Watch

우리는 종종 반응형 상태가 변경 되었을때에 감지하여 다른 작업(api call 등)을 수행해야 하는 경우가 있습니다.

예를 들어 어떠한 상태가 변경 되었을때 DOM을 변경하거나 비동기 작업을해서 다른 상태를 변경하는 것입니다.

- `watch` **함수**를 사용하여 반응형 속성이 변경될 때마다 함수를 실행할 수 있습니다:
- watch의 경우 실제 데이터 변경이 일어나기 전까지는 실행되지 않습니다.(초기에 할당된 값이 반드시 변경이 일어나야만 watch가 실행됩니다.

Watch Source Type

```
// 첫 번째 매개변수는 다양한 타입이 될 수 있습니다.
// (ref, reactive, computed, getter함수, array)
watch(/* Source Type */, (newValue, oldValue) => {});
```

src/views/ VueWatch.vue

ref가 하나일 경우

```
<template>
  <div>
    <button type="button" @click="changeX">{{ x }}</button>
  </div>
</template>

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

export default {
  setup() {
    const x = ref(0)

    // single ref
```

```
    watch(x, (newX) => {
      console.log(`x is ${newX}`)
    }, { immediate: true }) //immediate옵션을 사용하여 최초에 즉시실행 할 수 있다

      //x값을 변경하는 함수
    const changeX = () => {
      x.value++; // x값을 증가
    }

    return {
      x,
      changeX
    }
  }
}
</script>
```

getter타입

```
<template>
  <div>
    <button type="button" @click="changeX">{{ x }}</button>
  </div>
</template>

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

export default {
  setup() {
    const x = ref(0)
    const y = ref(10)

    watch(x, (newX) => {
      console.log(`x is ${newX}`)
    })

      // getter
```

```
    watch(
      () => x.value + y.value,
      (sum) => {
        console.log(`sum of x + y is: ${sum}`)
      }
    )

    const changeX = () => {
      x.value++;
    }

    return {
      x,
      changeX
    }
  }
}
</script>
```

source가 여러개 일 경우

```
<template>
  <div>
    <button type="button" @click="changeX">{{ x }}</button>
  </div>
</template>

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

export default {
  setup() {
    const x = ref(0)
    const y = ref(10)

    watch(x, (newX) => {
      console.log(`x is ${newX}`)
    })
```

```
    // getter
  watch(
    () => x.value + y.value,
    (sum) => {
      console.log(`sum of x + y is: ${sum}`)
    }
  )

  // array of multiple sources
  watch([x, () => y.value], ([newX, newY]) => {
    console.log(`x is ${newX} and y is ${newY}`)
  })

  const changeX = () => {
    x.value++;
  }

  return {
    x,
    changeX
  }
 }
}
</script>
```

반응형 객체의 속성은 볼 수 없습니다.

```
import { ref, watch, reactive } from 'vue';

const obj = reactive({ count: 100 });

watch(obj.count, (newValue) => {
  console.log(newValue)
}, { immediate: true })
```

```
//getter를 사용하면 됩니다
watch(() => obj.count, (newValue) => {
  console.log(newValue)
}, { immediate: true })
```

버튼을 클릭으로 이름 변경

```
<template>
  <div>
    <!-- 버튼을 클릭하면 이름이 변경됩니다 -->
    <button type="button" @click="changeName">변경</button>
    <h1>{{ fullName }}</h1>
  </div>
</template>

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

export default {
  setup() {
    const firstName = ref('Minkyu')
    const lastName = ref('Jang')
    const fullName = ref('')

    // firstName과 lastName이 변경될 때 fullName을 업데이트
    watch([firstName, lastName], () => {
      fullName.value = firstName.value + ' ' + lastName.value;
    })

    // 이름을 변경하는 함수
    const changeName = () => {
      firstName.value = 'Eunsol' // firstName을 Eunsol로 변경
    }

    // 초기 fullName을 설정
```

```
    fullName.value = firstName.value + ' ' + lastName.value;

    return {
      firstName,
      lastName,
      fullName,
      changeName
    }
  }
}
</script>
```

실습 예제(watch)

노란 부분을 제외한 부분은 같이 작성 → 문제는 맨 밑에 있음

router/index.js

```
...
import WatchEx from './views/WatchEx.vue';
import Profile from './views/Profile.vue';

const routes = [
  ...
  { path: '/watchex', component: WatchEx },
  { path: '/profile', component: Profile }
];
```

App.vue

```
...
<span> | </span>
<RouterLink to="/watchex">WatchEx</RouterLink>
```

views/Profile.vue

```
<template>
  <div>
    <h1>Profile Page</h1>
    <p>Welcome to your profile!</p>
  </div>
</template>
```

views/WatchEx.vue

```
<template>
  <div>
    <h1>Home Page</h1>
    <label>FirstName: <input v-model="firstName" type="text" /></label>
    <label>LastName: <input v-model="lastName" type="text" /></label>
    <p>Full Name: {{ fullName }}</p>
  </div>
</template>

<script>
import { ref, computed, watch } from 'vue';
import { useRouter } from 'vue-router';

export default {
  setup() {
    const firstName = ref('');
    const lastName = ref('');
    // this.$router.push는 setup 안에서는 사용할 수 없으므로, `router`를 직접 사용
해야 함
    const router = useRouter();

    // fullName은 firstName과 lastName을 결합하여 계산됨
```

```
    const fullName = computed(() => {
      return firstName.value + ' ' + lastName.value;
    });

    // watch로 fullName을 감시하고, fullName이 특정 조건을 만족하면 Profile 페이지로 이동
    watch(fullName, (newFullName) => {
      if (newFullName === 'Minkyu Jang') {
        // fullName이 'Minkyu Jang'이면 Profile 페이지로 라우팅
        router.push('/profile');
      }
    });

    return {
      firstName,
      lastName,
      fullName
    };
  }
};
</script>
```

문제 : watch로 fullName을 감시하고, fullName이 특정 조건(본인 이름)을 만족하면 Profile 페이지로 이동

앞서 배웠던 computed와 watch는 비슷한 역할을 하고 있다

- Vue 인스턴스의 상태(ref, reactive 변수)의 종속 관계를 자동으로 세팅하고자 할 때는 `computed`로 구현하는 것이 좋다.

```
//예시
//reverseMessage는 message값에 따라 결정되어지는 종속관계에 있다
const reverseMessage = computed(() =>
  message.value.split('').reverse().join('')
);
```

- Vue 인트턴스의 상태(ref, reactive 변수)의 변경 시점에 특정 액션(call api, push route 등)을 취하고자 할때 적합하다.

  대부분 computed로 구현이 가능한 것이라면 computed로 구현 하는게 대부분 옳다