React’teki olay yönetimi

React’teki olay yönetimi, DOM elementlerindeki olay yönetimi ile oldukça benzerdir. Sadece, bazı küçük sözdizimi farklılıkları bulunmaktadır:

- Olay isimleri, DOM’da lowercase iken, React’te camelCase olarak adlandırılır.

- DOM’da fonksiyon isimleri, ilgili olaya string olarak atanırken, JSX’te direkt fonksiyon olarak atanır.

Örneğin HTML’de aşağıdaki gibi olan kod:

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

React’te biraz daha farklıdır:

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

React’teki diğer bir farklılık ise, olaylardaki varsayılan davranışın false değeri döndürülerek engellenemiyor oluşudur. Bunun için preventDefault şeklinde açıkça yazarak tarayıcıya belirtmeniz gerekir. Örneğin düz bir HTML kodunda, form elemanının varsayılan davranışı olan submiti engellemek için aşağıdaki gibi bir kod yazabilirsiniz:

```
<form onsubmit="console.log('You clicked submit.'); return false">
  <button type="submit">Submit</button>
</form>
```

React’te ise varsayılan form elementi davranışını e.preventDefault() kodu ile engellemeniz gerekir:

```
function Form() {
  function handleSubmit(e) {
    e.preventDefault();
    console.log('You clicked submit.');
  }
```

```
  return (
    <form onSubmit={handleSubmit}>
      <button type="submit">Submit</button>
    </form>
  );
}
```

Burada e, bir sentetik olaydır. React, bu sentetik olayları W3C şartnamesine göre tanımlar. Bu sayede, tarayıcılar arası uyumsuzluk problemi oluşmaz. React olayları, doğal (native) olaylarla tam olarak aynı şekilde çalışmaz. Bu konuda daha fazla bilgi edinmek için Sentetik Olaylar rehberini inceleyebilirsiniz.

React ile kod yazarken, bir DOM elementi oluşturulduktan sonra ona bir dinleyici (listener) atamak için, addEventListener fonksiyonunu çağırmanıza gerek yoktur. Bunun yerine render fonksiyonunda, ilgili element ilk kez render olduğunda ona bir dinleyici (listener) atamanız doğru olacaktır.

ES6 sınıfı kullanarak bir bileşen oluşturulduğunda, ilgili olayın tanımlanması için en yaygın yaklaşım, ilgili metodun o sınıf içerisinde oluşturulmasıdır. Örneğin aşağıdaki Toggle bileşeni, “ON” ve “OFF” durumlarının gerçekleştirilmesi için bir butonu render etmektedir:

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};
    // Callback içerisinde `this` erişiminin çalışabilmesi için, `bind(this)` gereklidir
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.setState(prevState => ({
      isToggleOn: !prevState.isToggleOn
    }));
  }
  render() {
```

```
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

JSX callback'lerinde this kullanırken dikkat etmeniz gerekmektedir. Çünkü JavaScript'te, sınıf metotları varsayılan olarak this'e bağlı değillerdir. Bu nedenle, this.handleClick'i bind(this) ile bağlamayı unutarak onClick'e yazarsanız, fonksiyon çağrıldığında this değişkeni undefined hale gelecek ve hatalara sebep olacaktır.

Bu durum, React'e özgü bir davranış biçimi değildir. Aslen, fonksiyonların JavaScript'te nasıl çalıştığı ile ilgilidir. Genellikle, onClick={this.handleClick} gibi bir metot, parantez kullanmadan çağırırken, o metodun bind edilmesi gerekir.

Eğer sürekli her metot için bind eklemek istemiyorsanız, bunun yerine farklı yöntemler de kullanabilirsiniz. Örneğin public class fields yöntemini kullanırsanız, callback'leri bağlamak için sınıf değişkenlerini kullanabilirsiniz:

```
class LoggingButton extends React.Component {
  // Bu yazım şekli, `this`'in handleClick içerisinde bağlanmasını sağlar.
  handleClick = () => {
    console.log('this is:', this);
  };
  render() {
    return (
      <button onClick={this.handleClick}>
        Click me
      </button>
    );
```

```
  }
}
```

Bu yöntem, Create React App ile oluşturulan geliştirim ortamında varsayılan olarak gelir. Böylece hiçbir ayarlama yapmadan kullanabilirsiniz.

Eğer bu yöntemi kullanmak istemiyorsanız, callback içerisinde ok fonksiyonunu da kullanabilirsiniz:

```
class LoggingButton extends React.Component {
  handleClick() {
    console.log('this is:', this);
  }
  render() {
    // Bu yazım şekli, `this`'in handleClick içerisinde bağlanmasını sağlar.
    return (
      <button onClick={() => this.handleClick()}>
        Click me
      </button>
    );
  }
}
```

Fakat bu yöntemin bir dezavantajı vardır. LoggingButton bileşeni her render edildiğinde, yeni bir callback oluşturulur. Birçok durumda bu olay bir sorun teşkil etmez. Ancak ilgili callback, prop aracılığıyla alt bileşenlere aktarılırsa, bu bileşenler fazladan render edilebilir. Bu tarz problemlerle karşılaşmamak için binding işleminin, ya sınıfın constructor'ında ya da class fields yöntemi ile yapılmasını öneririz.

* Olay Yöneticilerine Parametre Gönderimi

Bir döngü içerisinde, olay fonksiyonuna fazladan parametre göndermek isteyebilirsiniz. Örneğin, bir satır ID'si için id parametresi, aşağıdaki kodlardan her ikisi de işinizi görecektir:

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```

Üstteki iki satır birbiriyle eş niteliktedir. Ve sırasıyla ok fonksiyonu ile Function.prototype.bind fonksiyonu kullanırlar.

Her iki durum için de e parametresi, ID'den sonra ikinci parametre olarak aktarılacak bir React olayını temsil eder. Ok fonksiyonunda bu parametre açık bir şekilde tanımlanırken, bind fonksiyonunda ise otomatik olarak diğer parametreler ile birlikte gönderilir.