Разработка frontend на Haskell без JS и производных

Прокопенко Андрей

Кусок Истории

- **Julius** (Template Haskell + JS): https://www.yesodweb.com/book/shakespearean-templates
- **Haste** (Latest commit on Sep 15, 2017): https://haste-lang.org
- …
- **GHCJS** (bundle size >1MB):
    - **Reflex:** https://reflex-frp.org
    - **Miso** (GHCJS + JSaddle)**:** https://haskell-miso.org
    - **JSaddle** (Bridge between GHC and GHCJS): https://hackage.haskell.org/package/jsaddle

Что делать

- **Fay:** https://github.com/faylang/fay
- Подмножество Haskell, есть поддержка ADT
- Не поддерживает населенные классы типов (а значит, монады и многое другое)
- Тривиальный FFI
- API для чтения данных на сервере

Fay: output

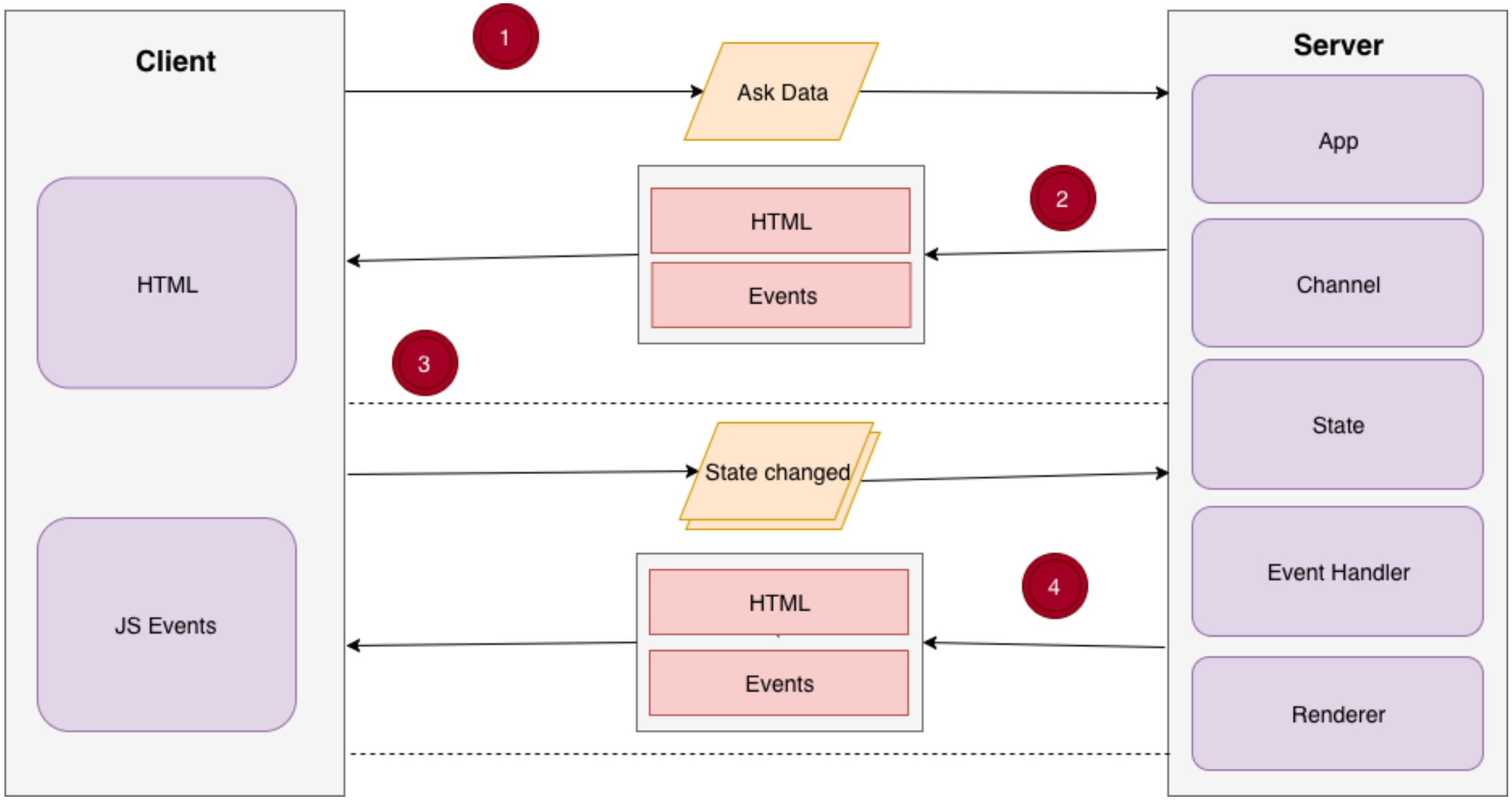
```
// Built-in ==.
function Fay$$eq(x){
  return function(y){
    return new Fay$$$(function(){
      return Fay$$equal(x,y);
    });
  };
}
```

Fay: data transcoding

```
data A

{"instance":"A"}

data A2 = A2 Int

{"instance":"A2", "slot1":123}
```

Fay: FFI

```
onKeyPress :: Element -> Fay () -> Fay ()
onKeyPress = ffi "%1.onkeypress=%2"

onKeyDown  :: Element -> Fay () -> Fay ()
onKeyDown = ffi "%1.onkeydown=%2"

setInnerHTML :: Element -> Text -> Fay ()
setInnerHTML = ffi "%1.innerHTML=%2"
```


Client
HTML
JS Events
1
Ask Data
Server
App
Channel
State
Event Handler
Renderer
HTML
Events
2
3
State changed
HTML
Events
4

Сервер

- websockets (как общаться с клиентом)
- ~ blaze-html (DOM с поддержкой событий)
- state:
    - как рендерить на клиенте
    - как обрабатывать события

Мост

```haskell
data In a = PingPong
  | Send (Action a)
  | AskEvents
  deriving (Data, Typeable)

data Out a = EmptyCmd
  | ExecuteClient ClientId (ClientTask a) ExecuteStrategy
  deriving (Data, Typeable)

data ExecuteStrategy =
  ExecuteAll | ExecuteExcept
  deriving (Data, Typeable, Eq)

data ClientTask a = ClientTask
  { executeRenderHtml :: [RenderHtml]
  , executeAction :: [CallbackAction (Action a)]
  } deriving (Data, Typeable)

data RenderHtml = AttachText ElementId HtmlText
  | AttachDOM ElementId HtmlText deriving (Data, Typeable)

data CallbackAction a = CallbackAction (EventHandler a)
#ifdef FAY
  deriving (Typeable, Data)
#else
  deriving (Typeable, Data)
```

```haskell
data EventHandler a
    = OnKeyDown    !a
    | OnKeyUp      !a
    | OnKeyPress   !a
    | OnFocus !a
    | OnChange     !a
    | ...
```

Клиент

```haskell
onMessage' :: WSEvent -> Fay ()
onMessage' evt = do
  ws <- target evt
  responseText <- eventData evt
  response <- parse responseText
  case response of
    EmptyCmd -> return ()
    ExecuteClient cid task strategy -> do
      if strategy == ExecuteAll
        then do
          forM_ (executeRenderHtml task) $ \html ->
            case html of
              AttachText eid val -> attachToElemById eid val
              AttachDOM eid val  -> attachToParentById eid val
          forM_ (executeAction task) $ \act -> addListener ws act
        else return ()

addListener :: WebSocket -> CallbackAction (Action a) -> Fay ()
addListener ws (CallbackAction eh) = handle ws eh
```

Пример: Model + Rendering

```haskell
data App = App
  { appModel :: TVar Model
  , appChannel :: TChan (Out (Action Msg))
  , appStatic :: Static
  }

-- * Model

data Model = Model
  { entries :: [Entry]
  , nextId :: Int
  }

data Entry = Entry
  { description :: Text
  , eid :: Int
  }

-- * Actions

data Msg = UpdateEntry Int Bool
  | Add
  | Complete Int
  | Update Int RecordValue
  deriving (Show, Typeable, Data)
```

```haskell
renderModel :: Model -> H.Markup (Action Msg)
renderModel Model{..} = do
  H.h1 $ "TODO MVC"
  H.br
  forM_ entries $ \entry -> renderEntry entry
  let btnId = "todo-add"
  H.button
    ! A.id "todo-add"
    ! A.type_ "submit"
    ! E.onClick (Action btnId ObjectAction Add)
    $ "Добавить"

renderEntry :: Entry -> H.Markup (Action Msg)
renderEntry Entry{..} = do
  let elemId = "todo-" <> (T.pack $ show eid)
      removeId = "remove-" <> (T.pack $ show eid)
  H.input
    ! A.id (toValue elemId)
    ! A.type_ "text"
    ! A.value (toValue description)
    ! E.onKeyUp (Action elemId RecordAction (Update eid ""))
  H.button
    ! A.id (toValue removeId)
    ! A.type_ "submit"
    ! E.onClick (Action removeId ObjectAction (Complete eid))
    $ "Завершить"
```

Пример: Event Handling

```haskell
onCommand cmd stateTVar = do
  m@Model{..} <- STM.readTVarIO stateTVar
  case readFromFay' cmd of
    Right (Send (Action _ _ acmd)) -> do
      case acmd of
        Add -> do
          let newTodo = Entry "TODO: " nextId
              newState = Model (newTodo : entries) (succ nextId)
              task = createTask "root" renderModel newState
          atomically $ void $ swapTVar stateTVar newState
          cid <- lift clientSession
          return (ExecuteClient cid task ExecuteAll)
        Complete _eid -> do
          let newState = Model (filter ((/=_eid) . eid) entries) nextId
              task = createTask "root" renderModel newState
          atomically $ void $ swapTVar stateTVar newState
          cid <- lift clientSession
          return (ExecuteClient cid task ExecuteAll)
        Update _eid val-> do
          let newState = Model ((upd _eid val) <$> entries) nextId
              upd _id val' e@Entry{..} =
                if eid == _id then e { description = val', eid = _id } else e
              task = createTask "root" renderModel newState
          atomically $ void $ swapTVar stateTVar newState
          cid <- lift clientSession
          return (ExecuteClient cid task ExecuteExcept)
    Right AskEvents -> do
      let task = createTask "root" renderModel m
      cid <- lift clientSession
      return $ ExecuteClient cid task ExecuteAll
```

Q&A

Спасибо!

- https://github.com/swamp-agr/front