9.2.2 sass-loader 설정 커스터마이징하기

<코드>**webpack.config.js – sassRegex 찾기**

```
{
  test: sassRegex,
  exclude: sassModuleRegex,
  use: getStyleLoaders(
    {
      importLoaders: 3,
      sourceMap: isEnvProduction
        ? shouldUseSourceMap
        : isEnvDevelopment,
    },
    'sass-loader'
  ),
  sideEffects: true,
},
```

</코드>

메모 포함[KM1]: 음.. 색상이 기존이랑 좀 바뀌긴 한 것 같은데요, 별로 안 중요해서, 그냥 이대로 하셔도 됩니다.

<코드>

```
{
  test: sassRegex,
  exclude: sassModuleRegex,
  use: getStyleLoaders({
    importLoaders: 3,
    sourceMap: isEnvProduction
      ? shouldUseSourceMap
      : isEnvDevelopment,
  }).concat({
    loader: require.resolve("sass-loader"),
    options: {
      sassOptions: {
        includePaths: [paths.appSrc + "/styles"],
      },
    },
  }),
  sideEffects: true,
},
```

</코드>

메모 포함[KM2]: 그 아래의 코드

<코드>

```
{
  test: sassRegex,
  exclude: sassModuleRegex,
  use: getStyleLoaders({
    importLoaders: 3,
    sourceMap: isEnvProduction
      ? shouldUseSourceMap
      : isEnvDevelopment,
  }).concat({
    loader: require.resolve("sass-loader"),
    options: {
      sassOptions: {
        includePaths: [paths.appSrc + "/styles"],
      },
      additionalData: "@import 'utils';",
    },
```

메모 포함[KM3]: pg. 231 의 코드

```
  }),
  sideEffects: true,
}
```
</코드>

10.3 기능 구현하기

pg. 276

설치하고 나서 크롬 개발자 도구를 열면 개발자 도구 탭에 components가 나타납니다. 이를 클릭하세요. 그리고 좌측에서 TodoInsert를 선택하면, 다음과 같이 인풋을 수정했을 때 Hooks의 State 부분에도 똑같은 값이 잘 들어가는 것을 확인할 수 있습니다.

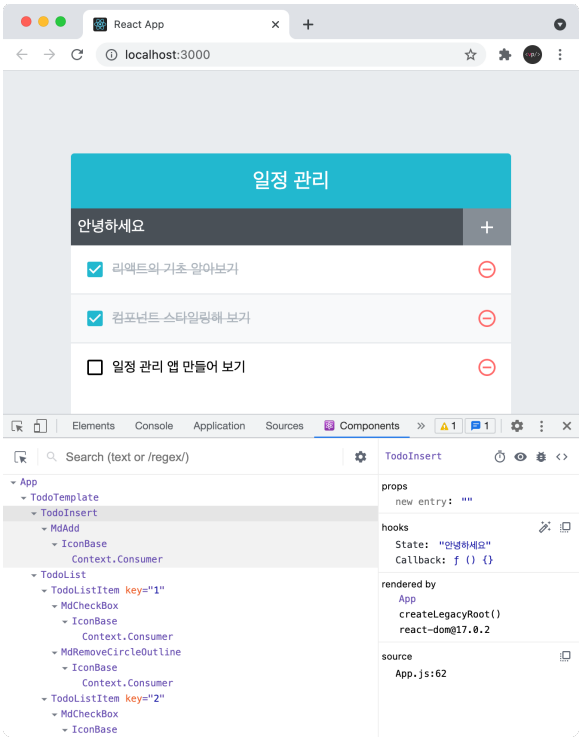


그림 10-16 리액트 개발자 도구

11.2 React DevTools를 사용한 성능 모니터링

성능을 분석해야 할 때는 느려졌다는 느낌만으로 충분하지 않습니다. 정확히 몇 초가 걸리는지 확인해야 하는데, 이는 React DevTools를 사용하여 측정하면 됩니다. 리액트 v17 전에는 브라우저에 내장된 성능 측정 도구의 User Timing API를 사용했었지만, v17 부터는 리액트 전용 개발자 도구인 React DevTools를 사용해야 성능 분석을 자세하게 할 수 있습니다. 10장에서 리액트 개발자 도구의 Components 탭을 열어봤었는데요, 그 우측에 Profiler라는 탭을 열어보세요. 이 탭을 열면 좌측 상단에 파란색 녹화 버튼이 보일 것입니다.

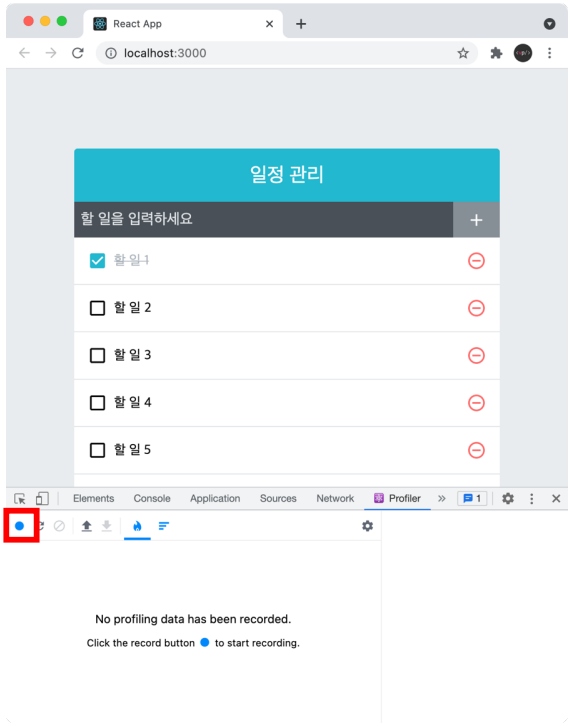


그림 11-3 React DevTools의 Profiler 탭

이 버튼을 누르고 '할 일 1' 항목을 체크한 다음, 화면에 변화가 반영되면 녹화 버튼을 한번 더 누르세요. 그러면 다음과 같이 성능 분석 결과가 나타납니다.

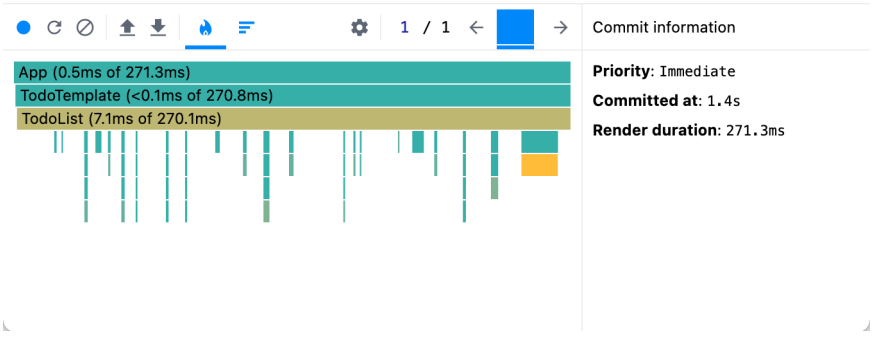
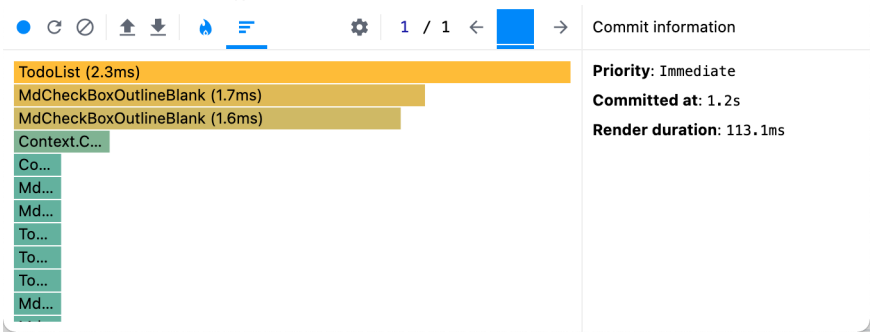


그림 11-4 성능 분석 결과

우측에서 Render duration이 리렌더링에 소요된 시간을 의미합니다. 방금 변화를 화면에 반영하는데 271.3ms가 (1ms 는 0.001초 입니다) 걸렸다는 의미이죠. 참고로, 소요 시간은 컴퓨터 환경에 따라 결과가 다르게 나타날 수 있습니다.

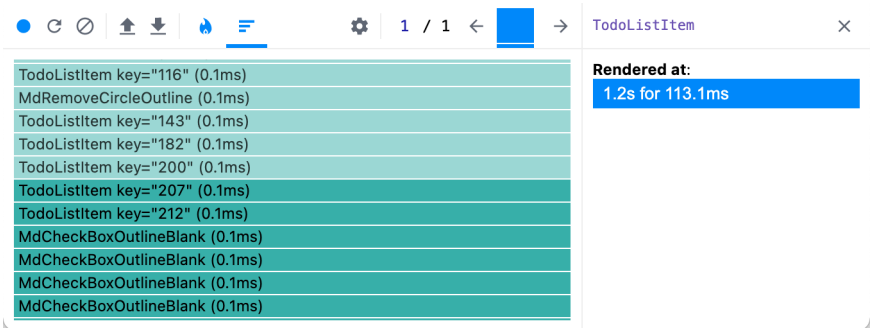
Profiler 탭의 상단에 있는 불꽃 모양 아이콘 우측의 랭크 차트 아이콘을 눌러보세요.

그림 11-5 Ranked Chart (i)



이 화면에서는 리렌더링된 컴포넌트를 오래걸린 순으로 정렬하여 나열해줍니다. 여기서 스크롤을 해보시면 정말 많은 컴포넌트가 리렌더링된 것을 확인 할 수 있을 것입니다. 현재 초록색 박스들은 너무 작아서 텍스트 내용이 잘려서 보이지 않을텐데요 클릭을 하시면 크기가 늘어나서 내용을 확인할 수 있습니다. 작은 초록색 박스를 누르고 아래로 쭉 스크롤을 내려보세요.

그림 11-6 Ranked Chart (ii)



이를 보면 우리가 이번에 변화를 일으킨 컴포넌트랑 관계없는 컴포넌트들도 리렌더링 된 것을 확인 할 수 있습니다.

(...)

pg.298
이제 코드를 저장하고, 조금 전 했던 것과 똑같이 Profiler 개발자 도구를 열고 성능을 측정해보세요.

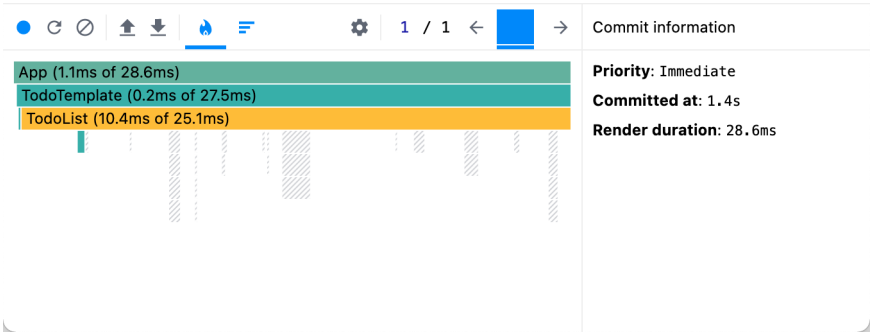


그림 11-7 최적화 이후 성능

성능이 훨씬 향상된 것을 확인하셨나요? 렌더링 소요시간이 113.1ms 에서 28.6ms로 줄었습니다. 왼쪽을 보시면 회색 빗금이 그어져있는 박스들이 있습니다. 이는 React.memo를 통하여 리렌더링이 되지 않은 컴포넌트를 나타냅니다.

랭크 차트 아이콘을 눌러서 이번에 리렌더링된 컴포넌트의 수를 보면 몇 개 없는 것을 확인하실 수 있을 것입니다.

메모 포함[KM4]: 개발 환경에서의 성능 노트는 제거! (프로덕션에선 별도 설정을 하지 않는 이상, Profiling 이 불가능해지기도 했고, 성능이 아주 크게 차이가 나지는 않아서요) 이를 통해서 11.2의 공간을 조금 확보할 수 있지 않을까 싶습니다. 300페이지에 이어지는 serve 사용 부분도 지웁니다.

메모 포함[KM5]: 그림 캡션 11-8 -> 11-9 로 변경

pg. 310

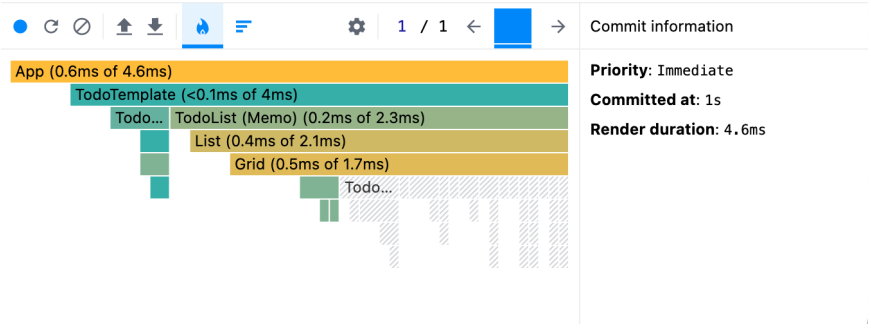


그림 11-10 react-virtualized를 통한 최적화 결과

React.memo를 통해 28ms까지 줄였는데, 이번에는 4.6ms로 줄었습니다!