



# 7/13 ~ 7/17 2주차 - 3

## Vue 객체에 대해서

### Vue 객체

- MVVM패턴에서 ViewModel 을 담당하는 Vue.js의 핵심 객체라고 말할 수 있다.
- Vue 생성자 함수에 옵션 객체를 전달해서 생성한다.

```
let vi = new Vue(  
  {  
    el : '#app',  
    data : model  
  }  
);
```

속성 명	설명
el	Vue로 만든 화면이 그려지는 인스턴스의 시작 지점 - CSS 선택자로 요소 지정(여러 개가 선택되더라도 맨 처음 요소만 사용됨 → ID 기반 접근 필요)
data	인스턴스의 데이터 속성으로 객체를 이용해 여러 값 저장
template	화면에 표시할 HTML, CSS등 마크업 요소를 정의하는 속성 - Vue의 데이터 및 기타 속성들도 함께 화면에 렌더링
methods	이벤트 및 화면 동작 메서드로 속성들은 function
created...	라이프 사이클 커스터마이징 로직

이 속성들을 Vue객체 외부에서 접근하기 위해서는 \$options라는 속성을 사용한다.

참고로 vue객체의 속성중 \$가 붙은 속성들을 내장 속성이라고 함

다음은 Vue 객체의 \$options의 값을 콘솔에 출력한 예이다.

```
▼ Object ⓘ
  ▶ components: {}
  ▶ computed: {sum: f}
  ▶ data: f mergedInstanceDataFn()
  ▶ directives: {}
    el: "#app"
  ▶ filters: {}
  ▶ render: f anonymous( )
  ▶ staticRenderFns: []
  ▶ _base: f Vue(options)
  ▶ __proto__: Object
```

## el

- element를 나타내며 View를 연결하는 속성이다
- el을 지정할 때에는 CSS 선택자를 이용해서 HTML의 DOM 요소를 지정한다.
- 이때 반드시 하나만 지정되어야한다.
- 여러개가 선택된다해도 맨 처음의 요소만 활용된다.
- ID속성을 지정하는 #이 사용된다.

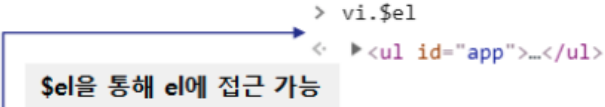
만약 Vue 객체 외부에서 el 요소에 직접 접근하기 위해서는 vue 객체가 가지는 \$el 속성을 이용한다.

```
<ul id="app">
  <li>{{message}}</li>
</ul>

<script>
  let model={
    message:"Hi Vue"
  }

  let vi = new Vue({
    el:"#app",
    data:model
  });

  console.log(vi.$el);
</script>
```



```
> vi.$el
< ><ul id="app">...</ul>
```

\$el을 통해 el에 접근 가능

## data

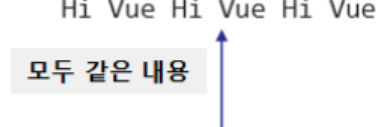
- Model을 연결하는 속성으로 JSON객체이다.
- Vue 객체 외부에서 data에 접근하기 위한 내장 옵션으로 #data를 사용할 수 있다.
- 하지만 좀더 편리하게 사용하기 위해서 data의 속성들은 모두 Vue 객체의 속성으로 관리된다.

```
<ul id="app">
  <li>{{message}}</li>
</ul>

<script>
  let model={
    message:"Hi Vue"
  }

  let vi = new Vue({
    el:"#app",
    data:model
  });

  console.log(model.message, vi.message, vi.$data.message);
</script>
```



```
Hi Vue Hi Vue Hi Vue
```

모두 같은 내용

## 반응형과 비 반응형 속성

- Vue는 model 값 변경시 결과가 화면에 바로 반영된다.

```
<script>
  let model = {
    emp : {
      name : "hong",
    },
    list : [ 1 ],
  };

  let vi = new Vue({
    el : "#app",
    data : model,
  });
</script>
```

```
<div id="app">
  <ul>
    <li>{{emp.name}}, {{emp.age}}</li>
    <li v-for="item of list">{{item}}</li>
  </ul>
</div>
```

- hong,
- 1

- emp와 list속성을 갖는 model이 선언되어있고 화면에서 사용과 출력은 맨 오른쪽에 있다.
- emp 객체에는 age속성이 없고 html에서 참조하려 하지만 실제 출력은 되지 않는다.
- 다만 emp의 name 속성을 홍길동으로 변경하면 잘 변경됨
- model 객체의 값을 변경하거나 \$data를 통해서 변경하거나 결과는 동일 🙌

```
> vi.emp.name="홍길동"
< "홍길동"
```

---

```
> vi.$data.emp.name="장길산"
< "장길산"
```

- 장길산,
- 1

하지만 model 객체의 속성 추가, 삭제 및 배열의 몇 가지 동작은 감지하지 못한다.

```
> vi.emp.age=30
```

모델 객체의 속성 추가

```
< 30
```

```
> vi.list[0]=1000
```

```
< 1000
```

```
> vi.list[1]=2000
```

배열 객체의 값 변경 및 추가

```
< 2000
```

```
> vi.list
```

```
< ▶ (2) [1000, 2000, __ob__: Observer]
```

- hong,
- 1

- 배열 형태로 데이터를 관리하다가 동적으로 변화하는 값을 반영하지 못한다면 문제가 될 수 있다.
- 이때는 Vue.set(obj, key, newValue) 함수를 이용해 비 반응형 속성으로 지정해서 처리해야함.

Vue.set함수는 vi.\$set 함수 형태로도 사용할수 있다.

## computed

- 복잡한 계산식의 결과를 바인딩하며 마치 data의 속성처럼 이름으로 사용한다.
- 이미 계산된 속성이기 때문에 별도로 실행하지 않는다.
- 책에서 읽은 바에 따르면 계산식에 포함된 값이 변경될 때 계산을 해놓는다.
- Vue외부에서 접근하기 위한 내장 객체는 \$options의 computed 속성을 이용한다.

```

<ul id="app">
  <li>{{message}}</li>
  <li>{{sum}}</li>
</ul>

```

```

<script>

```

```

  let model={ message:"Hi Vue"}

```

```

  let vi = new Vue({

```

```

    el:"#app",

```

```

    data:model,

```

```

    computed:{

```

```

      sum(){

```

```

        let result = 0;

```

```

        for(let i=1; i<=10; i++){

```

```

          result+=i;

```

```

        }

```

```

        return result;

```

```

      }

```

```

    }

```

```

  });

```

```

</script>

```

sum() 처럼 실행하지 않아도 동작  
data의 message 처럼 사용

```

> vi.$options.computed.sum

```

```

< f sum() {

```

```

  let result = 0;

```

```

  for (let i = 1; i <= this.num; i++) {

```

```

    result += i;

```

```

  }

```

```

  return result;

```

```

  ...

```

```

> vi.sum

```

```

< 1

```

→ computed 속성은 Vue 객체의 \$options.computed.sum으로 확인해보면 실제 동작할 함수가 출력되고 Vue 객체에는 sum으로 계산된 값이 연결되어 있음을 확인할 수 있다.

## methods

- Vue객체에서 사용할 메서드들을 객체로 등록하는 속성.

등록된 메서드는 Vue 객체에서 직접 호출하거나 디렉티브 표현식 등에서 data처럼 사용 가능하다.

```
<div id="app">
  1부터 <input type="number" v-model="num"/> 의 합은 {{sum()}}
</div>
```

computed는 선언만 했었죠!

```
<script>
let vi = new Vue({
  el:"#app",
  data:{num:0},
  methods:{
    sum(){
      let result=0;
      for(let i=1; i<=this.num; i++){
        result+=i;
      }
      return result;
    }
  }
});
</script>
```

- computed에서 sum을 썼을 때는 단순히 선언만 했지만 methods의 속성들은 반드시 ()를 이용해서 호출해줘야 한다.
- methods의 내용을 화면에도 사용될 수 있고 이벤트 발생시 동작할 핸들러를 작성할 때 도 사용된다.

## 주의할 점

- 화면에서 methods의 내용을 사용할 때 화면이 다시 렌더링 될 때마다 화면에서 사용되고 있는 method들은 모두 다시 실행됨 🔍

## computed vs methods

- computed와 methods와 가장 큰 차이점  
→ 바로 computed는 캐싱된 값을 사용한다.
- computed는 종속 대상이 변경되지 않는 한 다시 값을 계산하지 않음.
- 값이 자주 변경되지 않는 경우 computed, 값이 자주 변경되서 캐시를 유지할 필요없을 경우는 methods를 사용

- 그리고 앞서 설명했던 것처럼 methods는 함수처럼 실행해주고 computed는 속성처럼 선언한다.

## watch

- watch도 computed, methods 와 유사하게 함수들을 관리하는 객체이다.
- watch는 이름 그대로 지켜보는 것이다. → 데이터를 본다.
- 데이터를 지켜보다가 값이 변경되면 무언가 처리할 함수들을 관리한다.
- watch는 computed와 유사하게 하나의 데이터를 기반으로 다른 데이터를 변경할 때 사용한다.

## computed와의 차이점

- computed는 동기로 값을 리턴하지만 watch는 비동기로 값을 리턴하지 않는다.
- computed는 선언되어야 동작하지만 watch는 모델이 변경되면 뒷단에서 바로 동작한다
- → 즉 데이터를 보고 있다가 알아서 움직인다.



watch에 함수를 등록할 때 함수의 이름은 변경을 모니터링할 객체의 속성 이름과 같아야 한다. 함수에는 파라미터를 선언할 수 있는데 파라미터에 변경된 값이 전달된다.

```
<div id="app">
  <input type="number" v-model="x">+<input type="number" v-model="y">={{sum}}
</div>
<script>
  let vi = new Vue({
    el:"#app",
    data:{x:0, y:0, sum:0},
    watch:{
      x(nv){
        console.log("x 변경됨!!");
        this.sum = parseInt(nv)+parseInt(this.y);
      },
      y(nv){
        console.log("y 변경됨!!");
        this.sum = parseInt(this.x)+ parseInt(nv);
      }
    }
  });
</script>
```