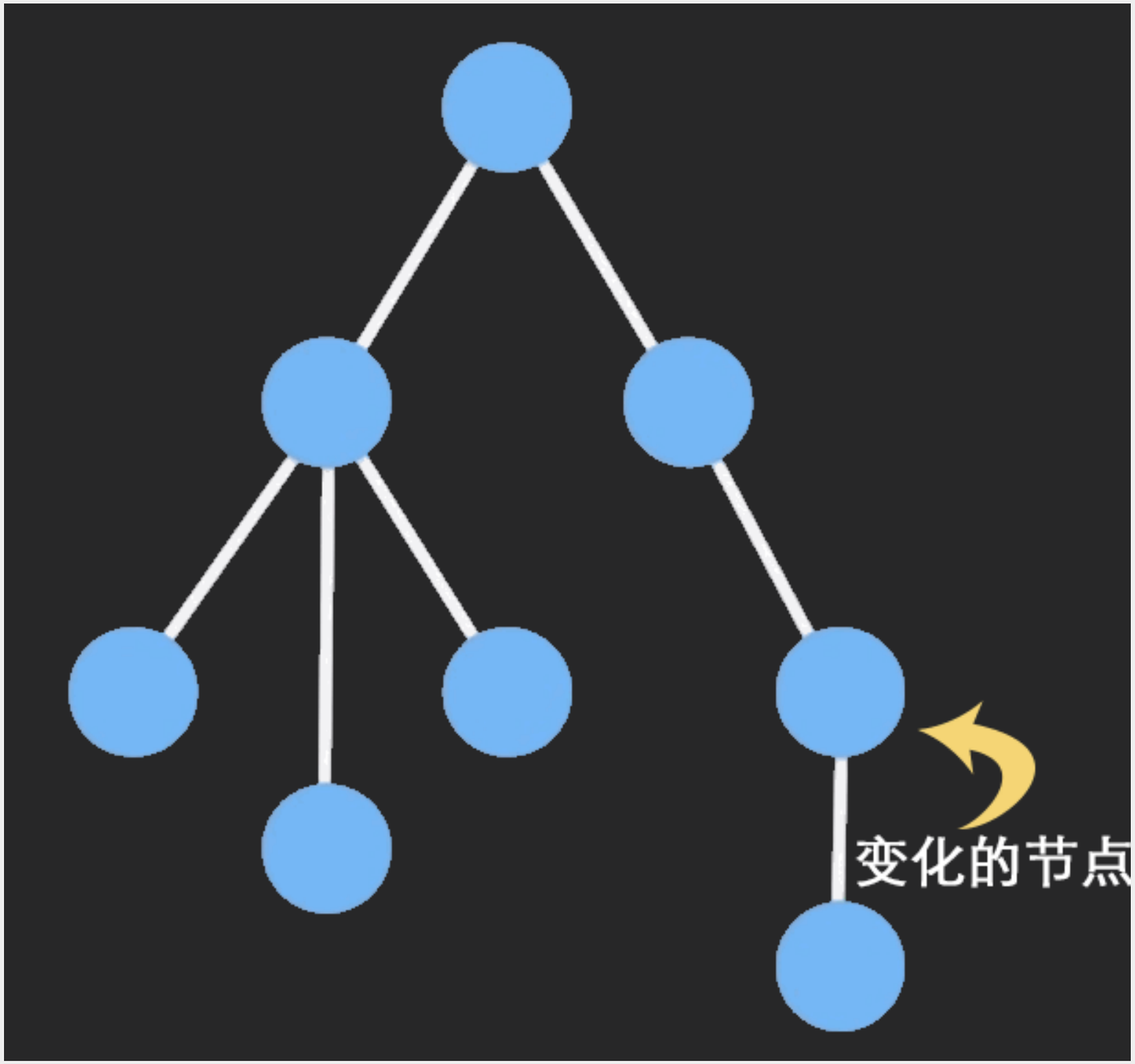
Compare方案

IMMUTABLE

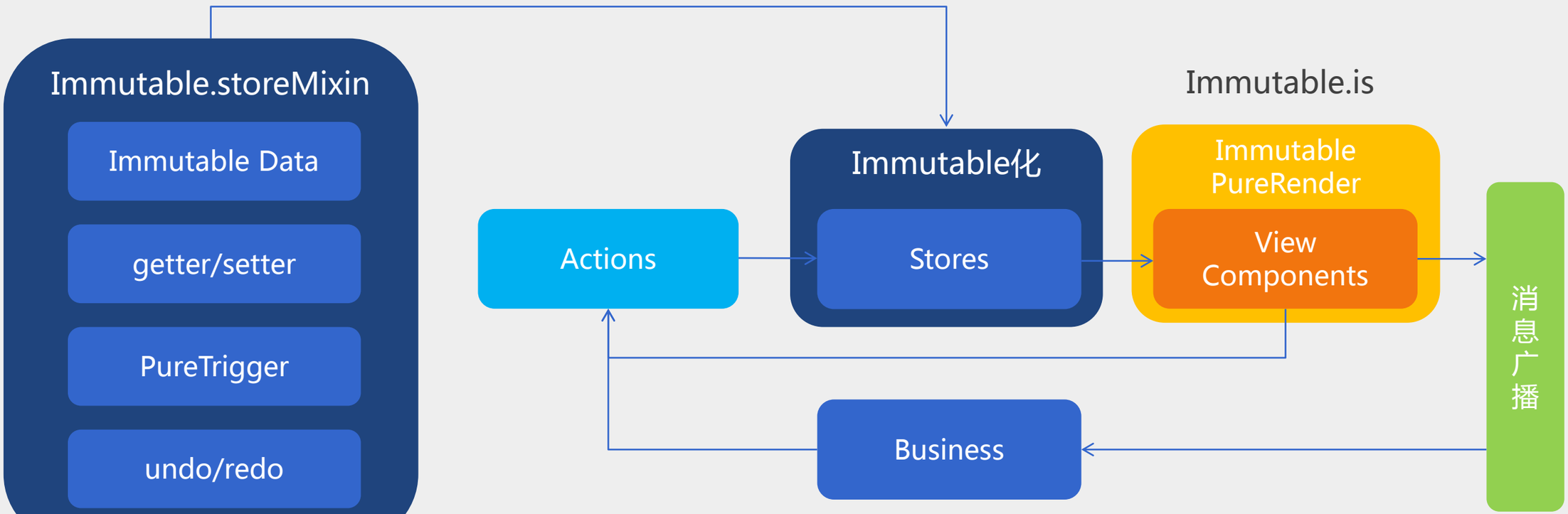
- 持久化数据结构
- 结构共享

提供的Compare方案：

- === 内存地址比较
- Immutable.is 值比较



Reflux + Immutable



开发实例及成效

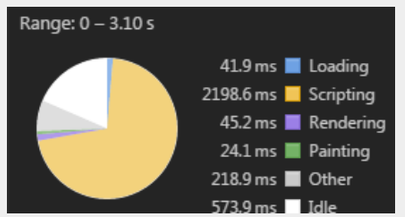
```
var option = getInitOption(param);

var store = reflux.createStore({
    listenables: [actions],
    getInitialState: function () {
        return this.initState(option);
    },
    mixins: [ImmStoreMixin],
    //初始化数据（第一次拉取接口，会带有所有数据）
    onInitData: function (json) {
        var p = formatPlayNumList(json.playNumList);
        console.info(p);
        this.set({
            'isFirst': false,
            'gameName': json.gameName,
            'playNumList': imm.fromJS(formatPlayNumList(json.playNumList)),
            'guessList': imm.fromJS(json.guessList)
        });
    },
    //设置投注信息
    onSetBetInfo: function (betInfo) {
        this.set({
            'betInfo': imm.fromJS(betInfo)
        });
    },
    //设置赛事信息
    onSetGuessList: function (guessList) {
        this.set({
            'isFirst': false,
            'guessList': imm.fromJS(guessList)
        });
    },
    //设置赛事信息
    onSetPlayNumList: function (playNumList) {
        this.set({
            'isFirst': false,
            'playNumList': imm.fromJS(formatPlayNumList(playNumList))
        });
    }
});
```

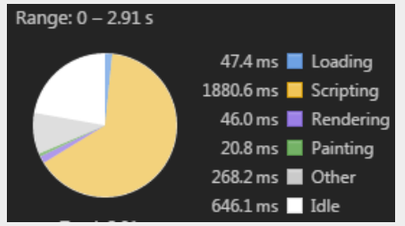
FIFA滚球项目

Before 

After 

- 30秒渲染次数 降低 82%

Before 

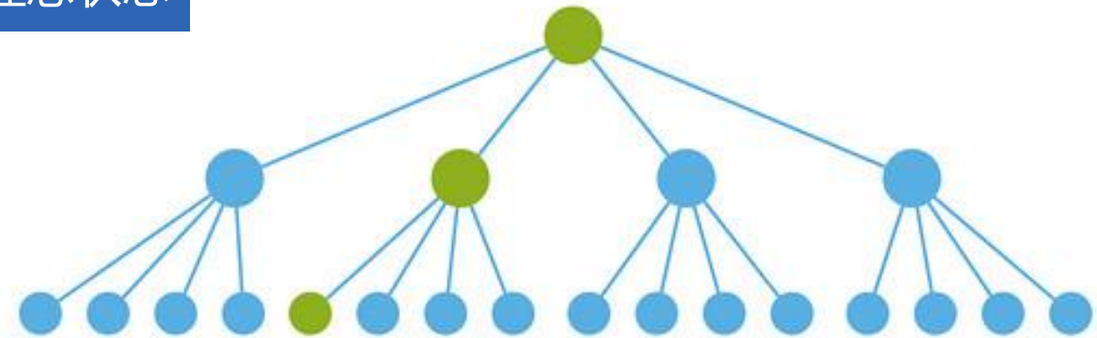
After 

- 首屏渲染Scripting 降低14%

React性能调优总结

- ✓ 不需要更新的组件return false
- ✓ 从根本上规避数据引用等等操作
- ✓ 根据数据类型来选择Compare方案
- ✓ 利用key属性来触发insertBefore移动节点
- ✓ 使用 React.addons.Perf 来做性能分析

理想状态

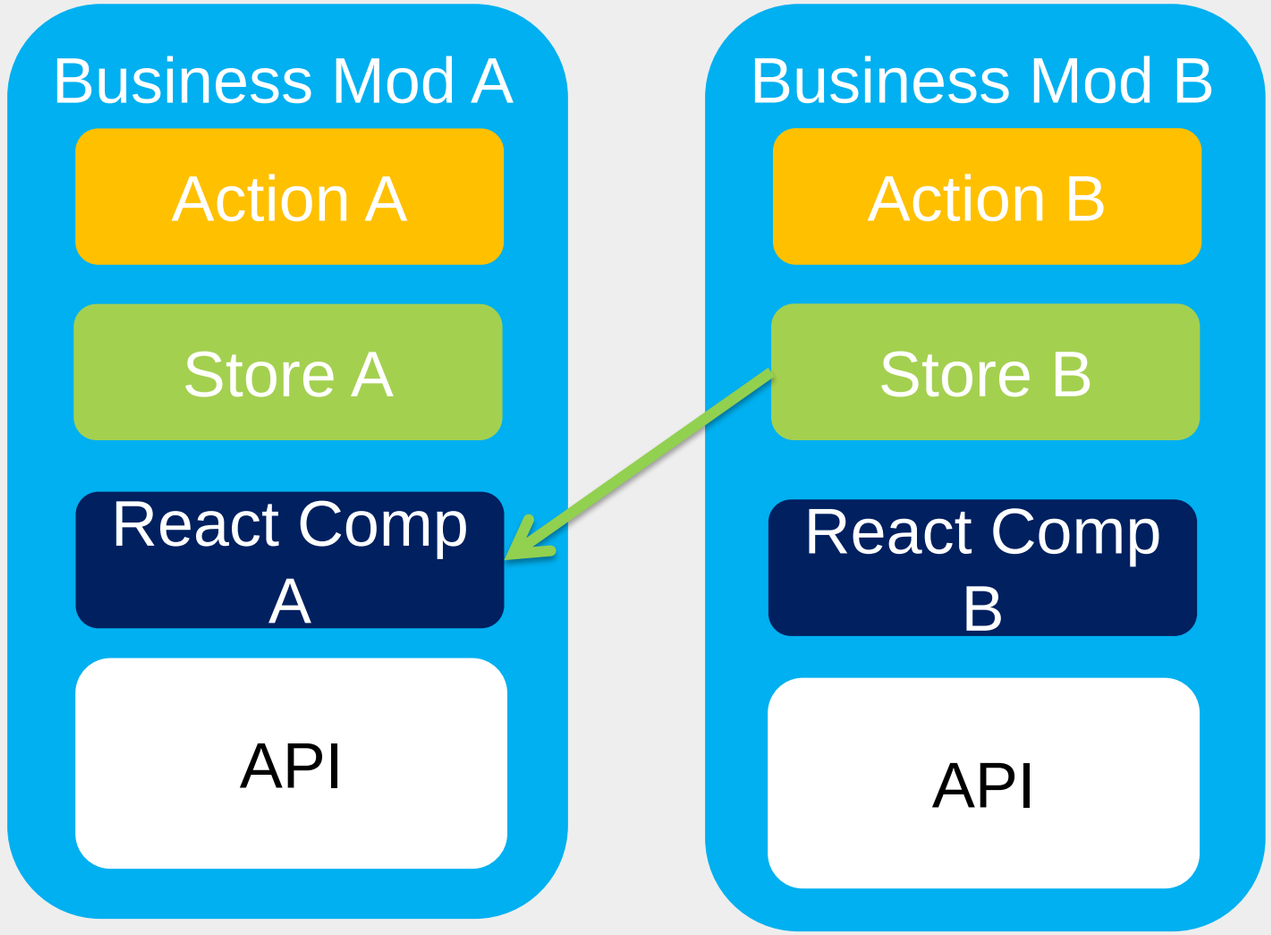


✓PureRender

Q&A

THANK YOU

附录1：业务模块之间的Store依赖

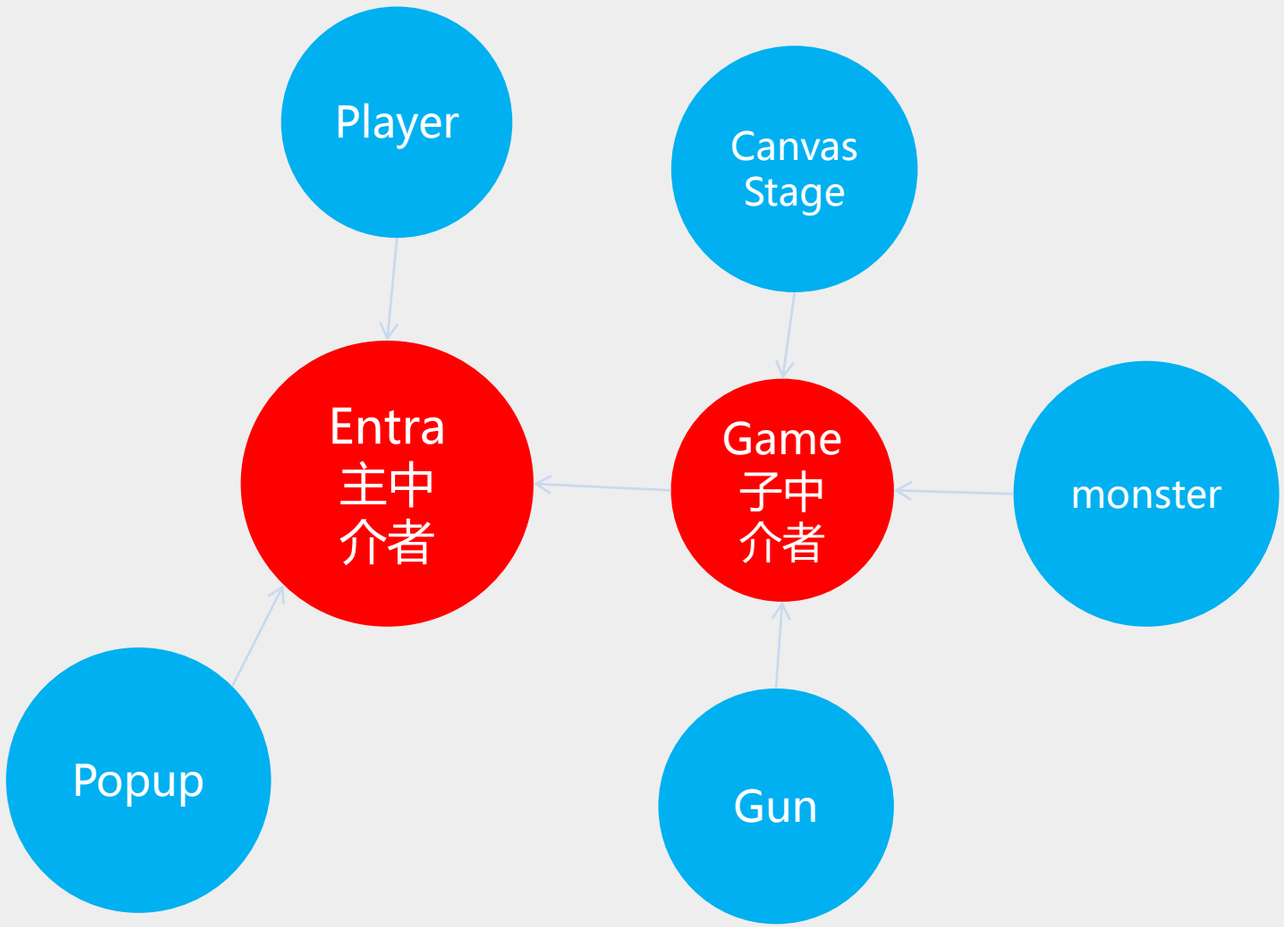


A模块的React模板依赖B模块的Store数据，该怎么办？

React Comp A需要connect StoreB

封装是把双刃剑，理论上A模块不允许知道B模块的Store B 。但实际情况下，可以做一些妥协

附录2：中介者臃肿庞大的解决方案



- entra是一个中介者,负责整体app的运转
- game子中介者负责游戏逻辑，包括了游戏的状态（类似于股指的状态模块），提供回调和Api。而它本身也算一个中介者（canvas starge 和 monster, gun模块互相之间相对独立，符合最少知识原则）
- 其他的业务模块，存在一些必要的联系，但各自的逻辑都比较简单