5 Polymer : Properties, Data Binding, Observer & Compute

Objektif :

- **Mahasiswa dapat memahami tentang Properties Polymer 3**
- **Mahasiswa dapat memahami tentang Data Binding Polymer 3**
- **Mahasiswa dapat memahami tentang Observer Polymer 3**
- **Mahasiswa dapat memahami tentang Compute Polymer 3**

5.1 Apa itu Web Components?

Web components adalah kumpulan fitur pada web browser yang memungkinkan untuk membuat widget (atau yang lebih dikenal dengan istilah component) dalam pengembangan web. Layaknya konsep plugin pada wordpress atau CMS serupa, component ini siap dipakai ulang pada halaman HTML manapun tanpa memerlukan library atau framework tambahan.

Tujuan utamanya adalah memberikan kemudahan untuk penulisan kode dengan struktur berbasis komponen (component-based software engineering). Setiap komponen mewakili satu atau lebih fungsionalitas sistem dan "harus" bersifat reusable. Beberapa framework javascript seperti vue, react atau angular yang sama-sama men-support penulisan kode berbasis komponen juga, namun web components merupakan fitur native di web browser.

Contoh sederhananya, misal ada banyak aplikasi web yang dibangun dengan framework yang berbeda-beda. Dengan satu codebase HTML yang sama kita bisa membuat sebuah footer menjadi component, sehingga footer tersebut dapat dipasang di semua aplikasi web menggunakan HTML Imports.

```
<link rel="import" href="my-footer.html">
```

Kemudian memanggilnya dengan Custom Element yang sudah didefinisikan.

```
<my-footer></my-footer>
```

Tidak hanya footer, component bisa saja berupa datepicker, login modal, ads banner, custom button bahkan proses autentifikasi single sign on. Setiap component tidak harus memiliki UI karena sebuah template umumnya terdiri dari blok HTML, CSS dan JS. Bisa saja isinya hanya berupa blok javascript.

Contoh markup elemen HTML Template :

```
<template>
  <style>
    /* BLOCK CSS */
  </style>
  <div>
      <!-- BLOCK HTML -->
  </div>
</template>
<script>
  // BLOCK JS
</script>
```

5.1.1 Apa itu Kerangka (framework) Javascript?

Di dunia development, sebuah "kerangka (framework)", dapat didefinisikan sebagai pustaka Javascript yang menampilkan dan menterjemahkan "data-driven", antarmuka interaktif. Dan digunakan untuk membantu menterjemahkan data ke pengguna saat interaksi dipicu. Semua pustaka (library) sedikit berbeda antara satu dengan yang lainnya, tapi tujuan mereka tetap sama, yaitu menampilkan data baru saat "interaksi" terjadi.

- Pemain-Pemain Kerangka (framework) Javascript

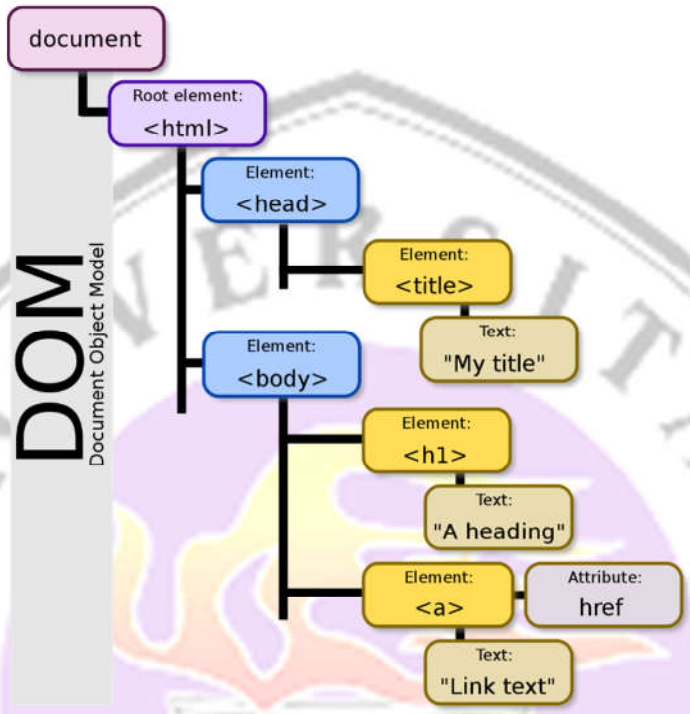
  Ada banyak pilihan terkait memilih kerangka, berikut ini adalah daftar yang ada, termasuk Vue, React, Angular, Ember, polymer dll. Masing masing dipilih berdasarkan perkembangan komunitas dan keberlanjutannya, aktifitas GitHub, seberapa lama proyek ini sudah tersedia untuk developer, dan apakah masih aktif digunakan di group developer dan perusahaan pengembangan.

5.1.2 Polymer JS

Pada bagian modul Google Go language beginner sudah dijelaskan mengenai pengenalan polymer, pada bagian pertemuan modul kali ini akan dibahas mengenai Data Properties, Binding, Observer & Compute pada polymer. Sebelum membahas itu semua, ada baiknya dibahas sedikit mengenai DOM.

DOM (Document Object Model) adalah model data standar. DOM adalah cara javascript melihat suatu halaman html. DOM adalah sebuah platform dan interface yang

memperbolehkan pengaksesan dan perubahan pada konten, struktur, dan style pada sebuah dokumen oleh program dan script. Istilah HTML DOM mengacu kepada dokumen html. Kasusnya disini ialah konten, struktur, dan style pada dokumen html dapat diakses dan dirubah dengan menggunakan sintaks javascript.



Gambar 5.1 DOM

DOM pada HTML adalah model objek standar dan antarmuka pemrograman untuk HTML, dan semuanya di definisikan pada :

- Elemen HTML sebagai objek
- Properti semua elemen HTML
- Metode untuk mengakses semua elemen HTML
- Event untuk semua elemen HTML

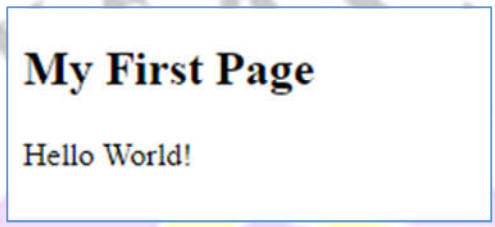
Method DOM HTML adalah tindakan yang dapat Anda lakukan (pada Elemen HTML). Properti DOM HTML adalah nilai (dari Elemen HTML) yang dapat diatur atau ubah. HTML DOM dapat diakses dengan JavaScript (dan dengan bahasa pemrograman lain). Pada DOM semua elemen HTML didefinisikan sebagai objek. Antarmuka pemrograman adalah properti dan metode masing-masing objek. Properti adalah nilai yang bisa Anda dapatkan atau setel

(seperti mengubah konten elemen HTML). Method/Metode adalah tindakan yang dapat dilakukan (seperti menambah atau menghapus elemen HTML).

Contoh berikut mengubah konten ***(innerHTML)*** **`<p>`** elemen dengan **`id ="demo"`:**

```
<html>
	<body>
		<p id="demo"></p>
	<script>
		document.getElementById("demo").innerHTML = "Hello 		World!";
	</script>
	</body>
</html>
```

Maka ketika di jalankan, output program tersebut adalah



Dalam contoh di atas, *getElementById* adalah sebuah method, sementara *innerHTML* adalah properti. Cara paling umum untuk mengakses elemen HTML adalah dengan menggunakan id elemen tersebut. Pada contoh di atas *getElementById* adalah metode yang digunakan id="demo"untuk menemukan elemen. cara termudah untuk mendapatkan konten suatu elemen dari Properti *innerHTML* adalah dengan menggunakan *innerHTML* properti. *innerHTML* properti ini berguna untuk mendapatkan atau mengganti isi dari elemen HTML. *innerHTML* properti dapat digunakan untuk mendapatkan atau mengubah setiap elemen HTML, termasuk **`<html>`** dan **`<body>`**.

5.2 Properties

Properti adalah atribut yang dapat digunakan dalam pembuatan tampilan yang akan dimunculkan pada bagian depan halaman website. Proses mendeklarasikan properti pada elemen adalah untuk menambahkan nilai default dan mengaktifkan berbagai fitur dalam sistem data. Properti yang dideklarasikan dapat menentukan:

- Jenis properti.
- Nilai standar. (Default)
- *Property change observer.* Memanggil metode kapan pun nilai properti berubah.
- *Read-only status*. Mencegah perubahan yang tidak disengaja pada nilai properti.

- *Two-way data binding support*. Menghidupkan suatu peristiwa setiap kali nilai properti berubah.
- *Computed property*. Secara dinamis menghitung nilai berdasarkan properti lainnya.
- *Property reflection to attribute*. Memperbarui nilai atribut yang sesuai ketika nilai properti berubah.

Banyak fitur-fitur ini terintegrasi erat ke dalam sistem data, dan didokumentasikan dalam bagian sistem data. Selain itu, properti yang dideklarasikan dapat dikonfigurasi dari markup menggunakan atribut. Dalam kebanyakan kasus, properti yang merupakan bagian dari API publik elemen Anda harus dideklarasikan di objek properti. Untuk mendeklarasikan properti, tambahkan pengambil properti statis ke dalam kelas elemen. Pengambil harus mengembalikan objek yang berisi deklarasi properti.

Contoh :

```
class XCustom extends PolymerElement {
  static get properties() {
    return {
      user: String,
      isHappy: Boolean,
      count: {
        type: Number, readOnly: true, notify: true
      }
    }
  }
}
customElements.define('x-custom', XCustom);
```

Keterangan:

Kata Kunci	Keterangan
type	Type: constructor Jenis atribut, digunakan untuk deserializing (Mengubah byte stream ke dalam sebuah objek yang dapat digunakan kembali.) dari suatu atribut. Mendukung Polimer jenis deserializing berikut: Boolean, Tanggal, Nomor, String, Array dan Objek. Anda bisa menambahkan dukungan untuk tipe lain dengan mengganti metode _deserializeValue elemen. Default: String
value	Type: boolean, number, string or function. Nilai default untuk properti. Jika nilai adalah fungsi, fungsi dipanggil dan nilai kembali digunakan sebagai nilai default properti. Jika nilai default harus berupa array atau objek unik untuk instance, untuk array

	atau objek di dalam suatu fungsi. Default: undefined
reflectToAttribute	Type: boolean Di setting ke true untuk menyebabkan atribut terkait diset pada node host ketika nilai properti berubah. Jika nilai properti adalah Boolean, atribut dibuat sebagai atribut boolean HTML standar (diset jika benar, tidak diset jika salah). Default: false
readOnly	Jika benar, properti tidak dapat ditetapkan secara langsung oleh penugasan atau pengikatan data. Default: false
notify	Type: boolean Jika benar, properti tersedia untuk data binding dua arah. Selain itu, suatu event, property-name-changed dilepaskan setiap kali properti berubah. Default: false
computed	Type: string Nilai ditafsirkan sebagai nama metode dan daftar argumen. Metode dipanggil untuk menghitung nilai setiap kali salah satu nilai argumen berubah. Properti yang dihitung selalu hanya read-only. Default: tidak ada
observer	Type: string Nilai ditafsirkan sebagai nama metode yang akan dipanggil ketika nilai properti berubah. Default: tidak ada

5.3 Konsep Sistem Data pada Polymer

Polymer memungkinkan developer mengamati perubahan pada properti elemen dan mengambil berbagai tindakan berdasarkan perubahan data. Tindakan ini, atau efek propertinya termasuk :

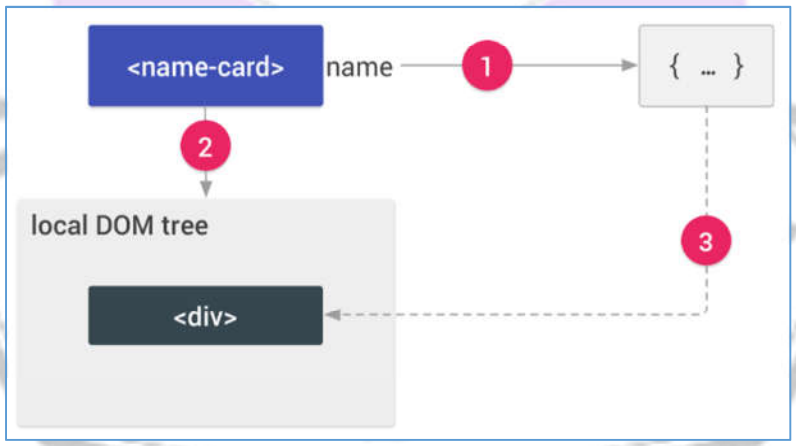
- ***Data bindings***. Anotasi yang memperbarui properti, atribut, atau konten teks dari simpul DOM saat data berubah.

- ***Observers***. Panggilan balik dipanggil saat data berubah.
- ***Computed properties***. Properti virtual dihitung berdasarkan properti lainnya, dan dihitung ulang saat input data berubah.

Setiap elemen Polymer mengelola model datanya sendiri dan elemen DOM lokal. Model untuk elemen adalah properti elemen. Binding data menghubungkan model elemen dengan elemen-elemen di DOM lokalnya.

Contoh :

```
class NameCard extends PolymerElement {
  constructor() {
    super();
    this.name = {first: 'Kai', last: 'Li'};
  }
  static get template() {
    return html `
      <div>[[name.first]] [[name.last]]</div>
    `;
  }
}
customElements.define('name-card', NameCard);
```



Gambar 5.2 Elemen DOM

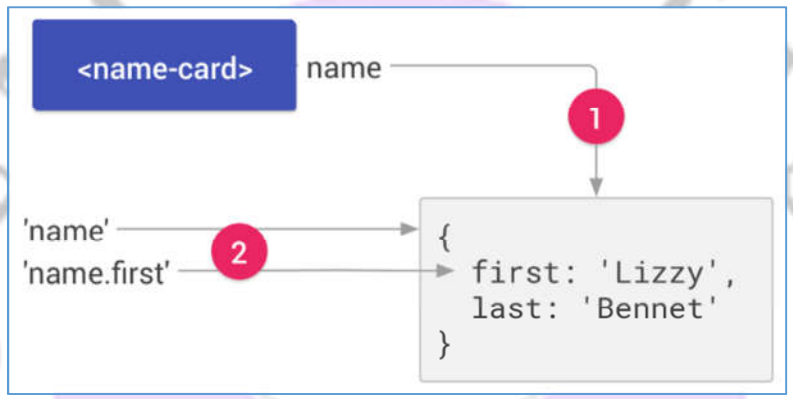
Keterangan :

1. Elemen **<name-card>** memiliki properti nama yang merujuk ke objek JavaScript.
2. Elemen **<name-card>** meng-host tree DOM lokal, yang berisi elemen ** ** tunggal.
3. Binding data dalam templat menautkan objek JavaScript ke elemen ** .**

Sistem data didasarkan pada path, bukan objek, di mana path mewakili properti atau subproperti relatif terhadap elemen host. Sebagai contoh, elemen **<name-card>** memiliki binding data untuk path "name.first" dan "name.last". Jika **<nama-card>** memiliki objek berikut dengan properti namanya:

```
{
  first: 'Lizzy',
  last: 'Bennet'
}
```

Path "name" merujuk ke properti nama elemen (objek). Path "name.first" dan "name.last" merujuk ke properti objek itu.



Gambar 5.3 Path

Keterangan :

1. Properti nama merujuk ke objek JavaScript.
2. Path "name" mengacu pada objek itu sendiri. Path "name.first" mengacu pada subproperti miliknya, pertama (string, Lizzy).

Polymer tidak secara otomatis mendeteksi semua perubahan data.

5.4 Data Binding

Data binding adalah pengikat data dari elemen yang dimodifikasi (element host) ke properti atau atribut elemen di dalam DOM lokal (elemen target). Data element host dapat berupa properti atau sub properti yang diwakili oleh jalur data atau data yang dihasilkan berdasarkan satu jalur atau lebih. Untuk membuat Data binding dengan menambahkan anotasi ke templat DOM lokal elemen, seperti skrip dibawah ini :

```
static get template() {
  return html`
```

```
    <target-element           target-property="{{hostProperty}}"></target-
element>`;
}
```

5.4.1 Anatomi Data Binding

Data binding muncul di templat DOM lokal sebagai atribut HTML, seperti skrip dibawah ini :

```
property-name = annotation-or-compound-binding
attribute-name$ = annotation-or-compound-binding
```

Sisi kiri dari binding, mengidentifikasi properti atau atribut target, agar dapat mengikat ke dalam properti, gunakan nama properti dalam bentuk atribut dash-case seperti **`my-element`** untuk HTML dan bukan camelCase untuk di Polymer seperti **`myProperty`** :

```
<my-element my-property = "{{hostProperty}}" ">
```

Contoh binding diatas mengarah ke properti target, **`my-property`** di **`<my-element>`**. Untuk mengikat atribut, dapat menggunakan nama atribut yang diikuti oleh simbol $:

```
<a href$="{{hostProperty}}">
```

Sisi kanan data binding terdiri dari *binding annotation* atau *compound (gabungan) binding*.

- **Binding annotation** adalah Teks yang dikelilingi oleh pembatas kurung kurawal ganda ({{…}}) atau pembatas kurung siku ganda ([[…]]). Mengidentifikasi data host yang sedang terikat.
- **Compound binding** yaitu sebuah string literal yang mengandung satu atau lebih penjelasan yang mengikat.

Apakah aliran data turun dari host ke target, naik dari target ke host, atau kedua cara dikendalikan oleh jenis anotasi yang mengikat dan konfigurasi properti target. Kurung kurawal ganda ({{…}}) mendukung aliran data ke atas dan ke bawah, sedangkan kurung siku kotak ganda ([[…]]) adalah satu arah, dan hanya mendukung aliran data ke bawah.

5.4.2 Binding ke target properti

Untuk mengikat ke properti target, tentukan nama atribut yang sesuai dengan properti, dengan anotasi binding atau compound binding sebagai nilai atribut.

```
<target-element name="{{myName}}"></target-element>
```

Pada contoh skrip diatas mengikat properti nama elemen target ke properti **myName** elemen host. Karena anotasi ini menggunakan pembatas dua arah atau "otomatis" ({{…}}), untuk membuat pengikatan dua arah jika properti nama dikonfigurasikan untuk mendukungnya.

Untuk memastikan binding satu arah, gunakan tanda kurung ganda:

```
<target-element name = "[[myName]]"> </target-element>
```

Untuk mengikat properti elemen camelCase, gunakan dash-case dalam nama atribut. Sebagai contoh:

```
<!-- Bind <user-view>.firstName to this.managerName; -->
<user-view first-name="{{managerName}}"></user-view>
```

Jika properti yang diikat adalah objek atau array, kedua elemen mendapatkan referensi ke objek yang sama. Ini berarti bahwa salah satu elemen dapat mengubah objek dan pengikatan satu arah yang benar adalah hal tidak mungkin.

5.4.3 Bind ke sub-properti host

Data binding juga dapat menyertakan jalur ke sub-properti, seperti yang ditunjukkan di bawah ini:

File : main-view.js

```
import { PolymerElement, html } from '@polymer/polymer/polymer-element.js';
import './user-view.js';

class MainView extends PolymerElement {
  static get properties() {
    return {
      user: {
        type: Object,
        value: function () { return { given: "San", family: "Zhang" }; }
      }
    };
```

```
  }
  static get template() {
    return html`
      <user-view  given="{{user.given}}"  family="{{user.family}}"></user-
view>
    `;
  }
}
customElements.define('main-view', MainView);
```

File : user-view.js

```
import  {  PolymerElement,  html  }  from  '@polymer/polymer/polymer-
element.js';

class UserView extends PolymerElement {
  static get properties() {
    return {
      given: String,
      family: String
    };
  }
  static get template() {
    return html`
      <div>[[given]] [[family]]</div>
    `;
  }
}
customElements.define('user-view', UserView);
```

File Pemanggilan : index.html

```
<main-view></main-view>
```

Perubahan sub-properti tidak dapat diamati secara otomatis . Jika elemen host melakukan update ke subproperti, maka perlu menggunakan metode set.

```
//  Change a subproperty observably
this.set('name.last', 'Maturin');
```

Jika pengikatannya dua arah dan elemen target memperbarui properti yang terikat, perubahan akan berpengaruh ke atas (main-view.js) secara otomatis.

5.4.4 Binding ke text content

Untuk mengikat ke konten teks elemen target, cukup menyertakan anotasi atau binding yang mengikat di dalam elemen target.

Contoh :

```
import { PolymerElement, html } from '@polymer/polymer/polymer-element.js';

class UserView extends PolymerElement {
  static get properties() {
    return {
      name: String
    };
  }
  static get template() {
    return html`
      <div>[[name]]</div>
    `;
  }
}
customElements.define('user-view', UserView);
```

Cara menggunakan di HTML nya adalah:

```
<!-- usage -->
<user-view name="Samuel"></user-view>
```

Hasil outputnya adalah Samuel.

5.4.5 Binding ke target attribute

Pada sebagian besar kasus, data binding ke elemen lain harus menggunakan binding properti, di mana perubahan disebarkan dengan menetapkan nilai baru ke properti JavaScript pada elemen. Namun, kadang-kadang perlu mengatur atribut pada elemen, sebagai lawan dari

properti. Misalnya, ketika penyeleksi atribut digunakan untuk CSS atau untuk interoperabilitas dengan API berbasis atribut, seperti standar Accessible Rich Internet Applications (ARIA) untuk informasi aksesibilitas. Untuk mengikat atribut, tambahkan tanda dolar ($) setelah nama atribut, seperti contoh dibawah ini :

```
<div style$="color: {{myColor}};">
```

Di mana nilai atribut adalah binding annotation atau sebuah compound binding. Atribut mengikat hasil dalam panggilan ke:

```
element.setAttribute (attr, value);
```

Sebagai gantinya :

```
element.property = value;
```

Contohnya :

```
static get template() {
  return html`
    <!-- Attribute binding -->
    <my-element selected$="[[value]]"></my-element>
    <!-- results in <my-element>.setAttribute('selected', this.value); --
>

    <!-- Property binding -->
    <my-element selected="{{value}}"></my-element>
    <!-- results in <my-element>.selected = this.value; -->
  `;
}
```

Binding atribut selalu satu arah, host-to-target dan nilai diserialisasi menurut jenis nilai saat ini. Ada beberapa properti elemen asli yang umum yang tidak dapat diikat oleh Polymer secara langsung, karena pengikatan menyebabkan masalah pada satu atau lebih browser.

Tabel 5.1 Atribut binding yang mempengaruhi properti

Attribute	**Property**	**Notes**
class	classList, className	Maps to two properties with different formats.
style	style	By specification, style is considered a read-only reference to a CSSStyleDeclaration object.
href	href	-

for	htmlFor	-
data-*	dataset	Custom data attributes (attribute names starting with data-) are stored on the dataset property.
value	value	Only for <input type="number">.

Daftar ini mencakup properti yang saat ini diketahui menyebabkan masalah dengan binding properti. Properti lainnya juga dapat terpengaruh.

Terdapat berbagai alasan mengapa properti tidak dapat diikat:

- Cross-browser mengeluarkan dengan kemampuan untuk menempatkan kurung kurawal {{...}} di beberapa nilai atribut ini.
- Atribut yang memetakan ke properti JavaScript yang diberi nama berbeda (seperti kelas).
- Properti dengan struktur unik (seperti style).

Atribut binding berikut ini adalah yang terikat ke nilai dinamis (gunakan $ =):

```
<!-- class -->
<div class$="[[foo]]"></div>
<!-- style -->
<div style$="[[background]]"></div>
<!-- href -->
<a href$="[[url]]">
<!-- label for -->
<label for$="[[bar]]"></label>
<!-- dataset -->
<div data-bar$="[[baz]]"></div>
<!-- ARIA -->
<button aria-label$="[[buttonLabel]]"></button>
```

5.4.6 Binding ke Array item

Untuk menggunakan computed binding untuk melakukan binding item array tertentu, atau ke subproperti item array, seperti `array[index].name`.

Contoh berikut menunjukkan cara mengakses properti dari item array menggunakan computed binding. Function computed perlu dipanggil jika nilai subproperti berubah, atau

jika array itu sendiri dimutasi/diganti, sehingga binding akan menggunakan wildcard path, **myArray.*.**

```
class XCustom extends PolymerElement {
  static get properties() {
    return {
      myArray: {
        type: Array,
        value: [{ name: 'Bob' }, { name: 'Wing Sang' }]
      }
    };
  }
  // first argument is the change record for the array change,
  // change.base is the array specified in the binding
  arrayItem(change, index, path) {
    // this.get(path, root) returns a value for a path
    // relative to a root object.
    return this.get(path, change.base[index]);
  }
  ready() {
    super.ready();
    // mutate the array
    this.unshift('myArray', { name: 'Fatma' });
    // change a subproperty
    this.set('myArray.1.name', 'Rupert');
  }
  static get template() {
    return html`
      <div>[[arrayItem(myArray.*, 0, 'name')]]</div>
      <div>[[arrayItem(myArray.*, 1, 'name')]]</div>
    `;
  }
}
```

5.4.7 Binding Dua Arah

Binding dua arah dapat menyebarkan perubahan data baik ke bawah (dari host ke target) dan ke atas (dari target ke host). Agar perubahan menyebar ke atas, Anda harus

menggunakan pembatas pengikatan data otomatis ({{…}}) dan properti target harus diset notify: true.

Saat mengikat properti array atau objek, kedua elemen memiliki akses ke array atau objek bersama, dan keduanya dapat membuat perubahan. Gunakan pembatas yang mengikat otomatis untuk memastikan bahwa efek properti merambat ke atas.

5.4.8 Binding Dua Arah non-Polymer elemen

Untuk binding dua arah ke elemen asli atau elemen non-Polimer yang tidak mengikuti konvensi penamaan event, untuk dapat menentukan nama event yang diubah dalam anotasi menggunakan sintaks berikut:

```
target-prop="{{hostProp::target-change-event}}"
```

Contoh :

```
<!-- Listens for `input` event and sets hostValue to <input>.value -->
<input value="{{hostValue::input}}">

<!-- Listens for `change` event and sets hostChecked to <input>.checked -
->
<input type="checkbox" checked="{{hostChecked::change}}">

<!-- Listens for `timeupdate ` event and sets hostTime to
<video>.currentTime -->
<video url="..." current-time="{{hostTime::timeupdate}}">
```

Saat binding ke properti notifikasi standar pada elemen Polimer, menentukan nama event tidak diperlukan, karena konvensi standar adalah menangkap event yang diubah properti.

```
<!-- Listens for `value-changed` event -->
<my-element value="{{hostValue::value-changed}}">

<!-- Listens for `value-changed` event using Polymer convention by
default -->
<my-element value="{{hostValue}}">
```

5.5 Observers properties

Observers adalah metode yang digunakan ketika perubahan yang dapat diamati terjadi pada data elemen. Terdapat dua tipe dasar Observers:

- Observers sederhana yaitu mengamati satu properti.
- Observers kompleks dapat mengamati satu atau lebih properti atau jalur.

Untuk menggunakan sintaks yang berbeda untuk mendeklarasikan dua jenis observers ini, tetapi dalam kebanyakan kasus keduanya berfungsi dengan cara yang sama. Setiap observers memiliki satu atau lebih ketergantungan. Metode observers berjalan ketika perubahan yang diamati dibuat untuk ketergantungan.

Properti yang dihitung adalah properti virtual yang didasarkan pada satu atau lebih bagian data elemen. Properti yang dikomputasi dihasilkan oleh fungsi komputasi, observers kompleks yang mengembalikan nilai. Kecuali ditentukan dengan ketentuan lain, catatan tentang observers berlaku untuk observers sederhana, observers kompleks, dan properti yang dihitung.

5.5.1 Observers dan element initialization

Panggilan pertama ke observers ditunda sampai kriteria berikut dipenuhi:

- Elemen ini sepenuhnya dikonfigurasikan (nilai default telah ditetapkan dan binding data disebarkan).
- Setidaknya salah satu dependensi (ketergantungan) didefinisikan (yaitu, tidak memiliki nilai yang tidak terdefinisi).

Setelah panggilan awal, setiap perubahan yang dapat diamati ke dependensi akan menghasilkan panggilan ke observers, bahkan jika nilai baru untuk dependensi tidak ditentukan. Ini memungkinkan elemen untuk menghindari menjalankan observers dalam kasus default.

5.5.2 Observe sebuah properti

Tetapkan observer sederhana dengan menambahkan kunci observer ke deklarasi properti, mengidentifikasi metode pengamat dengan nama.

Contoh :

```
import { PolymerElement } from '@polymer/polymer/polymer-element.js';

class XCustom extends PolymerElement {
  static get properties() {
```

```
    return {
      active: {
        type: Boolean,
        // Observer method identified by name
        observer: '_activeChanged'
      }
    }
  }
  // Observer method defined as a class method
  _activeChanged(newValue, oldValue) {
    this.toggleClass('highlight', newValue);
  }
}
customElements.define('x-custom', XCustom);
```

Method observer biasanya didefinisikan pada kelas itu sendiri, meskipun metode observer juga dapat didefinisikan oleh superclass, subclass, atau mixin kelas, selama metode yang disebutkan ada pada elemen.

5.5.3 Observers Kompleks

Observers kompleks dideklarasikan dalam array observers. Observers yang kompleks dapat memonitor satu atau lebih path. Path ini disebut dependensi observers.

```
static get observers() {
  return [
    // Observer method name, followed by a list of dependencies, in
parenthesis
    'userListChanged(users.*, filter)'
  ]
}
```

Setiap dependensi (ketergantungan) mewakili:

- Properti tertentu (misalnya, firstName).
- Subproperti tertentu (misalnya, address.street).
- Mutasi pada array tertentu (misalnya, users.splices).
- Semua perubahan sub-properti dan mutasi array di bawah jalur yang diberikan (misalnya, pengguna. *).

Method observers dipanggil dengan satu argumen untuk setiap ketergantungan. Berikut ini adalah jenis argumen bervariasi tergantung pada jalur yang diamati, yaitu :

- Untuk properti sederhana atau dependensi subproperti, argumennya adalah nilai baru dari properti atau subproperti.
- Untuk mutasi array argumen adalah catatan perubahan yang menjelaskan perubahan.
- Untuk lintasan wildcard, argumen adalah catatan perubahan yang menjelaskan perubahan, termasuk lintasan persis yang berubah.

Perhatikan bahwa argumen mana pun dapat tidak terdefinisi saat observers dipanggil. observers yang kompleks seharusnya hanya bergantung pada dependensi yang menyatakan.

5.5.4 Observe mengubah pada banyak properti

Observers dapat mengubah pada set properti, maka dapat menggunakan array observers. observers ini berbeda dari pengamat properti tunggal dalam beberapa cara, yaitu:

- Observers multi-properti dan properti yang dihitung berjalan sekali pada inisialisasi jika ada dependensi yang ditentukan. Setelah itu, observers pindah setiap kali ada perubahan yang dapat diamati untuk ketergantungan.
- Observers tidak menerima nilai lama sebagai argumen, hanya nilai baru. Hanya observers properti tunggal yang didefinisikan dalam objek properti yang menerima nilai lama dan baru.

Contoh :

```
import { PolymerElement } from '@polymer/polymer/polymer-element.js';

class XCustom extends PolymerElement {
  static get properties() {
    return {
        preload: Boolean,
        src: String,
        size: String
    }
  }

  // Each item of observers array is a method name followed by
  // a comma-separated list of one or more dependencies.
  static get observers() {
    return [
        'updateImage(preload, src, size)'
```

```
    ]
  }
  // Each method referenced in observers must be defined in
  // element prototype. The arguments to the method are new value
  // of each dependency, and may be undefined.
  updateImage(preload, src, size) {
    // ... do work using dependent values
  }
}

customElements.define('x-custom', XCustom);
```

5.6 Computed properties

Computed properties adalah properti virtual yang dihitung berdasarkan satu atau beberapa path. Fungsi komputasi untuk computed properties mengikuti aturan yang sama dengan observer yang kompleks, kecuali bahwa ia mengembalikan nilai, yang digunakan sebagai nilai computed properties.

Seperti observer yang kompleks, fungsi komputasi dijalankan sekali pada inisialisasi jika ada dependensi (ketergantungan) yang telah ditentukan. Setelah itu, fungsi berjalan setiap kali ada perubahan observer pada dependensinya.

5.6.1 Menentukan Computed Properties

Polymer mendukung properti virtual yang nilainya dihitung dari properti lain. Untuk mendefinisikan properti yang dikomputasi, tambahkan ke objek properti dengan key mapping yang dikomputasi ke fungsi komputasi:

```
fullName: {
  type: String,
  computed: 'computeFullName(first, last)'
}
```

Fungsi ini disediakan sebagai string dengan dependensi sebagai argumen yang berada dalam tanda kurung. Seperti halnya observer yang kompleks, fungsi komputasi tidak dipanggil sampai setidaknya satu dependensi didefinisikan (! == tidak terdefinisi).

Contoh :

```
import { PolymerElement, html } from '@polymer/polymer/polymer-element.js';

class XCustom extends PolymerElement {
  static get template() {
    return html`
      <p>My name is <span>{{fullName}}</span></p>
    `;
  }
  static get properties() {
    return {
      first: String,
      last: String,
      fullName: {
        type: String,
        // when `first` or `last` changes `computeFullName` is called once
        // and the value it returns is stored as `fullName`
        computed: 'computeFullName(first, last)'
      }
    }
  }
  computeFullName(first, last) {
    return first + ' ' + last;
  }
}
customElements.define('x-custom', XCustom);
```

Argumen untuk function computed dapat berupa properti sederhana pada elemen, serta salah satu dari tipe argumen yang didukung oleh observer, termasuk path, path dengan wildcard, dan path ke susunan array. Argumen yang diterima oleh fungsi komputasi cocok dengan yang dijelaskan dalam bagian contoh di atas.