(디지털 컨버전스) 스마트 콘텐츠와 웹 융합 응용 SW개발자 양성과정

훈련기간 : 2021.05.07 ~ 2021.12.08

KH정보교육원 | 전승리

앞으로 출력은 올리지 않겠다 어차피 코드로 예상 안되면 다시 공부해야된다.

v-for문

HTML

```
<div id="app">
    <p>{{ message }}</p>
    <ol>
        <li v-for="item in list">{{ item }}</li>
    </ol>
```

Vue

```
var app = new Vue ({
    el: '#app',
    data: {
        message: '우왕 뷰 짱',
        initMsg: '양방향 연결이 뭔지 함 볼까 ?',
        list: ['사과', '바나나', '딸기', '수박', '참외', '포도', '망고', '블루베리', '체리'],
```

Clickbutton

HTML

```
<button v-on:click="buttonClickTest">클릭해봐!</button>
```

Vue

```
buttonClickTest: function (event) {
    alert('뷰 짱')
},
```

HTML

```
<button v-on:click="increment">숫자를 카운팅 해보자!</button>
<p>{{ count }}번 클릭했습니다.</p>
```

Vue

```
data: {
count: 0,
```

```
methods: {
increment: function () {
    this.count += 1
},
```

v-model 양방향 연결

비슷한 예 이전 게시판 작성에 사용한 Thymleaf의 th:field *{} 같은 느낌

HTML

```
data: {
    message: '우왕 뷰 짱',
    initMsg: '양방향 연결이 뭔지 함 볼까 ?',
```

Vue

```
<input v-model="initMsg">
<p>{{ initMsg }}</p>
```

Transition

HTML

```
<transition>
    <p v-if="show">invisibility도 쉽게 제어 되네 ?</p>
</transition>
```

Vue

```
.v-enter-active, .v-leave-active {
transition: opacity 2s;
}

/* 투명도를 2초동안 Fade In & Fade Out */
.v-enter, .v-leave-to {
    opacity: 0;
}
```

v-enter: 실제 HTML 요소가 삽입되기 이전에 적용되고 한 프레임 후에 제거 됨
v-enter-active: enter에 대한 활성 및 종료 상태에서
HTML 요소가 삽입 되기 이전에 적용됨
transition 이 완료되면 제거됨

list[] 사용가능

HTML

```
<p>Hello {{ message }}</p>
<p>{{ message.length }}</p>
<p>{{ list[3] }}</p>
<p>{{ list[num] }}</p>
```

Vue

```
data: {
    message: '우왕 뷰 짱',
    initMsg: '양방향 연결이 뭔지 함 볼까 ?',
    list: ['사과', '바나나', '딸기', '수박', '참외', '포도', '망고',
    show: true,
    num: 3,
```

SVG, XML에 대해

svg는 백터(vector) 이미지를 표현하기 위한 포맷으로 w3c에서 만든 백터 이미지 표준입니다. SVG 자체는 CSS가 아닙니다만

기존의 HTML과 달리 웹상에서 구조화된 문서를 전송가능하도록 설계되었다. 이게 무슨 뜻이냐면 예를 들어 HTML에서는 CPU 2.83GHz라는 데이터를 표기할 때 어디부터가 데이터 명이고 어디부터가 실제 데이터인지 표시할 수 있는 마땅한 방법이 없다.
즉, 데이터에 의미를 부여하는 메타데이터를 기술할 수 있다. XML은 바로 이러한 목적으로 탄생했다. 위의 예를 XML로 바꾸면 데이터 명은 <dataname>CPU</dataname>가 되고 데이터 값은 <datavalue>2.83</datavalue>이 된다

쉽게 말해 데이터의 구조를 기술하는 언어의 한 가지이다.

v-bind

v-model과 다르게 값을 단방향 입력받음

HTML

```
data: {
radius: 50,
```

Vue

```
<p>{{ count }}번 클릭했습니다.</p>
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
    <circle cx="100" cy="75" v-bind:r="radius" fill="lightblue"/>
</svg>
<input type="range" min="0" max="100" v-model="radius"><br><br><br>
```

SVG, XML에 대해

reduce

reduce()를 사용하면 알아서 내부에 있는 요소를 전부 낱개로 쪼개서 하나하나 비교하게됨
그래서 function(a, b)에 입력으로 들어가는 a, b는
 모두 monsters 배열에 있는 각각의 낱개 요소들임
계속 낱개 단위로 끝까지 비교를 한다면
reduce는 전체 순회를 끝낼때까지 동작이 값이 증발되지 않으며
최종 결과를 얻을때까진 이전 결과를 유지하며 연산이 진행됨

Vue

```
var max = this.monsters.reduce( function (a, b) {
    return a > b.id ? a : b.id
}, 0)
```

```
var arr = [1, 2, 3, 4];
// reduce ( function (유지되는 값, 반복하며 비교하거나 연산 대상이 되는 값))
var sum = arr.reduce( function (accumulator, currentValue) {
    alert(accumulator)
    return accumulator + currentValue
}, 0)

alert(sum)
```

arr.reduce(function (accumulator, currentValue)
에서 초기 accumulator값은 0부터 시작한다.
차례로 배열에 들어가서 진행되는 과정
sum = 0 + 0
sum = 1 + 2
sum = 3 + 3
sum = 6 + 4
최종 sum = 10

VDOM

```
beforeCreate() {
    console.log('Vue 객체를 만들기 이전입니다.')
},
created() {
    console.log('Vue 객체를 만들었습니다.')
},
beforeMount() {
    console.log('HTML 요소를 붙이기 전입니다.')
},
mounted() {
    console.log('HTML 요소를 붙입니다.')
},
beforeUpdate() {
    console.log('VDOM의 변화를 감지합니다.')

    var i
    for (i = 0; i < this.monsters.length; i++) {
        if (this.monsters[i].hp <= 0) {
            this.monsters.splice(i, 1)
        }
    }
},
updated() {
    console.log('VDOM의 변화를 적용합니다.')
},
beforeDestroy() {
    console.log('Vue 객체를 파괴하기 이전입니다.')
},
destroyed() {
    console.log('Vue 객체를 파괴하였습니다.')
}
```

자바스크립트 코드를 보면 전체에서 찾기때문에 시간이 뷰에비해 상대적으로 오래걸림.
뷰에서는 가상돔을 사용해서 사용하지않는 객체를 넣고 빼고 하는 것이기때문에 빠름.

```
Vue 객체를 만들기 이전입니다.
Vue 객체를 만들었습니다.
HTML 요소를 붙이기 전입니다.
HTML 요소를 붙입니다.
```

```
VDOM의 변화를 감지합니다.
VDOM의 변화를 적용합니다.
VDOM의 변화를 감지합니다.
VDOM의 변화를 적용합니다.
```