React Native uygulamanızı hızlandırmak için 6 basit yol.

Önceki makalemde uygulamanızın performansını iyileştirmek için etkinlik döngüsünü korsanlık hakkında konuştuk. Amacınız boyunca 60 FPS korumaktır. Bunların hepsi, yerel uygulamaya da tepki vermek veya tepki vermek için geçerlidir. Eğer henüz okumadıysanız, belki de bunu yapmak için doğru zaman.

Daha yüksek FPS'yi korumak için tepkiyi kullanırken veya yerel tepki verdiğinizde yapabileceğiniz birkaç şey vardır.

1. PureComponent veya shouldComponentUpdate kullanın

Reaksiyondaki PureComponent'in durumu yok, sadece bileşeninizi sahne üzerinden geçirilen verilere göre oluşturuyorlar. Sadece sahne değiştirirse yeniden işler.
shouldComponentUpdate life-cycle yöntemi, belirli senaryolarda false döndürerek yeniden oluşturmayı iptal etmek için normal non pure React Bileşeninde kullanılır.

İki kod örneğini takip etmek aynı şeyi yapar.

```
  MyComponent sınıfı, React.PureComponent {öğesini
genişletir
  //
  }
```

```
  sınıfım MyComponent, React.Component {öğesini genişletir
  //
  shouldComponentUpdate (nextProps, nextState) {
  if (this.props.firstProp === nextProps. firstProp &&
  this.props.secondProp === nextProps.secondProp) {
  yanlış dönüş;
  }
  gerçek geri dönüş
  }
  //
  }
```

Yukarıdaki örneklerin her ikisi de, boşa harcanmış bazı işleyicileri kurtarmanıza yardımcı olabilir. İlk örnek, zaten sizin için shouldComponentUpdate mantığını uygular. İkinci örnek size biraz

daha kontrol sağlar. Durumu bileşende tutabilir ve durum değişmezse yeniden oluşturmayı durdurabilirsiniz. Bunun gibi

```
  sınıfım MyComponent, React.Component {öğesini genişletir
  shouldComponentUpdate (nextProps, nextState) {
  if (this.state.isLoading === nextState .Yükleniyor) {
  yanlış dönüş;
  }
  gerçek geri dönüş
  }
  }
```

2. Liste öğelerinde anahtar özelliğini kullan

Liste, herhangi bir uygulamada en yaygın kullanılan şeydir. Her liste öğesi için benzersiz anahtar belirtmezseniz, tepkiler listeden herhangi bir öğe eklendiğinde veya çıkarıldığında her öğeyi yeniden oluşturur. Her liste öğesinde benzersiz bir tuşa sahip olmak, yeniden kaydetmeyi yeniden kaydetmekten kurtarır.

```
  MyComponent sınıfı, React.PureComponent {öğesini
genişletir
  render () {
  this.props.data.map değerini döndür ((öğe, i) => {
  dönüş <Text key = {item.id}> {item.title} </ Text>
  });
  }
  }
```

3. Erken bağlayın ve render içinde işlevler oluşturmayın.

Bunu yap

```
  MyComponent sınıfı, React.PureComponent {öğesini
genişletir

  yapıcı (sahne) {
  Süper (sahne);
  this.doWork = this.doWork.bind (this);
  }

  işi yapmak() {
  // burada biraz iş yapmak.
```

```
  // this.props.dispatch ....
  }

  render () {
  <Text onPress = {this.doWork}> Bazı Çalışmalar Yap </
Text>
  }

  }
```

Bunu render yapmayın.

```
  <Text onPress = {() => this.doWork ()}> Bazı Çalışmalar
Yapın </ Text>
```

veya

```
  <Text onPress = {this.doWork.bind (this)}> Bazı Çalışmalar
Yap </ Text>
```

İşleme çok sık denir ve yukarıdaki iki şeyden herhangi birini her yaptığınızda, yeni bir işlev oluşturulur.
Eğer argümanlar doWork işlevine geçmek istiyorsanız, bunun gibi bir kapatma kullanabilirsiniz.

```
  doWork = (param) => () => {
  console.log (param)
  }

  <Text onPress = {this.doWork (someVarParam)} Bazıları Args
ile Çalışıyor </ Text>
```

4. componentWillUpdate içindeki durumu veya gönderim eylemlerini güncellemeyin

componentWillUpdate yaşam döngüsü yöntemi, bir güncellemeye hazırlamak için kullanılır, başka bir tanesini tetiklemez. Durumu ayarlamak veya herhangi bir redux eylemini göndermek istiyorsanız, bunları componentWillReceiveProps yerine yapın.

5. Büyük veri kümeleri için VirtualizedList, FlatList ve SectionList kullanın.

Tepki ana belgelerine göre, VirtualizedList, FlatList ve SectionList, daha az bellek ayak izi kullandıkları için listeler oluşturmak için performans ara yüzüdür. Yüzlerce satır içeren bir listeniz varsa, aşağı kaydırılana kadar bunların hepsi ekrana yüklenmez.
VirtualizedList, hem FlatList hem de SectionList için temeldir. Değişken veri kümeniz varsa, doğrudan VirtualizedList kullanmalısınız.

6. FPS izlemek için Perf monitor kullanın

Geliştirici araçlarını açın ve mükemmel monitörü etkinleştirin. Artık uygulamanızı etkileşime geçmeye başladığınızda, gezinirken veya gezinirken, bazen FPS'nin düştüğünü görebilirsiniz. Ve çoğunlukla kendi yerel UI iş parçacığı üzerinde değil, JS iş parçacığında. Bu yüzden FPS'in düştüğü acı noktaları aramaya başlayın. Durumu ayarlıyor veya redux eylemlerini yanlış yere gönderiyor olabilirsiniz. Ya da JS iş parçacığı üzerinde çok fazla eşzamanlı çalışma yapıyor.

Yerelleştirmeye yönelik bir sonraki makalemde daha gelişmiş teknikleri paylaşır. Yeni hikayeler yayınlandığında güncellemeleri almak için beni takip edebilirsiniz :)
Herhangi bir şey eklemek isterseniz, lütfen yorum bölümünde paylaşın.