

EINSTIEG IN ELM



Carsten König - @carstenk_dev

Code / Slides unter github.com/CarstenKoenig/DWX2017

ELM

- Web-Frontendentwicklung
- nach JavaScript übersetzt
- **rein** funktionale Sprache
- ML Syntaxfamilie (F#, Haskell, OCaml, ...)

TOOLS UND INSTALLATION

INSTALLATION

- Installer für **Windows** und **Mac**
- Alle Plattformen: `npm install -g elm`
- empfehlenswert: `npm install -g elm-format`

EDITOR

- Online:
 - TryElm <http://elm-lang.org/try>
 - Ellie <https://ellie-app.com/new>
- Editor-Support
 - VS.code mit [vscode-elm](#)
 - Atom, Brackets, Emacs, IntelliJ, ... siehe [Elm Guide](#)

ELM MAKE

- initialisieren eines Projekt `elm make`
- kompilieren eines Projekts nach JavaScript `elm make`
`Main.elm --output=main.js`

ELM REPL

Read Eval Print Loop

- Konsole `elm repl`
- Online <http://elmrepl.cuberooot.in/>

ELM REACTOR

- Kompilieren und Anzeigen von Elm Modulen im Browser
- *Debugger*

ELM PACKAGE

- Herunterladen von Packages `elm package install elm-lang/core`
- Veröffentlichen von Packages `elm package publish`
 - erzwingt **semver**
- zeigt Unterschiede zwischen Versionen `elm package diff elm-lang/core 3.0.0 4.0.0`

THE ELM ARCHITECTURE

MODEL

Beschreibung des gesamten Zustands eines Programms

```
1: type alias Model =  
2:     { augenzahl : Int  
3:     }
```

VIEW

Wandelt das **Modell** in eine HTML Darstellung um, die Elm dann anzeigt

```
1: view : Model -> Html Never
2: view model =
3:     viewDice model.augenzahl
```

MESSAGES

zeigt eine gewünschte Zustandsänderung an

```
1: type Msg
2:     = AugenzahlAendern Int
```

werden z.B. in der View über Events ausgelöst

```
1: viewSwitchButton n =
2:     button
3:         [ onClick (AugenzahlAendern n)
```

UPDATE

verknüpft eine **Message** mit aktuellen Zustand und liefert neuen Zustand

```
1: update : Msg -> Model -> Model
2: update msg model =
3:   case msg of
4:     AugenzahlAendern n ->
5:       { model | augenzahl = n }
```

der neue Zustand wird dann über **view** angezeigt

NEUE MAIN

```
1: main : Program Never Model Msg
2: main =
3:     beginnerProgram
4:         { model = init
5:           , update = update
6:           , view = view
7:         }
```

EFFEKTE

- bisher alles **pure**
- wie bekommen wir Seiteneffekte?
 - Zufallszahl
 - HTTP Requests
 - Systemzeit
 - ...

SUBSCRIPTIONS

Wenn wir über nicht von der View ausgelöste Ereignisse benachrichtigt werden wollen

- WebSocket Nachricht
- Taste gedrückt
- Maus bewegt
- Timer
- ...

SUBSCRIPTIONS

```
1: subscriptions : Model -> Sub Msg
2: subscriptions model =
3:   Time.every second Tick
```

COMMANDS

Lassen uns Seiteneffekte auslösen

- HTTP Request
- Zufallszahl erzeugen
- ...

COMMANDS

```
1: update msg model =
2:     case msg of
3:         ...
4:         ZufaellichAendern ->
5:             ( model, zufallszahlErzeugen )
6:
7:
8: zufallszahlErzeugen : Cmd Msg
9: zufallszahlErzeugen =
10:     Rnd.generate AugenzahlAendern (Rnd.int 1 6)
11:
```

NEUE MAIN

```
1: main : Program Never Model Msg
2: main =
3:   program
4:     { init = init
5:       , update = update
6:       , view = view
7:       , subscriptions = always Sub.none
8:     }
```

KOMPONENTEN MIT DER TEA

DICE - KOMPONENTE

- in eigenes Modul mit entsprechenden `init`, `update`, `view`, `Cmd` Funktionen
- und eigenem `Model`, `Msg` Typ

```
1: module Dice exposing (Model, Msg, init, update, view, random)
```

IN MAIN

Komponenten-Model in **Model** aufnehmen:

```
1: type alias Model =  
2:     { dice : Dice.Model  
3:     }
```


IN MAIN

Komponenten-*Messages* wrappen

```
1: type Msg
2:     = DiceMsg Dice.Msg
3:     | ZufaellichAendern
```

IN MAIN

Mappen der Komponenten-Messages, Commands, Subscriptions mit `Html.map`, `Cmd.map` und `Sub.map`

```
1: init : ( Model, Cmd Msg )
2: init =
3:   let
4:     ( diceModel, diceInitCmd ) =
5:       Dice.init
6:   in
7:     ( Model diceModel, Cmd.map DiceMsg diceInitCmd )
```

DEMO MEHRERE KOPIEN

EINBETTEN IN HTML

```
1: <div id="main"></div>
2: <script src="app.js"></script>
3: <script>
4:     var node = document.getElementById('main');
5:     var app = Elm.App.embed(node);
6: </script>
```

PORTS

- sollten in eigenem `port module` liegen
- `port name : output -> Cmd msg` für Elm nach JS
- `port name : (input -> msg) -> Sub msg` für JS nach Elm

```
1: port module Prompt exposing (..)
2:
3:
4: port showPrompt : String -> Cmd msg
5:
6: port promptResult : (String -> msg) -> Sub msg
7:
```

Javascript kann **Cmd** Ports *subscriben* und an **Sub** Ports *senden*:

```
1: var app = Elm.TeaDemoPorts.embed(node);  
2: app.ports.showPrompt.subscribe (function(frage) {  
3:   var antwort = prompt(frage);  
4:   app.ports.promptResult.send(antwort);  
5: });
```


- Daten über *Ports* übertragen
- es gelten **Einschränkungen**
- **Value** zum Austausch empfohlen
 - Verwendung über **Decoder**
 - und **decodeValue**

LINKS UND CO.

- Elm Guide Online: <https://guide.elm-lang.org/>
- Installieren: <https://guide.elm-lang.org/install.html>
- Package Verzeichnis / Docs: <http://package.elm-lang.org/>
- *fancy Search* <https://klaftertief.github.io/elm-search/>

VIELEN DANK