OPITZ CONSULTING

Menschen. Innovationen. Lösungen.

APEX und JavaScript

Beispiele, Pattern und Best Practices

Hendrik Gossens

Consultant

OPITZ CONSULTING GmbH

Nürnberg, 20.11.2014

Agenda

1. „Pimp my APEX with JavaScript“ - Beispiele
2. JavaScript in APEX
3. Gefahren bei der Verwendung von JavaScript
4. Good Practice – Bad Practice
5. Fazit

1 Pimp my APEX with JavaScript

Excelize my Tabular Form

- **Verhalten der Tabular Form wird um aus Excel bekannte Funktionalitäten angereichert**
  - Navigation in den Zellen über Tastaturpfeile und Enter-Taste
  - Automatisches Ausfüllen von Zellen
  - Vertikale Spaltenüberschriften
  - Fixierte Spaltenüberschriften

Formsize my APEX

- **Verhalten einer APEX-Maske wird um aus Oracle Forms bekannte Funktionalitäten angereichert**
  - Hotkey-Support (z.B. Suche ausführen, Zeilen einfügen, ...)

User Experience

- **Features, die das Nutzererlebnis steigern**
  - Inline-Validierungsmeldungen verschwinden beim Klick in das betreffende Feld

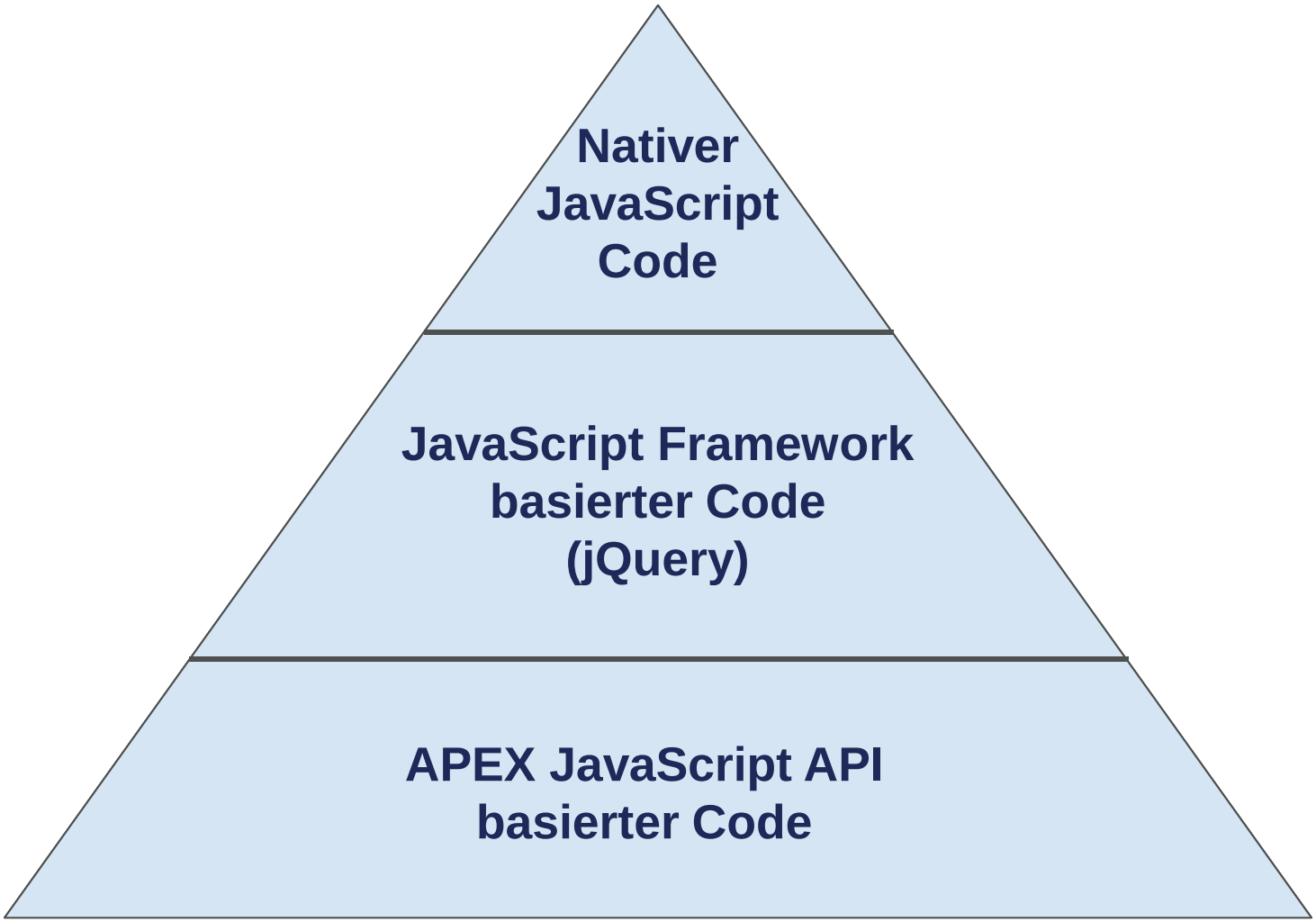
Menschen. Innovationen. Lösungen.

Showcase

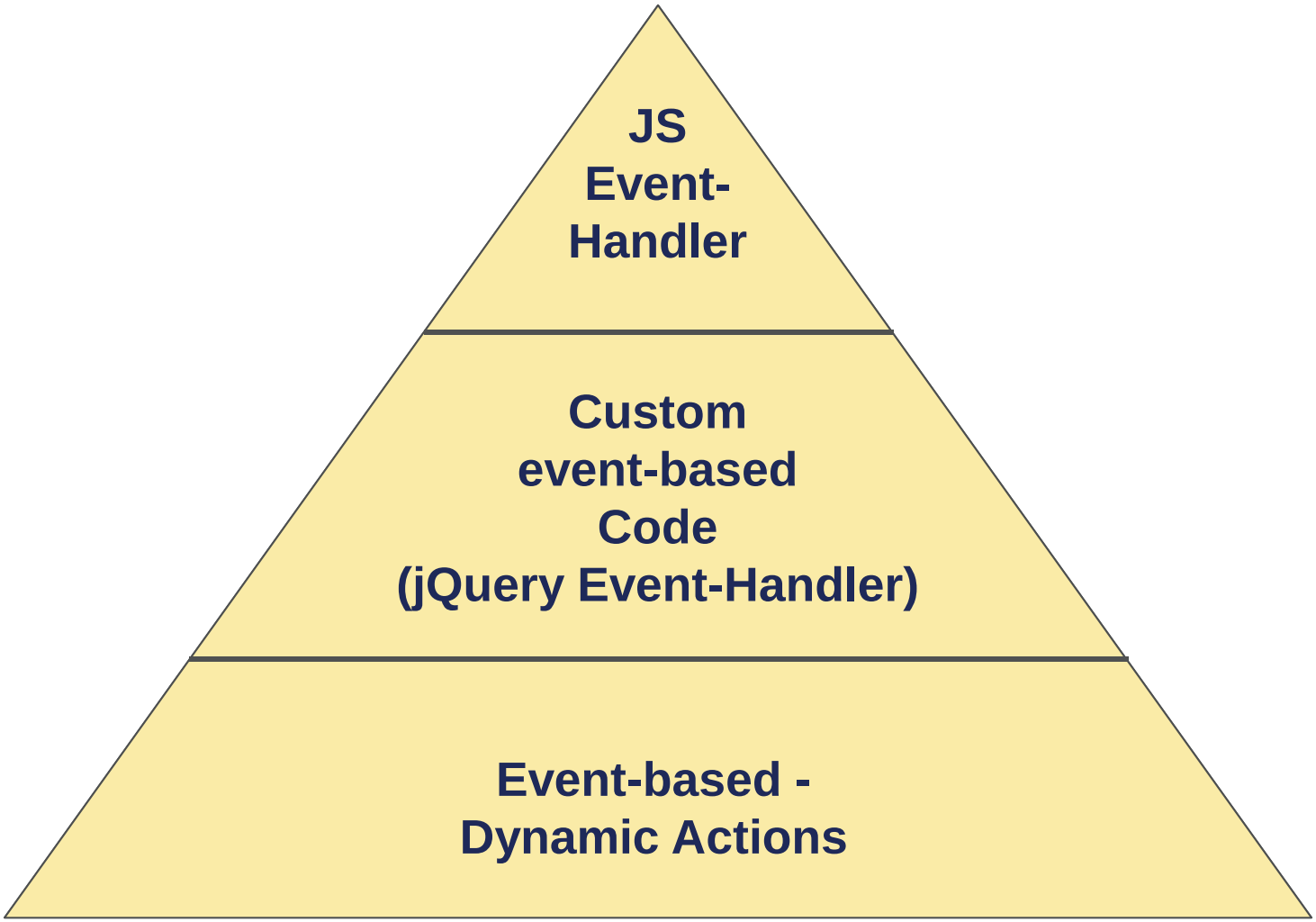
OPITZ CONSULTING

2 JavaScript in APEX

JavaScript in APEX – Wie?



JavaScript in APEX – Aufruf



JavaScript in APEX - Mögliche Schwierigkeiten

- **Datentypen**
- **Debugging**
- **Wartbarkeit**
- **Browserabhängig unterschiedliches Verhalten desselben Codes**
- **Zusammenspiel einzelner Codeteile unklar (Event-Handler)**
- **Ein Syntaxfehler verhindert die komplette Ausführung (ohne sichtbare Fehlermeldung)**
- **Code spezifisch für spezielle APEX oder Browser-Version implementiert (Upgrade-Sicherheit?)**

Datentypen

- **JavaScript ist nicht typenstreng!**
- **Vergleiche von Variablen syntaktisch mittels == oder === möglich**
  - `1 == “1“; // true`
  - `1 === “1“; // false`

  => Für Vergleiche immer === verwenden!

> *"If two operands are of the same type and value, then === produces true and !== produces false."*
>
> - JavaScript: The Good Parts

Debugging

- **Logging**
  - Client-seitiges Logging
  - Logging in Datenbank-Tabelle
- **Tool-Unterstützung**
  - Firefox: Firebug
  - Chrome: Developer Tools
  - IE: eher rudimentär
  - Alle Browser: Konsole für Log-Ausgaben

Namespaces verwenden

- **Eigenen Namespaces verwenden**
  - Fachliche Strukturierung des Quellcodes möglich
  - Aufbau von Namespace-Hierarchien möglich
  - Vermeidung von Konflikten mit anderen JavaScript Funktionen
- **jQuery extend Funktion nutzen um nested namespaces / Namespace Hierarchien aufzubauen**

Namespaces verwenden

```
// top-level namespace
var myApp = myApp || {};
myApp.library = {
    foo:function(){ /*..*/}
};

// deep extend/merge namespace
$.extend(true, myApp, {
    library: {
        bar: function() {
            /*..*/
        }
    }
});
```

Wartbarkeit

- **Sprechende Namespaces verwenden**
- **Sprechende Methodennamen verwenden**
- **Sprechende Variablennamen verwenden**
  - ggf. mit Kennzeichnung des Datentypen da JavaScript nicht typenstreng ist
  - Camel-Case Notation => Neues Wort mit Großbuchstabe beginnen lassen
- **Sinnvolle Kommentare verwenden**
- **Annotationen (z. B. JSDoc)**
  - Was macht die Funktion?
  - Welche Parameter werden erwartet
  - Was wird zurückgegeben?
  - Wer ist Ansprechpartner?
  - ...

Bad Practice

```
function doSomething(msg) {
  if (console) {
    var cname = arguments.callee.caller.toString();
    var msgout = msg;
    if (typeof cname != "undefined")       {
      cname = cname.match(/function ([^\(]+)/)[1];
      msgout = "[" + cname + "]: " + msgout;
    }
    console.log(msgout);
  }
}
```

Good Practice

```
/**
* Loggt eine Message auf der JavaScript Konsole des Browsers und gibt
* dabei an, aus welcher Methode heraus das Logging aufgerufen wurde
*
* @param {string} message - Die zu loggende Message
* @author Hendrik Gossens
* @version 1.0
*/
Logger.log = function(message) {
  if (console) {
    /* IE kennt nicht in allen Versionen arguments.callee.name,
       daher den Funktionsnamen aus dem Caller-Objekt extrahieren
       (Läuft browserübergreifend!)    */
    var szCallerName = arguments.callee.caller.toString();
    var szMessage = message;

    // Aufrufende Methode als Präfix, falls diese ermittelt werden kann
    if (typeof szCallerName != "undefined")        {
      szCallerName = szCallerName.match(/function ([^\(]+)/)[1];
      szMessage = "[" + szCallerName + "]: " + szMessage;
    }
    console.log(szMessage);
  }
}
```

Exception Handling

- **Nicht abgefangene Fehler führen dazu, dass der gesamte Code nicht mehr ausgeführt wird!**
- **try-catch-Blöcke im Code verwenden**
- **Fehler/Warnungen loggen**
- **Fehler anzeigen, falls sie die weitere Verarbeitung unmöglich machen**

Bad Practice

```
var badFunction = function(x) {
  doSomething(x);
}
```

Not That Bad Practice

```
var notThatBadFunction = function(x) {
  try {
    if(isNaN(x)) {
      throw "not a number";
    }
    doSomething(x);
  }
  catch(error) {
       alert (error);
  }
}
```

Good Practice

```
var goodFunction = function(x) {
  try {
    if(isNaN(x))  {
          throw {
                  name: "Ungültiger Wert",
                  message: x + " ist keine Zahl!"
                };
    }
    doSomething(x);
  }
  catch(error) {
       alert (error.name + ':' + error.message);
  }
}
```

Error Objekt

Upgrade-Sicherheit

- **Wenn möglich Seitenelemente immer über die ID, nicht die Struktur, adressieren**
  - wo möglich statische ID in APEX definieren: z. B. für Regions
- **Abstrahierende Technologien nativem JavaScript vorziehen**
  - APEX JavaScript API
  - jQuery
  - Dynamic Actions

Bad Practice

- **Pure JavaScript**
  - Item Wert setzen

```
var textElement = document.getElementById('P1_TEXT'));
var textElementValue = textElement.value;
```

  - Item Wert auslesen

```
var textElement = document.getElementById('P1_TEXT'));
textElement.value = 'Neuer Wert';
```

Not That Bad Practice

- **jQuery**
  - Item Wer setzen: `$('#P1_TEXT').val('Neuer Wert');`
  - Item Wert auslesen: `$('#P1_TEXT').val();`

- **Diese Vorgehensweise adressiert das Item über seine ID. Ändert sich intern der Aufbau des Items (v.a. bei komplexeren Items als Textfeldern), so läuft der Code u.U. nicht mehr!**

Good Practice

- **APEX JavaScript API**
  - Item Wert setzen: `$s('P1_TEXT', 'Neuer Wert');`
  - Item Wert auslesen: `$v('P1_TEXT');`
- **Die APEX JavaScript API kann mit den verschiedenen Item Typen umgehen und ermittelt (upgrade-sicher) den aktuellen Wert oder setzt diesen.**

Fazit

- **JavaScript kann die Funktionalität von APEX erheblich erweitern!**
- **Abstraktion von nativem JavaScript Code durch Dynamic Actions und Frameworks wie jQuery => Code bleibt auch bei Updates / Browserwechsel lauffähig!**
- **JavaScript nur ergänzend zur APEX Standardfunktionalität verwendet werden, diese aber nicht ersetzen oder nachbauen!**
- **Pattern und Best Practices machen den Code lesbarer und wartbarer!**

Fragen und Antworten

Kontakt

Hendrik Gossens

Consultant Oracle Based Solutions

OPITZ CONSULTING GmbH
Kirchstr. 6 | 51647 Gummersbach
Tel. +49 (2261) 60 01-0
hendrik.gossens@opitz-consulting.com

youtube.com/opitzconsulting

@OC_WIRE

slideshare.net/opitzconsulting

xing.com/net/opitzconsulting