assetsStore.ts 기술 보고서

1. 개요

자산(Asset) 관련 상태와 비즈니스 로직을 관리하는 Zustand 스토어입니다. 자산 리스트의 추가, 수정, 삭제, 개별 자산 조회 등 자산군 관리 기능을 제공합니다. 특히 미국 자산군의 환율 반영 상태를 전략 스토어와 연동하여 전체 환율 반영 여부를 동적으로 관리합니다.

2. State

State Key	Type	설명
assetList	AssetData[]	전체 자산 데이터 리스트
actions	object	자산 관련 액션 함수 모음

`AssetData` 타입:

필드명	타입	설명
id	string	자산 고유 식별자
type	AssetType	자산 종류
assetGroup	string	자산 그룹명
weight	string	자산 비중
exchangeRateState	boolean	환율 반영 여부

3. Action

Action	Params	설명
addAsset	()	새로운 자산을 리스트에 추가 (기본값으로 생성)
updateAsset	(id: string, newData: Partial)	특정 자산의 일부 정보를 수정
deleteAsset	(id: string)	자산 리스트에서 해당 id의 자산을 삭제
allCheck	()	미국 자산군 전체 환율 반영 체크박스 상태 반환
updateAllCheck	(checked: boolean)	미국 자산군 전체 환율 반영 상태 일괄 변경
getAssetItem	(id: string)	id로 자산 데이터 반환
getData	()	전체 자산 리스트 반환
reset	()	자산 리스트 초기화

4. 사용 예시

```
import { useAssetStore } from 'store/assetsStore';

const assetList = useAssetStore(state => state.assetList);
const { addAsset, updateAsset, deleteAsset, allCheck, updateAllCheck,
getAssetItem, getData, reset } = useAssetStore(state => state.actions);

addAsset();
updateAsset('uuid', { weight: '10', exchangeRateState: true });
deleteAsset('uuid');
```

5. 설계 의도

- **구조**: 상태(assetList)와 액션(actions 객체)를 분리하여, 필요한 값만 selector로 구독하고 액션은 구조분해 할당으로 효율적으로 사용.
- **장점**: 코드 가독성, 유지보수성, 액션 관리가 명확함. 불필요한 리렌더링 최소화.
- **단점**: 여러 스토어 간 의존성이 생길 수 있으며, getState/setState 사용 시 순환 참조에 주의 필요.

6. 주의 사항/한계

- ◦ 값 변경 시 필드값 구독으로 인한 불필요한 리랜더링이 발생하지 않도록 유의
- ◦ 특히 assetList의 일부 요소 변경시 selecter를 사용해서 다른 컴포넌트의 랜더링에 영향을 주지 않도록 유의