美团网
meituan.com

Thinking in React

美团技术团队 陈益国

w3ctech 美团 React 专场

目录

拥抱未来

React or Web Components

Hello world of Web Component

```
<hello-world></hello-world>
```

Hello world of React

```
<HelloWorld></HelloWorld>
```

Web Components

- 封装性（Shadow DOM）
- 可移植性（原生支持）

React

- 高性能（Vitural DOM）
- 松耦合

React ~~or~~ + Web Components

同构

“一次编写，多处运行”

Client + Server Rendering

- 更好的用户体验
- 可维护性
- SEO

ES6(2015) & ES7(2016)

Classes、static、Arrow Function

```
class HelloWorld extends React.Component {
    constructor (props) {
        super(props);
        this.state = { name: 'mxTeam' };
    }
    sayHello = () => {
        console.log(`Hello ${this.state.name}`);
    }
    static defaultProps = {
        content: 'World!'
    };
    render () {
        return (
            <div onClick={this.sayHello}>Hello {this.props.content}</div>
        )
    }
};
```

设计原则

JSX

Render a HTML tag

```
var myDivElement = <div className="foo" />;
```

Render a Component

```
var MyComponent = React.createClass({/*...*/});
var myElement = <MyComponent someProperty={true} />;
```

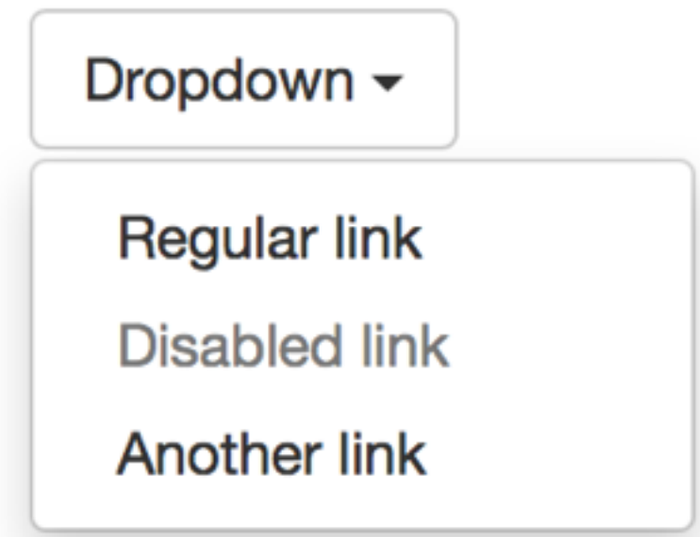
状态机

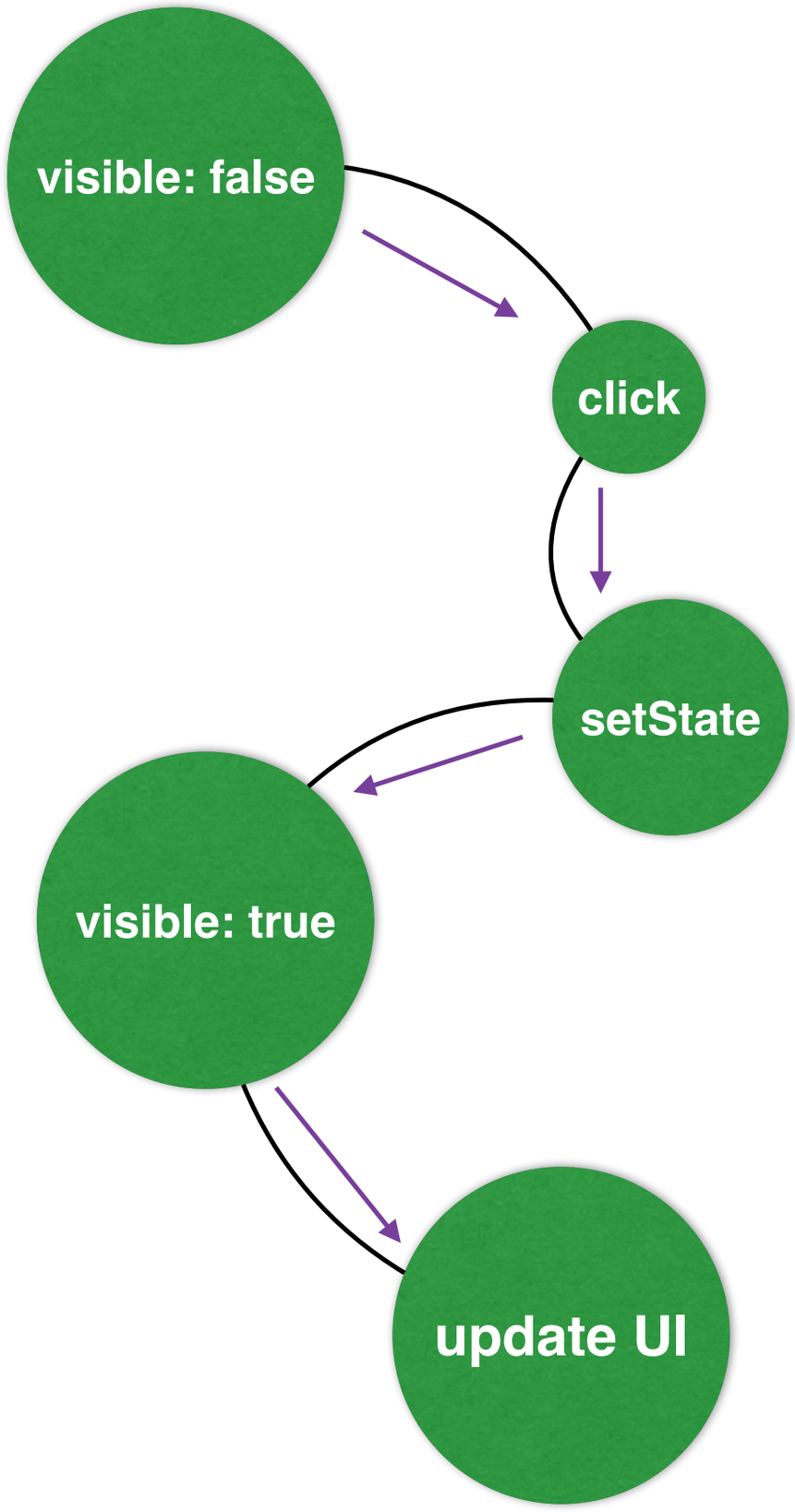
“抛弃繁琐的*DOM*操作，仅需维护数据的状态”

一个下拉菜单的两种状态

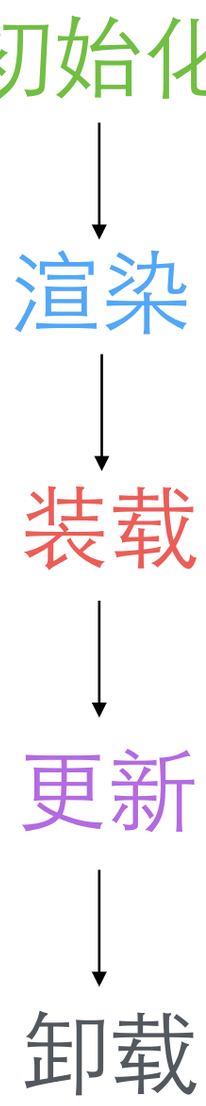
state: 收起

state: 展开




visible: false
click
setState
visible: true
update UI

生命周期



设计原则

- 化繁为简
- 可预测性
- 单向数据流

高性能

当我们谈论 *JavaScript* 慢的时候，我们都在谈论些什么

Virtural DOM

Create ReactElement(Virutal DOM)

```
var root = React.createElement('div');
```

```
var root = (
    <div></div>
);
```

4 个简单的属性，没有 prototype

```
ReactElement {
    type: 'div',
    ref: null,
    key: null,
    props: Object
}
```

通过 React.render 将 ReactElement 渲染成常规的 DOM

Diff

同层级 diff

node类型 `<div />` `<span />`

component类型 `<Header />` `<Content />`

attribute `<div foo="bar" />` `<div foo="biz" />`

style `<div style={{color:'#666'}} />`

`<div style={{fontWeight:'bold'}} />`

列表 diff

```
<ul>
    <li>a</li>
    <li>b</li>
    <li>d</li>
    <li>e</li>
<ul>
```

```
<ul>
    <li>a</li>
    <li>b</li>
    <li>d</li>
    <li>e</li>
    <li>c</li>
<ul>
```

```
<ul>
    <li key="a">a</li>
    <li key="b">b</li>
    <li key="c">c</li>
    <li key="d">d</li>
    <li key="e">e</li>
<ul>
```

key is key

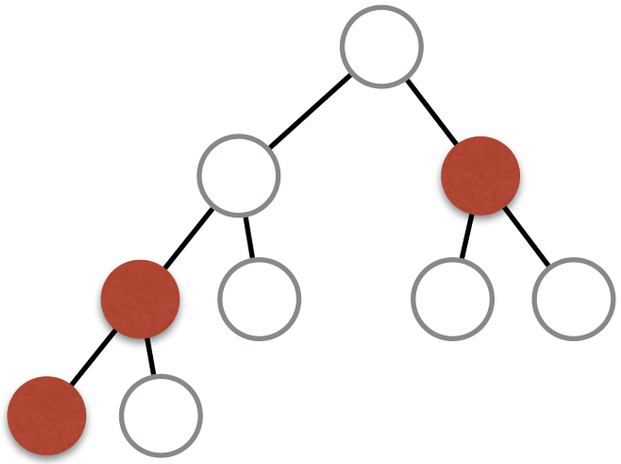
Render

为什么 setState 是一个异步操作

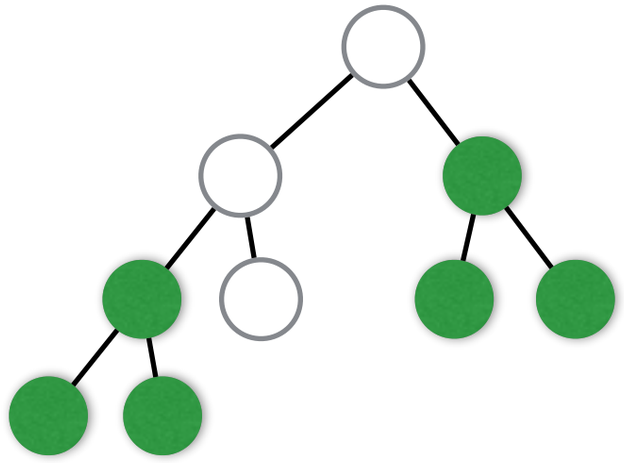
```
this.state = {
    foo: null
};

...

this.setState({
    foo: 'bar'
});

console.log(this.state.foo); // null
```

批量操作

Render 流程



标记 dirty



重新 render 子树

shouldComponentUpdate

```
shouldComponentUpdate: function (nextProps, nextState) {
    // return nextProps.id === this.props.id;
    // return nextState.id === this.state.id;
}
```

更优的 事件代理

- 绑定至文档的根节点
- 仅迭代组件的根节点

生态圈

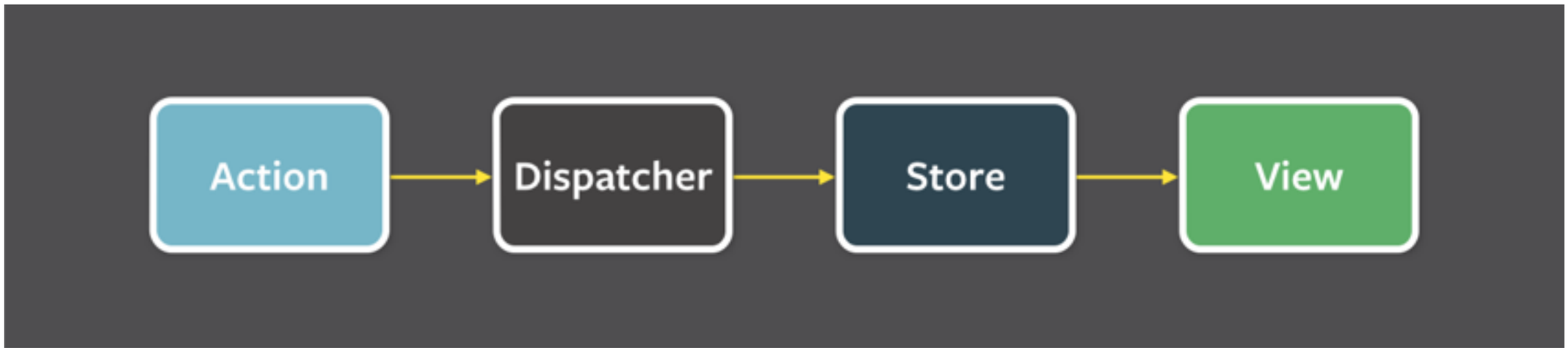
双向绑定 VS 单向数据流

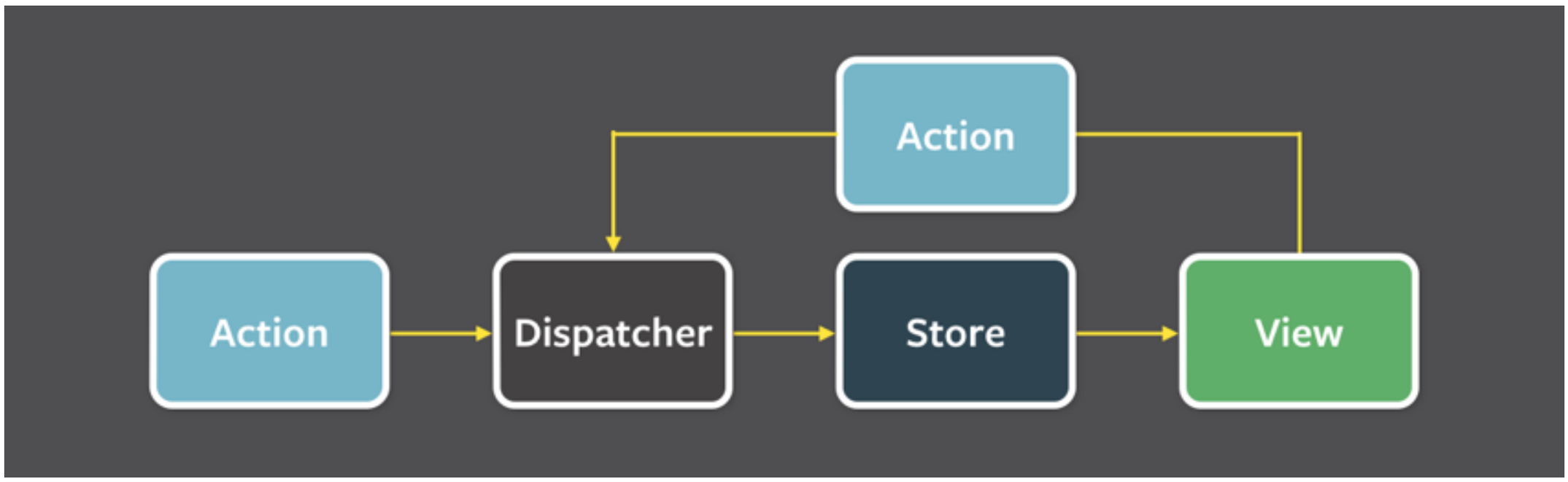
- 脏检查 & Object.observe
- 复杂度高，数据不可预测

- setState & immutable
- 简单清晰，数据可预测

Flux

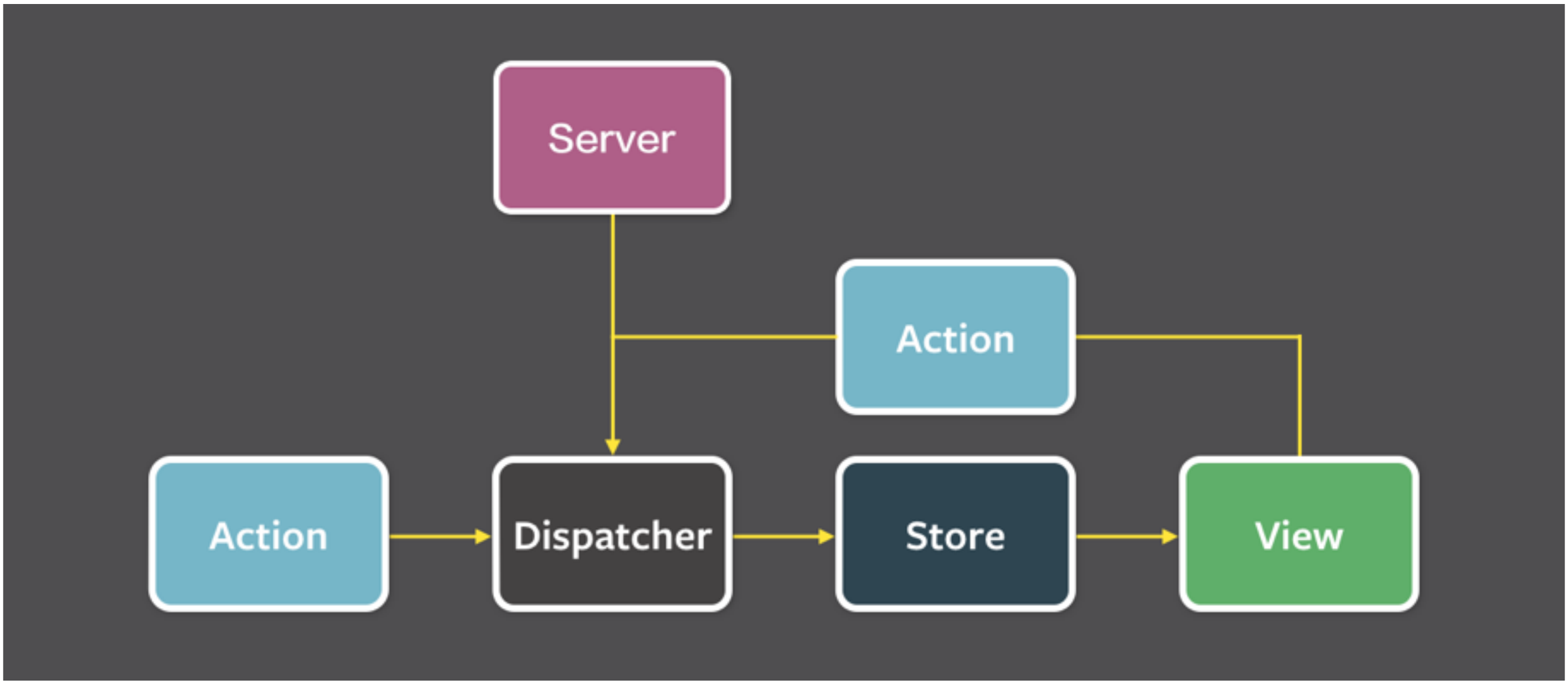
Flux 的流程结构


Action
Dispatcher
Store
View


Action
Action
Dispatcher
Store
View

- **Action** 用户交互、数据交互都是一个Action
- **Dispatcher** 分发动作给 Store
- **Store** 数据存储中心，响应 Action，对数据进行修改
- **View** 监听数据变化的事件，从 Store 获取数据，触发 Action

- Flux
- Reflux
- Redux
- ...


Server
Action
Action
Dispatcher
Store
View

```
class FriendInfo {
    statics = {
        queries: {
            user () {
                return graphql `
                    User {
                        name,
                        mutual_friends { count }
                    }
                `;
            }
        }
    };

    render () {
        var user = this.props.user;

        return (
            <div>
                <span>{user.name}</span>
                <span>{user.mutual_friends.count} mutual friends</span>
            </div>
        );
    }
};
```

跨平台／终端

Thanks & End

欢迎关注美团技术团队获取更多优质技术内容