

# 원쌤의 Vue.js 퀵스타트

## 4. Vue 인스턴스



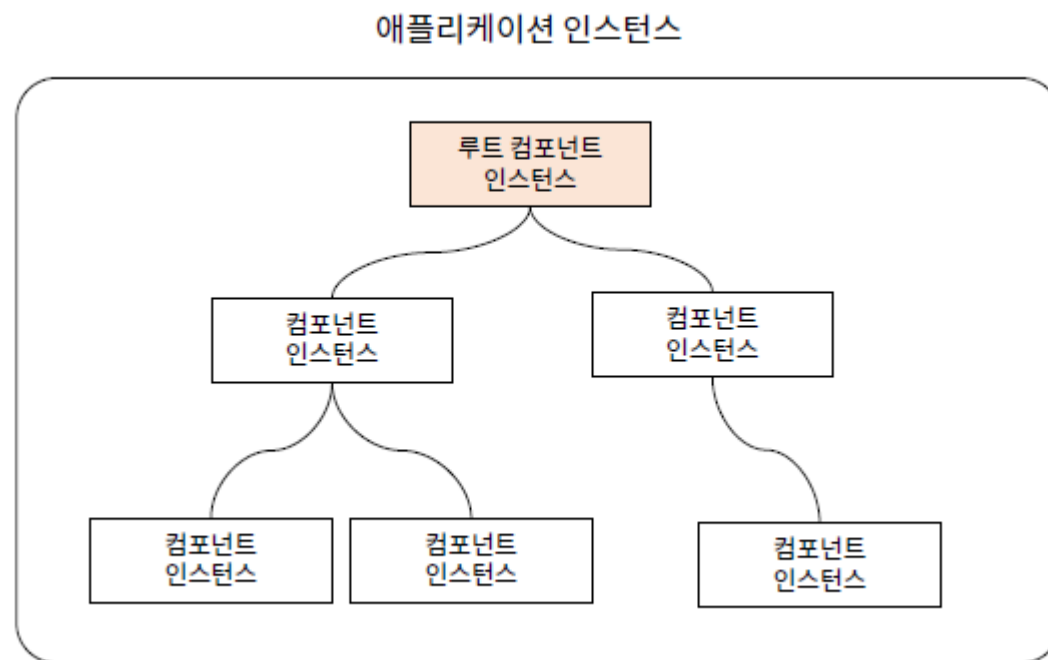
# 1. Vue 인스턴스 개요

## ❖ Vue 인스턴스

- createApp()으로 만든 객체

```
var vm = Vue.createApp({
  name : "App",
  data() {
    return { name : "" };
  }
}).mount('#app')
```

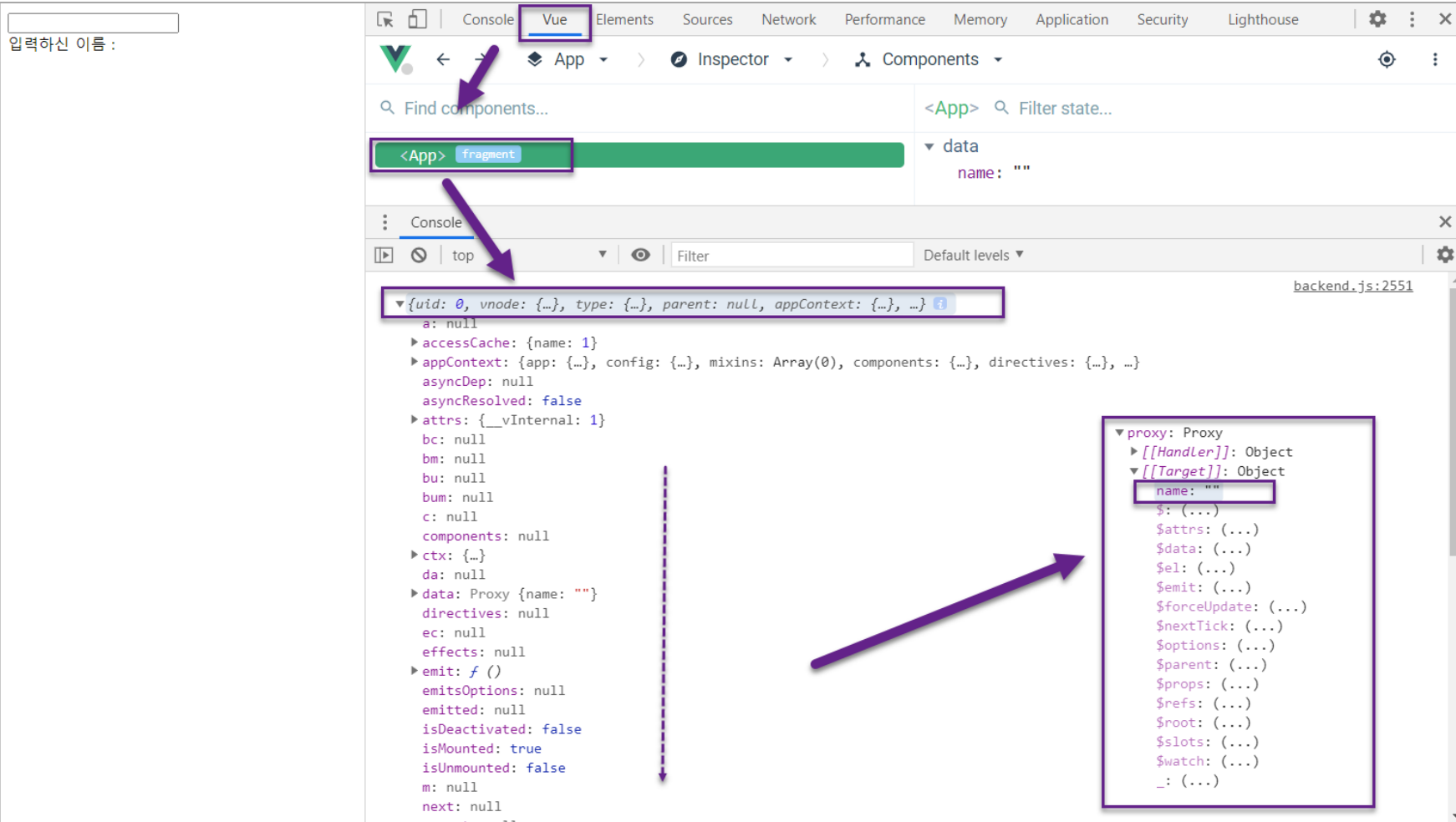
- 옵션 객체
  - Vue 인스턴스를 생성할 때 전달하는 속성들을 담은 객체
  - name 옵션, data 옵션
- 루트 인스턴스
  - 최상위 Vue 인스턴스
  - mount() 메서드로 DOM 트리에 마운트



# 1. Vue 인스턴스 개요

## ❖예제 04-01 실행 후 Vue 인스턴스 확인

입력하신 이름 :



The screenshot shows the Vue DevTools interface. The component inspector shows a single component with the name 'fragment'. The console shows the Vue instance object, with the 'data' property highlighted. A red box highlights the 'data' property, and a red arrow points to a detailed view of the 'data' property in the right-hand pane.

```
{uid: 0, vnode: {...}, type: {...}, parent: null, appContext: {...}, ...}
  a: null
  accessCache: {name: 1}
  appContext: {app: {...}, config: {...}, mixins: Array(0), components: {...}, directives: {...}, ...}
  asyncDep: null
  asyncResolved: false
  attrs: {__vInternal: 1}
  bc: null
  bm: null
  bu: null
  bum: null
  c: null
  components: null
  ctx: {...}
  da: null
  data: Proxy {name: ""}
  directives: null
  ec: null
  effects: null
  emit: f ()
  emitsOptions: null
  emitted: null
  isDeactivated: false
  isMounted: true
  isUnmounted: false
  m: null
  next: null
```

```
> vm
< Proxy {...}
  ▶ [[Handler]]: Object
  ▼ [[Target]]: Object
    name: (...)
    $: (...)
    $attrs: (...)
    $data: (...)
    $el: (...)
    $emit: (...)
    $forceUpdate: (...)
    $nextTick: (...)
    $options: (...)
    $parent: (...)
    $props: (...)
    $refs: (...)
    $root: (...)
    $slots: (...)
    $watch: (...)
    _: (...)
```

### ❖ data 옵션

- 컴포넌트가 관리하고 추적해야 할 데이터를 등록할 때 사용
- 변경을 탐지하고 추적함

### ❖ 주의 사항

- 반드시 객체를 리턴하는 함수로 지정해야 함
  - 만일 그렇지 않다면(직접 객체를 지정한다면) 다음 오류 발생

```
⚠ [Vue warn]: The data option must be a function. Plain object usage is no longer supported.  
at <App>
```

```
✖ Uncaught TypeError: dataFn.call is not a function  
  at resolveData (vue@next:6614)  
  at applyOptions (vue@next:6447)  
  at finishComponentSetup (vue@next:7152)  
  at setupStatefulComponent (vue@next:7088)  
  at setupComponent (vue@next:7028)  
  at mountComponent (vue@next:5314)  
  at processComponent (vue@next:5290)  
  at patch (vue@next:4905)  
  at render (vue@next:6018)  
  at mount (vue@next:4382)
```

- 반응성을 가진 데이터는 반드시 미리 data 옵션으로 지정해야 함
  - 인스턴스가 생성될 때 Proxy가 만들어짐

## 2. data 옵션

- data에 대한 접근 방법
  - vm.\$data.name
  - vm.name
- 이름에 \$, \_ 이 포함된 데이터를 작성할 수 없음
  - 경고, 오류 발생

```
> vm.name = "Hello"
< "Hello"
> vm.$data.name
< "Hello"
```

```
<div id="app">
  <input id="a" type="text" v-model="$name" />
  <br />
  입력하신 이름 : <span>{{ $name }}</span>
</div>
<script type="text/javascript" src="https://unpkg.com/vue"></script>
<script type="text/javascript">
  var vm = Vue.createApp({
    name: "App",
    data() {
      return { $name: "" };
    },
  }).mount("#app");
</script>
```

2 ▶ [Vue warn]: Property "\$name" must be accessed via \$data because it starts with a reserved character and is not proxied on the render context.  
at <App>

### 3. 계산된 속성

#### ❖ 계산된 속성 (Computed Property)

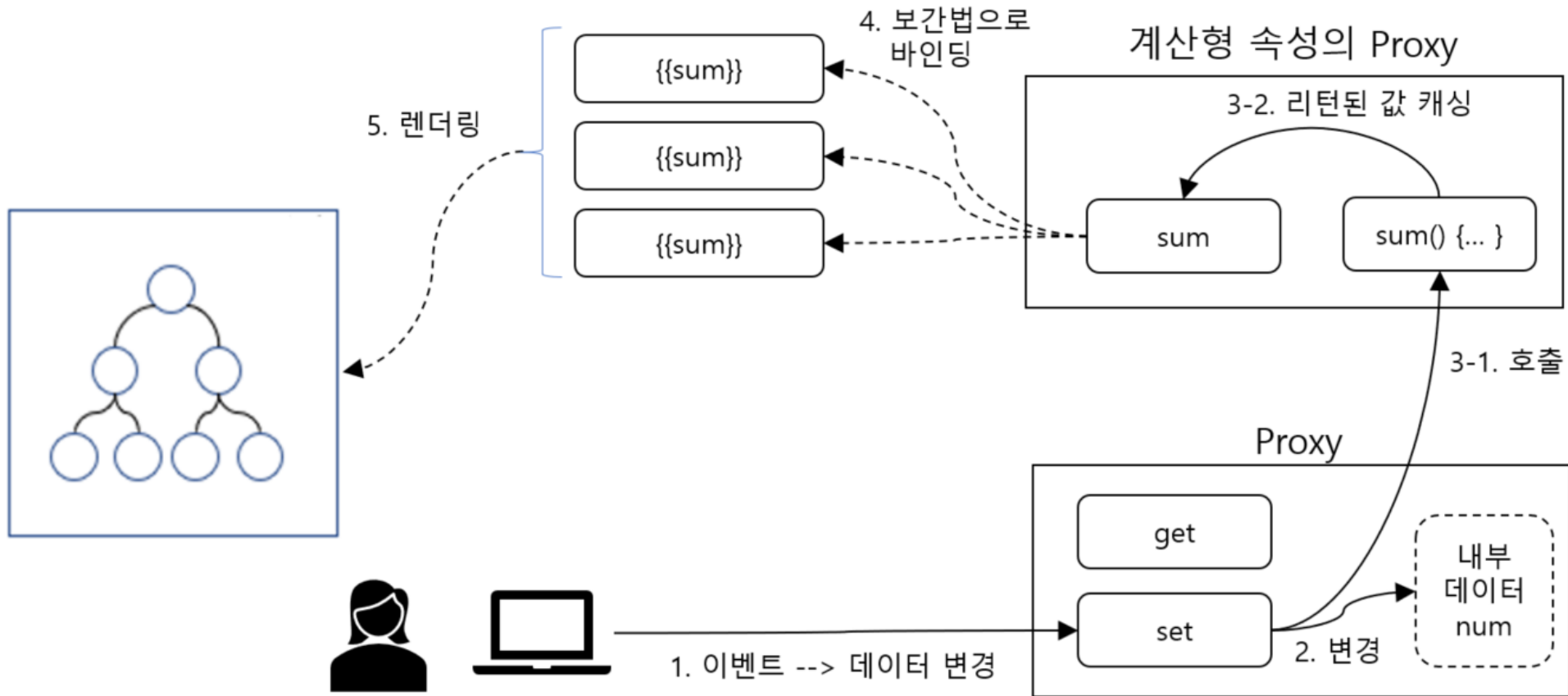
- data나 다른 속성이 변경될 때 함수가 실행되어(계산된) 저장된 캐싱된 값
- 함수의 실행은 의존하고 있는 속성이 변경될 때 한번 호출됨

#### ❖ 예제 04-03

```
<div id="app">
  1보다 큰수 : <input id="a" type="text" v-model.number="num" />
  <br />
  1부터 입력한 값까지의 합 : <span>{{sum}}</span><br />
  1부터 입력한 값까지의 합 : <span>{{sum}}</span><br />
  1부터 입력한 값까지의 합 : <span>{{sum}}</span><br />
</div>
<script type="text/javascript" src="https://unpkg.com/vue"></script>
<script type="text/javascript">
  var vm = Vue.createApp({
    name: "App",
    data() {
      return { num: 0 };
    },
    computed: {
      sum() {
        console.log("## num : " + this.num);
        var n = parseInt(this.num);
        if (Number.isNaN(n)) return 0;
        return (n * (n + 1)) / 2;
      },
    },
  }).mount("#app");
</script>
```

### 3. 계산된 속성

#### ❖ 예제 04-03 실행 구조

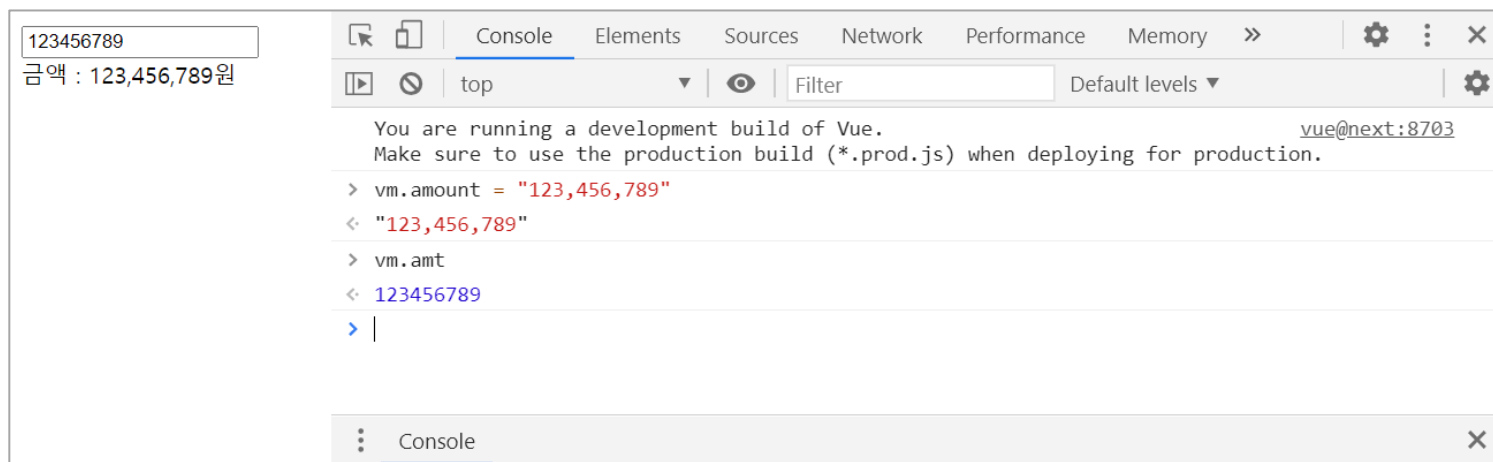




### 3. 계산된 속성

#### ❖예제 04-04 : getter, setter

```
<div id="app">
  <input type="text" v-model.number="amt" /><br />
  금액 : <span>{{amount}}원</span>
</div>
<script type="text/javascript" src="https://unpkg.com/vue"></script>
<script type="text/javascript">
  var vm = Vue.createApp({
    name: "App",
    data() {
      return { amt: 99999 };
    },
    computed: {
      amount: {
        get() {
          var regexp = /\B(?=(\d{3})+(?!\d))/g;
          return this.amt.toString().replace(regexp, ",");
        },
        set(amount) {
          var amt = parseInt(amount.replace(/,/g, ""));
          this.amt = Number.isNaN(amt) ? 0 : amt;
        },
      },
    },
  }).mount("#app");
</script>
```





### 3. 계산된 속성

- ❖ 계산된 속성에 등록하는 함수가 화살표 함수일려면
  - 인자가 vue 인스턴스인 함수로...

---

```
computed : {  
  sum : (vm) => {  
    console.log("## num : " + vm.num)  
    var n = parseInt(vm.num);  
    if (Number.isNaN(n)) return 0;  
    return (n * (n+1))/2  
  }  
}
```

---

### ❖ 메서드

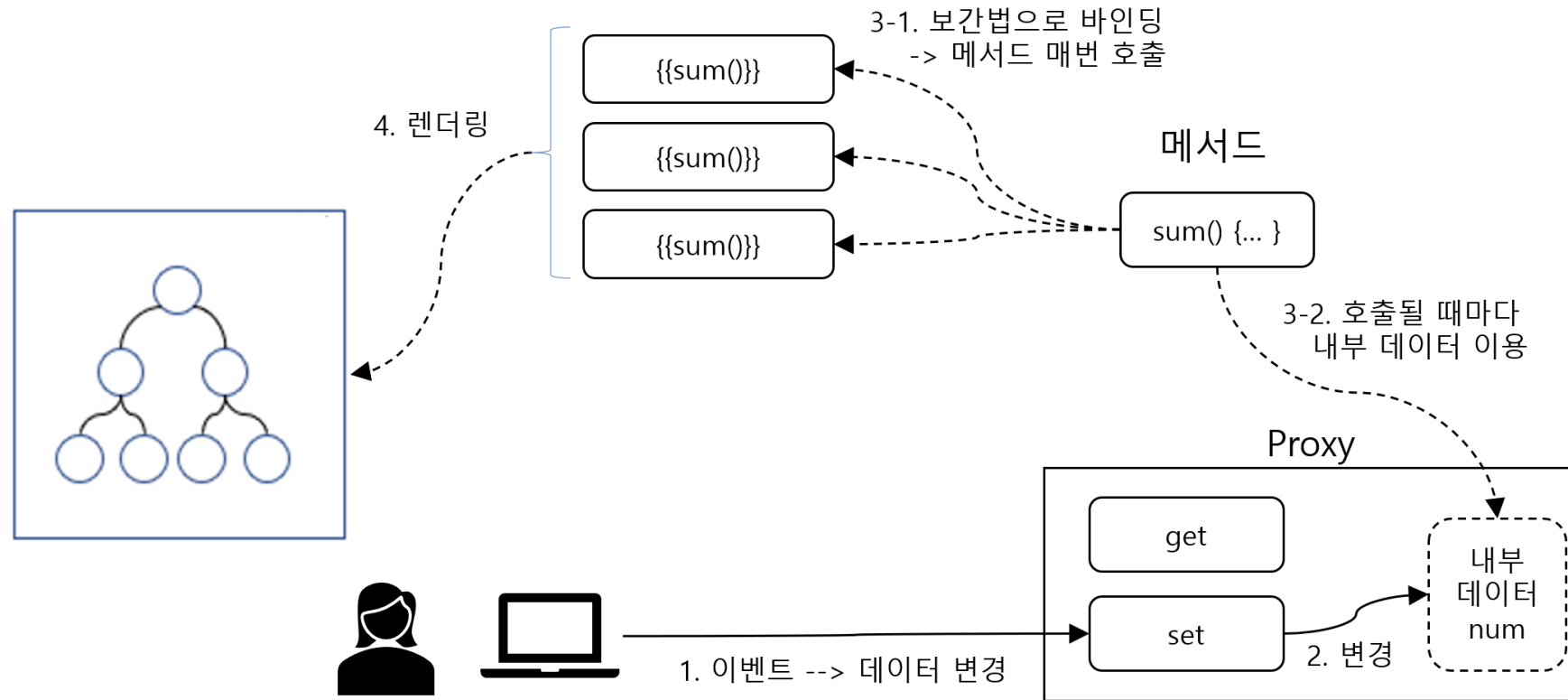
- Vue 인스턴스에서 사용할 메서드를 등록하기 위해 methods 옵션 사용
- 모두 사용 가능
  - 인스턴스 내에서 직접 호출
  - 보간법에서 사용
  - 디렉티브 표현식에서 사용
  - 이벤트 핸들러에서 사용

### ❖ 예제 04-05

```
<div id="app">
  1보다 큰수 : <input id="a" type="text" v-model.number="num" />
  <br />
  1부터 입력한 값까지의 합 : <span>{{sum()}}</span><br />
  1부터 입력한 값까지의 합 : <span>{{sum()}}</span><br />
  1부터 입력한 값까지의 합 : <span>{{sum()}}</span><br />
</div>
<script type="text/javascript" src="https://unpkg.com/vue"></script>
<script type="text/javascript">
  var vm = Vue.createApp({
    name: "App",
    data() {
      return { num: 0 };
    },
    methods: {
      sum() {
        console.log("## num : " + this.num);
        var n = parseInt(this.num);
        if (Number.isNaN(n)) return 0;
        return (n * (n + 1)) / 2;
      },
    },
  }).mount("#app");
</script>
```

## 4. 메서드

- ❖ 예제 04-05 실행 결과
  - 계산된 속성과 비교할 것



- ❖ 화살표 함수로 메서드를 작성할 경우 this가 전역을 참조 --> 사용하지 않을 것

### ❖관찰 속성(Watched Property)

- 하나의 데이터(data, 속성)를 기반으로 다른 데이터를 변경할 필요가 있을 때 사용
- 특히 긴 시간이 필요한 비동기 처리에 적합

### ❖예제 04-06

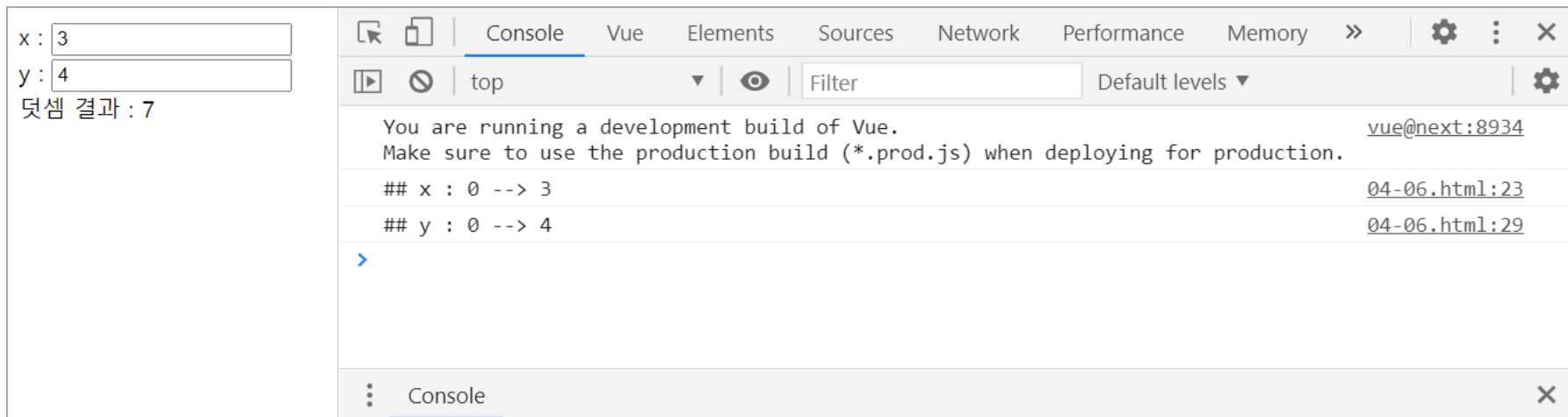
data 또는 속성명      변경후 x값      변경전 x값

```
x ( current , old ) {  
  console.log(`## x : ${old} --> ${current}`)  
  var result = Number(current) + Number(this.y);  
  if (isNaN(result)) this.sum = 0;  
  else this.sum = result;  
}
```

```
<div id="app">  
  x : <input type="text" v-model.number="x" /><br />  
  y : <input type="text" v-model.number="y" /><br />  
  덧셈 결과 : {{sum}}  
</div>  
  
<script type="text/javascript" src="https://unpkg.com/vue"></script>  
<script type="text/javascript">  
  var vm = Vue.createApp({  
    name: "App",  
    data() {  
      return { x: 0, y: 0, sum: 0 };  
    },  
    watch: {  
      x(current, old) {  
        console.log(`## x : ${old} --> ${current}`);  
        var result = Number(current) + Number(this.y);  
        if (isNaN(result)) this.sum = 0;  
        else this.sum = result;  
      },  
      y(current, old) {  
        console.log(`## y : ${old} --> ${current}`);  
        var result = Number(this.x) + Number(current);  
        if (isNaN(result)) this.sum = 0;  
        else this.sum = result;  
      },  
    },  
  }).mount("#app");  
</script>
```

## 5. 관찰 속성

### ❖예제 04-06 실행 결과



- 하지만 코드는 계산형 속성이 간단해보임

## 5. 관찰 속성

### ❖ 계산형 속성이 할 수 없는 것

- 긴 시간이 필요한 비동기 처리
  - 계산형 속성은 반드시 값을 리턴해야 하는데, 리턴된 값이 캐싱되어 사용되므로 동기적!
- 변경 전/후 값의 활용
  - 계산된 속성은 불가능

### ❖ 예제 04-08

- 테스트용 백엔드 API 이용
  - 1초의 지연시간 후에 응답
- axios를 이용한 비동기 처리 예

이름 :

- Courtney Johnson : 010-3456-8254
- Kelley Jones : 010-3456-8269
- Penelope Johnson : 010-3456-8273

```
<div id="app">
  이름 : <input type="text" v-model.trim="name" placeholder="영문 두글자 이상을 입력하세요" /><br />
  <ul>
    <li v-for="c in contacts">{{c.name}} : {{c.tel}}</li>
  </ul>
  <div v-show="isLoading">검색중</div>
</div>

<script type="text/javascript" src="https://unpkg.com/vue"></script>
<script type="text/javascript" src="https://unpkg.com/axios"></script>
<script type="text/javascript" src="https://unpkg.com/lodash"></script>
<script type="text/javascript">
  const BASEURL = "https://contactsvc.bmaster.kro.kr";
  var vm = Vue.createApp({
    name: "App",
    data() {
      return { name: "", contacts: [], isLoading: false };
    },
    watch: {
      name(current) {
        if (current.length >= 2) {
          this.fetchContacts();
        } else {
          this.contacts = [];
        }
      },
    },
    methods: {
      fetchContacts: _.debounce(function () {
        this.isLoading = true;
        axios.get(BASEURL + `/contacts_long/search/${this.name}`).then((response) => {
          this.isLoading = false;
          this.contacts = response.data;
        });
      }, 300),
    },
  }).mount("#app");
</script>
</body>
```

### ❖ 관찰 속성 사용시 주의사항

- 복잡한 구조의 객체인 경우 감시 기능이 작동하지 않을 수 있음
  - 참조형 대상은 메모리 주소가 변경되어야 함
- 예제 04-09
- 해결책 : deep compare

```
watch: {
  values: {
    handler: function (current) {
      var result = Number(current.x) + Number(current.y);
      if (isNaN(result)) this.sum = 0;
      else this.sum = result;
    },
    deep: true,
  },
},
```

```
<div id="app">
  x : <input type="text" v-model.number="values.x" /><br />
  y : <input type="text" v-model.number="values.y" /><br />
  덧셈 결과 : {{sum}}
</div>
<script type="text/javascript" src="https://unpkg.com/vue"></script>
<script type="text/javascript">
  var vm = Vue.createApp({
    name: "App",
    data() {
      return { values: { x: 0, y: 0 }, sum: 0 };
    },
    watch: {
      values(current) {
        var result = Number(current.x) + Number(current.y);
        if (isNaN(result)) this.sum = 0;
        else this.sum = result;
      },
    },
  }).mount("#app");
</script>
```



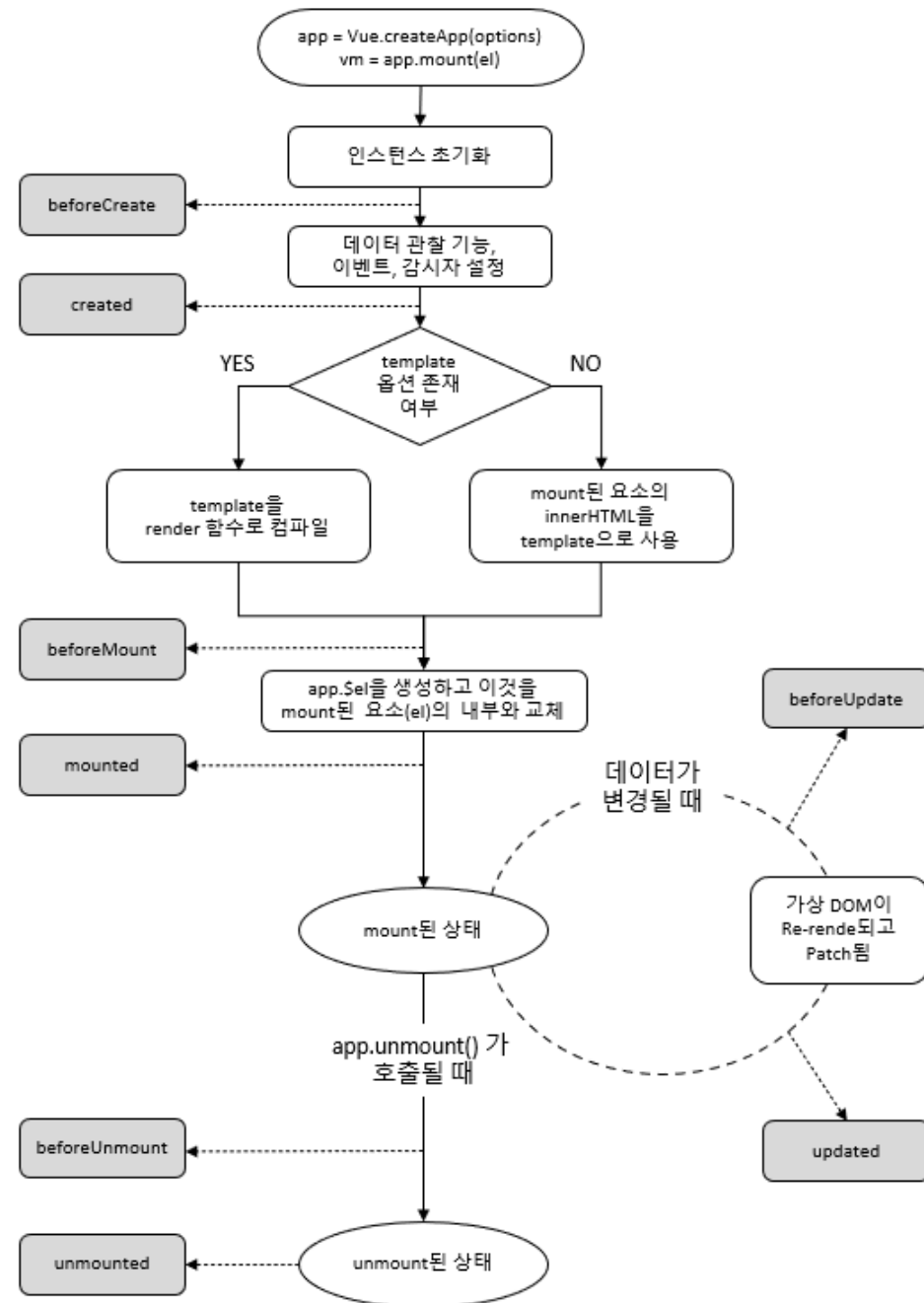
### ❖관찰 속성 정리

- 비동기 처리에는 장점이 있음
- 가급적 사용하지 않는 것이 바람직함.
  - data에 대한 의존 관계가 복잡해져서 코드의 실행을 분석하기 어려움
  - 대안) 메서드 + 이벤트의 조합 --> 5장
- 예제 : 예제 04-08-2를 참조

## 6. 생명 주기 메서드

### ❖ 생명주기 메서드

- 컴포넌트, Vue 인스턴스의 생성부터 언마운트될 때까지의 이벤트시에 실행되는 메서드
- 간단한 애플리케이션에서는 중요하지 않지만 실전 프로젝트에서는 중요함
  - 복잡한 비동기 처리, 상태 관리, 라우팅이 사용되면 중요하게 다루어짐

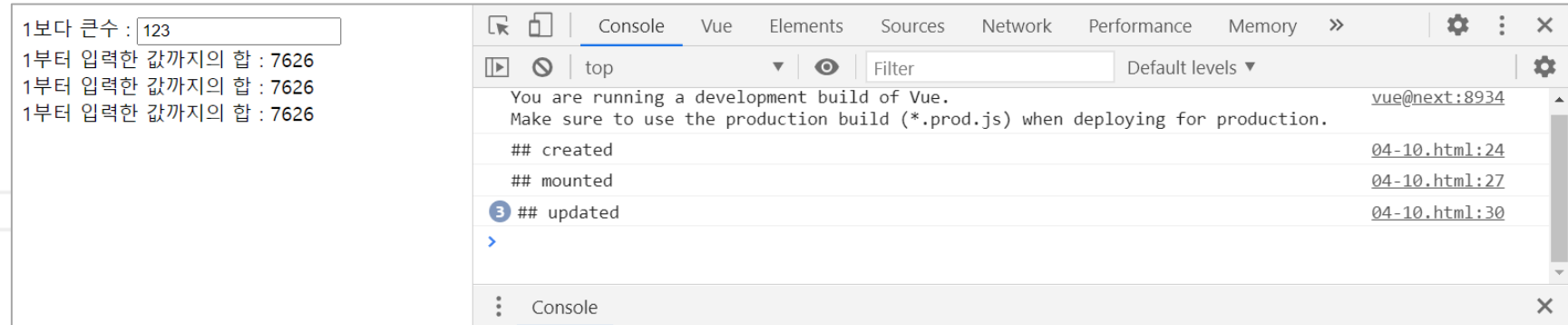


## 6. 생명 주기 메서드

### ❖ 생명주기 메서드 예제 : 예제 04-10

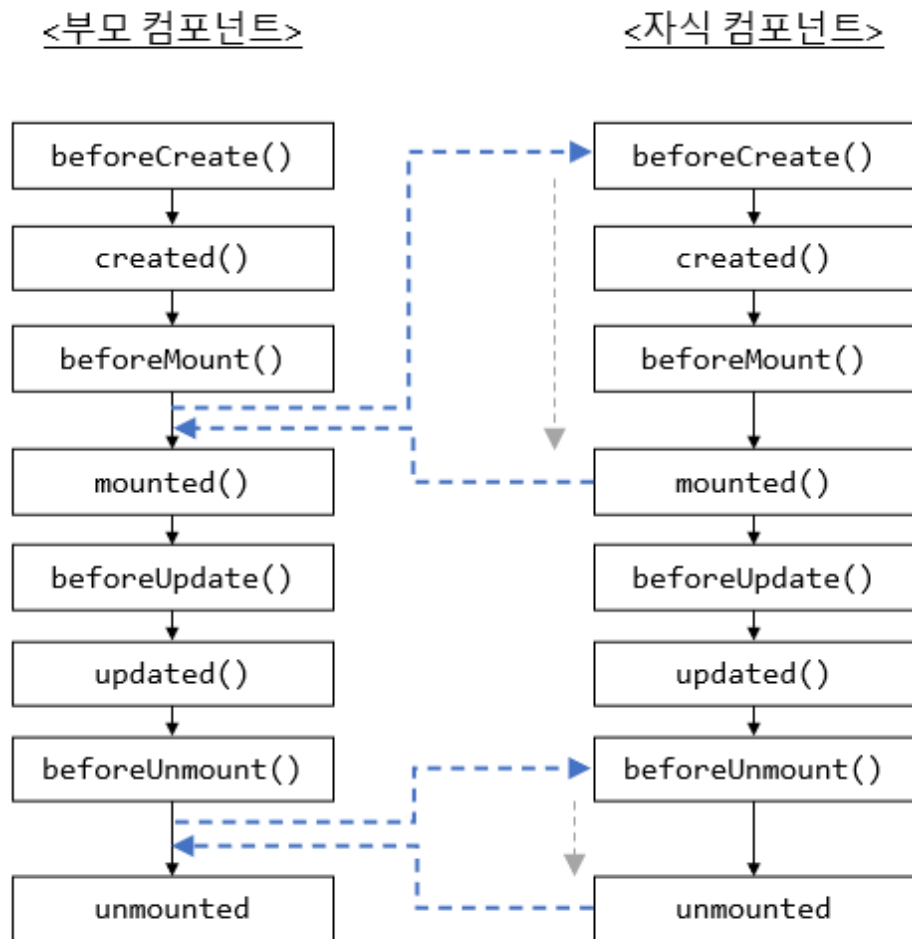
```
<script type="text/javascript">
var vm = Vue.createApp({
  name: "App",
  data() {
    return { num: 0 };
  },
  created() {
    console.log("## created");
  },
  mounted() {
    console.log("## mounted");
  },
  updated() {
    console.log("## updated");
  },
  computed: {
    sum() {
      var n = parseInt(this.num);
      if (Number.isNaN(n)) return 0;
      return (n * (n + 1)) / 2;
    },
  },
}).mount("#app");
</script>
```

1보다 큰수 : 123  
1부터 입력한 값까지의 합 : 7626  
1부터 입력한 값까지의 합 : 7626  
1부터 입력한 값까지의 합 : 7626

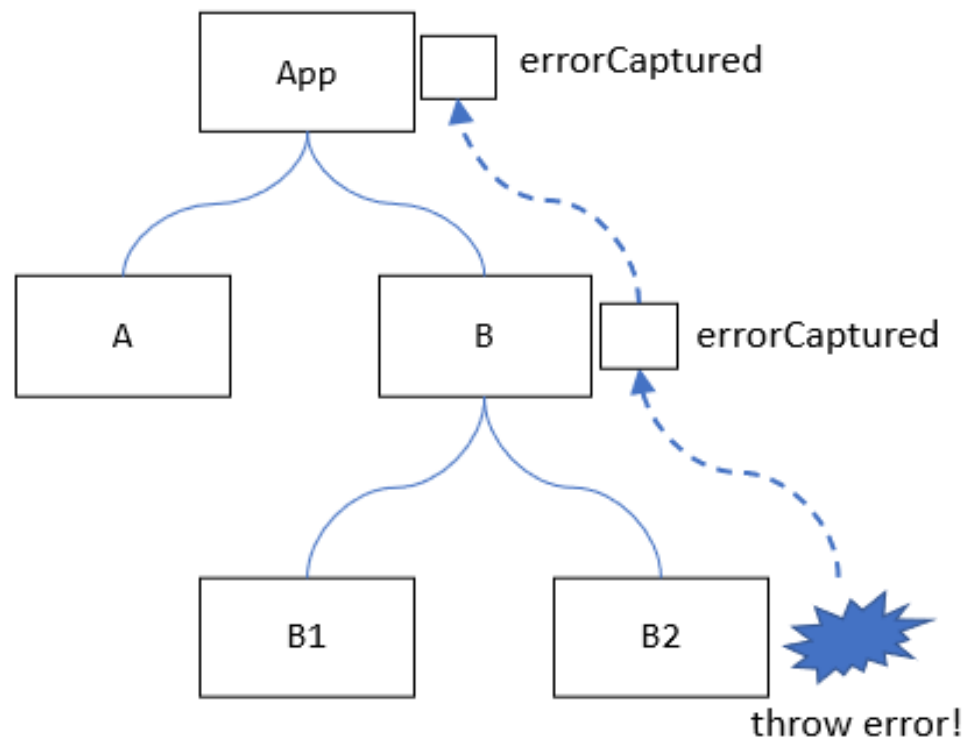


## 6. 생명 주기 메서드

### ❖ 부모-자식 컴포넌트 생명주기 메서드



### ❖ 컴포넌트 트리에서의 에러 처리



지금까지 Vue 인스턴스의 기본적인 옵션들에 대해 살펴보았습니다. 일반적인 경우라면 watch 옵션을 사용하는 관찰 속성보다는 계산된 속성이나 메서드가 더 편리합니다. 하지만 긴 작업 시간이 필요한 비동기 처리가 요구되는 경우에는 관찰 속성을 사용해야 하는 경우도 있습니다.

methods 옵션을 이용해 컴포넌트 인스턴스에 메서드를 정의할 수 있습니다. 등록된 메서드는 콧수염(Mustache) 표현식의 템플릿 문자열로도 사용할 수 있으며, 다음 장에서 살펴볼 이벤트에서도 사용할 수 있습니다.