컴포넌트 Basic

생성일	@2024년 5월 16일 오후 1:03
태그	

- 컴포넌트란?
  - View(HTML), Data, Code(JavaScript)의 세트
  - 재사용이 가능 ⇒ 다른 컴포넌트에 import해서 사용

▼ 컴포넌트 구조 이해하기

▼ 컴포넌트 기본 구조

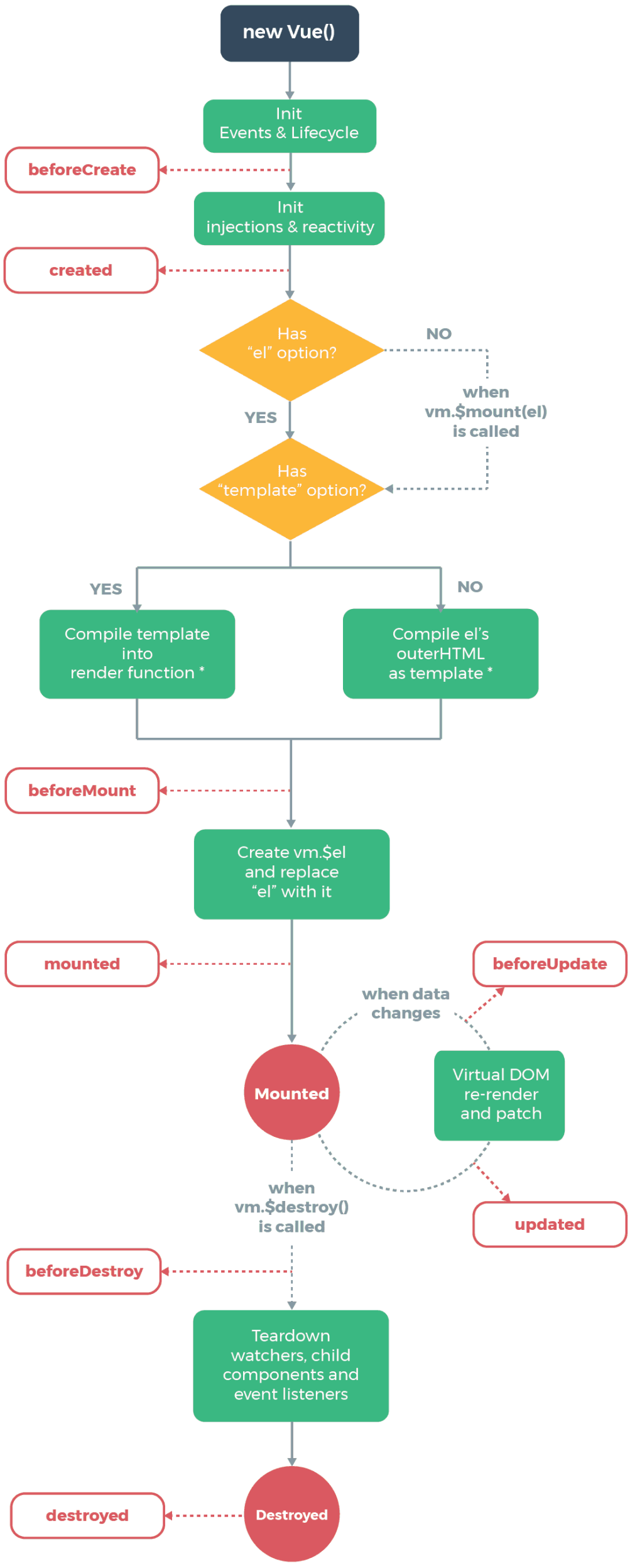
```
<template>
  <div></div>
</template>

<script>
export default {
  components: {}, // 다른 컴포넌트 사용 시 컴포넌트를 import하고, 배열로 저장
  data() {
    // HTML과 JS 코드에서 사용할 데이터 변수 선언
    return {
      sampleData: '',
    };
  },
  setup() {}, // Composition API
  created() {}, // 컴포넌트가 생성되면 실행
  mounted() {}, // template에 정의된 HTML 코드가 렌더링된 후 실행
  unmounted() {}, // unmount가 완료된 후 실행
  methods: {}, // 컴포넌트 내에서 사용할 메소드 정의
};
</script>
```

항목	설명
<template>	View에 해당하는 html 코드 작성
name	컴포넌트 이름
components	외부 컴포넌트를 사용하게 되면 해당 컴포넌트를 import 한 후 이곳에 배열로 등록
data	- html 코드와 JS 코드에서 전역 변수로 사용하기 위해 선언하는 데이터 - 데이터 바인딩을 통해 html과 JS 코드 간 양방향 통신 지원 - 이곳에 정의된 변수는 this를 통해 접근
setup	컴포지션 API 구현
created	컴포넌트 생성 시 실행
mounted	템플릿에 작성한 HTML 코드가 랜더링 된 후 실행
unmounted	컴포넌트를 빠져나갈 때 실행
methods	컴포넌트 내에서 사용할 메소드, this를 통해서 접근

▼ Lifecycle Hooks

▼ 라이프사이클(lifecycle)

- Vue 인스턴스나 컴포넌트가 생성되며 거치는, 미리 사전에 정의된 몇 단계의 과정
- Vue 인스턴스가 생성된 후 우리 눈에 보여지고, 사라지기까지의 단계
- 과정



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

▼ Lifecycle Hooks

- 각각의 단계에서 훅(Hook)을 할 수 있도록 API를 제공
- 종류

훅(Hook)	설명
beforeCreate	Vue 인스턴스가 초기화 된 직후에 발생 - 컴포넌트가 DOM에 추가되기도 전이어서 this.$el에 접근 불가 - data, event, watcher에도 접근하기 전이라 data, methods에도 접근 불가
created	인스턴스가 모든 상태 관련 옵션 처리를 완료된 후 발생 - data를 반응형으로 추적할 수 있음 - computed, methods, watch 등이 활성화되어 접근 가능 - 사용 1) 컴포넌트 초기에 외부에서 받아온 값들로 data를 세팅 2) 이벤트 리스너를 선언
beforeMount	DOM에 부착하기 직전에 호출 - 템플릿을 렌더링 한 상태로 가상 DOM이 생성 - 실제 DOM에 부착되지는 않은 상태
mounted	가상 DOM의 내용이 실제 DOM에 부착되고 난 이후에 실행 - data, computed, methods, watch 등 모든 요소에 접근 가능
beforUpdate	data의 값이 변해 DOM에도 그 변화를 적용시키기 직전에 호출
updated	가상 DOM을 렌더링 하고 실제 DOM이 변경된 이후에 호출 - 변경된 data가 DOM에도 적용된 상태
beforDestroy	인스턴스가 해체되기 직전에 호출 - 아직 인스턴스가 작동하므로 모든 속성에 접근 가
destroyed	인스턴스가 해체되고 난 직후에 호출 - 인스턴스의 속성에 접근 불가

  - 사용
    - created
      - 화면 접속 시 제일 먼저 보여줘야 하는 데이터를 서버에서 받아옴
    - mounted
      - 화면 로딩 이후에 삽입되어도 되는 데이터 혹은 HTML 객체 정의

▼ 데이터 바인딩

▼ 사용하는 경우

- 데이터가 html tag 안에 텍스트로 바인딩되는 경우
- 데이터가 html tag의 속성(attribute)로 바인딩되는 경우
- 데이터가 html의 Form element의 value에 바인딩되는 경우
- 다중 데이터가 html의 다중 element를 생성하기 위해서 바인딩 되는 경우

- 문자열 데이터 바인딩
  - 이중괄호

```
{{ property }}
```

- raw(원시) HTML 데이터 바인딩
- Form 입력 데이터 바인딩

▼ v-model 디렉티브

```
<input-tag v-model="property"></input-tag>
```

- 주의사항
  - 내부적으로 서로 다른 속성을 사용

    - 서로 다른 입력 요소에 대해 서로 다른 이벤트를 전송

태그	바인딩 속성	바고
input type=text	value	
input type=number	value	수식어(.number)로 타입변환
TextArea	value	
Select	value	
체크박스 (input type=checkox)	checked	체크되었을 때 값을 설정 ⇒ v-bind:value 사용 (혹은 true-value / false-value 사용)
라디오 (input type=radio)	checked	체크되었을 때 값을 설정 ⇒ v-bind:value 사용

▼ v-bind 디렉티브

```
<tag v-bind:attribute="property"></tag>
```

- 속성(Attribute)
    - Img 객체의 src
    - button 객체의 disabled
        - 데이터 타입 : boolean
        - 조회화면에서 조회 조건 중 필수 입력 조건이 모두 입력되었을 때만 활성화
        - 등록화면에서 필수 입력 조건이 모두 입력되었을 때만 활성화
        - 권한이 있는 사용자에게만 허용되는 버튼을 활성화
- 클래스 바인딩
    - 클래스 속성의 경우 다중 값을 허용
    - 사용방법
        - 반드시 적용되어야 하는 값은 기존 html에 사용하던 방식 사용
        - 조건에 따라 바인딩할 값의 경우 v-bind 디렉티브를 사용

```
// 객체를 사용하여 바인딩될 값이 적용여부를 제어
<tag v-bind:class="{ property : true }"></tag>

// 다중 값이 필요한 경우 배열을 사용할 수 있음
<tag v-bind:class="[property1, property2, ...]"></tag>
```

- 인라인 스타일 바인딩

```
// 객체를 사용하여 바인딩
// 단, 기존 css 적용시 사용했던 속성과 값은 그대로 사용
//     케밥 표기법은 카멜 표기법으로 변환해서 작성
<tag v-bind:class="{ property : value }"></tag>

// 다중 값이 필요한 경우 배열을 사용할 수 있음
<tag v-bind:class="[property1, property2, ...]"></tag>
```

- 리스트 랜더링(v-for)
    - 다중 데이터를 처리하는 부분

```
<template v-for="(item, index) in items">
    <tag v-bind:attribute="item"></tag>
</template>
```

```
items : 데이터 배열로, 컴포넌트가 가지고 있는 프로퍼티
item  : 반복되면서 하나씩 읽어드린 데이터를 담는 임시변수
index : 배열의 현재 index 값
```

▼ 랜더링 문법(v-if, v-show)

- 조건에 따라 렌더링 하는 방법

```
<tag v-if="property == 'A'"></tag>
<tag v-else-if="property == 'B'"></tag>
<tag v-else></tag>

// v-if, v-else-if, v-else 디렉티브 사이에 관련 디렉티브가 없는 태그가 존재 불가
```

- v-if
- v-show 보이느냐 보이지 않느냐

```
<tag v-show="property"></tag>
```

- v-if 와 v-show의 차이점
  - 렌더링 시 태그의 생성 여부
    - v-if
      - 조건이 만족될 경우에만 태그를 생성
    - v-show
      - 조건 만족 여부와 상관없이 태그를 생성
      - 조건에 따라 css의 display 속성 값을 제어
  - 사용 시 선택 기준
    - 페이지 내에서 toggle의 빈도 수

▼ 이벤트 처리(v-on)

- 심볼 @으로 사용가능

```
// default
<tag v-on:event="EventHandler" />

// 심볼 @ 사용
<tag @event="EventHandler" />

// 필요에 따라 매개변수를 넘기는 형태로 구현가능
<tag @event="EventHandler(parameter)" />

// 하나의 이벤트에 여러 개의 이벤트핸들러를 연결할 수 있음
<tag @event="EventHandler1(), EventHandler2()" />

// 이벤트 객체가 필요한 경우
<tag @event="EventHandler($event)"/>
```

- 주요 이벤트
  - Click 이벤트
  - Change 이벤트
  - Key 이벤트

```
<tag @keyup.target="EventHandler"/>

<tag @keyup.system.target="EventHandler"/>

// target : 입력키 수식어
// system : 시스템 입력키 수식어
```

▼ 입력키 수식어

- .enter
- .tab
- .delete (키보드에서 Del키, Backspace키)
- .esc
- .space
- .up
- .down
- .left
- .right

▼ 시스템 입력키 수식어

- .ctrl
- .alt
- .shift
- .meta (키보드에서 window 키)

◦ 이벤트 수식어

- .stop (Event 객체의 stopPropagation() 과 같은 역할)
- .prevent (Event 객체의 preventDefault() 와 같은 역할)

▼ computed와 watch

- 공통점
  - Vue 인스턴스 내의 정의된 데이터 값의 변경을 감시
  - 변경될 때마다 정의된 함수가 실행
- computed
  - 함수이자 동시에 Vue 인스턴스의 데이터
  - 기존에 정의된 데이터 값을 기반으로 새로운 데이터 값 활용
  - 실행되는 경우
    - 인스턴스 생성 시 최초 실행
    - 사용(호출)과 상관없이
      참조하는 데이터 값이 변경되었을 때만 동작
- watch
  - 정의된 데이터 값 하나만을 감시하기 위한 용도
  - 실행되는 경우
    - 실제 데이터 변경이 일어났을 때만 동작