React.js شروع و درک مفاهیم

React چیست؟

Reactیک کتابخانه UI توسط فیسبوک ساخته شده است برای راحتی و ایجاد تعامل در استفاده مجدد اجزاء است. فیسبوک در ساخت محصول همچون Instagram از آن استفاده میکند.

یکی از نقاط منحصر به فرد آن این است که، نه تنها کارها را در سمت سرویس گیرنده انجام می دهد، بلکه در سمت سرور هم کار می کند، و اینها در کنار هم بصورت سازگار کار می کنند.

برای کار با اجزای صفحه از مفهومی به نام Virtual DOM استفاده می کند که این امکان را میدهد تا Node انتخابی DOM را بر اساس تغییرات اخیر ارائه دهد که باعث کمترین بروزرسانی در DOM خواهد شد.

Virtual DOM چگونه کار می کند؟

تصور کنید شی دارید که از افراد اطراف حد مدل سازی کرده اید. این شی هر اموالی که یک شخص واقعی می تواند داشته باشید را دارد، و وضعیت فعلی شخص را منعکس می کند. این دقیقا چیزی است که React با DOM ها می کند.

حال به این فکر کنید که شی ما زمان دارد و برخی از تغییرات در آن انجام شده است. به عنوان مثال، سبیل، برخی از عضلات دو سر و چشم ... اضافه شده است. در دنیایReact ، وقتی این تغییرات را اعمال می کنیم، دو چیز اتفاق می افتد. اولی، Reactاز الگوریتم diffing برای شناسایی تغییرات استفاده می کند. دومی، اصلاح کردن، جایی که DOM ها با نتایج متفاوت اصلاح می شوند.

به جای در نظر گرفتن یک شخص واقعی و ساخت آنها از ابتدا، تنها لازم است صورت و بازوها را تغییر دهید. به این معنی که اکر شما متنی را در ورودی داشته باشید، تا زمانی که گره مادر برای اصلاح برنامه ریزی نشده است، متن دست نخورده باقی خواهد ماند.

بخاطر اینکه React از DOM جعلی استفاده می کند نه واقعی، ما می توانیم آن DOM جعلی را روی سرور نمایش دهیم.

شروع

راه اندازی React را همانند دانلود آن از React Starter Kit آسان است. همچنین می توانید بر روی مثالی که در JSFiddleقرار داده شده است آزمون و خطا کنید.

راه اندازی صفحه

هنگامی که صفحه را راه اندازی کردید، حال می خواهید که react.js و JSXTransformer.js شامل صفحه شود، و اجزاء خود را در یک اسکریپت node با تنظیم type به text/jsx بنویسید:

```
<!DOCTYPE html>
<html>
  <head>
    <script src="build/react.js"></script>
    <script src="build/JSXTransformer.js"></script>
  </head>
  <body>
    <div id="mount-point"></div>
    <script type="text/jsx">
      // React Code Goes Here
    </script>
  </body>
</html>
<!DOCTYPE html>
<html>
  <head>
    <script src="build/react.js"></script>
    <script src="build/JSXTransformer.js"></script>
  </head>
  <body>
```

```
    <div id="mount-point"></div>
    <script type="text/jsx">
      // React Code Goes Here
    </script>
  </body>
</html>
```

درReact ، اجزاء در یک عنصر قرار می گیرند، بنابراین در این مثال ما از #mount-point به عنوان در برگیرنده ی مادر استفاده کرده ایم.

در حالی که این یک راه آسان برای شروع کار است، هنگامی که زمان ساخت یک چیز واقعی فرا برسد، احتمالا ایده خوبی است که Browserify یا webpack استفاده کنیم و مجموعه اجزاء خود را در فایل های جداگانه قرار دهیم.

مبانی

بلوک های React اجزاء یا components نامیده می شود، بیایید یکی از آنها را بنویسیم:

```
<script type="text/jsx">
    /** @jsx React.DOM */
    React.renderComponent(
        <h1>Hello, world!</h1>,
        document.getElementById('myDiv')
    );
</script>
<script type="text/jsx">
    /** @jsx React.DOM */
    React.renderComponent(
        <h1>Hello, world!</h1>,
        document.getElementById('myDiv')
    );
</script>
```

اگر تا به حال یکی از اینها را قبلا مشاهده نکرده اید، شاید تعجب کنید که در حال حاظر چجور جادوگری در حال وقوع است.

JSX

آن را JSX می نامند، و یک تبدیل سینتکس XML جاوا اسکریپت است. این به شما اجازه می دهد که تگ های HTML را در جاوا اسکریپت بنویسیم. شما تنها به نمایندگی شی XML می نویسید.

برای تگ های منظمHTML ، صفت class برای className است و صفت for برای htmlFor است. در JSX بخاطر اینکه این کلمات در جاوا اسکریپت از پیش رزور شده است. برای تفاوت های بیشتر می توانید اینجا را مشاهده کنید.

حالا شما نیاز به استفاده از JSX ندارید، در اینجا سینتکسی را مشاهده می کنید که بدون آن انجام می شود:

```
/** @jsx React.DOM */
React.renderComponent(
  React.DOM.h1(null, 'Hello, world!'),
  document.getElementById('myDiv')
);
/** @jsx React.DOM */
React.renderComponent(
  React.DOM.h1(null, 'Hello, world!'),
  document.getElementById('myDiv')
);
```

می توانید اطلاعات بیشتر درباره پشتیبانی از عناصر را در اینجا مشاهده کنید.

در اولین تکه کد ما، آیا متوجه /* @jsx React.DOM **/ در بالای اسکریپت شده اید؟ این مهم است، به React می گوید که ما از JSX استفاده می کنیم و این نشانه گذاری ها نیاز به تبدیل شدن دارند، بنابراین شما باید آن را هنگام استفاده از JSX به کد اضافه کنید.

اجزاء یا Components

وقتی از متد renderComponent بالا استفاده می کنیم، اولین آرگومان اجزایی است که می خواهیم نمایش داده شود، و دومین آرگومان، گره ی DOM است که می خواهیم اجزاء در آن محل قرار گیرد. ما می توانیم با استفاده از متد createClassبرای ساخت اجزاء سفارشی استفاده کنیم. شی مشخصات را به عنوان آرگومان قبول می کند. بیایید یکی از آنها را درست کنیم:

```
var MyComponent = React.createClass({
  render: function(){
    return (
      <h1>Hello, world!</h1>
    );
  }
});
var MyComponent = React.createClass({
  render: function(){
    return (
      <h1>Hello, world!</h1>
    );
  }
});
```

بعد از اینکه کلاسی را ایجاد کردیم حال می توانیم آن را در سند خود بصورت زیر نمایش دهیم:

```
React.renderComponent(
   <MyComponent/>,
   document.getElementById('myDiv')
);
React.renderComponent(
   <MyComponent/>,
   document.getElementById('myDiv')
);
```

جالب است، نه؟

Props

هنگامی که اجزاء خود را تعریف کردی، می توانیم خواصی به نام props را اضافه کنیم. این خواص در component ما به عنوان this.props قابل دسترس است و می توانیم در متد نمایش برای نمایش داده ها بصورت پویا استفاده کنیم:

```
var MyComponent = React.createClass({
   render: function(){
      return (
         <h1>Hello, {this.props.name}!</h1>
      );
   }
});

React.renderComponent(<MyComponent name="Handsome" />, document.getElementById('myDiv'));
var MyComponent = React.createClass({
```

```
  render: function(){
    return (
      <h1>Hello, {this.props.name}!</h1>
    );
  }
});

React.renderComponent(<MyComponent name="Handsome" />, document.getElementById('myDiv'));
```

Spacs، Lifecycle و State

متد render تنها تنظیمات مورد نیاز برای ایجاد یک component است، اما چندین متد lifecycle و specs وجود دارد که می تواند مفید باشد هنگامی که شما در واقع می خواهید component هر کاری را انجام دهد.

متد هایLifecycle

componentWillMountیک بار قبل از render شدن در سرویس گیرنده و سرور اجرا می شود.

componentDidMountفقط یک بار بعد از render شدن در سمت سرویس گیرنده اجرا می شود.

shouldComponentUpdateمشخص کردن مقدار بازگشت اینکه آیا component باید به روز رسانی شود یا نه.

componentWillUnmountاستناد قبل از پیاده کردن جزء.

Specs

getInitialStateبازگشت مقدار اولیه.

getDefaultProps تنظیم شکست مقدارهای props اگر props عرضه نشده باشد.

mixinsآرایه از از شی ها، که در گسترش قابلیت های کامپوننت جاری استفاده می شود.

موارد بسیاری هم وجود دارد که می توانید در اینجا مشاهده کنید.

State

هر کامپوننتی دارای یک شی state و یک شی props است State .با استفاده از متد setState تنظیم می شود.
فراخوانی setState باعث به روز رسانی UI می شود و نون و پنیر تعامل با React است. اکر بخواهیم یک حالت اولیه قبل از هر گونه تعامل رخ دهد می توانیم از متد getInitialState استفاده کنیم. در زیر حالت component خود را نشان می دهیم:

```
var MyComponent = React.createClass({
   getInitialState: function(){
      return {
         count: 5
      }
   },
   render: function(){
      return (
         <h1>{this.state.count}</h1>
      )
   }
});
var MyComponent = React.createClass({
```

```
  getInitialState: function(){
    return {
      count: 5
    }
  },
  render: function(){
    return (
      <h1>{this.state.count}</h1>
    )
  }
});
```

رویدادها (Events)

ها می توانند متدها را اجرا کنند. کد زیر اجازه می دهد یک شمارشگر با component همچنین دارای یک سیستم رویداد است. رویدادها به عنوان خواص وابسته به React استفاده از رویدادها ایجاد کنیم:

```
/** @jsx React.DOM */

var Counter = React.createClass({
 incrementCount: function(){
  this.setState({
   count: this.state.count + 1
  });
 },
 getInitialState: function(){
  return {
   count: 0
  }
```

```
  },
  render: function(){
    return (
      <div class="my-component">
        <h1>Count: {this.state.count}</h1>
        <button type="button" onClick={this.incrementCount}>Increment</button>
      </div>
    );
  }
});

React.renderComponent(<Counter/>, document.getElementById('mount-point'));
/** @jsx React.DOM */

var Counter = React.createClass({
  incrementCount: function(){
    this.setState({
      count: this.state.count + 1
    });
  },
  getInitialState: function(){
     return {
       count: 0
     }
  },
  render: function(){
    return (
      <div class="my-component">
        <h1>Count: {this.state.count}</h1>
        <button type="button" onClick={this.incrementCount}>Increment</button>
      </div>
```

```
    );
  }
});
```

```
React.renderComponent(<Counter/>, document.getElementById('mount-point'));
```

گردش داده یک طرفه

درReact ، برنامه های جزیان داده یک طرفه از طریق شی state و props صورت می گیرد. به این معنی که، در یک componentسلسله مراتبی، یک component مادر باید state را مدیریت کند و آن را از طریق props به پایین زنجیر انتقال دهد state .شما باید با استفاده از متد setState به روز رسانی شود تا اگر لازم باشد، اطمینان حاصل شود که UI تازه رخ خواهد داد.

مثال زیر را مشاهده کنید که این مفهوم را نشان می دهد:

```
var FilteredList = React.createClass({
  filterList: function(event){
    var updatedList = this.state.initialItems;
    updatedList = updatedList.filter(function(item){
      return item.toLowerCase().search(
        event.target.value.toLowerCase()) !== -1;
    });
    this.setState({items: updatedList});
  },
  getInitialState: function(){
     return {
       initialItems: [
         "Apples",
         "Broccoli",
         "Chicken",
         "Duck",
```

```
        "Eggs",
        "Fish",
        "Granola",
        "Hash Browns"
      ],
      items: []
    }
  },
  componentWillMount: function(){
    this.setState({items: this.state.initialItems})
  },
  render: function(){
    return (
      <div className="filter-list">
        <input type="text" placeholder="Search" onChange={this.filterList}/>
      <List items={this.state.items}/>
      </div>
    );
  }
});


var List = React.createClass({
  render: function(){
    return (
      <ul>
      {
        this.props.items.map(function(item) {
          return <li key={item}>{item}</li>
        })
      }
      </ul>
```

```
    )
  }
});

React.renderComponent(<FilteredList/>, document.getElementById('mount-point'));

var FilteredList = React.createClass({
  filterList: function(event){
    var updatedList = this.state.initialItems;
    updatedList = updatedList.filter(function(item){
      return item.toLowerCase().search(
        event.target.value.toLowerCase()) !== -1;
    });
    this.setState({items: updatedList});
  },
  getInitialState: function(){
    return {
      initialItems: [
        "Apples",
        "Broccoli",
        "Chicken",
        "Duck",
        "Eggs",
        "Fish",
        "Granola",
        "Hash Browns"
      ],
      items: []
    }
  },
  componentWillMount: function(){
```

```
    this.setState({items: this.state.initialItems})
  },
  render: function(){
    return (
      <div className="filter-list">
        <input type="text" placeholder="Search" onChange={this.filterList}/>
      <List items={this.state.items}/>
      </div>
    );
  }
});

var List = React.createClass({
  render: function(){
    return (
      <ul>
      {
        this.props.items.map(function(item) {
          return <li key={item}>{item}</li>
        })
      }
      </ul>
    )
  }
});

React.renderComponent(<FilteredList/>, document.getElementById('mount-point'));
```