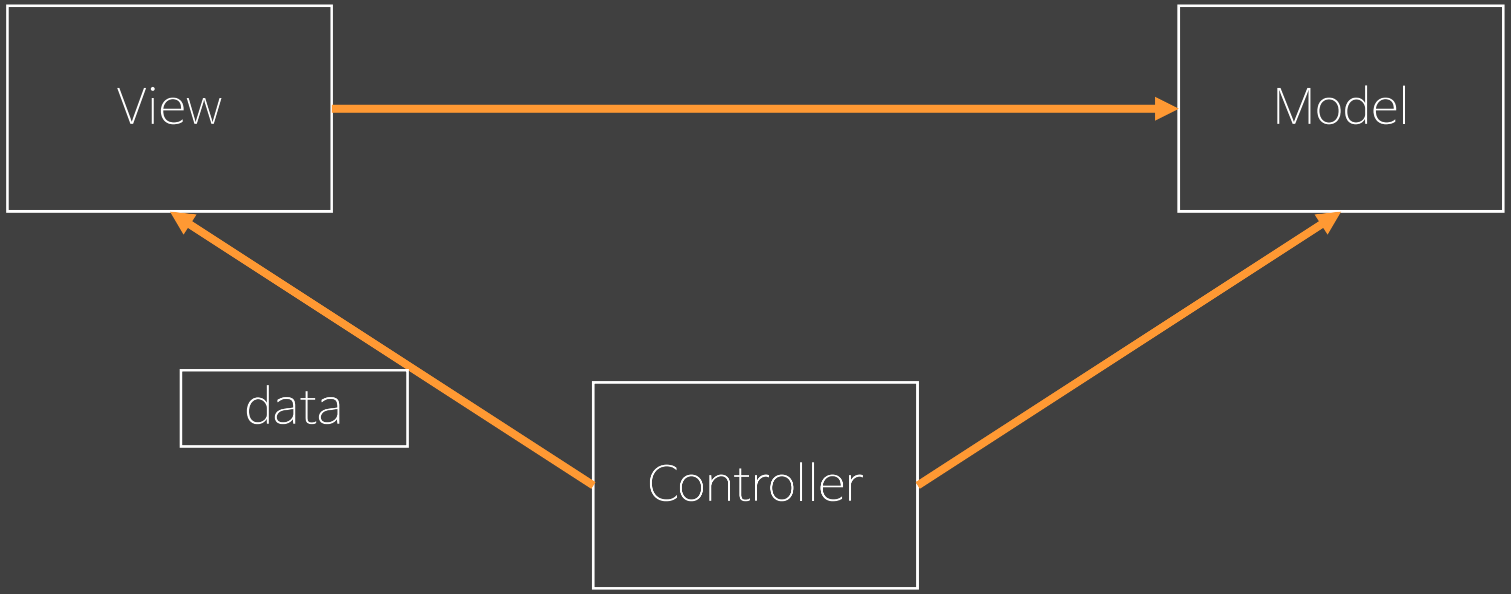
CODE SPITZ 86

OBJECT ORIENTED JAVASCRIPT

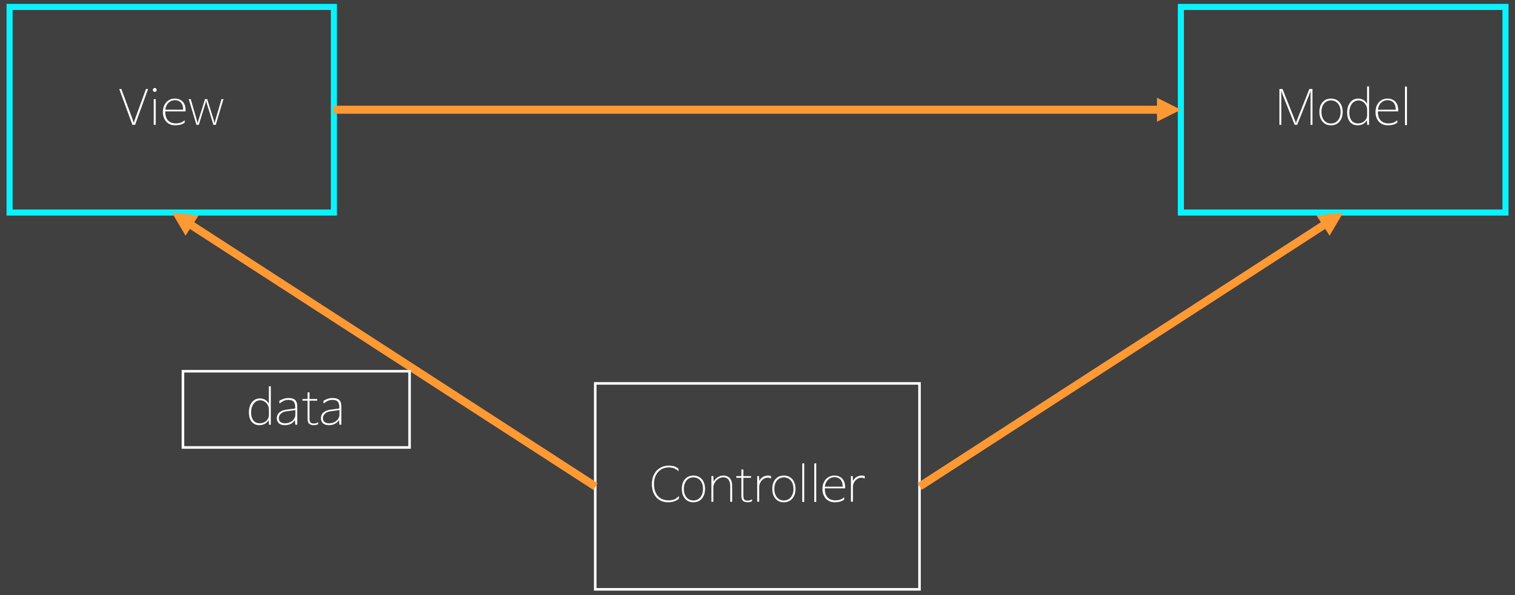
1 2 3 4 5

MVVM

Model View Controller



Model View Controller



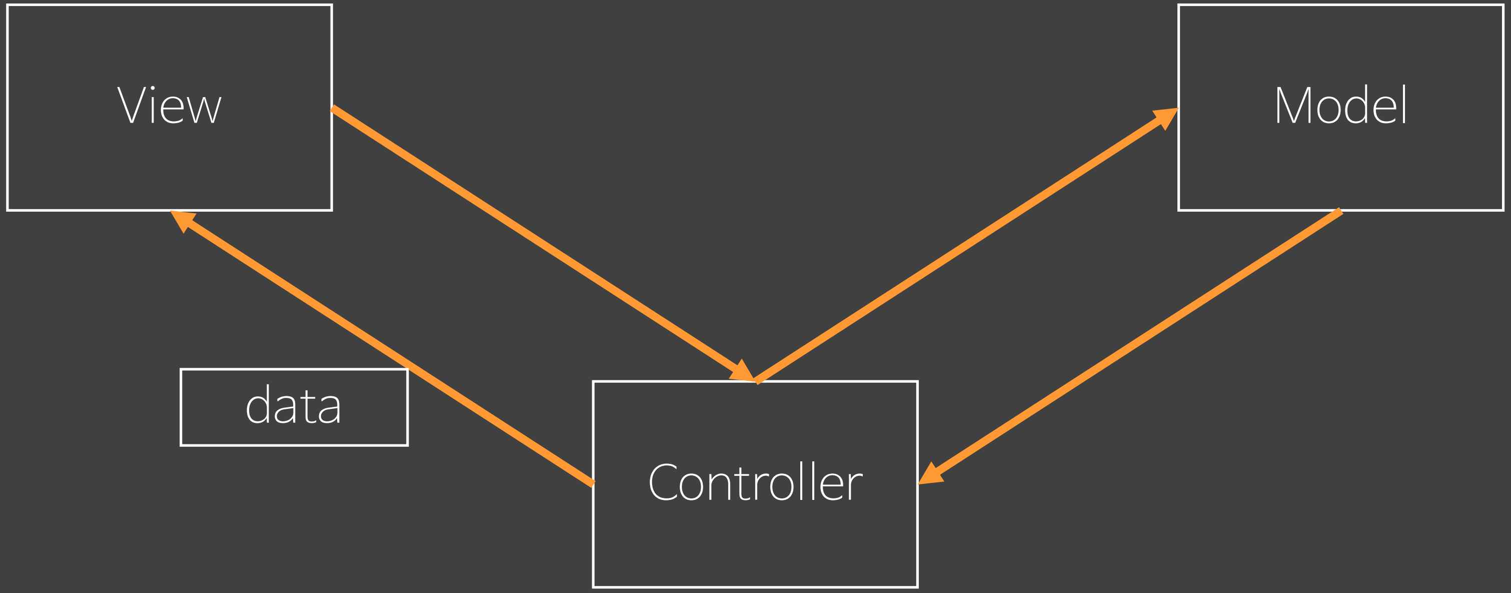
View

Model

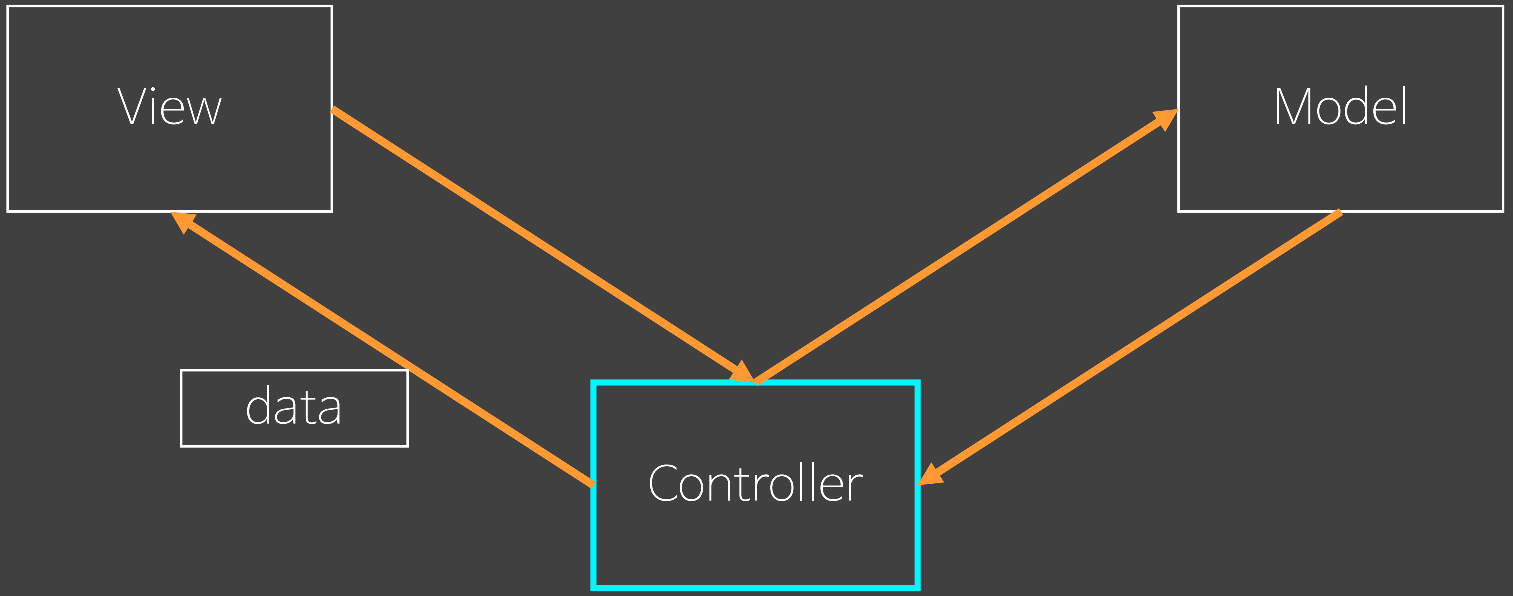
data

Controller

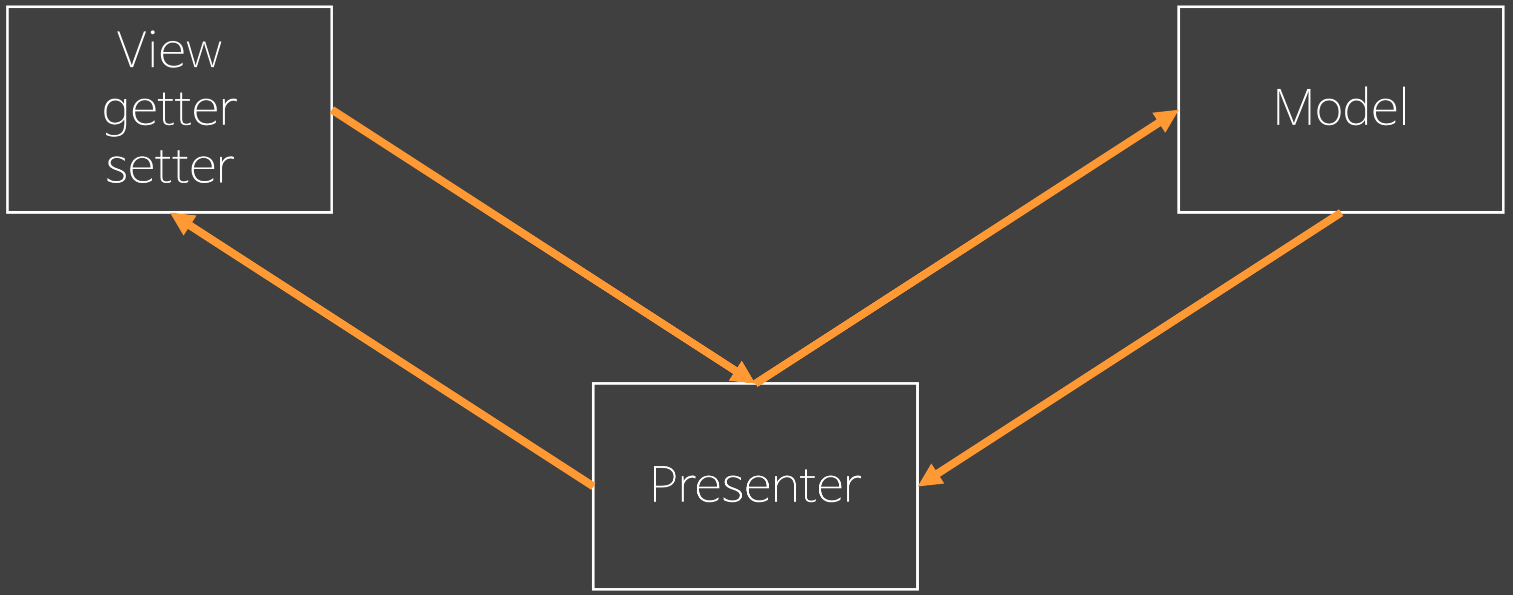
Model View Controller



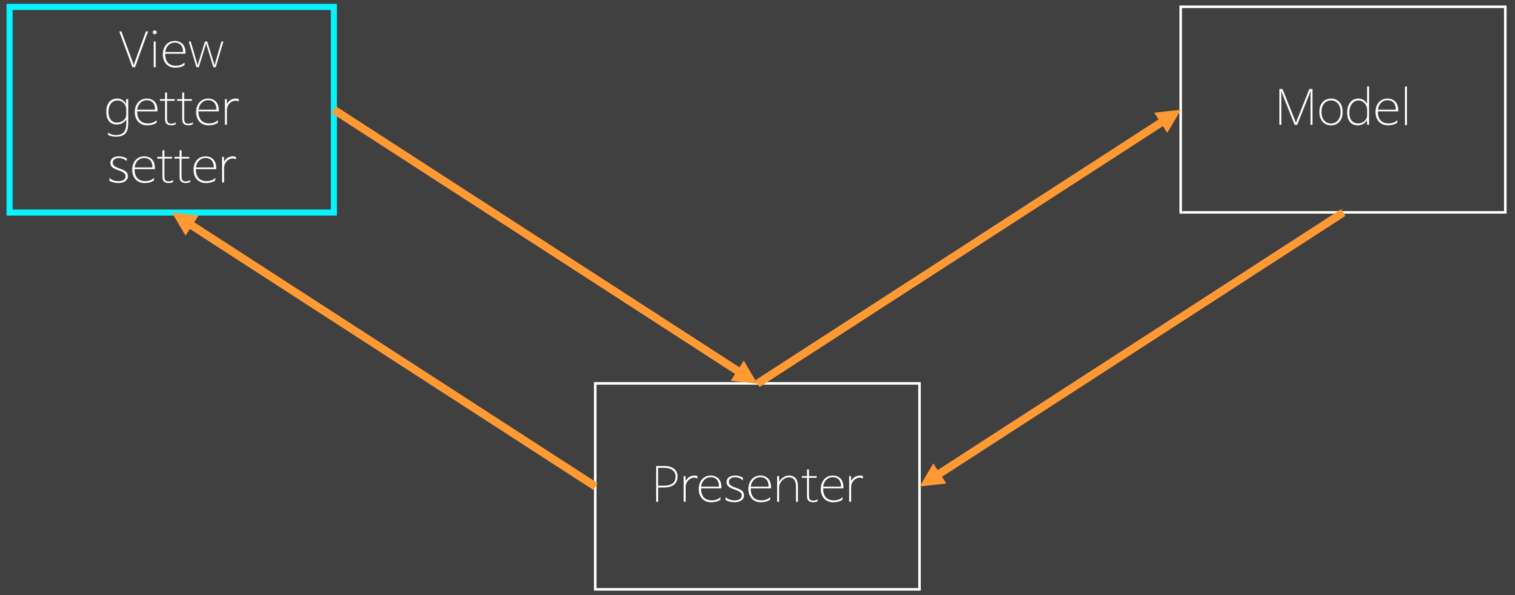
Model View Controller



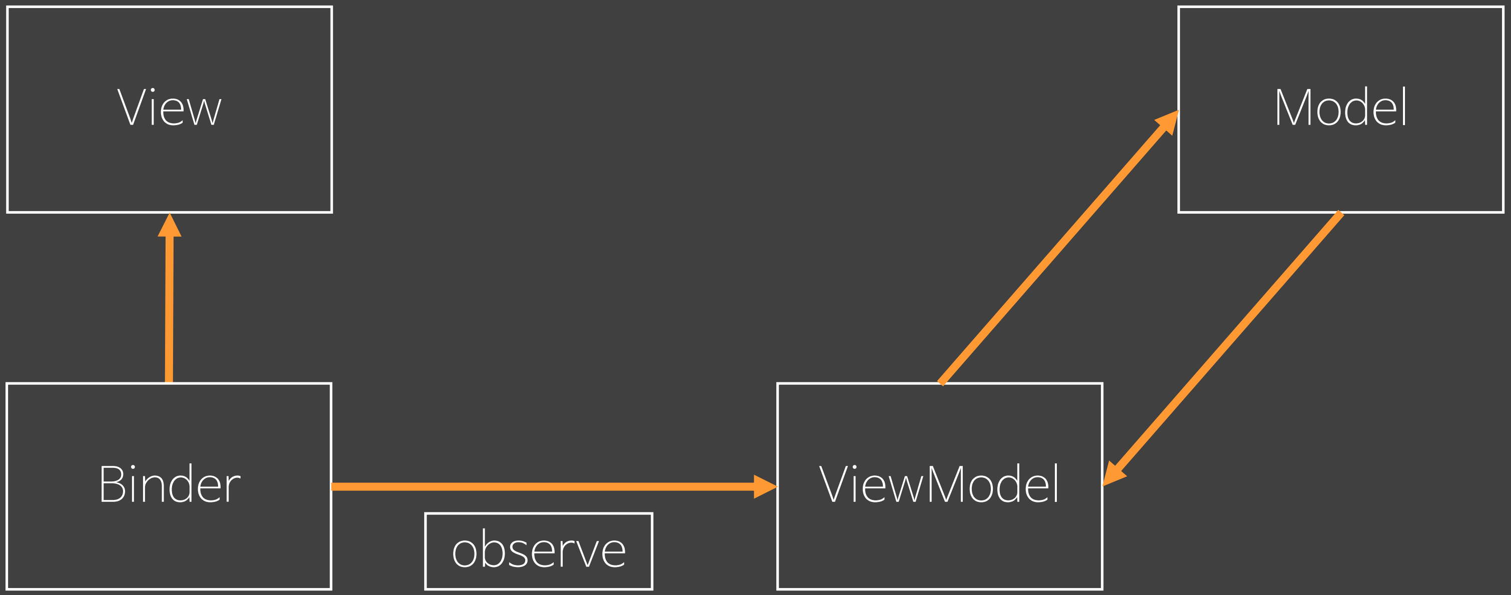
Model View Presenter



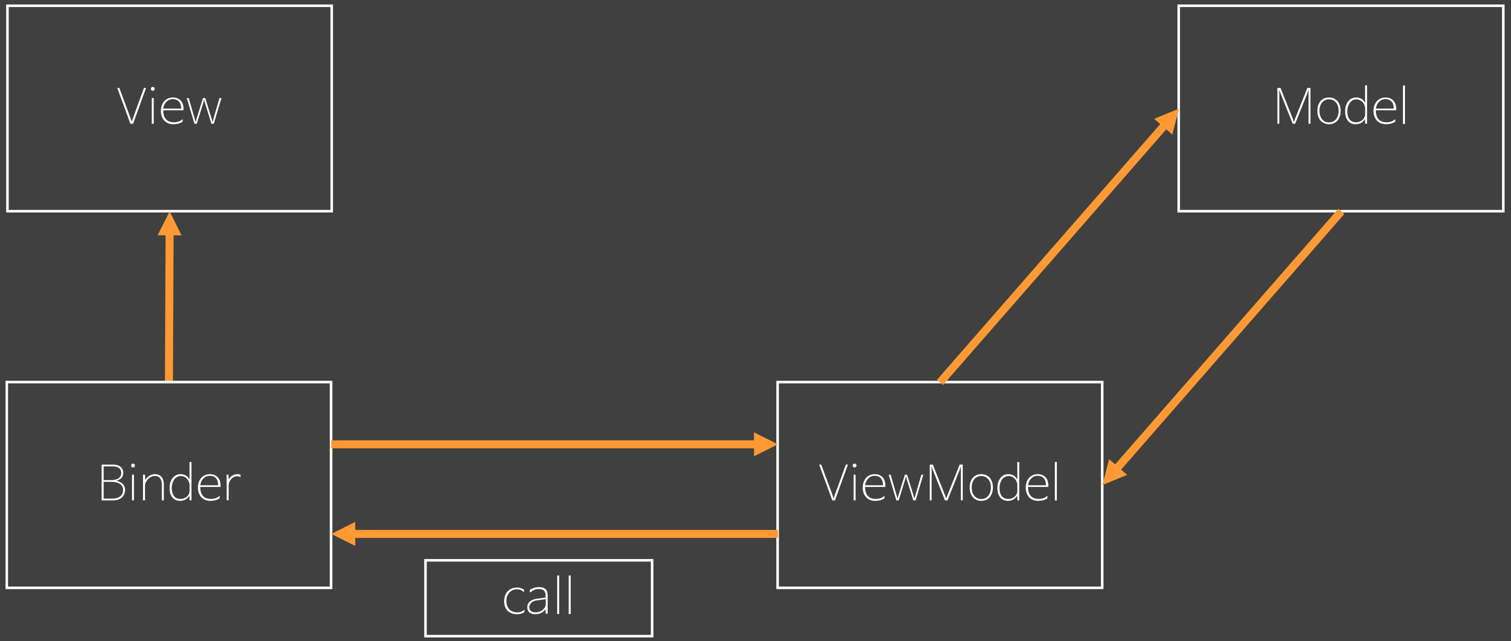
Model View Presenter



Model View ViewModel



Model View ViewModel



TypeCheck

```
const type = (target, type)=>{
    if(typeof type == "string"){
        if(typeof target != type) throw `invaild type ${target} : ${type}`;
    }else if(!(target instanceof type)) throw `invaild type ${target} : ${type}`;
    return target;
};
```

```
const type = (target, type)=>{
    if(typeof type == "string"){
        if(typeof target != type) throw `invaild type ${target} : ${type}`;
    }else if(!(target instanceof type)) throw `invaild type ${target} : ${type}`;
    return target;
};
```

```
type(12, "number");
type("abc", "string");
type([1,2,3], Array);
type(new Set, Set);
type(document.body, HTMLElement);
```

```
const type = (target, type)=>{
    if(typeof type == "string"){
        if(typeof target != type) throw `invaild type ${target} : ${type}`;
    }else if(!(target instanceof type)) throw `invaild type ${target} : ${type}`;
    return target;
};
```

```
type(12, "number");
type("abc", "string");
type([1,2,3], Array);
type(new Set, Set);
type(document.body, HTMLElement);
```

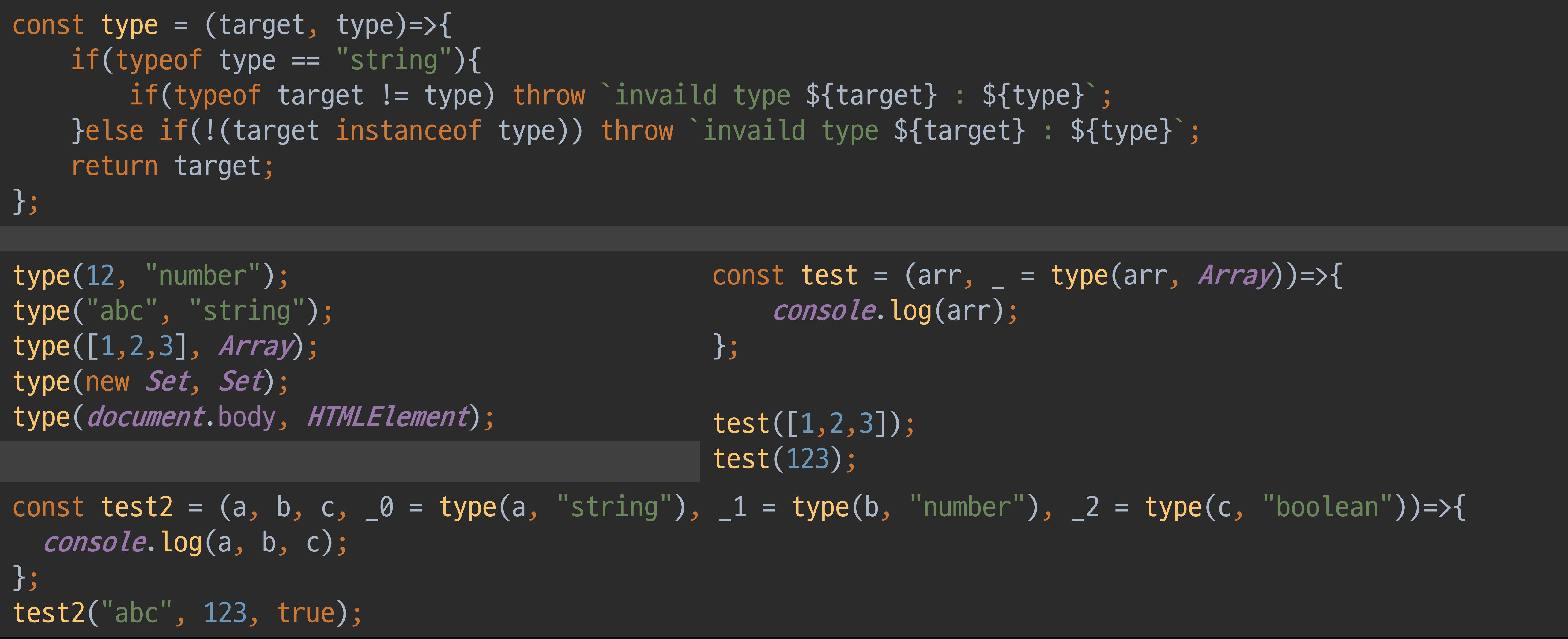
```
const test = (arr, _ = type(arr, Array))=>{
    console.log(arr);
};
```

```
const type = (target, type)=>{
    if(typeof type == "string"){
        if(typeof target != type) throw `invaild type ${target} : ${type}`;
    }else if(!(target instanceof type)) throw `invaild type ${target} : ${type}`;
    return target;
};
```

```
type(12, "number");
type("abc", "string");
type([1,2,3], Array);
type(new Set, Set);
type(document.body, HTMLElement);
```

```
const test = (arr, _ = type(arr, Array))=>{
    console.log(arr);
};

test([1,2,3]);
test(123);
```



```javascript
const type = (target, type)=>{
    if(typeof type == "string"){
        if(typeof target != type) throw `invaild type ${target} : ${type}`;
    }else if(!(target instanceof type)) throw `invaild type ${target} : ${type}`;
    return target;
};
```

```javascript
type(12, "number");
type("abc", "string");
type([1,2,3], Array);
type(new Set, Set);
type(document.body, HTMLElement);
```

```javascript
const test = (arr, _ = type(arr, Array))=>{
    console.log(arr);
};

test([1,2,3]);
test(123);
```

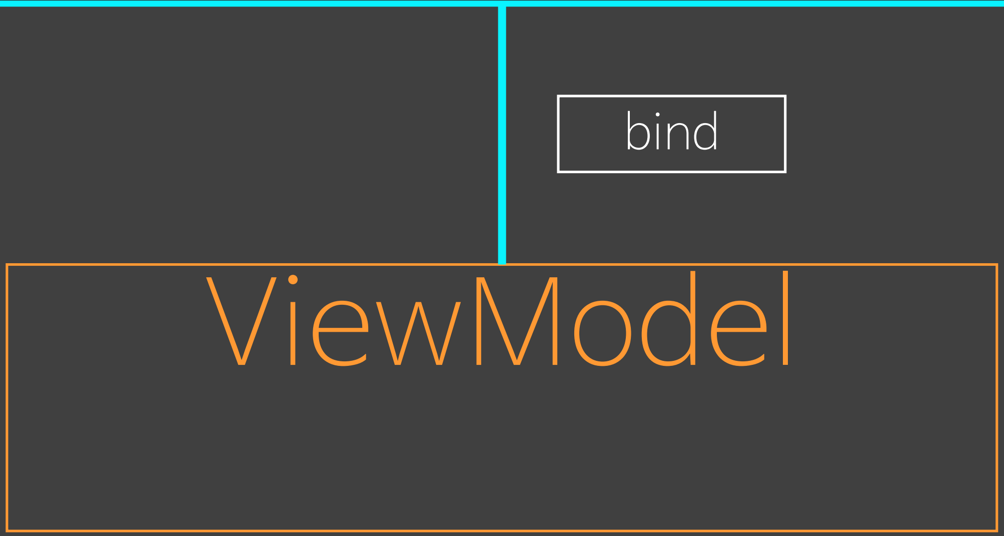
```javascript
const test2 = (a, b, c, _0 = type(a, "string"), _1 = type(b, "number"), _2 = type(c, "boolean"))=>{
  console.log(a, b, c);
};
test2("abc", 123, true);
```

View hook & bind

```html
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```

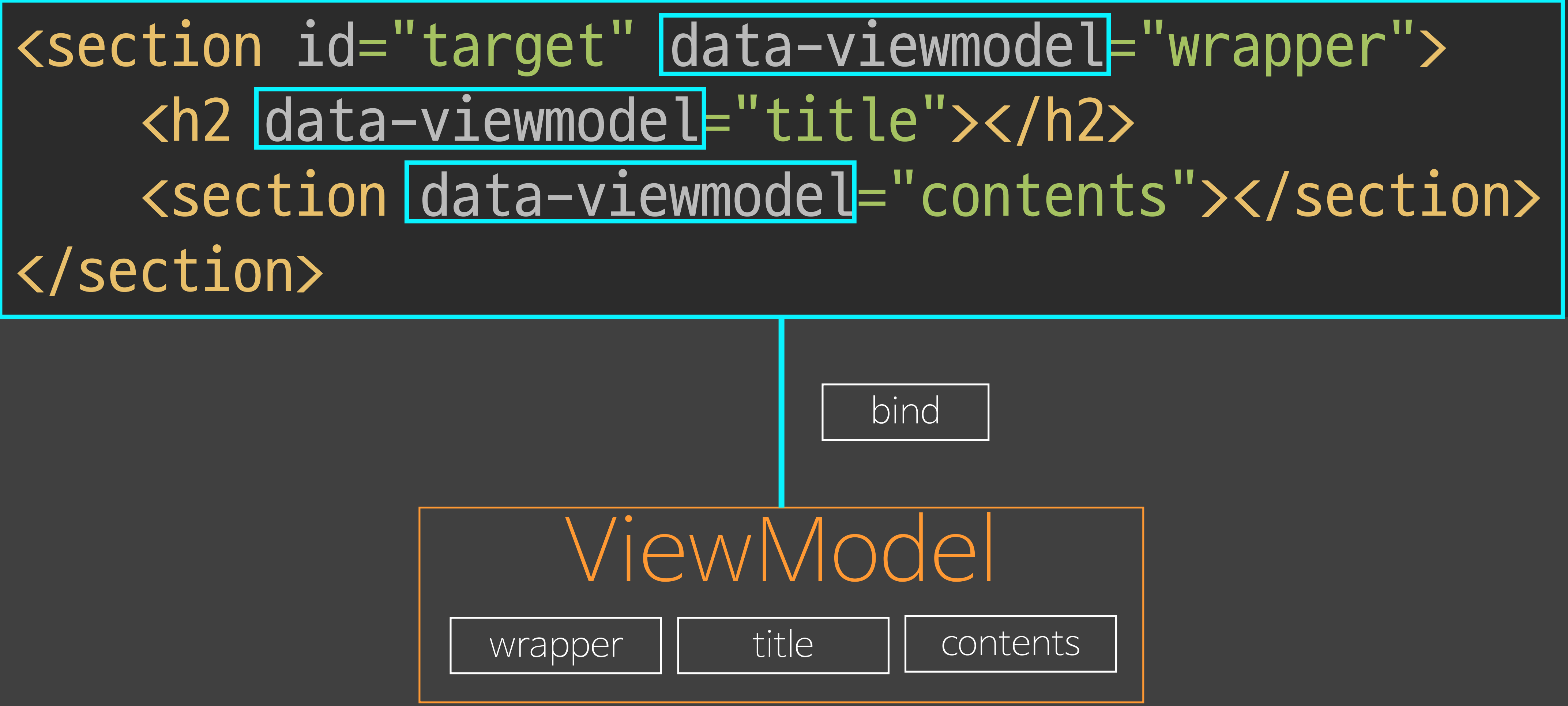
```html
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```

```html
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```



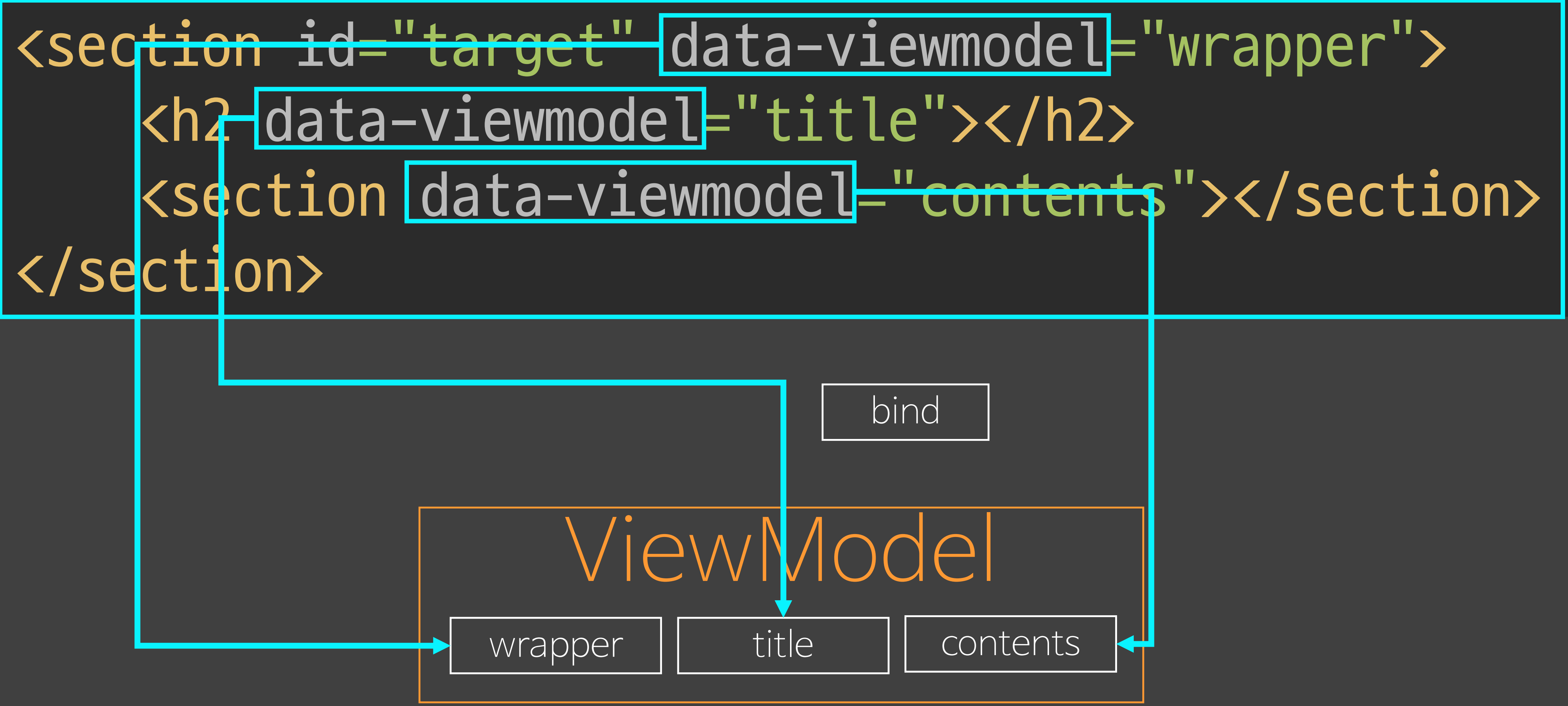
bind

ViewModel



```
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```

bind
ViewModel
wrapper
title
contents

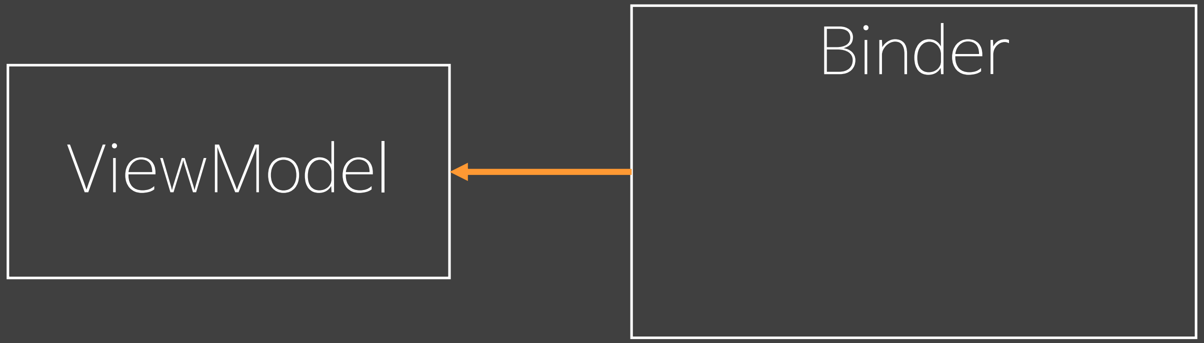


```
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```

bind
ViewModel
wrapper
title
contents

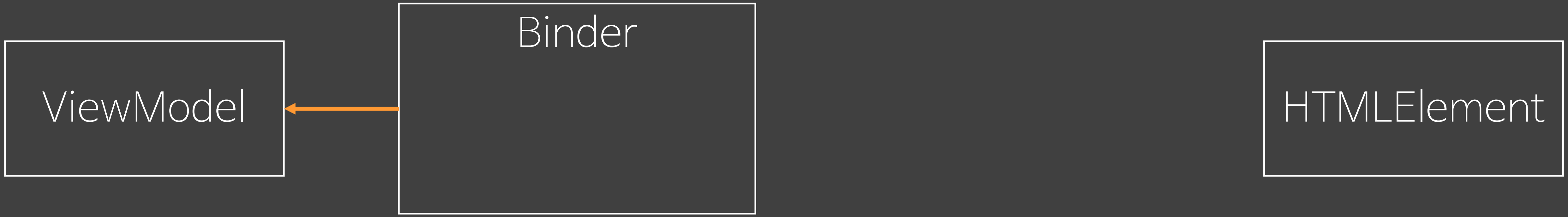
Role Design

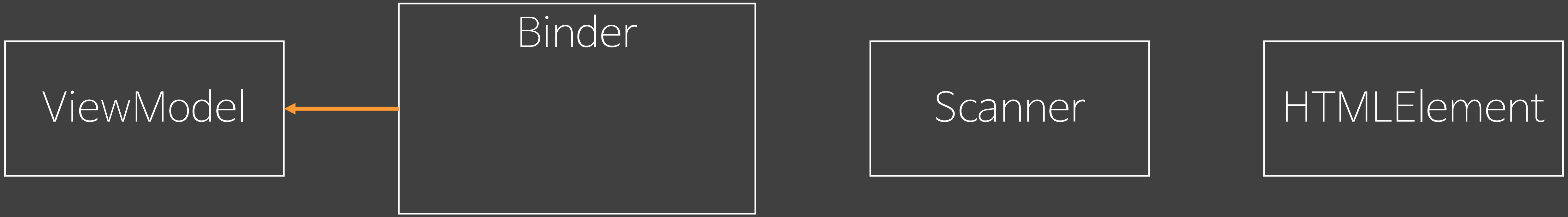
Binder



ViewModel

Binder


ViewModel
Binder
HTMLElement

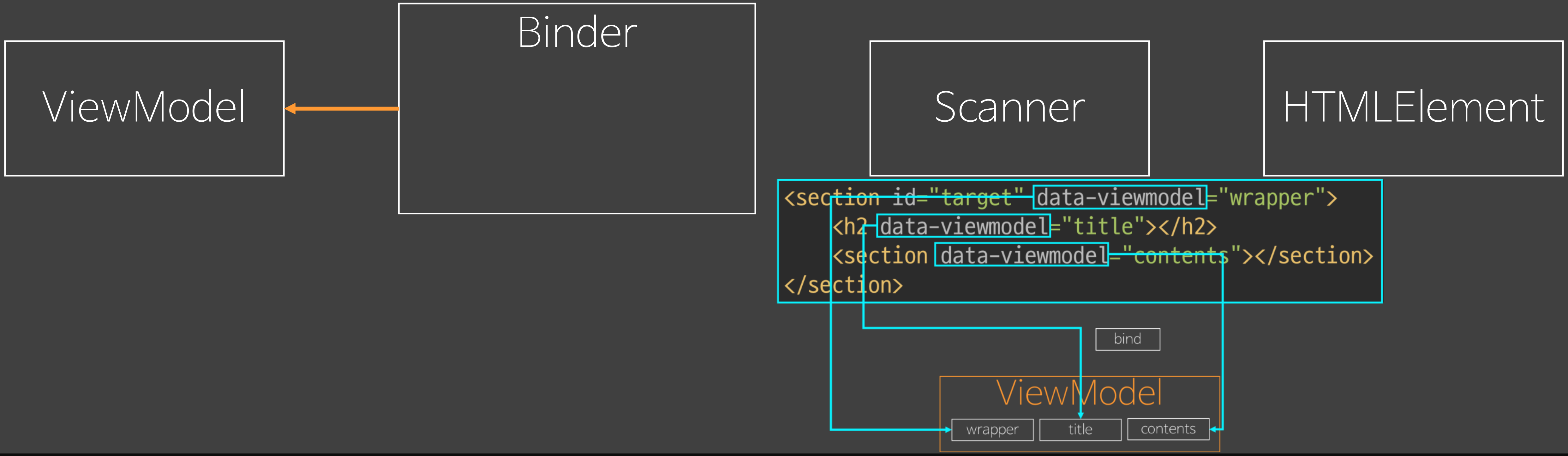


ViewModel

Binder

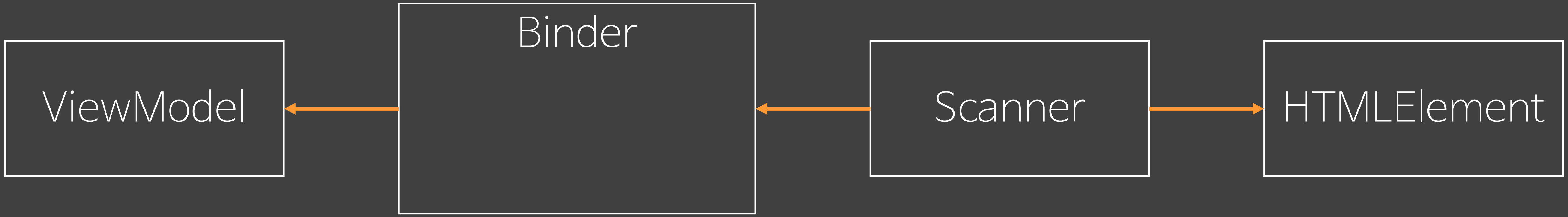
Scanner

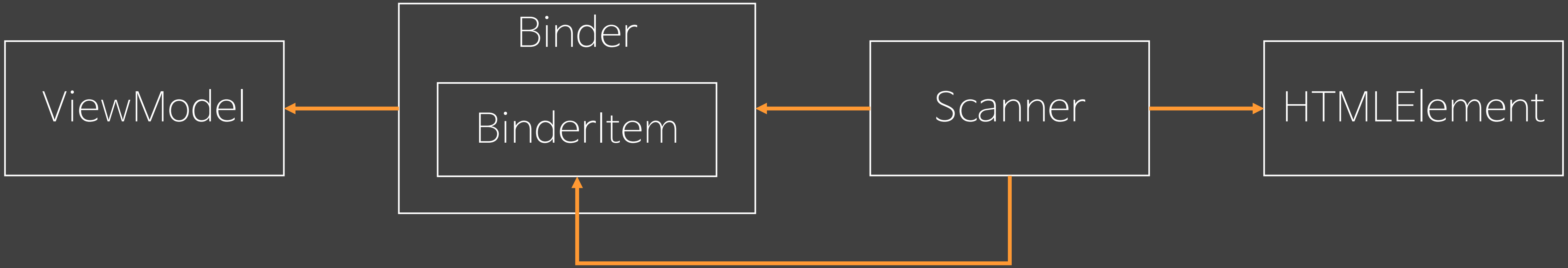
HTMLElement



ViewModel
Binder
Scanner
HTMLElement
bind
ViewModel
wrapper
title
contents

```
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```


ViewModel
Binder
Scanner
HTMLElement



ViewModel

Binder

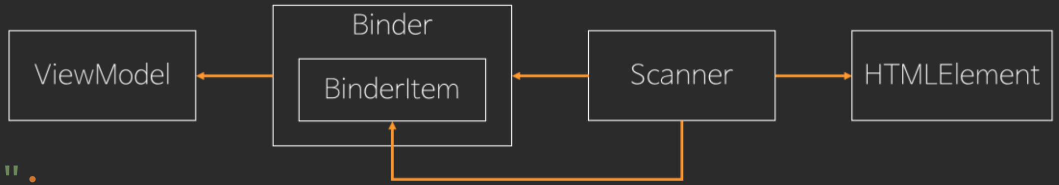
BinderItem

Scanner

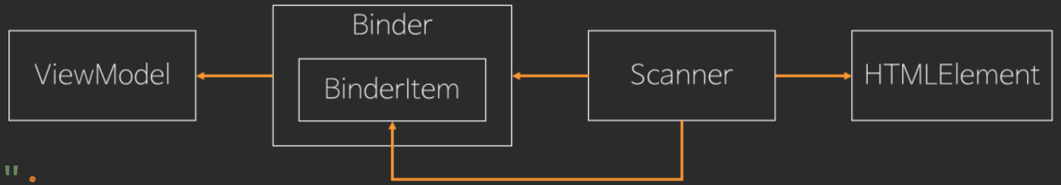
HTMLElement

ViewModel

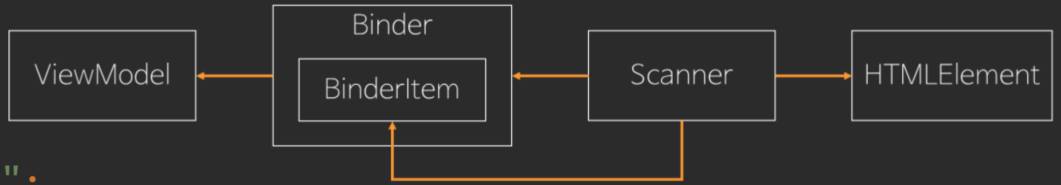
```
const ViewModel = class{
    static #private = Symbol();
    static get(data){
        return new ViewModel(this.#private, data);
    }
    styles = {}; attributes = {}; properties = {}; events = {};
    constructor(checker, data){
        if(checker != ViewModel.#private) throw "use ViewModel.get()!";
        Object.entries(data).forEach(([k, v])=>{
            switch(k){
            case"styles": this.styles = v; break;
            case"attributes": this.attributes = v; break;
            case"properties": this.properties = v; break;
            case"events": this.events = v; break;
            default: this[k] = v;
            }
        });
        Object.seal(this);
    }
};
```



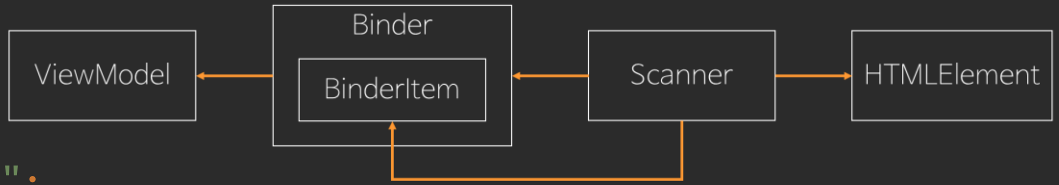
```
const ViewModel = class{
    static #private = Symbol();
    static get(data){
        return new ViewModel(this.#private, data);
    }
    styles = {}; attributes = {}; properties = {}; events = {};
    constructor(checker, data){
        if(checker != ViewModel.#private) throw "use ViewModel.get()!";
        Object.entries(data).forEach(([k, v])=>{
            switch(k){
            case"styles": this.styles = v; break;
            case"attributes": this.attributes = v; break;
            case"properties": this.properties = v; break;
            case"events": this.events = v; break;
            default: this[k] = v;
            }
        });
        Object.seal(this);
    }
};
```



```
const ViewModel = class{
    static #private = Symbol();
    static get(data){
        return new ViewModel(this.#private, data);
    }
    styles = {}; attributes = {}; properties = {}; events = {};
    constructor(checker, data){
        if(checker != ViewModel.#private) throw "use ViewModel.get()!";
        Object.entries(data).forEach(([k, v])=>{
            switch(k){
            case"styles": this.styles = v; break;
            case"attributes": this.attributes = v; break;
            case"properties": this.properties = v; break;
            case"events": this.events = v; break;
            default: this[k] = v;
            }
        });
        Object.seal(this);
    }
};
```

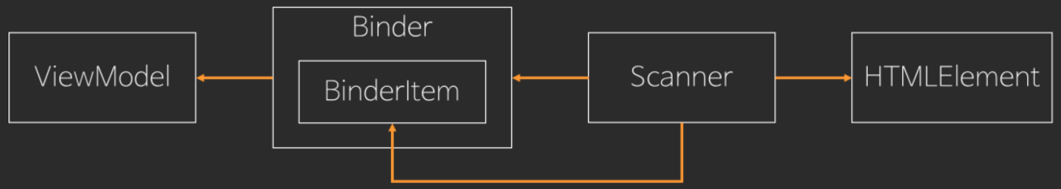


```
const ViewModel = class{
    static #private = Symbol();
    static get(data){
        return new ViewModel(this.#private, data);
    }
    styles = {}; attributes = {}; properties = {}; events = {};
    constructor(checker, data){
        if(checker != ViewModel.#private) throw "use ViewModel.get()!";
        Object.entries(data).forEach(([k, v])=>{
            switch(k){
            case"styles": this.styles = v; break;
            case"attributes": this.attributes = v; break;
            case"properties": this.properties = v; break;
            case"events": this.events = v; break;
            default: this[k] = v;
            }
        });
        Object.seal(this);
    }
};
```

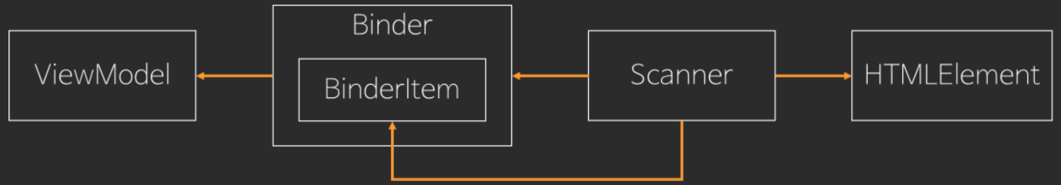


Binder

```
const BinderItem = class{
  el; viewmodel;
  constructor(el, viewmodel, _0=type(el, HTMLElement), _1=type(viewmodel, "string")){
      this.el = el;
      this.viewmodel = viewmodel;
      Object.freeze(this);
  }
};
```

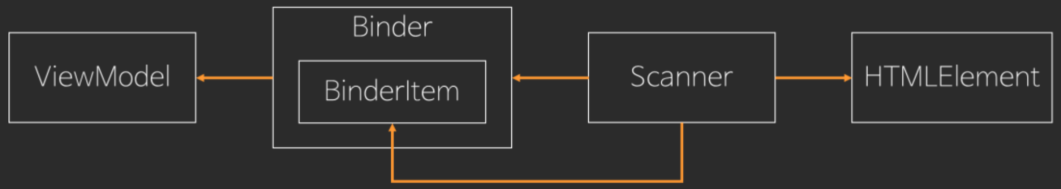


```
const BinderItem = class{
  el; viewmodel;
  constructor(el, viewmodel, _0=type(el, HTMLElement), _1=type(viewmodel, "string")){
      this.el = el;
      this.viewmodel = viewmodel;
      Object.freeze(this);
  }
};
```



```
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```

```
const BinderItem = class{
  el; viewmodel;
  constructor(el, viewmodel, _0=type(el, HTMLElement), _1=type(viewmodel, "string")){
      this.el = el;
      this.viewmodel = viewmodel;
      Object.freeze(this);
  }
};
```

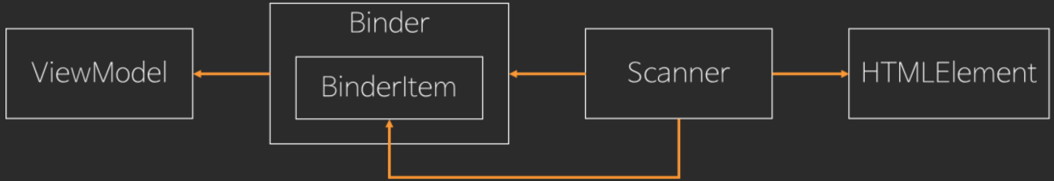


```
new BinderItem(section, "wrapper");
new BinderItem(h2, "title");
new BinderItem(section2, "contents");
```

```
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```

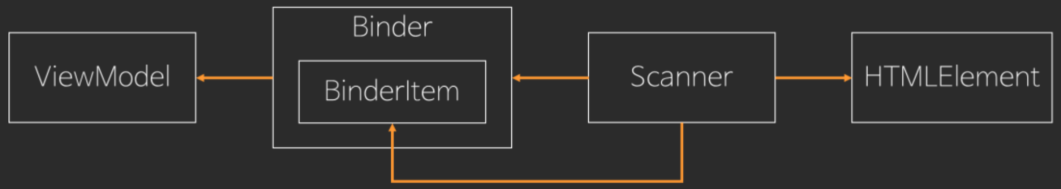
```
const BinderItem = class{
  el; viewmodel;
  constructor(el, viewmodel, _0=type(el, HTMLElement), _1=type(viewmodel, "string")){
      this.el = el;
      this.viewmodel = viewmodel;
      Object.freeze(this);
  }
};

const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```



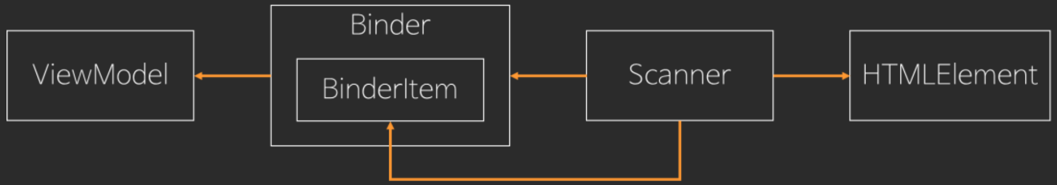
```
const BinderItem = class{
  el; viewmodel;
  constructor(el, viewmodel, _0=type(el, HTMLElement), _1=type(viewmodel, "string")){
      this.el = el;
      this.viewmodel = viewmodel;
      Object.freeze(this);
  }
};

const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```



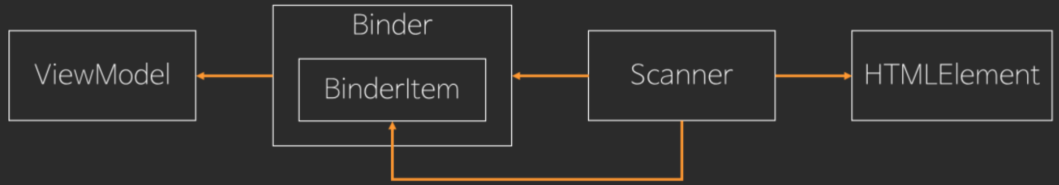
```
const BinderItem = class{
  el; viewmodel;
  constructor(el, viewmodel, _0=type(el, HTMLElement), _1=type(viewmodel, "string")){
      this.el = el;
      this.viewmodel = viewmodel;
      Object.freeze(this);
  }
};

const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```



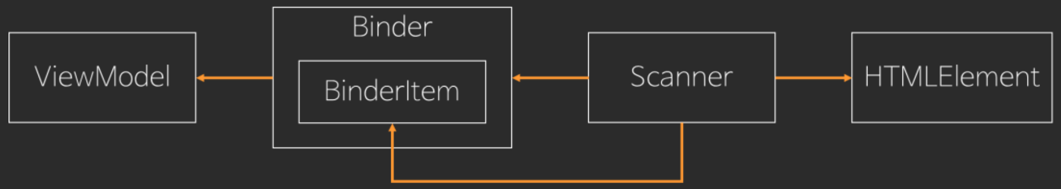
```
const BinderItem = class{
  el; viewmodel;
  constructor(el, viewmodel, _0=type(el, HTMLElement), _1=type(viewmodel, "string")){
      this.el = el;
      this.viewmodel = viewmodel;
      Object.freeze(this);
  }
};

const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```



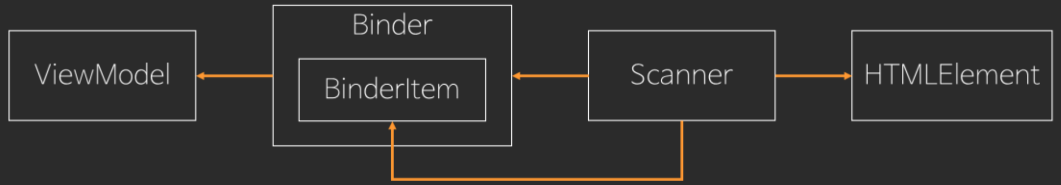
```
const BinderItem = class{
  el; viewmodel;
  constructor(el, viewmodel, _0=type(el, HTMLElement), _1=type(viewmodel, "string")){
      this.el = el;
      this.viewmodel = viewmodel;
      Object.freeze(this);
  }
};

const Binder = class{
  #items = new Set;
  add(v, _ = type(v, BinderItem)){this.#items.add(v);}
  render(viewmodel, _ = type(viewmodel, ViewModel)){
      this.#items.forEach(item=>{
          const vm = type(viewmodel[item.viewmodel], ViewModel), el = item.el;
          Object.entries(vm.styles).forEach(([k, v])=>el.style[k] = v);
          Object.entries(vm.attributes).forEach(([k, v])=>el.setAttribute(k, v));
          Object.entries(vm.properties).forEach(([k, v])=>el[k] = v);
          Object.entries(vm.events).forEach(([k, v])=>el["on" + k] =e=>v.call(el, e, viewmodel));
      });
  }
};
```

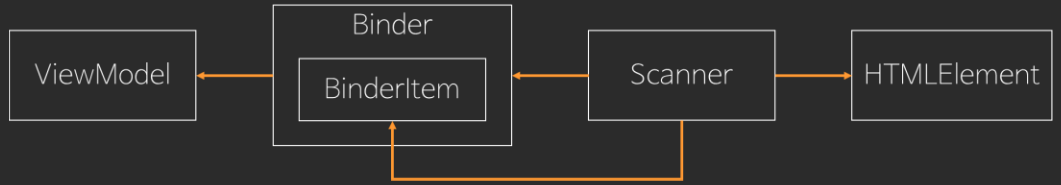


Scanner

```
const Scanner = class{
  scan(el, _ = type(el, HTMLElement)){
      const binder = new Binder;
      this.checkItem(binder, el);
      const stack = [el.firstElementChild];
      let target;
      while(target = stack.pop()){
          this.checkItem(binder, target);
          if(target.firstElementChild) stack.push(target.firstElementChild);
          if(target.nextElementSibling) stack.push(target.nextElementSibling);
      }
      return binder;
  }
  checkItem(binder, el){
      const vm = el.getAttribute("data-viewmodel");
      if(vm) binder.add(new BinderItem(el, vm));
  }
};
```



```
const Scanner = class{
  scan(el, _ = type(el, HTMLElement)){
      const binder = new Binder;
      this.checkItem(binder, el);
      const stack = [el.firstElementChild];
      let target;
      while(target = stack.pop()){
          this.checkItem(binder, target);
          if(target.firstElementChild) stack.push(target.firstElementChild);
          if(target.nextElementSibling) stack.push(target.nextElementSibling);
      }
      return binder;
  }
  checkItem(binder, el){
      const vm = el.getAttribute("data-viewmodel");
      if(vm) binder.add(new BinderItem(el, vm));
  }
};
```



ViewModel

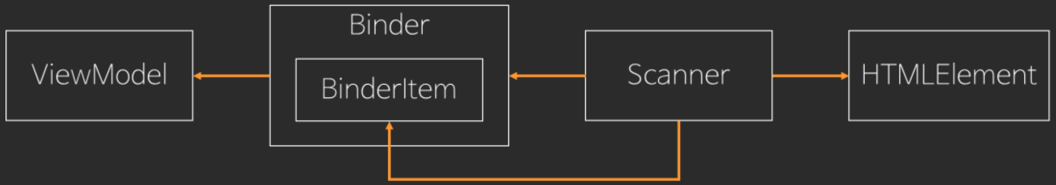
Binder

BinderItem

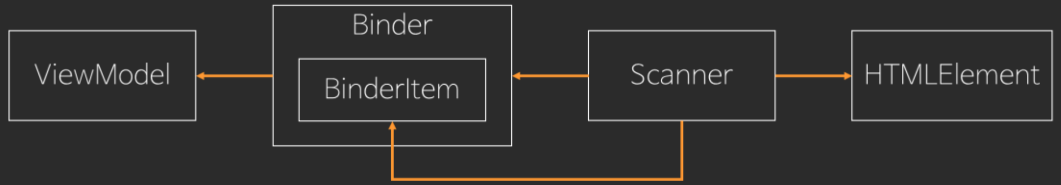
Scanner

HTMLElement

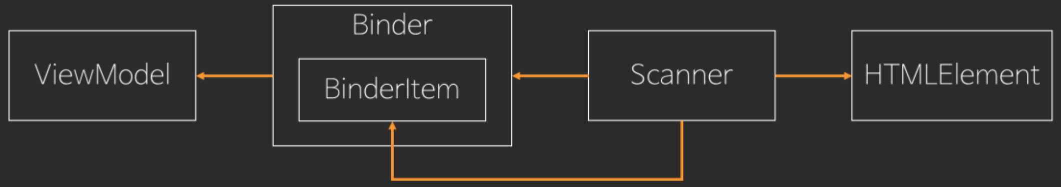
```
const Scanner = class{
  scan(el, _ = type(el, HTMLElement)){
      const binder = new Binder;
      this.checkItem(binder, el);
      const stack = [el.firstElementChild];
      let target;
      while(target = stack.pop()){
          this.checkItem(binder, target);
          if(target.firstElementChild) stack.push(target.firstElementChild);
          if(target.nextElementSibling) stack.push(target.nextElementSibling);
      }
      return binder;
  }
  checkItem(binder, el){
      const vm = el.getAttribute("data-viewmodel");
      if(vm) binder.add(new BinderItem(el, vm));
  }
};
```



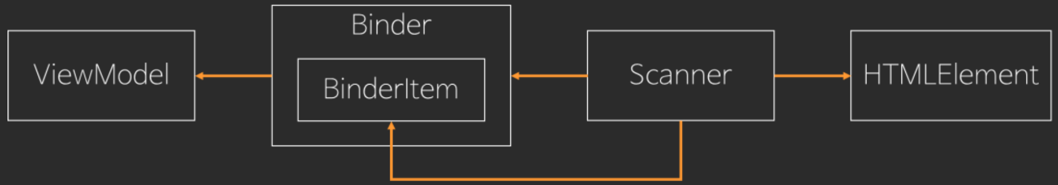
```
const Scanner = class{
  scan(el, _ = type(el, HTMLElement)){
      const binder = new Binder;
      this.checkItem(binder, el);
      const stack = [el.firstElementChild];
      let target;
      while(target = stack.pop()){
          this.checkItem(binder, target);
          if(target.firstElementChild) stack.push(target.firstElementChild);
          if(target.nextElementSibling) stack.push(target.nextElementSibling);
      }
      return binder;
  }
  checkItem(binder, el){
      const vm = el.getAttribute("data-viewmodel");
      if(vm) binder.add(new BinderItem(el, vm));
  }
};
```



```
const Scanner = class{
  scan(el, _ = type(el, HTMLElement)){
      const binder = new Binder;
      this.checkItem(binder, el);
      const stack = [el.firstElementChild];
      let target;
      while(target = stack.pop()){
          this.checkItem(binder, target);
          if(target.firstElementChild) stack.push(target.firstElementChild);
          if(target.nextElementSibling) stack.push(target.nextElementSibling);
      }
      return binder;
  }
  checkItem(binder, el){
      const vm = el.getAttribute("data-viewmodel");
      if(vm) binder.add(new BinderItem(el, vm));
  }
};
```



```
const Scanner = class{
  scan(el, _ = type(el, HTMLElement)){
      const binder = new Binder;
      this.checkItem(binder, el);
      const stack = [el.firstElementChild];
      let target;
      while(target = stack.pop()){
          this.checkItem(binder, target);
          if(target.firstElementChild) stack.push(target.firstElementChild);
          if(target.nextElementSibling) stack.push(target.nextElementSibling);
      }
      return binder;
  }
  checkItem(binder, el){
      const vm = el.getAttribute("data-viewmodel");
      if(vm) binder.add(new BinderItem(el, vm));
  }
};
```

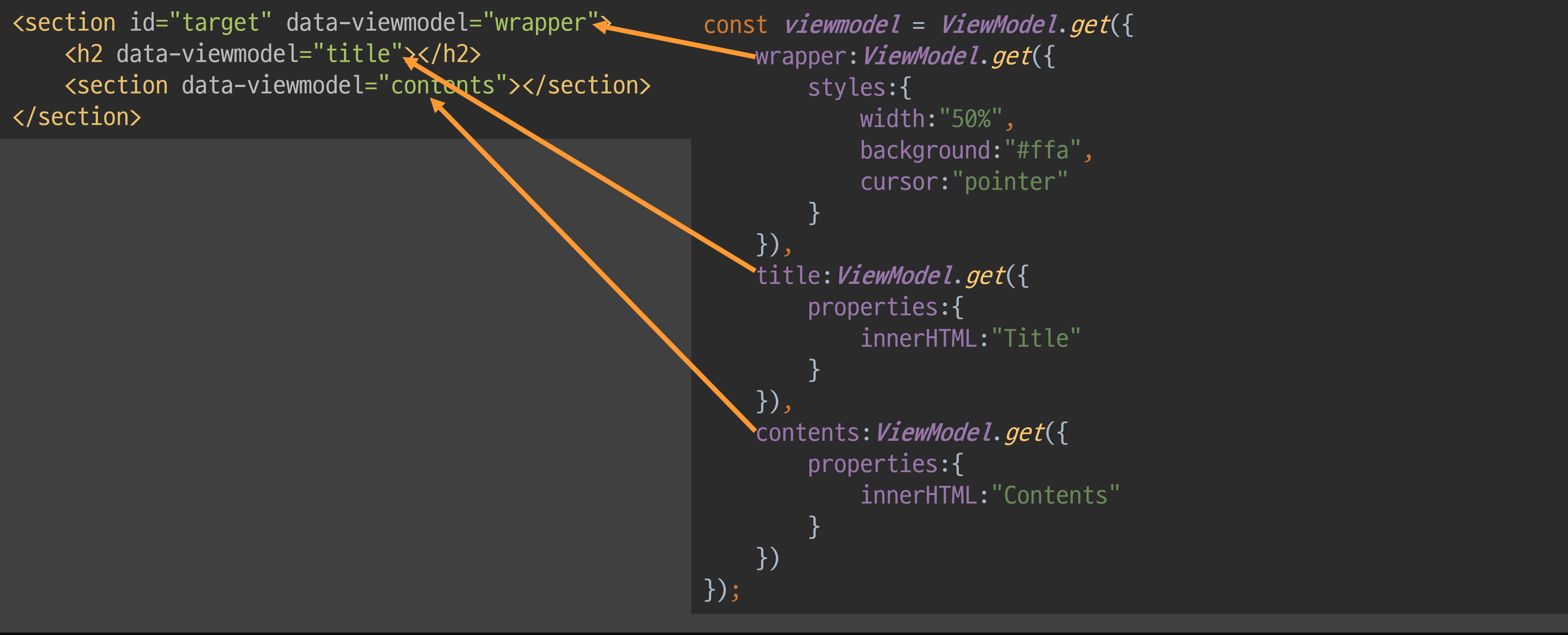


client

```
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```

```html
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```

```javascript
const viewmodel = ViewModel.get({
    wrapper:ViewModel.get({
        styles:{
            width:"50%",
            background:"#ffa",
            cursor:"pointer"
        }
    }),
    title:ViewModel.get({
        properties:{
            innerHTML:"Title"
        }
    }),
    contents:ViewModel.get({
        properties:{
            innerHTML:"Contents"
        }
    })
});
```



```html
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```

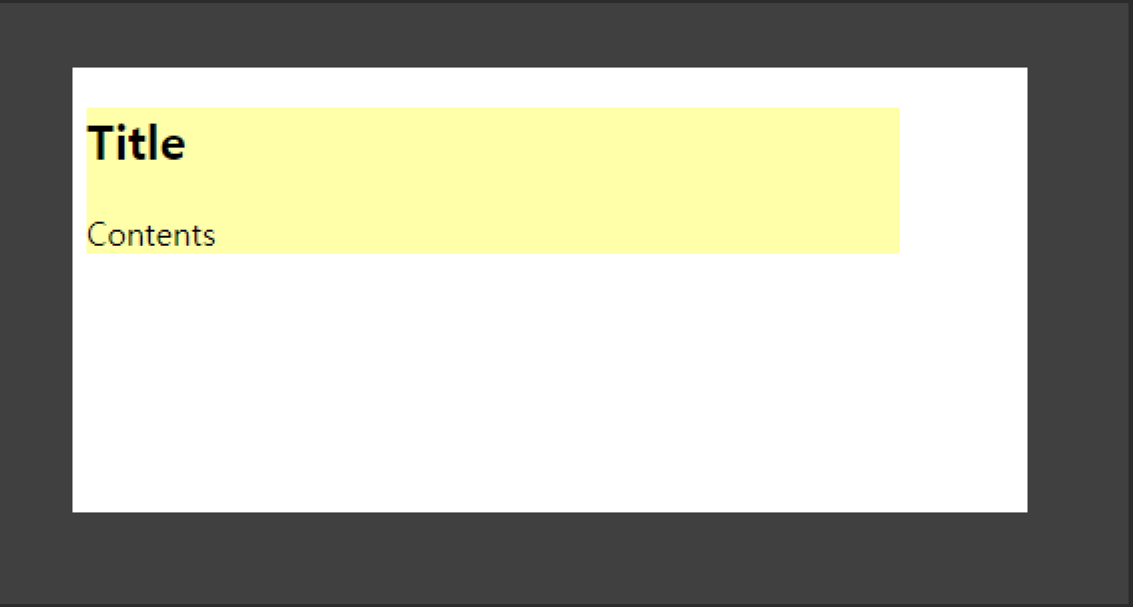
```javascript
const viewmodel = ViewModel.get({
    wrapper:ViewModel.get({
        styles:{
            width:"50%",
            background:"#ffa",
            cursor:"pointer"
        }
    }),
    title:ViewModel.get({
        properties:{
            innerHTML:"Title"
        }
    }),
    contents:ViewModel.get({
        properties:{
            innerHTML:"Contents"
        }
    })
});
```

```html
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```

```javascript
const viewmodel = ViewModel.get({
    wrapper:ViewModel.get({
        styles:{
            width:"50%",
            background:"#ffa",
            cursor:"pointer"
        }
    }),
    title:ViewModel.get({
        properties:{
            innerHTML:"Title"
        }
    }),
    contents:ViewModel.get({
        properties:{
            innerHTML:"Contents"
```

```javascript
const scanner = new Scanner;
const binder = scanner.scan(document.querySelector("#target"));
binder.render(viewmodel);
```

```html
<section id="target" data-viewmodel="wrapper">
    <h2 data-viewmodel="title"></h2>
    <section data-viewmodel="contents"></section>
</section>
```

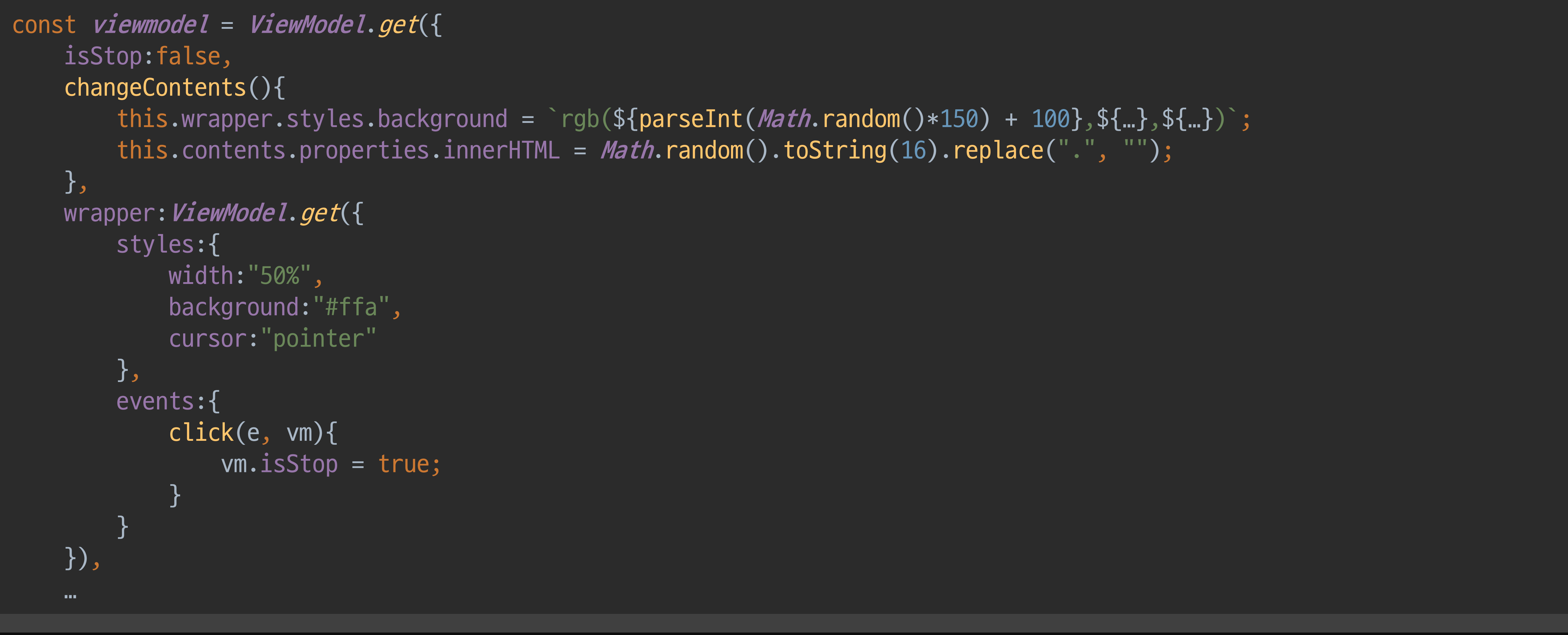


Title

Contents

```javascript
const viewmodel = ViewModel.get({
    wrapper:ViewModel.get({
        styles:{
            width:"50%",
            background:"#ffa",
            cursor:"pointer"
        }
    }),
    title:ViewModel.get({
        properties:{
            innerHTML:"Title"
        }
    }),
    contents:ViewModel.get({
        properties:{
            innerHTML:"Contents"
```

```javascript
const scanner = new Scanner;
const binder = scanner.scan(document.querySelector("#target"));
binder.render(viewmodel);
```



```
const viewmodel = ViewModel.get({
    isStop:false,
    changeContents(){
        this.wrapper.styles.background = `rgb(${parseInt(Math.random()*150) + 100},${…},${…})`;
        this.contents.properties.innerHTML = Math.random().toString(16).replace(".", "");
    },
    wrapper:ViewModel.get({
        styles:{
            width:"50%",
            background:"#ffa",
            cursor:"pointer"
        },
        events:{
            click(e, vm){
                vm.isStop = true;
            }
        }
    }),
    …
```

```
const viewmodel = ViewModel.get({
    isStop:false,
    changeContents(){
        this.wrapper.styles.background = `rgb(${parseInt(Math.random()*150) + 100},${…},${…})`;
        this.contents.properties.innerHTML = Math.random().toString(16).replace(".", "");
    },
    wrapper:ViewModel.get({
        styles:{
            width:"50%",
            background:"#ffa",
            cursor:"pointer"
        },
        events:{
            click(e, vm){
                vm.isStop = true;
            }
        }
    }),
    …
```

```
const f =_=>{
  viewmodel.changeContents();
  binder.render(viewmodel);
  if(!viewmodel.isStop) requestAnimationFrame(f);
};
requestAnimationFrame(f);
```