개발자경험 향상을 위한 상태관리모델 제안

임수원[O], 모지식[*], 권재환[*], 김명호(교신저자)[*]
[O]숭실대학교 소프트웨어학부,
[*]숭실대학교 소프트웨어학과
e-mail: limsoft0401@kakao.com[O], 1102278011@soongsil.ac.kr[*], jaehwan@soongsil.ac.kr[*], kmh@ssu.ac.kr[*]

A proposal of State management model, to improve the developer experience

suwon Lim[O], jisik Mo[*], jaewhan Kwon[*], myungho Kim(Corresponding Author)[*]
[O]Dept. of Software Science, Soongsil University,
[*]Dept. of Software Science, Soongsil University

● 요 약 ●

본 연구는 상태관리를 위한 새로운 모델, Quantum State Management (QSM)을 제안한다. QSM은 어플리케이션의 상태를 Quantum이라는 최소 단위로 나누어 비동기 및 병렬처리를 최적화하며, 상태의 추적가능성을 높이는 모델이다. 본 연구에서는 QSM의 개념을 제시하고 이를 Flux 패턴의 Redux와 비교하여 QSM이 갖는 장점과 비교한다.

키워드: 상태관리(State Management), QSM(Quantum State Management), 개발자경험(developer experience)

I. Introduction

웹 어플리케이션의 복잡도가 증가함에 따라서, 상태관리는 웹 어플리케이션 개발의 중요한 부분이 되었다. Flux구조를 따르는 상태관리 라이브러리 중 가장 점유율이 높은 Redux는 어플리케이션 상태를 효과적으로 관리하는 방법을 제공하지만, 복잡한 구조로 인해서 높은 러닝 이슈와 성능문제를 초래할 수 있다. 본 연구에서는 이러한 문제를 해결하기 위해서 Quantum객체를 통해 상태를 관리하고 변경을 감지하는 리스너방식인 새로운 상태관리 모델, QSM (Quantum State Management)을 소개한다.

II. Preliminaries

1. Flux Pattern

표에서 볼 수 있듯이, Flux 패턴은 일방향 데이터흐름을 갖는다. 사용자의 입력이나 서버응답 등으로 인해 Action이 생성되면 Dispatcher를 통해 Store에 전달되고, Store의 상태 업데이트는 다시 View에 반영되는 방식이다. 이런 일방향적 흐름은 상태변경을 예측가능하게 하고 추적가능한 코드를 작성할 수 있게 도와줌으로서 버그를 줄인다. 그러나 Flux패턴을 사용하는 라이브러리 (Redux)는 액션생성자와 reducer의 복잡성, 동기식 동작으로 인해 미들웨어를 추가해야하는 등 여러 문제를 가진다.

Table 1. Components of Flux Pattern

구성요소	설명
Actions	사용자의 입력이나 시스템에서 발생한 이벤트를 나타내는 객체이다. 사용자의 클릭이나 서버 응답과 같은 특정 이벤트가 발생하면, 이에 해당하는 Action이 생성된다.
Dispatcher	모든 Action이 통과하는 중앙허브이다. Action이 발생하면 이를 통하여 Store에 전달된다.
Store	어플리케이션의 상태와 로직을 관리한다. Dispatcher에서 전달받은 Action에 따라 상태를 업데이트한다.
View	사용자에게 보여지는 부분으로, Store의 상태에 따라서 업데이트 된다. Store의 상태가 변경되면 View는 이를 반영하여 리렌더링된다.

III. The Proposed Scheme

1. Quantum State Management

Quantum State Management(이하 QSM)는 상태관리를 Quantum단위로 쪼개는 것을 제안한다. Quantum(이하 퀀텀)은 상태

의 최소조각을 의미하며 간결한 구조, 상태변경용이성, 그리고 병렬성에 그 의의를 두었다.

Table 2. Components of QSM

구성요소	설명
Quantum	어플리케이션 상태의 가장 작은 단위로, 독립적으로 동작하고 필요에 따라서 다른 퀀텀과 상호작용할 수 있다. 독립적이므로 추가와 제거가 쉽다.
State	각 Quantum이 가지고 있는 데이터로서, 어플리케이션에서 필요에 따라서 변경될 수 있다.
State Change Function	퀀텀의 상태를 변경하는 함수로서, 사용자의 입력이나 API 호출결과, 타이머 등에 의해서 호출될 수 있다.
Subscriber	퀀텀의 상태변경을 구독하며, 상태가 변경되면 새로운 상태를 받아서 바로 구독하는 컴포넌트에 제공한다.

2. Quantum의 작동

QSM은 다음과 같은 단계로 작동한다. 1. 퀀텀생성 , 퀀텀 객체를 생성하고 초기 상태를 설정한다. 이 상태는 퀀텀이 관리할 데이터를 나타낸다. 2. 리스너등록, 각 퀀텀은 상태변경을 추적하는 하나이상의 리스너를 가질 수 있다. 리스너는 주로 컴포넌트나 다른 상태 의존을 가진 함수가 될 수 있다. 퀀텀의 subscribe 메서드를 호출하면 리스너를 등록할 수 있다. 3. 상태변경, 어플리케이션에서 이벤트(사용자입력, api응답 등)가 발생하면 퀀텀의 setState메서드가 호출되어 상태가 업데이트된다. 4. 리스너에게 상태변경알림, 상태가 업데이트되면, 퀀텀은 모든 리스너들에게 새로운 상태를 알린다. 이를 통해서 리스너는 상태 변경에 따라서 필요한 작업을 수행할 수 있다. 5. 리스너제거, 필요에 따라 퀀텀의 unsubscribe메서드를 호출하여 특정 리스너를 제거할 수 있다. 리스너를 제거하면, 해당리스너는 더이상 상태변경에 대한 알림을 받지 않는다.

Table 3. Comparison of Flux(Redux) and QSM

	Flux(Redux)	QSM
분산처리	중앙집중식의 상태관리를 제공하고, 모든 상태변경이 단일 스토어에서 처리된다	각 퀀텀은 독립적으로 동작하며, 상태변경작업을 분산처리할 수 있다. 이는 병렬처리를 가능하게 만드며, 어플리케이션의 성능을 향상 시킬 수 있다.
상태추적	action log를 통해 상태변경을 추적	각 퀀텀에서 상태변경을 추적한다.
유연성	store의 상태구조를 미리 설정해야한다. 동적 상태구조변경이 번거로움	각 퀀텀은 독립적으로 동작하므로, 퀀텀을 동적으로 추가하거나 제거하는 것이 쉽다.
코드의 간결성	action 생성자와 reducer를 정의해야하고, 상태관리 한번을 위해서 준비해야 하는 코드가 많다.	상태변경 로직을 퀀텀 안에 캡슐화하여 코드의 간결성이 높다.

3. QSM과 Flux(Redux)비교

QSM과 Flux(Redux)는 모두 웹 어플리케이션의 상태를 관리하는 방법론이지만, 그 동작 방식과 특징이 다르다.

상태변경 작업이 비동기적이거나, 병렬로 처리될 필요가 있는 경우에 QSM의 분산처리가 Flux(Redux)에 비해서 우수하다. 개발의 변경이 잦은 프로젝트의 경우에도 퀀텀을 유연하게 추가/제거/변경할 수 있는 QSM이 유리하다.

IV. Conclusions

현대의 기술 트렌드는 빠르게 변화하며, 이에 따라서 프로젝트도 지속적으로 수정의 필요성을 요구받는다. 점유율이 가장 높은 Flux패턴은 상태관리에 있어서 예측가능하고 추적가능한 모델을 제공하지만, 복잡한 작동방식과 많은 보일러플레이트 코드를 작성해야 하기 때문에 개발자에게 혼란과 피로를 초래한다. 반면 QSM은 간결하고 이해하기 쉬운 구조, 간결한 메서드를 통해 개발자에게 더 나은 개발 경험을 제공하며 쉬운 변경을 가능하게 한다. 미래의 트렌드는 더욱 빠르게 바뀔 것이고 그럴수록 변화에 빠르게 대응할 수 있는 간결하고 이해가 쉬운 구조의 상태관리 모델의 중요성은 점점 더 커질 것이다.

ACKNOWLEDGEMENT

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학사업의 연구결과로 수행되었음(2018-0-00209)

REFERENCES

[1] "Flux Overview, "https://haruair.github.io/flux/docs/overview.html"