

COMENIUS UNIVERSITY IN BRATISLAVA

FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

TWO-WAY DATABINDING OF MODELS AND
VIEWS IN DART

Diploma thesis

2014

Bc. Jakub Uhrík

COMENIUS UNIVERSITY IN BRATISLAVA

FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS

TWO-WAY DATABINDING OF MODELS AND VIEWS IN DART

Diploma thesis

Study programme: Computer Science

Field of Study: 9.2.1. Computer Science, Informatics

Department: FMFI.KI - Department of Computer Science

Thesis supervisor: RNDr. Tomáš Kulich, PhD.

Bratislava, 2014

Bc. Jakub Uhrík



Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Jakub Uhrík
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Obojsmerné previazanie dát s pohľadmi v jazyku Dart / *Two-way databinding of models and views in Dart*

Cieľ: Porovnajte rôzne spôsoby obojsmerného previazania modelov s pohľadmi vo webových aplikáciách. Vyberte si prístup navrhnutý Angular-om resp. Model Driven Views, alebo prístup, ktorý používa Facebook React, implementujte tento prístup v jazyku Dart. Zdôvodnite svoje rozhodnutie a odôvodnite, prečo je zvolený prístup pre Dart vhodnejší.

Vedúci: RNDr. Tomáš Kulich, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.
Dátum zadania: 28.10.2013

Dátum schválenia: 29.10.2013

prof. RNDr. Branislav Rován, PhD.
garant študijného programu

.....
študent

.....
vedúci práce



Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Jakub Uhrík
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Obojsmerné previazanie dát s pohľadmi v jazyku Dart / *Two-way databinding of models and views in Dart*

Cieľ: Porovnajte rôzne spôsoby obojsmerného previazania modelov s pohľadmi vo webových aplikáciách. Vyberte si prístup navrhnutý Angular-om resp. Model Driven Views, alebo prístup, ktorý používa Facebook React, implementujte tento prístup v jazyku Dart. Zdôvodnite svoje rozhodnutie a odôvodnite, prečo je zvolený prístup pre Dart vhodnejší.

Vedúci: RNDr. Tomáš Kulich, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.
Dátum zadania: 28.10.2013

Dátum schválenia: 29.10.2013

prof. RNDr. Branislav Rován, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

I would like to thank my supervisor RNDr. Tomáš Kulich, PhD. for his guidance, support, and encouragement throughout writing this thesis.
Special thanks belong to my family for all their support.

Bc. Jakub Uhrík

Abstract

Abstract in english.

Key words: Databinding, Dart, Facebook React, AngularJS, ...

Abstrakt

Abstrakt v slovenčine.

Kľúčové slová: Databinding, Dart, Facebook React, AngularJS, ...

Contents

Introduction	1
1 Motivation - why databinding	2
1.1 History	2
1.1.1 Plain documents	2
1.1.2 Simple PHP	2
1.1.3 Server side frameworks	2
1.1.4 Simple JavaScript/jQuery	2
1.1.5 JavaScript MVC frameworks	2
1.2 Objectives	2
1.2.1 Server-side rendering	3
1.2.2 Programmer friendly API	3
1.2.3 Easy concept	3
1.2.4 Two way databinding	3
2 Databinding	4
2.1 One way databinding	4
2.2 Two way databinding	4
3 Existing solutions	5
3.1 Template driven	5
3.2 Component driven	5
4 Our solution	6
4.1 Requirements	6
4.2 Architecture	6
4.2.1 High level idea	6
4.2.2 Core	6
4.2.3 Life-cycle	6
4.2.4 Events	11

4.2.5	Rendering	11
4.2.6	Infecting	11
4.3	API	11
4.3.1	Component	11
4.3.2	Browser specific API	11
4.3.3	Server specific API	11
5	Performance	12
6	Benchmarks	13
	Conclusion	14
	Bibliography	15

List of Figures

Introduction

As one of the results of this magister thesis is our new databinding library in dart, which is called **tiles**. In next text, we will use only **tiles** to mention *our new databinding library in dart*.

Chapter 1

Motivation - why databinding

The first question, as always should be, is the motivation of this work. What is the motivation to create another library, that will handle databinding in dart?

We will start with small introduction to history of how websites and later web-applications was created. Then we define a set of features required for **tiles**.

1.1 History

1.1.1 Plain documents

1.1.2 Simple PHP

1.1.3 Server side frameworks

1.1.4 Simple JavaScript/jQuery

1.1.5 JavaScript MVC frameworks

1.2 Objectives

From previous overview of "history" we can produce set of features, which should be contained in **tiles**.

1.2.1 Server-side rendering

1.2.2 Programmer friendly API

1.2.3 Easy concept

1.2.4 Two way databinding

Chapter 2

Databinding

In this chapter we will introduce problematics of databinding more deeply then in introduction.

2.1 One way databinding

Discuss one way databinding.

2.2 Two way databinding

Discuss two way databinding.

Chapter 3

Existing solutions

3.1 Template driven

Discuss databinding based on filling some type of template with model. This approach is used in standard MVC frameworks like AngularJS, Ember or UI libraries like Polymer.dart.

3.2 Component driven

Discuss databinding based on component approach used for example in React from facebook or our library.

Chapter 4

Our solution

In this chapter we will introduce and deeply describe our Dart library **Tiles**.

4.1 Requirements

In this section we write down a list of requirements on our library.

4.2 Architecture

We decided to stay with the idea of component, so the main class of our library is **Component**.

4.2.1 High level idea

4.2.2 Core

4.2.3 Life-cycle

Every instance of **Component** have own life-cycle. As every object, first it is created. Then, when component is mounting or rendering into text, it is rendered, and then it is mounted. Then it lives it's own life.

When something "higher" want to update it, it will first receive props, then it is asked, if it should be updated, and if yes, then it is rendered. After that, it was updated, of course.

At the end of component's life, component should be notified about that it will be unmounted(e.g. from DOM), to be able, to do some modifications to it's refs, destroy timers and so on.

Create

Create part of life-cycle is implemented by constructor of `Component`. It will receive props and optionally children as arguments and it should do whatever it needs to prepare whole state of object to live.

An trivial example of constructor of `Component` is

```
class MyComponent extends Component {
  MyComponent(props, [children]): super(props, children) {}
}
```

which only call's constructor of super class `Component`

Example of more complex constructor should be e.g. component which maintain example `Todo` instance.

```
class MyTodoComponent extends Component {
  Todo todo;
  MySearchComponent(props, [children]): super(props, children) {
    if (props != null && props.todo is Todo) {
      this.todo = props.todo
    } else {
      this.todo = new Todo();
    }
  }

  // ...
}
```

Did mount

`Component` life-cycle **Did mount** is implemented by method `didMount`. It is called after component is mounted to DOM.

This is the correct place to initialize for example timers, stream listeners and so on.

For example, in our `MyTodoComponent` we should listen for change of `todo` on server, and if it was changed, we can redraw component.

```
class MyTodoComponent extends Component {
  Todo todo;
  StreamSubscription subscription;
```

```

// ...

didMount() {
  this.subscription = this.todo.changedOnServer.listen((change) {
    this.redraw();
  });
}

// ...

}

```

Will receive props

Will receive props life-cycle method is `willReceiveProps`. It is called every time, when component will receive new `props`, except first time, when these `props` are passed to constructor.

This is place, where old props and new props can be compared, so this is right place to make changes based on difference in old and new props.

Example of `willReceiveProps` in our `MyTodoComponent` should compare `todo` of old and new props and there are not equal, it can update change listener.

```

class MyTodoComponent extends Component {
  Todo todo;
  StreamSubscription subscription;

  // ...

  willReceiveProps(dynamic newProps) {
    if (this.todo !== newProps.todo) {
      this.subscription.cancel();
      this.subscription = newProps.todo.changedOnServer.listen((change) {
        this.redraw();
      });
    }
  }

  // ...
}

```

Should update

Should update is partly lifecycle, partly not. It is a question, if component should update on this props-change.

This "life-cycle" is implemented by method `shouldUpdate`. This method is used mainly for speed up performance. By default it returns true, so if it is not implemented in custom component, it will update always.

In basic scenario this method recognize, if it will be rendered differently with new props. If not, it return false, else it return true.

Example in `MyTodoComponent` should look like this:

```
class MyTodoComponent extends Component {
  Todo todo;
  StreamSubscription subscription;

  // ...

  shouldUpdate (newProps, oldProps) {
    if (newProps.todo == oldProps.todo) {
      return false;
    }
    return true;
  }

  // ...
}
```

Render

Render is the main part of the `Component`.

It is implemented by method `render`, which have no attributes. It should return array of component descriptions which should be considered as *"this is how this component should look like"*.

For example, in our `MyTodoComponent` render will return `<div>` which contains title and description of `todo`.

```
class MyTodoComponent extends Component {
  Todo todo;
  StreamSubscription subscription;
```

```

// ...

render () {
  return div ({class: "todo"}, [
    h2 ({}, todo.title),
    p ({}, todo.description)
  ]);
}

// ...

}

```

Did update

When life-cycle method `didUpdate`, by which is implemented this life-cycle event, is triggered, component, and programmer, can be sure, that component is mounted and there exist elements in DOM for each `DomComponent` descendant.

Will unmount

This event is implemented by method `willUnmount`, which contain no arguments.

It is called right before it is unmounted from dom.

This is the correct place to stop all timers and listeners.

```

class MyTodoComponent extends Component {
  Todo todo;
  StreamSubscription subscription;

  // ...

  willUnmount () {
    subscription.cancel();
  }

  // ...

}

```

4.2.4 Events

4.2.5 Rendering

Server side

In browser

4.2.6 Infecting

4.3 API

Documentation of offered API of our library.

4.3.1 Component

4.3.2 Browser specific API

4.3.3 Server specific API

Chapter 5

Performance

Chapter 6

Benchmarks

Conclusion

Here will be conclusion of wholw thesis

Bibliography

- [Aja10] AjaxPatterns.org Wiki. *RESTful Service*, 2010.
http://ajaxpatterns.org/RESTful_Service.
- [jav12] *Java web frameworks discussed*, 2012.
<http://entjavastuff.blogspot.com/2012/01/java-web-frameworks-discussed.html>.
- [JQU12] JQUERY FOUNDATION AND THE JQUERY UI TEAM. *jQueryUI Demos & Documentation*, 2012.
<http://jqueryui.com/demos/>.
- [Mic] Microsoft Developer Network. *Model-View-Controller*.
<http://msdn.microsoft.com/en-us/library/ff649643.aspx>.
- [Nie03] Jakob Nielsen. *Usability 101: Introduction to Usability*, 2003.
<http://www.useit.com/alertbox/20030825.html>.
- [Ste07] Stefan Tilkov. *A Brief Introduction to REST*, 2007.
<http://www.infoq.com/articles/rest-introduction>.
- [Sun02] Sun Microsystems, Inc. All Rights Reserved. *Java BluePrints: Model-View-Controller*, 2002.
<http://java.sun.com/blueprints/patterns/MVC-detailed.html>.
- [TS] Jupiter Consulting JavaScriptMVC Training and Support. *JavaScriptMVC Documentation*.
<http://javascriptmvc.com/docs.html>.
- [zen] zenexity & Typesafe. *Play 2.0 documentation*.
<http://www.playframework.org/documentation/2.0.1/Home>.