KB금융그룹 | 국민의 평생 금융파트너

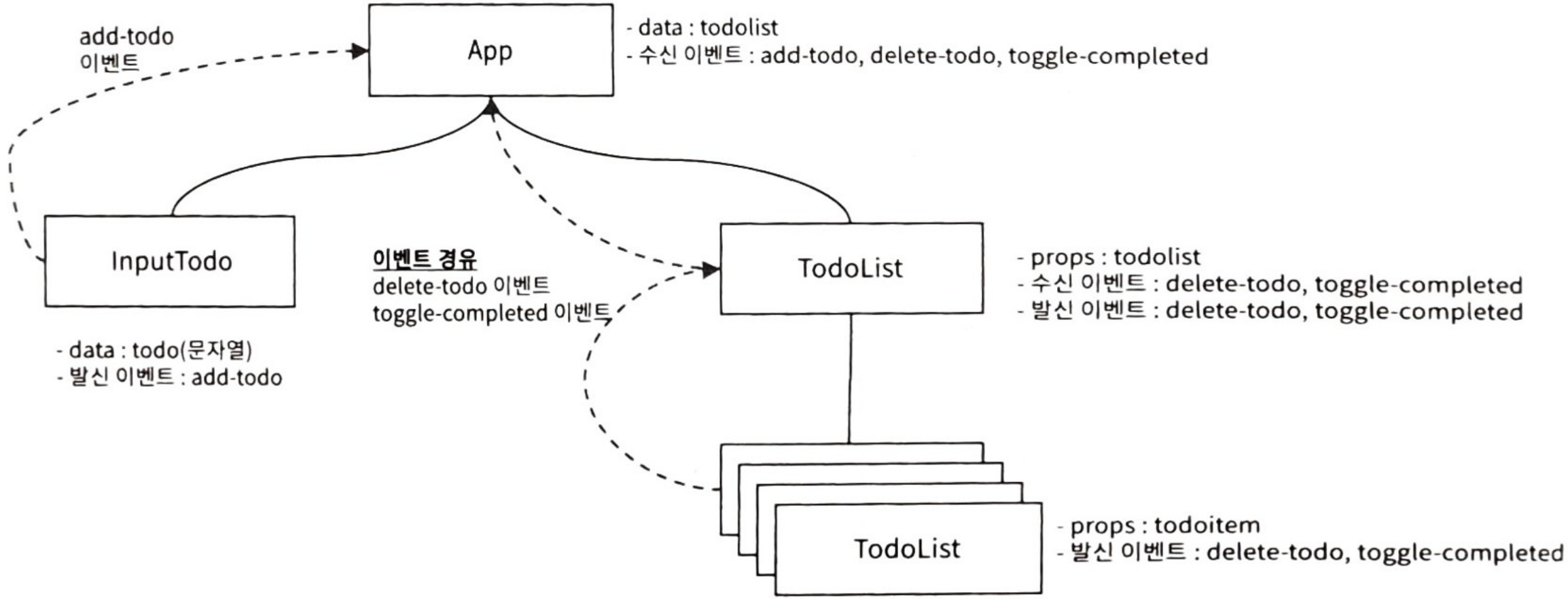
2025년 상반기 K-디지털 트레이닝

TodoList App 리팩토링(Composition API)

[KB] IT's Your Life

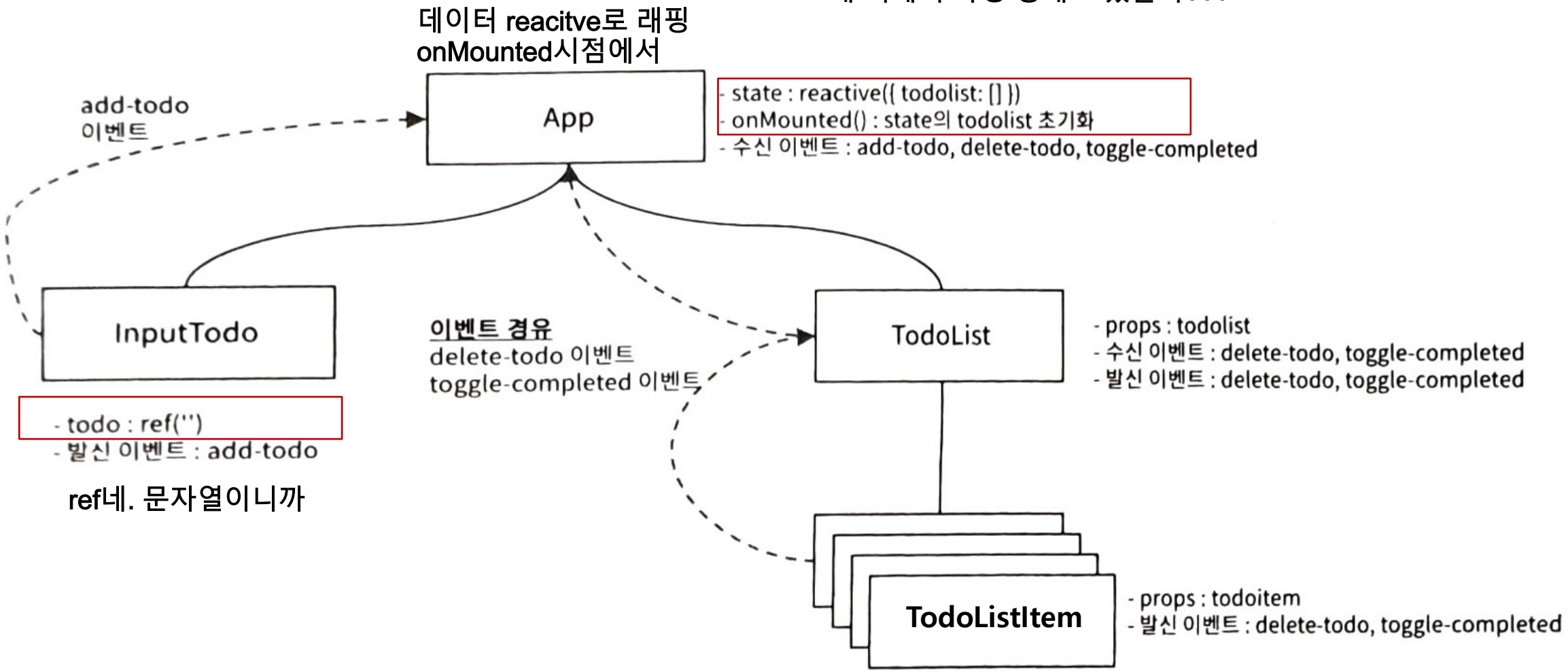
KB 국민은행

TodoList 앱의 계층 구조(기존)



리팩토링된 TodoList 앱의 계층 구조

reactive([ ]) 될 것을
왜 객체의 속성 형태로 했을까???



ref네. 문자열이니까

src/App.vue

템플릿 파트는 변화X

```
<template>
  <div class="container">
      <div class="card card-body bg-light">
          <div classe="title">:: Todolist App</div>
      </div>
      <div class="card card-default card-borderless">
          <div class="card-body">
              <InputTodo @add-todo="addTodo"></InputTodo>
              <TodoList :todoList="state.todoList" @delete-todo="deleteTodo"
                 @toggle-completed="toggleCompleted"></TodoList>
          </div>
      </div>
  </div>
</template>
```

src/App.vue

script setup태그 사용X

```
<script>
import { reactive, onMounted } from 'vue'
import InputTodo from './components/InputTodo.vue'
import TodoList from './components/TodoList.vue'

export default {
  name : "App",
  components : { InputTodo, TodoList },
  setup() {
    const ts = new Date().getTime();
    const state = reactive({ todoList : [] })

    onMounted(()=>{
      state.todoList.push({ id: ts, todo:"자전거 타기", completed: false })
      state.todoList.push({ id: ts+1, todo:"딸과 공원 산책", completed: true })
      state.todoList.push({ id: ts+2, todo:"일요일 애견 카페", completed: false })
      state.todoList.push({ id: ts+3, todo:"Vue 원고 집필", completed: false })
    })
```

초기화

백엔드로부터 서버에서 데이터를 받는 상황.

src/App.vue

```
    const addTodo = (todo)=> {
      if (todo.length >= 2) {
          state.todoList.push({ id: new Date().getTime(),
               todo:todo, completed: false });
      }
    }

    const deleteTodo = (id) => {
      let index = state.todoList.findIndex((item)=> id === item.id);
      state.todoList.splice(index,1);
    }

    const toggleCompleted = (id) => {
      let index = state.todoList.findIndex((item)=> id === item.id);
      state.todoList[index].completed = !state.todoList[index].completed;
    }

    return { state, addTodo, deleteTodo, toggleCompleted }
  }
}
</script>
```

내부함수로 다 들어옴

리턴

src/components/InputTodo.vue

```
<template>
…
</template>

<script>
import { ref } from 'vue';

export default {
  name: 'InputTodo',
  setup(props, context) {
    const todo = ref('');
    const addTodoHandler = () => {
      if (todo.value.length >= 3) {
        context.emit('add-todo', todo.value);
        todo.value = '';
      }
    };
    return { todo, addTodoHandler };
  },
};
</script>
```

setup함수 방식과
script setup 방식과
가장 다른 점

this. 도 금지 $emit금지

setup함수 안에서는
this는 이제 vue인스턴스가 아니다

그렇다면vue인스턴스는 어떻게 접근해?
context로 접근

script setup에서 vue인스턴스 접근은?
별도의 함수를 제공한다

Composition API 사용하기

장점

- 적은 상용구 코드 사용으로 간결한 코드를 작성
- 순수 타입스크립트 언어를 사용해 props, 이벤트를 선언
- 런타임 성능이 더 좋음
- IDE에서의 타입 추론 성능이 더 뛰어남

2 <script setup> 사용하기

기존과 다른점

템플릿에서 사용하는 값

- 최상위의 변수, 함수는 직접 템플릿에서 사용

컴포넌트 등록

- import한 컴포넌트는 바로 템플릿에서 지역 컴포넌트로 사용

속성과 발신 이벤트 처리

- defineProps, defineEmits 함수를 이용해 속성과 emit 함수를 생성

```
//기존 방식
setup(props, context) {
   //이벤트를 발신할 때
   context.emit('add-todo', todo)
```

```
// <script setup> 방식
const props = defineProps({
   todoItem : { type : Object, required: true }
})
const emit = defineEmits(['delete-todo','toggle-completed'])
//이벤트를 발신할 때는 다음과 같이
emit('delete-todo', id)
```

파일명

```
<template>
…
</template>

<script setup>
import { reactive, onMounted } from 'vue';
import InputTodo from './components/InputTodo.vue';
import TodoList from './components/TodoList.vue';

const ts = new Date().getTime();
const state = reactive({ todoList: [] });

onMounted(() => {
  state.todoList.push({ id: ts, todo: '자전거 타기', completed: false });
  state.todoList.push({ id: ts + 1, todo: '딸과 공원 산책', completed: true });
  state.todoList.push({ id: ts + 2, todo: '일요일 애견 카페', completed: false });
  state.todoList.push({ id: ts + 3, todo: 'Vue 원고 집필', completed: false });
});
```

components등록은 자동으로

템플릿에서 사용을 위한
반환도 자동으로
TOP레벨의 식별자 모두 자동 반환

src/App.vue

```
const addTodo = (todo) => {
  if (todo.length >= 2) {
    state.todoList.push({
      id: new Date().getTime(),
      todo: todo,
      completed: false,
    });
  }
};

const deleteTodo = (id) => {
  let index = state.todoList.findIndex((item) => id === item.id);
  state.todoList.splice(index, 1);
};

const toggleCompleted = (id) => {
  let index = state.todoList.findIndex((item) => id === item.id);
  state.todoList[index].completed = !state.todoList[index].completed;
};
</script>
```

src/components/InputTodo.vue

```
<template>
…
</template>

<script setup>
import { ref } from 'vue';

const emit = defineEmits(['add-todo']);
const todo = ref('');

const addTodoHandler = () => {
  if (todo.value.length >= 3) {
    emit('add-todo', todo.value);
    todo.value = '';
  }
};
</script>
```

defineProps defineEmits는 임포트없이 사용가능하죠?
전역함수죠?

src/components/TodoList.vue

```
<template>
  <div class="row">
    <div class="col">
      <ul class="list-group">
        <TodoListItem v-for="todoItem in todoList" :key="todoItem.id"
          :todoItem="todoItem"
          @delete-todo="emit('delete-todo', todoItem.id)"
          @toggle-completed="emit('toggle-completed', todoItem.id)"
        />
      </ul>
    </div>
  </div>
</template>

<script setup>
import TodoListItem from './TodoListItem.vue';

const props = defineProps({
  todoList: { type: Array, required: true },
});

const emit = defineEmits(['delete-todo', 'toggle-completed']);
</script>
```

부모에게 넘겨줄 데이터(==인자값)

등록한 emit활용

reactive([ ]) 될 것을
왜 객체의 속성 형태로 했을까???

src/components/TodoListItem.vue

```
<template>
  <li
    class="list-group-item"
    :class="{ 'list-group-item-success': todoItem.completed }"
    @click="emit('toggle-completed', todoItem.id)" >
    <span class="pointer" :class="{ 'todo-done': todoItem.completed }">
      {{ todoItem.todo }} {{ todoItem.completed ? '(완료)' : '' }}
    </span>
    <span
      class="float-end badge bg-secondary pointer"
      @click.stop="emit('delete-todo', todoItem.id)"
      >삭제</span>
  </li>
</template>

<script setup>
const props = defineProps({
  todoItem: { type: Object, required: true },
});

const emit = defineEmits(['delete-todo', 'toggle-completed']);
</script>
```

부모에게서 받을
데이터명과 정의
그리고 검증

props: { ~: ~~, ~: ~~, ... }

예를 들어 페이지 네이션에 필요하고, 데이터가 서버에 있다고 치자.

todoList = reactive([ ]); 시 참조가 변화면 안되기에
한꺼번에 바꾸는 연산인 todoList= [새로운 배열 정보]; 연산이 안됨.
참조말고 속성값을 바꾸는 것은 맞지만
todoList= reactive({ todo: [ ] });로하면
todoList.todo= [새로운 배열 정보]; 연산이 가능

혹은 todoList= ref([ ]);로 해도
한방에 참조를 바꿀 수 있다.
todoList.value= [새로운 배열 정보] ; 연산이 가능.
그 전 파일에서 말한 참조형인데 ref가 좋을 때가 이때다