31.8.–3.9.2015
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

JSF meets JS (2. ed.)

JSF-Komponenten mit JavaScript

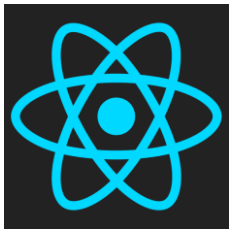
Sven Kölpin

open knowledge GmbH

JSF in Webanwendungen?

JavaScript & SPA

- Single page application (SPA)
  - Nur eine HTML-Seite
  - DOM-Manipulation hintergründig
  - Business-Logik im Client
  - Status im Browser
- Rolle des Servers
  - Stateless



JavaScript & SPA

- Hypegefahr
  - „Is angularJs dead?“
  - „ReactJs is the next big thing!“
- Architekturelle Probleme
  - Geschwindigkeit
  - Browserkompatibilität
- Pitfalls
  - Pionierarbeit (v.a. Enterpriseumfeld)

Serverseitige Webanwendungen

- Berüchtigt im Enterprise Umfeld
  - Bekanntes Programiermodell
  - Weniger Medienbrüche
- Rich Internet Applications
  - Ajax
  - Presentations-Logik
- Rolle von JavaScript
  - DOM-Manipulation

Serverseitige Webanwendungen

- Berüchtigt im Enterprise Umfeld
  - Bekanntes Programiermodell
  - Weniger Medienbrüche
- Rich Internet Applications
  - Ajax
  - Presentations-Logik
- Rolle von JavaScript
  - DOM-Manipulation

→ BEST OF BREED

JSF meets JavaScript

JSF meets JS: Why not Primefaces?

JSF meets JS: Why not Primefaces?

- Kundenanforderungen an die GUI
  - Individuelle Probleme
- Frameworks sind Blackboxes
- Third-Party-Libraries
  - Boilerplate
  - Bugs
  - Abhängigkeiten

Ziele des Vortrags

- Best-Practices vermitteln
  - JavaScript-Pattern
  - JSF-JS-Komponenten
  - JSF-JavaScript-Kommunikation

→ Den Einsatz von 3rd Party Libraries überdenken

Was im Standard geschah...

JSF Composite Components

- JSF-Code in Komponenten auslagern
  - Parametrisierbar
  - Wiederverwendbar
- Ziel: deklarative Kapselung von GUI-Logik

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ok="http://xmlns.jscp.org/jsf/composite/ok">

<ok:labelTextBox
        value="#{someBean.firstName}"
        name="Firstname">
</ok:labelTextBox>
```

JSF Composite Components

- „Keep simple things simple“
  - XHTML Interface
  - XHTML Implementierung

- „Convention over code & configuration“
  - Namenskonventionen
  - Ressourcenzugriff

Keep simple things simple

- Interface & Implementierung

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:composite="http://java.sun.com/jsf/composite"
      xmlns:jsf="http://xmlns.jcp.org/jsf">

<composite:interface>
    <composite:attribute name="name" />
    <composite:attribute name="value"/>
</composite:interface>

<composite:implementation>
    <label jsf:id="lbl" jsf:for="inputComponent"
                        jsf:value="#{cc.attrs.name}"/>
    <input  type="text" jsf:id="inputComponent"
                        jsf:value="#{cc.attrs.value}"/>
</composite:implementation>
```

Convention over configuration

- webapp/resources/ok/labelTextBox.xhtml
  → xmlns:ok="http://xmlns.jcp.org/jsf/composite/ok"

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:composite="http://java.sun.com/jsf/composite"
      xmlns:jsf="http://xmlns.jcp.org/jsf">

<composite:interface>
    <composite:attribute name="name" />
    <composite:attribute name="value"/>
</composite:interface>

<composite:implementation>
    <label jsf:id="lbl" jsf:for="inputComponent"
                        jsf:value="#{cc.attrs.name}"/>
    <input  type="text" jsf:id="inputComponent"
                        jsf:value="#{cc.attrs.value}"/>
</composite:implementation>
```

Und JavaScript?

JSF und JS: Der Standard

- Alles ist deklarativ in JSF (XML-Kapselung)
  - Komponenten

```
<h:dataTable value="#{someBean.list}" var="row">
    <h:column>#{row.valA}</h:column>
    <h:column>#{row.valB}</h:column>
</h:dataTable>
```

  - JavaScript

```
<h:commandButton value="Submit">
    <f:ajax execute="@this" render="panelB"/>
</h:commandButton>
```

  - Standardisierte JavaScript-Integration: Clientbehaviors

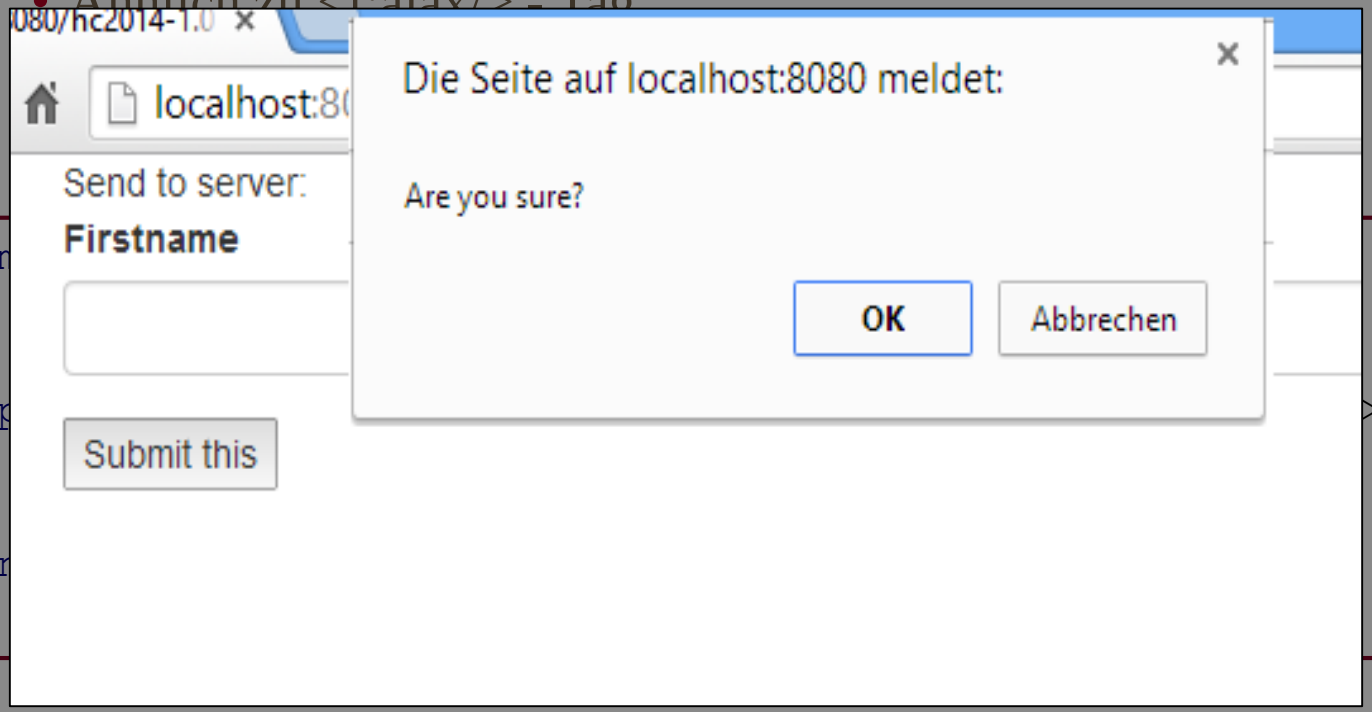
Client Behaviors

- JavaScript deklarativ kapseln
  - Ähnlich zu <f:ajax/> - Tag
  - Komponenten um clientseitige Funktionalität erweitern

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:okb="http://openknowledge.de/behavior" ...>

<input type="submit" jsf:id="submit" value="Submit this">
    <okb:confirm message="Are you sure?"/>
    <f:ajax render="parentForm"/>
</input>
```

Client Behaviors

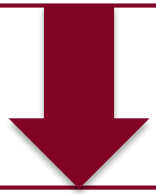
- JavaScript deklarativ kapseln
  - Ähnlich zu <f:ajax/> - Tag



Client Behaviors

- Was passiert da?

```
<input type="submit" jsf:id="submit" value="Submit this">
    <okb:confirm message="Are you sure?"/>
    <f:ajax render="parentForm"/>
</input>
```



```
<input id="parentForm:submit"
       onclick="jsf.util.chain(this,event,
           'return confirm(\'Are you sure?\')',
           'mojarra.ab(this,event,\'action\',0,\'parentForm\')');"/>
```

Client Behaviors

- Was braucht man?
  - Behavior-Klasse
    - extends ClientBehaviorBase
    - @FacesBehavior(“de.my.behavior“)
  - taglib.xml
    - Konfiguration ☹
  - JavaScript-Code
    - Das eigentliche Herzstück

Client Behaviors

```
@FacesBehavior("de.openknowledge.SimpleBehavior")
public class SimpleBehavior extends ClientBehaviorBase {
  private String message;

  @Override
  public String getScript(ClientBehaviorContext behaviorContext) {
    return "return confirm('"+getMessage()+"')";
  }

  public String getMessage() {
    return message;
  }

  public void setMessage(String msg) {
    this.message = msg;
  }
...
```

Client Behaviors

- JavaScript in Java?

```
@FacesBehavior("de.openknowledge.SimpleBehavior")
@ResourceDependencies(value = {
    @ResourceDependency(name = "ok/confirm.js")
})
public class SimpleBehavior extends ClientBehaviorBase {
  private String message;

  @Override
  public String getScript(ClientBehaviorContext behaviorContext) {
    return "ok.confirmDialog.show();";
  }
 ...
}
```

Client Behaviors

- META-INF/ok.taglib.xml

```
<facelet-taglib version="2.2"  … >
    <namespace>http://openknowledge.de/behavior</namespace>
     <tag>
         <tag-name>confirm</tag-name>
      <behavior>
         <behavior-id>
            de.openknowledge.SimpleBehavior
         </behavior-id>
      </behavior>
      <attribute>
         <name>message</name>
      </attribute>
     </tag> ...
```

Client Behaviors

- Zeitgemäß?
  - Konfiguration ☹
  - Boilerplate ☹
  - Eventbasiert ☹
  - Keine zwingenden JS-Pattern

JSF & JavaScript – Ein Chaos vermeiden

JSF & JavaScript

- Umittelbare Integration von JavaScript in JSF
  - Nicht standardisiert

- Inline Scripting

```
<input type="text" jsf:id="dp"/>

<script type="text/javascript">
        var dpElm = document.getElementById("form:dp");
        dpElm.dataset.datepicker = true;
        dpElm.dataset.dateformat = #{someBean.dateformat};
        //...
</script>

<div jsf:id="someOtherPanel">
    <h1>Bad bad code...</h1>
</div>
```

JSF & JavaScript

- Umittelbare Integration von JavaScript in JSF
  - Nicht standardisiert

- Inline Scripting

```
<input type="text" jsf:id="dp"/>

<script type="text/javascript">
        var dpElm = document.getElementById("form:dp");
        dpElm.dataset.datepicker = true;
        dpElm.dataset.dateformat = #{someBean.dateformat};
        //...
</script>

<div jsf:id="someOtherPanel">
    <h1>Bad bad code...</h1>
</div>
```

JSF & JavaScript

- Why Inline-Scripting sucks
  - Wartbarkeit?
  - DRY & Separation of concerns?
  - Fehlersuche?
  - Weiche Faktoren
    - Auslagerung = Modularisierung
    - Performanz + Traffic

Enterprise JavaScript

be unobtrusive

use consistent pattern

use build systems

minimize globals

write tests

Enterprise JavaScript

be unobtrusive

use consistent pattern

use build systems

minimize globals

write tests

Enterprise JS: Globals

- Globals are evil!

```
var myglobal = "hello"; // antipattern
console.log(myglobal); // "hello"
console.log(window.myglobal); // "hello"
console.log(window["myglobal"]); // "hello"
console.log(this.myglobal); // "hello"
```

- Kollisionen
  - Libraries
  - Andere JS-Dateien

Enterprise JS: Globals

- Namespace pattern

```
var Application = {};
Application.SubModul = {};
Application.SubModul = {
    SomeClass : function(){...}
};
```

- Module
  - Mit Tools (z.B. Browserify)
  - ES2015-Modules

```
//------ myFunc.js ------
export default function () { ... };
//------ main1.js ------
import myFunc from 'myFunc';
myFunc();
```

Enterprise JavaScript

be unobtrusive

use consistent pattern

use build systems

minimize globals

write tests

Enterprise JS: Pattern

- JavaScript === dynamisch
  - 1000 Wege führen nach Rom
- Das richtige Pattern finden
  - !== Inline-Scripting, !== Globals
  - Wohlbefinden des Teams
  - „Framework Faktoren“
- Pattern konsistent einsetzen
  - Explizit
  - Implizit

Enterprise JS: Module-Pattern

- Von vielen JS-Bibliotheken verwendet
  - Minimiert globale Variablen
  - Strukuriert Code
- Information Hiding & Vererbung
- Herzstück
  - Self-Invoking Functions
  - Function-Scope von JavaScript

```
SomeNameSpace.Module = (function () {
    //...
    return PublicApi;
})();
```

Module-Pattern: Klassensimulation

```
SomeNameSpace.DatePicker = (function () {
    //constructor
    var DatePicker = function(param){
        this.param = param;
    };
    //public method
    DatePicker.prototype.init = function(){
        //use call to keep this reference
        privateMethod.call(this);
    };
    //private method
    var privateMethod = function(){
        console.log(this.param);
    };
    return DatePicker;
})();
//usage
new SomeNameSpace.DatePicker("hello").init(); //prints hello
```

ES 2015: Class-Syntax

```
class DatePicker extends BaseConent{
    constructor(param){
        super();
        this.param = param;
    }
    get otherParam(){
        return this.otherParam;
    }
    set otherParam(param){
        this.otherParam = param;
    }
    init(){
        this._init();
    }
    _init(){
        console.log(this.param);
    }
}
new DatePicker("hello").init();
```

Enterprise JavaScript

be unobtrusive

use consistent pattern

use build systems

minimize globals

write tests

Separation of structure and behavior

Inline-Scripting

Separation of structure and behavior

- Problem: JavaScript wo und wie initialisieren?
  - Jede Komponente hat eigenes Script-Modul
  - Parameter aus JSF (Java) an JS übergeben

Separation of structure and behavior

- Problem: JavaScript wo und wie initialisieren?
  - Jede Komponente hat eigenes Script-Modul
  - Parameter aus JSF (Java) an JS übergeben

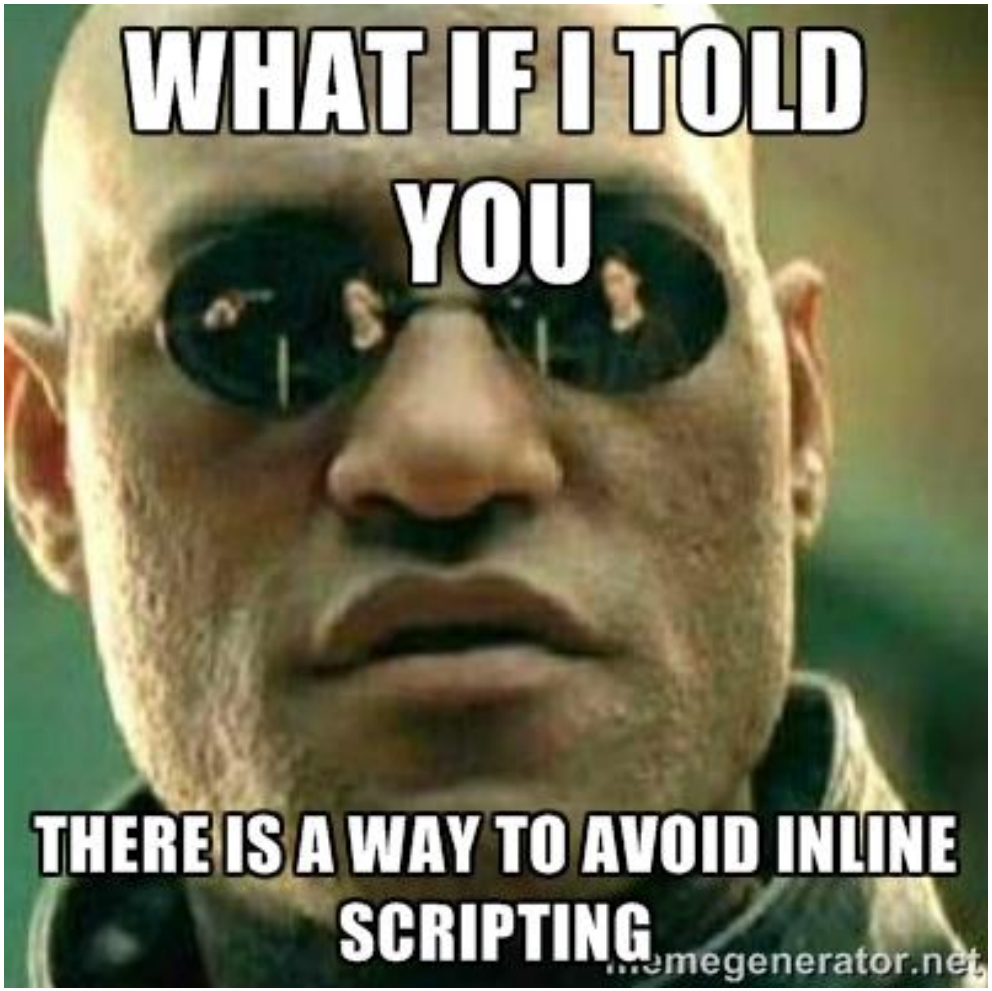
```
<composite:interface>
    <composite:attribute name="value" type="java.lang.Date"/>
    <composite:attribute name="format" type="java.lang.String"/>
</composite:interface>

<composite:implementation>
    <input type="text" jsf:id="dp" jsf:value="#{cc.attrs.value}"/>
    <script>
        new ok.DatePicker({
            clientId: "#{cc.clientId}",
            format: "#{cc.attrs.format}"
        });
    </script>
</composite:implementation>
```

Separation of structure and behavior

- Im Browser...

```
<input type="text" id="form:component:dp" value="1.1.2015"/>
<script>
    new ok.DatePicker({
        clientId: "form:component",
        format: "dd.mm.yyyy"
    });
</script>
```

JS-Initialisierung mit Nicole



JS-Initialisierung mit Nicole

- True separation of structure and behavior

```
<composite:interface>
    <composite:attribute name="value" type="java.lang.Date"/>
    <composite:attribute name="format" type="java.lang.String"/>
</composite:interface>

<composite:implementation>
    <input type="text" jsf:id="dp" jsf:value="#{cc.attrs.value}"/>

    <nicole:module modulename="DatePicker">
        <nicole:jsparameter name="format" value="#{cc.attrs.format}"/>
     </nicole:module>

</composite:implementation>
```

JS-Initialisierung mit Nicole

- Im Browser...

```
<input id="form:component:dp" value="1.1.2015" type="text"/>

<input id="form:component:nicoleCC:nicole"
       type="hidden"
       data-modulename="DatePicker"
       data-format="dd.mm.yyyy"
       data-clientid="form:component"/>
```

JS-Initialisierung mit Nicole

- Und JavaScript:

```
Nicole.module("DatePicker", function () {
    this.$elm("datePicker").datepicker({
        dateFormat: this.parameter("format")
    });
});
```

Nicole Zusammenfassung

- JSF bleibt JSF
- HTML bleibt HTML
- JavaScript: erzwungenes Pattern

→ github.com/dskgry/nicole

Enterprise JavaScript

be unobtrusive

use consistent pattern

use build systems

minimize globals

write tests

Enterprise JS: Build & Test

- Build-Systeme
  - Concatenation & Minification
  - Transpiling
  - Linting
  - Testing

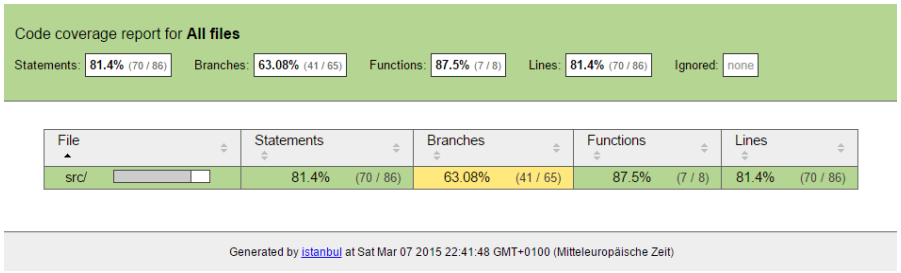
Enterprise JS: Build & Test

- Test: What do we test?

```
describe("the chartComponent", function () {
    it("does not support the type bar", function () {
        expect(function(){
            new ok.ChartComponent({
                clientId: "client:id",
                chartType: "bar"
            });
        }).toThrow();
    });
});
```

- Toolsets
  - Runner
  - Assertions%
  - Spy/Mock

- CI
  - Jenkins Integration
  - Statische Codeanalysen
    - Coverage



JSF & JavaScript: JSON-Kommunikation

JSF & JavaScript: JSON-Kommunikation

- JSF (Ajax) Kommunikation basiert auf XML
  - semantische Daten

```
<partial-response id="j_id1"><changes><update
id="parentForm"><![CDATA[
<form id="parentForm" method="post" class="container">
<input type="hidden" name="parentForm" value="parentForm" />
```

- JS-Komponenten nutzen JSON
  - Übertragung „reiner“ Daten nötig

JSF & JavaScript: JSON-Kommunikation

- Warum nicht mit JAX-RS
  - Stateless vs. Stateful
  - JSF-Lifecycle
  - CDI-Scopes
    - Viewscope?

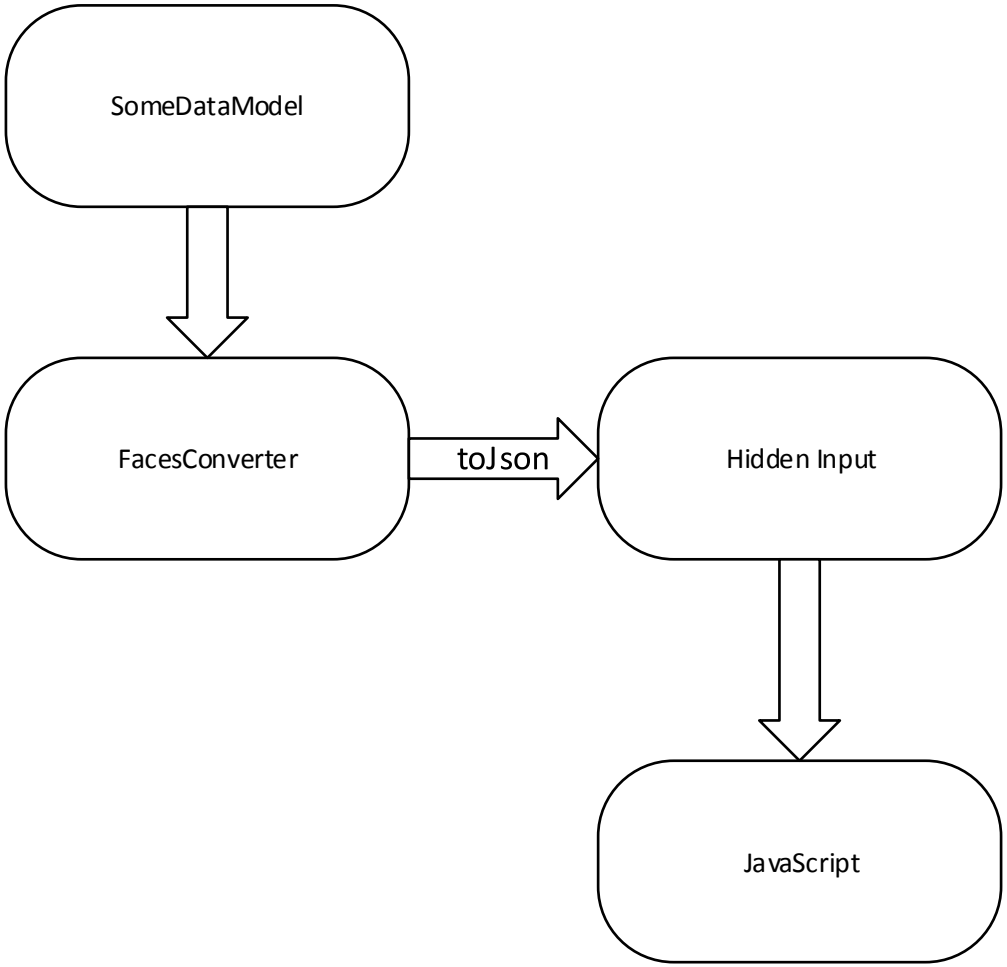
JSF & JavaScript: JSON-Kommunikation

- Übertragung des JSON in Hidden-Input-Fields
  - Nutzung von FacesConvertern (Java zu JSON konvertieren)

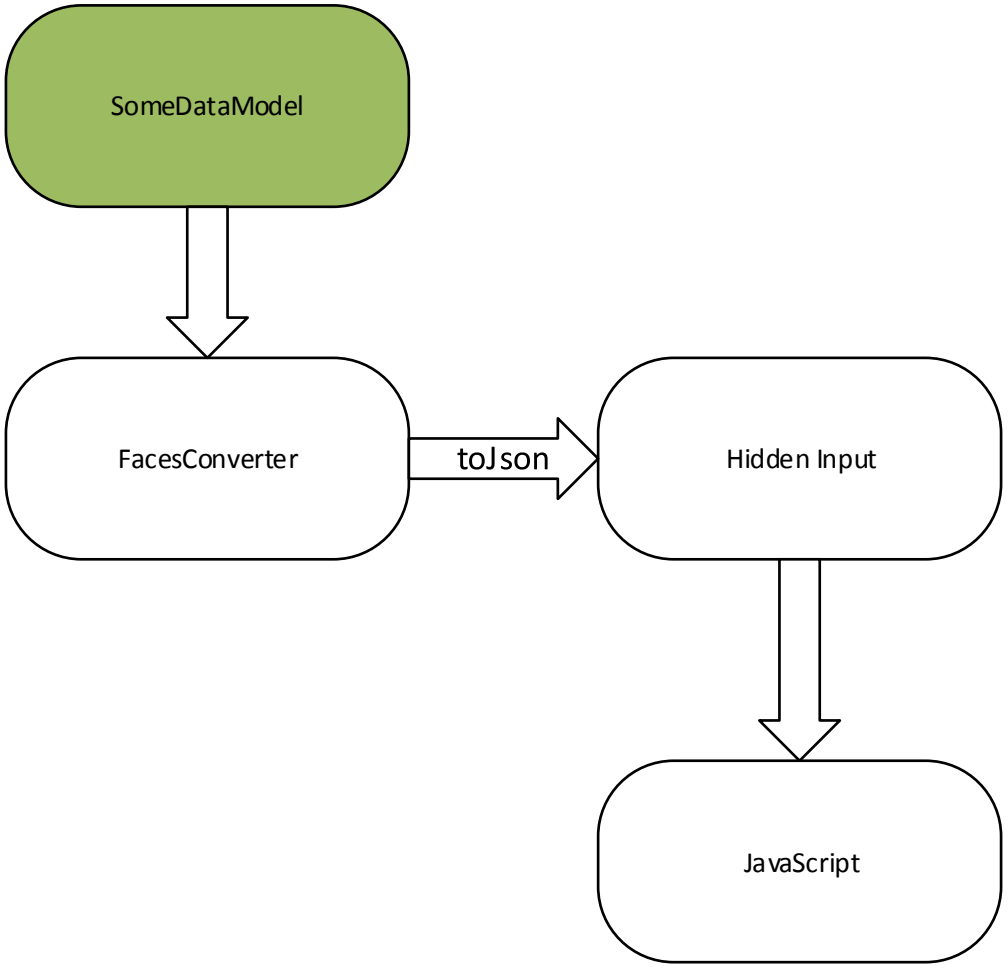
```
{"data":[{"a":"b"},{"a":"b"}]}
```

- Nutzung von JSF im Standard
- Minimaler Overhead

JSF & JavaScript: JSON-Kommunikation



JSF & JavaScript: JSON-Kommunikation

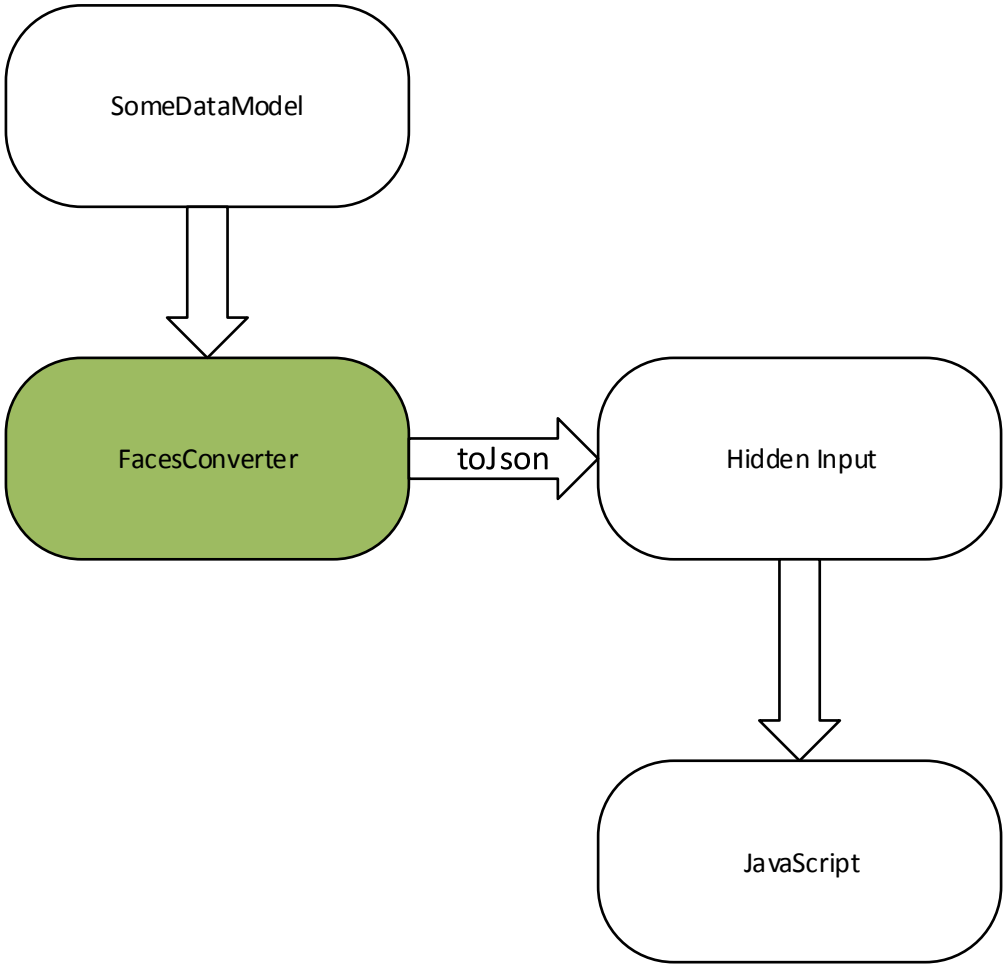


JSF & JavaScript: JSON-Kommunikation

- Komponente hat ein geeignetes Datenmodell
  - z.B. Chart - Menge aus x und y Werten
  - z.B. Map - Menge aus Lat. und. Lon. Werten

```
public abstract class ChartDataModel<T, X, Y> {
  private Map<X, Y> dataSeries;

  public Map<X, Y> getDataSeries() {
    return dataSeries;
  }
  protected abstract X getXValue(T t);

  protected abstract Y getYValue(T t);
}
```

JSF & JavaScript: JSON-Kommunikation

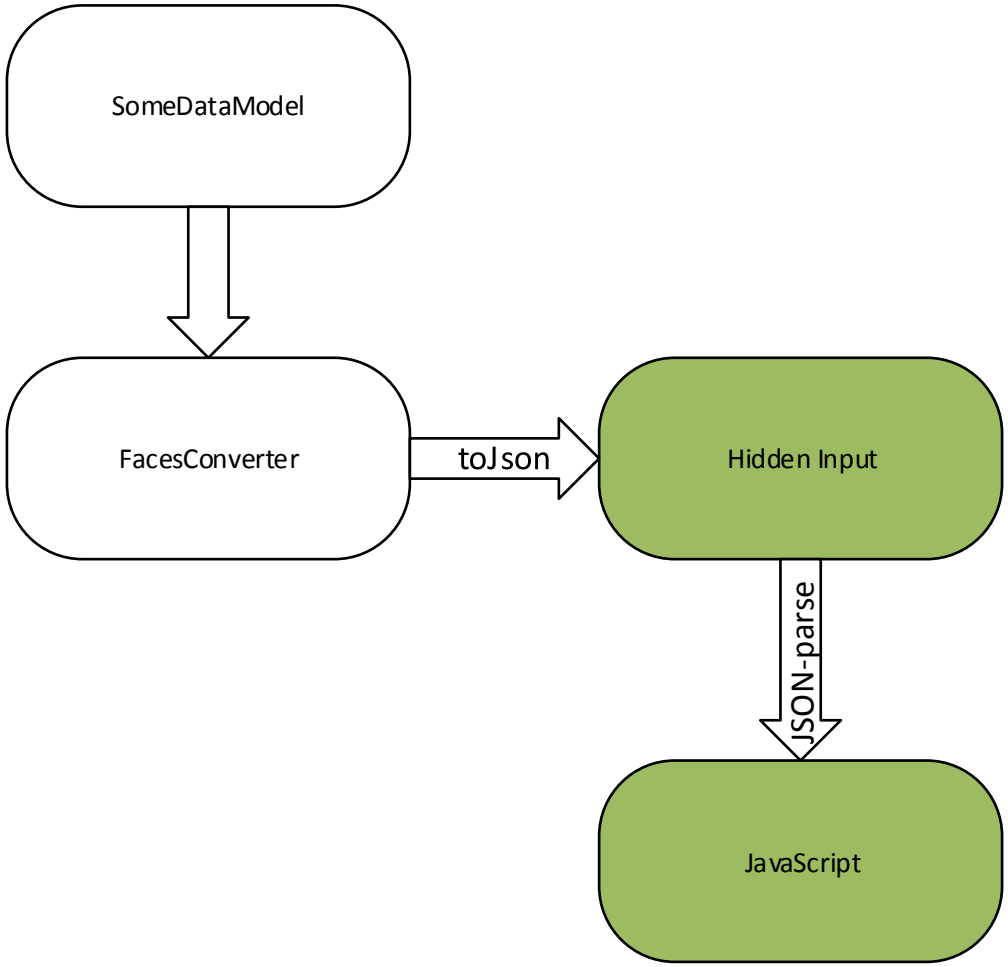


JSF & JavaScript: JSON-Kommunikation

- FacesConverter: Datenmodell → JSON
  - JavaEE 7 JSON-API

```
@FacesConverter(value = "de.openknowledge.ChartConverter")
public class ChartConverter implements Converter {
 ...
public String getAsString(FacesContext ctx, UIComponent comp,
                                                    Object value) {
    ChartDataModel dataModel = (ChartDataModel) value;

    JsonArrayBuilder dataArray = Json.createArrayBuilder();
     ...
    return dataArray.build().toString();
  }
  ...
```

JSF & JavaScript: JSON-Kommunikation



JSF & JavaScript: JSON-Kommunikation

- Converter am Hidden-Input-Feld registrieren
- Hidden-Input-Field mit JSON „befüllen“

```
<input
      type="hidden"
      jsf:id="jsonHelper"
      jsf:converter="de.openknowledge.ChartConverter"
      jsf:value="#{somebean.chartDataModel}"/>
```

JSF & JavaScript: JSON-Kommunikation

- HTML

```
<input value="{"xAxes":["04.09.14","06.09.14"],
        "yAxes":["8.74","85.77"]}"/>
```

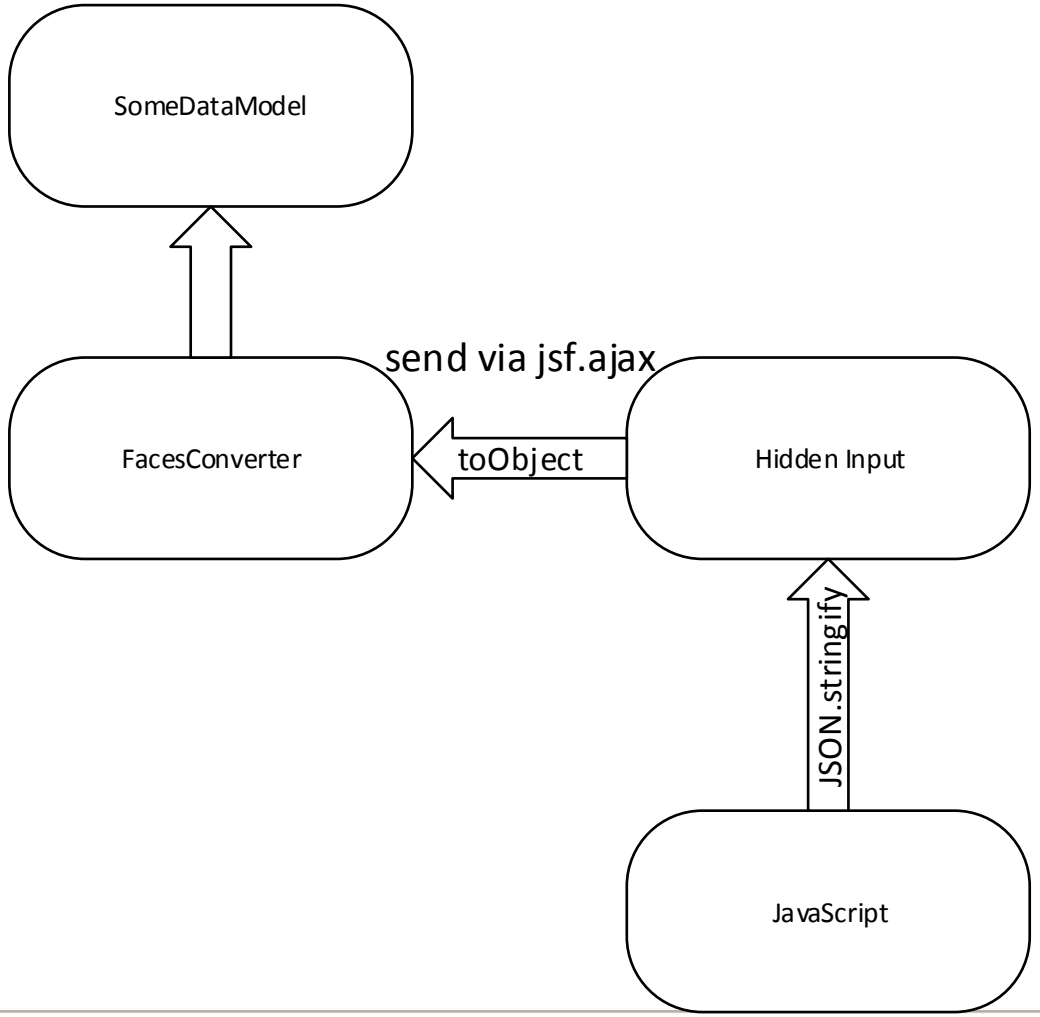
- JavaScript, jQuery

```
var chartDataJson =
   JSON.parse(this.$elm("jsonHelper").val());
```

JSF & JavaScript: JSON-Kommunikation

- Und andersrum?

JSF & JavaScript: JSON-Kommunikation




"That's all Folks!"

Ausblick

- JSF 2.3: Evolution statt Revolution
  - Ajax method invokation
- MVC 1.0
  - Auch hier: Chaos vermeiden
  - Weg von DOM-Manipulation?
- EcmaScript 2015 / 2016
  - class syntax / promises
  - privates

Fazit

- JSF & JavaScript
  → Pattern essentiell
- JSON-Kommunikation
  → JavaEE 7 Boardmittel
- Eigene Komponenten bringen enorme Flexibilität
  → initialer Aufwand höher
- Einsatz von 3rd-Party Libraries abwägen
  → 0 8 15 vs. fancy web

Vielen Dank!

Sven Kölpin
open knowledge GmbH