1-1~2

const app = new Vue({

        el:'#root',

        data:{

            liked:false,

        },

        methods:{

            onClickButton(){

                //method에서 this는 data를 가르킨다.

                this.liked = true;

            },

        },

    });

Vue, react를 할 때는 Data만 관리한다.

Data 중심으로 생각하는 방식을 가져야 한다.

Data만 관리하면 화면은 Vue가 알아서 바꿔주는 것이 Vue의 강점이다.

<div v-if="liked">

V가 붙은 속성의 값(따옴표 안)에는 자바스크립트 코드를 쓸 수가 있다.

1-3~4

<span>{{first \* 100}} 이렇게 적는것도 가능하다.</span>

<input type="number" v-model="value">

v-model으로 input값과 data를 연결할 수 있다.

항상 어느 부분이 바뀌는지 생각해야 한다. (바뀌는 data를 항상 생각해야 한다.)

<input type="number" v-model="value" ref="answer">

this.$refs.answer.focus();

input 태그에 focus를 주고 싶은대 focus 같은 경우는 data라 보기 애매하다.

그래서 vue가 식별할 수 있게 ref 속성에 값을 주고, javascript에서 $refs.이름.focus();

이렇게 할 수 있다.

주의!! : $refs 으로 data를 바꾸면 vue가 자동으로 값을 바꿔주지 않아서 화면의 불일치가 일어나니 위와 같은 행동은 하지 않는 것이 좋다.

1-5

항상 화면은 처음에 만들어놓고 데이터 중심으로 생각!!

1-6

<div v-if="route === 'main'">메인화면</div>

    <div v-else-if="route === 'sub'">서브화면</div>

    <div v-else-if="route === 'third'">세번째화면</div>

data:{

            word:'제로초',

            result:'',

            value:'',

            route:'main',

와 같은 형식으로 화면 전환하며 보여주기 가능하다.

2-1

반복되는 부분은 컴포넌트로 만든다.

<script>

Vue.component('word-relay', {

data:{

},

methods:{

}

})

</script>

컴포넌트를 생성할 때는 data 부분에서 다른점이 존재한다.

2-2

데이터를 Vue 인스턴스에서는 객체형식으로 표현해왔는데, 컴포넌트에서는 함수 형식으로 만들어야 한다.

template:`

<div>{{word}} </div>

<form v-on:submit="onSubmitForm">

<input type="text" v-model="value" ref="answer">

<button type="submit">입력!</button>

</form>

<div>{{result}}</div>

`,

template에서는 위 처럼 동등한 형제 테그가 있으면 안된다.

template:`

<div>

<div>{{word}} </div>

<form v-on:submit="onSubmitForm">

<input type="text" v-model="value" ref="answer">

<button type="submit">입력!</button>

</form>

<div>{{result}}</div>

</div>

위처럼 template은 항상 하나의 부모 테그로 묶어줘야한다는 제약조건이 있다.

컴포넌트는 같은 것을 랜더링하지만 중복을 제거하는 것. 컴포넌트로 생성된 서로 다른 것들은 서로 독립적인 각각의 data를 가진다.

컴포넌트를 만드는 스크립트를 어떤 스크립트보다 상단에 두어야 한다.

2-3

댓글 컴포넌트가 있다면 쓴사람, 쓴내용, 쓴날짜가 다름

그럴때는 아래처럼 컴포넌트를 쓰는 쪽에서 startWord를 넣어주고

<word-relay startWord="제로초"> </word-relay>

컴포넌트를 정의하는 부분에서 props에 startWord를 받겠다 정의할 수 있다.

props: ['startWord'],

그러면 data에서 아래와 같이 접근할 수 있다.

data(){

return{

word:this.startWord,

**특이사항은**

**<word-relay start-word="바보"> </word-relay>**

**Vue의 경우 HTML 속성 부분에 케밥케이스로 작성하여야 한다.**

**props: ['startWord'],**

**그러면 자바스크립트 소스 부분에서는 자동으로 카멜케이스로 바뀐다.**

**엄청 특이하다..**

이러면 공통된 모양을 가지지만 데이터는 따로 따로 가지기에 따로따로 논다.

3-1 웹팩 사용하기

Node를 설치해야 한다.

콘솔에서 npm init 입력하고 설정 입력하면

Package.json 파일이 생성된다.

이걸 사용하는 이유는, 실무를 하다보면 각각의 소스코드마다 버전이 있다.

실무에서 남의 소스코드를 백개 이상 쓰는 경우 해당 소스코드마다 버전관리하기가 엄청 힘들다.

<Script> 태그가 너무 많아지는 것이 힘들어서 웹펙을 쓰는것이다.

npm으로 package 관리를 할 것이다.

명령어 : npm install vue

명령어 : npm install webpack webpack-cli -D(--save-dev)

-d 옵션은 개발할때만 webpack이랑 webpack-cli를 쓰겠다는 의미이다.

Npm install로 라이브러리 설치하면 해당 위치에 node\_modules 폴더랑 package-lock.json 파일이 생긴다.

Webpack.config.js에 webpack에 대한 설정들을 적는 것이다.

Webpacking을 할 때 위 파일에 만들어놓은 모듈을 사용한다.

**웹팩 설정은 크게 4개로 설정한다.**

1. Entry:{},
2. Module:{},
3. Plugins:[],
4. Output:{},
5. module.exports = {
6. entry:{
7. app:'./main.js',
8. },
9. module:{
10. },
11. plugins:[],
12. output:{
13. filename: 'app.js',
14. path: './dist',
15. },
16. };
17. Entry 에서 main.js에 존재하는 스크립트 여러 개가 app으로 합쳐서 사용할 수 있게 된다.
18. Main.js에서 꼬리를 물고 연결된 파일들이 output으로 dist/app.js으로 합쳐진다.
19. Module이 웹팩의 핵심이다.  
    자바스크립트 파일들을 합칠 때 어떻게 합칠 건지, 어떻게 처리할 건지를 rules:[{}] 안에다 적어준다.