

Normalize data in Redux



Seedlink Tech



Corporate Website

- **N**atural **L**anguage **P**rocessing
- **A**rtificial **I**ntelligence
- **H**uman **R**esources

William Cai

William Cai



Wechat

- **F**ront **E**nd **D**eveloper
- **S**crum **M**aster
- **C**lean **C**ode

Survey

- **Normalizr**
- **Redux**
- **Programming**

Motivation

- **Nested network response data**
- **Reusable, flexible & scaleable**
- **Single source of truth**

Solution

Normalizr

```
[{
  id: 1,
  title: 'Some Article',
  author: {
    id: 1,
    name: 'Dan'
  }
}, {
  id: 2,
  title: 'Other Article',
  author: {
    id: 1,
    name: 'Dan'
  }
}]
```

normalize

```
{
  result: [1, 2],
  entities: {
    articles: {
      1: {
        id: 1,
        title: 'Some Article',
        author: 1
      },
      2: {
        id: 2,
        title: 'Other Article',
        author: 1
      }
    },
    users: {
      1: {
        id: 1,
        name: 'Dan'
      }
    }
  }
}
```

Schema

```
[{
  id: 1,
  title: 'Some Article',
  author: {
    id: 1,
    name: 'Dan'
  }
}, {
  id: 2,
  title: 'Other Article',
  author: {
    id: 1,
    name: 'Dan'
  }
}]
```

User:

```
import { normalize, schema } from 'normalizr';

// Define a users schema
const user = new schema.Entity('users');
```

Article:

```
// Define your article
const article = new schema.Entity('articles', {
  author: user,
});
```

Normalize & Denormalize

```
[{
  id: 1,
  title: 'Some Article',
  author: {
    id: 1,
    name: 'Dan'
  }
}, {
  id: 2,
  title: 'Other Article',
  author: {
    id: 1,
    name: 'Dan'
  }
}]
```

```
// Normalize source data to result & entities
const { result, entities } = normalize(sourceData, [article]);
```

normalize

denormalize

```
// De-normalize result & entities to new source data
const newSourceData = denormalize(result, [article], entities);
```

```
{
  result: [1, 2],
  entities: {
    articles: {
      1: {
        id: 1,
        title: 'Some Article',
        author: 1
      },
      2: {
        id: 2,
        title: 'Other Article',
        author: 1
      }
    },
    users: {
      1: {
        id: 1,
        name: 'Dan'
      }
    }
  }
}
```

Redux Integration

Action

```
// Add schema to action to let middleware normalize the response data  
const action = {  
  types: [REQUEST, SUCCESS, FAILURE],  
  promise: client => client.get('api/articles'),  
  schema: [article],  
}
```

Middleware

```
// Make client middleware by different network client
const createClientMiddleware = (client) => ({ dispatch, getState }) => next => (action) => {
  const { types: [REQUEST, SUCCESS, FAILURE], promise, schema } = action;

  const clientPromise = promise(client);
  dispatch({ type: REQUEST });

  return clientPromise.then((response) => {
    const { result, entities } = normalize(response.data, schema);
    dispatch({ type: SUCCESS, payload: result, entities });
    return result;
  }).catch((error) => {
    dispatch({ type: FAILURE, error });
    throw error;
  });
}
```

middleware

```
// Add schema to action to let middleware normalize
const action = {
  types: [REQUEST, SUCCESS, FAILURE],
  promise: client => client.get('api/articles'),
  schema: [article],
}
```

action

Reducer

```
// Entities reducer for handle all action with entities field-
const entitiesReducer = (state = initialState, action = {}) => {
  const { entities } = action;
  // Ignore action without entities
  if (!entities) return state;
  // Deep merge entities into state
  return mergeDeep(state, entities);
};
```

Entities Reducer

```
// Business reducer-
const articlesReducer = (state = initialState, action = {}) => {
  const { type, payload } = action;
  switch (type) {
    case SUCCESS:
      // Replace state with new result
      return payload;
    default:
      return state;
  }
};
```

Business Reducer

Selector

```
{
  articles: [1, 2],
  entities: {
    articles: {
      1: {
        id: 1,
        title: 'Some Article',
        author: 1
      },
      2: {
        id: 2,
        title: 'Other Article',
        author: 1
      }
    },
    users: {
      1: {
        id: 1,
        name: 'Dan'
      }
    }
  }
}
```

Global State

```
import { denormalize, schema } from 'normalizr';

// Define a users schema
const user = new schema.Entity('users');
// Define your article
const article = new schema.Entity('articles', {
  author: user,
});

// select article object list
const getArticles = (globalState) => {
  const { articles, entities } = globalState;
  return denormalize(articles, [article], entities);
}
```

Articles Selector

Exercise



Thanos

- **Save the Universe**
- **Wipe Out Half Population**
- **Gather Infinity Stones**

Exercise



Thanos

- **Save the Universe**
- **Wipe Out Half Population**
- **Gather Infinity Stones**

Exercise

```
{
  id: 1,
  title: 'Gather Infinity Stones',
  owner: {
    id: 1,
    name: 'Thanos'
  },
  dependence: {
    id: 2,
    title: 'Wipe Out Half Population',
    owner: {
      id: 1,
      name: 'Thanos'
    },
    dependence: {
      id: 3,
      title: 'Gather Infinity Stones',
      owner: {
        id: 1,
        name: 'Thanos'
      },
    },
  },
}
```

- Save the Universe
 - Wipe Out Half Population
 - Gather Infinity Stones

Exercise

```
{
  id: 1,
  title: 'Gather Infinity Stones',
  owner: {
    id: 1,
    name: 'Thanos'
  },
  dependence: {
    id: 2,
    title: 'Wipe Out Half Population',
    owner: {
      id: 1,
      name: 'Thanos'
    },
    dependence: {
      id: 3,
      title: 'Gather Infinity Stones',
      owner: {
        id: 1,
        name: 'Thanos'
      },
    }
  }
}
```

```
import { normalize, schema } from 'normalizr';

// Define a user schema
const user = new schema.Entity('users');
// Define a todo schema
const todo = new schema.Entity('todos', {
  owner: user
});
```

```
// Instance method useful for creating circular references in schema
todo.define({ dependence: todo });
```

```
// Normalize!!!
const data = normalize(todoData, todo);
```

Exercise

```
{
  id: 1,
  title: 'Gather Infinity Stones',
  owner: {
    id: 1,
    name: 'Thanos'
  },
  dependence: {
    id: 2,
    title: 'Wipe Out Half Population',
    owner: {
      id: 1,
      name: 'Thanos'
    },
    dependence: {
      id: 3,
      title: 'Gather Infinity Stones',
      owner: {
        id: 1,
        name: 'Thanos'
      },
    },
  },
}
```

```
{
  result: [1, 2, 3],
  entities: {
    todos: {
      1: {
        id: 1,
        title: 'Gather Infinity Stones',
        owner: 1,
        dependence: 2,
      },
      2: {
        id: 2,
        title: 'Wipe Out Half Population',
        owner: 1,
        dependence: 3,
      },
      3: {
        id: 3,
        title: 'Gather Infinity Stones',
        owner: 1,
      },
    },
    users: {
      1: {
        id: 1,
        name: 'Thanos'
      },
    },
  },
}
```



Advance

- Immutable
- Reselect
- Entity HOC
- ORM

```
{
  result: [1, 2, 3],
  entities: {
    todos: {
      1: {
        id: 1,
        title: 'Gather Infinity Stones',
        owner: 1,
        dependence: 2,
      },
      2: {
        id: 2,
        title: 'Wipe Out Half Population',
        owner: 1,
        dependence: 3,
      },
      3: {
        id: 3,
        title: 'Gather Infinity Stones',
        owner: 1,
      },
    },
    users: {
      1: {
        id: 1,
        name: 'Thanos'
      },
    },
  },
}
```

Summary

- **Using Normalizr to flatten nest data**
- **Using middleware to centralized convert**
- **Using reducer to watch & update store**
- **Using reselect to avoid unnecessary change**

Recruiting



Digital Interview

- Devops Engineers
- Backend Engineers
- Frontend Engineers
- Test Engineers
- Data Privacy Officer
- VP Engineering
- Research Scientist
- Behavioural Scientist
- Data Scientist



More Jobs

Q & A



Corporate Website



Digital Interview

Q & A



Wechat

Thank You

