

# Reaktivní programování

## Table of Contents

|                             |   |
|-----------------------------|---|
| 1. Koncept .....            | 1 |
| 1.1. Glosář .....           | 2 |
| 1.2. Statické vztahy .....  | 2 |
| 1.3. Dynamické vztahy ..... | 3 |
| 2. API .....                | 4 |
| 2.1. react .....            | 4 |
| 2.2. is .....               | 5 |
| 3. Příklady .....           | 6 |

## 1. Koncept

Ve verzi 0.6.0 byla zavedena podpora pro "reaktivní programování" se záměrem snížit velikost a složitost programů Red. Reaktivní model Redu se při sestavování orientovaného (directed) grafu a propagaci změn u objektů opírá o tok dat a události objektů. Používá model "push". Přesněji řečeno, Red používá model [objektově orientovaného programování](#), kde pouze pole objektů mohou být zdrojem změn.

Reaktivní API a jeho používání je jednoduché i když jeho popis je poněkud abstraktní. Zde je několik ilustrací reaktivních vztahů.

[react-simple] | [react-simple.png](#)

*Grafy A a B zobrazují jednoduché vztahy mezi jedním nebo několika reaktory (objekty, které slouží jako zdroj reaktivity)*

[react-graph] | [react-graphs.png](#)

*Grafy C, D a E zobrazují zřetěžené reakce, kde některé cíle jsou samy reaktory, tvoříce tak řetěz vztahů, jež mohou mít různý tvar*

Reakce probíhají asynchronně při změně hodnoty zdrojového pole. Reakční vztahy jsou zachovány dokud není reakce explicitně zničena podnětem `react/unlink` nebo `clear-reactions`.

V reaktivním výrazu musí být pouze zdrojový objekt reaktorem. Cílem může být jednoduchý objekt. Je-li cíl rovněž reaktorem, dochází ke zřetěžení reakcí, což se projeví automatickým vytvořením vztahového grafu.

## NOTE

- Reaktivní podpora Redu může být v budoucnu rozšířena o model "pull".
- Toto není framework FRP ([Funkcional reactive programming](#)), i když proudy událostí mohou být v budoucnu podporovány.
- Strojek Red/View GUI se při operacích s grafickými objekty opírá o objekty (piškoty) typu **face**! Piškoty jsou reaktory a mohou být použity pro ustavení reaktivních vztahů mezi piškoty nebo s nereaktorovými objekty.

## 1.1. Glosář

| Expression                    | Definition   |
|-------------------------------|--|
| <b>reaktivní programování</b> | Programovací paradigma, podmnožina programování toku dat, založené na událostech, které "push" (posílání) změny. |
| <b>reakce</b>                 | Blok kódu, který obsahuje jeden nebo víc reaktivních výrazů.   |
| <b>reaktivní výraz</b>        | Výraz, který odkazuje alespoň na jeden reaktivní zdroj.  |
| <b>reaktivní vztah</b>        | Vztah mezi dvěma či více objekty, zavedený s použitím reaktivních vztahů.  |
| <b>reaktivní zdroj</b>        | Hodnota <b>path!</b> , odkazující na pole reaktivního objektu.   |
| <b>reaktivní formule</b>      | Reakce, která vrací poslední výsledek vyhodnoceného výrazu.  |
| <b>reaktivní objekt</b>       | Objekt, jehož pole mohou být použita jako reaktivní zdroje.  |
| <b>reaktor</b>                | Alias pro "reaktivní objekt".  |

## 1.2. Statické vztahy

Nejjednodušší formou reakcí je *statický vztah*, vytvořený mezi *pojmenovanými* objekty. Říkáme statický proto, že staticky spojuje objekty. Vztahuje se výlučně na své zdrojové reaktory a nemůže být použit pro jiné objekty.

### Příklad 1

```
view [  
  s: slider return  
  b: base react [b/color/1: to integer! 255 * s/data]]
```

Tento příklad zavádí reaktivní vztah mezi posuvníkem (slider), zvaným **s** a bázovým piškotem (base face), zvaným **b**. Při pohybu posuvníku se odpovídajícím způsobem mění barva pozadí bázového piškotu. Reaktivní výraz nemůže být použit pro jinou sadu piškotů. Toto je nejjednodušší forma reaktivního chování grafických objektů v Red/View.

### Příklad 2

```
vec: reactor [x: 0 y: 10]
box: object [length: is [square-root (vec/x ** 2) + (vec/y ** 2)]]
```

Jiná forma statického vztahu může být definována s použitím operátoru **is**, kde hodnota reaktivního výpočtu je vztažena ke slovu (v jakékoli souvislosti).

Tento příklad nesouvisí se systémem GUI. Počítá délku vektoru, určeného parametry **vec/x** a **vec/y**, s použitím reaktivního výrazu. Zde opět je zdrojový objekt staticky určen svým jménem (**vec**) v reaktivním výrazu.

### Příklad 3

```
a: reactor [x: 1 y: 2 total: is [x + y]]
```

Slovo **total** má svou hodnotu vztaženu k výrazu **x + y**. S každou změnou hodnot **x** nebo **y** je okamžitě aktualizována hodnota slova **total**. Všimněte si, že v tomto případě není zapotřebí určovat cesty k reaktivním zdrojům, neboť funkce **is** je použita přímo uvnitř těla reaktoru, jenž zná její souvislosti.

### Příklad 4

```
a: reactor [x: 1 y: 2]
total: is [a/x + a/y]
```

Tato variace příkladu 3 ukazuje, že i globální slovo může být cílem reaktivního vztahu (i když nemůže být zdrojem). Tato forma je nejbližší modelu, používaného v aplikaci Excel (a jiných spreadsheetech).

Poznámka: vzhledem k velikosti globálního kontextu, může mít jeho reaktivní forma (jako nahoře s **total**) výrazně negativní vliv na rychlost výpočtu, i když se to v budoucnu může změnit.

## 1.3. Dynamické vztahy

Statické vztahy se snadno definují ale nesnadno se přizpůsobují při potřebě poskytnout tutéž reakci určitému počtu reaktorů nebo jsou-li tyto anonymní (připomínka: všechny objekty jsou implicitně anonymní). V takových případech by měla být reakce určena s použitím funkce **follow** a **react/link**.

### Příklad

```

;-- Pohybuj červeným balonem myši nahoru a dolů. Pozoruj, jak reagují ostatní balony..

win: layout [
  size 400x500
  across
  style ball: base 30x30 transparent draw [fill-pen blue circle 15x15 14]
  ball ball ball ball ball ball ball b: ball loose
  do [b/draw/2: red]
]

follow: func [left right][left/offset/y: to integer! right/offset/y * 108%]

faces: win/pane
while [not tail? next faces][
  react/link :follow [faces/1 faces/2]
  faces: next faces
]
view win

```

V tomto příkladě je reakcí funkce (**follow**), která je aplikována na piškoty balonů (by pairs). To vytváří řetězec vztahů, které spojují všechny balony. Termíny v reakcích jsou názvy parametrů, takže mohou být použity pro jiné objekty (narozdíl od statických vztahů).

## 2. API

### 2.1. react

#### Syntaxe

```

react <code>
react/unlink <code> <source>

react/link <func> <objects>
react/unlink <func> <source>

react/later <code>

<code>      : blok kódu, který obsahuje alespoň jeden reaktivní zdroj (block!).
<func>      : funkce, která obsahuje alespoň jeden reaktivní zdroj (function!).
<objects>   : seznam objektů, použitých jako argumenty reaktivní funkce (block! of
object! values).
<source>    : slovo 'all nebo objekt či seznam objektů (word! object! block!).

Returns     : <code> nebo <func> pro další odkazy na reakci

```

#### Popis

Funkce **react** zavádí nový reaktivní vztah, který obsahuje alespoň jeden reaktivní zdroj, jímž může být blok kódu (zavádí "statický vztah") nebo funkce (zavádí "dynamický vztah" a vyžaduje upřesnění **/link**). V obou případech je kód staticky analyzován za účelem určení reaktivních zdrojů (s hodnotami typu **path!**), které odkazují na pole reaktoru.

Implicitně je nově vytvářená reakce volána ihned po vytvoření před odezvou funkce **react**. To může být v některých případech nežádoucí a lze se tomu vyhnout volbou **/later**.

Reakce obsahuje libovolný kód Redu, jeden nebo více reaktivních zdrojů a jeden nebo více reaktivních výrazů. Je na uživateli, aby určil výběr vztahů, které nejlépe vyhovují jeho potřebám.

Volba **/link** přijímá funkci jako reakci a seznam objektů jako argumenty, které mají být použity při výpočtu reakce. Tato alternativní forma umožňuje dynamické reakce, jehož kód lze opětovně použít pro jinou sadu objektů (základní funkce **react** umí pracovat pouze s pojmenovanými objekty).

Reakci lze odebrat upřesněním **/unlink** a jedním z následujících **<zdrojových>** argumentů:

- Slovo 'all' - odebere všechny reaktivní vztahy vytvořené reakcí.
- Hodnota typu objekt - odebere pouze ty vztahy, ve kterých je tento objekt reaktivním zdrojem.
- Seznam objektů - odebere pouze ty vztahy, ve kterých jsou uvedené objekty reaktivními zdroji.

Funkce **/unlink** přijímá reaktivní blok nebo funkci jako argument, takže jsou odebrány pouze vztahy, vytvořené z této reakce.

## 2.2. is

### Syntaxe

```
<word>: is [<body>]
```

```
<word>: is [[<default>] <body>]
```

**<word>** : slovo, které má být přiřazeno k výsledku reakce (**set-word!**).

**<body>** : libovolný kód Redu, který obsahuje alespoň jeden reaktivní zdroj.

**<default>** : libovolný kód Redu, který vrátí počáteční implicitní hodnotu, použitou pro **<word>**.

### Popis

Operátor **is** vytváří reaktivní formuli, jejíž výsledek bude přiřazen ke slovu. Blok **<code>** může obsahovat odkazy jak na pole obalujícího (wrapping) objektu (pokud je použit v těle bloku reaktoru), tak na pole externího reaktoru. Jako první element reaktivní formule bloku může být vložen blok s implicitní hodnotou. To je zejména důležité při použití dopředného odkazu pro reaktivní zdroje, které nejsou zadány v okamžiku vyhodnocení formule.

Poznámka: Tento operátor vytváří reaktivní formule, napodobující formule modelu Excel.

### 3. Příklady

a: reactor [x: 1 y: 2 total: is [x + y]]

a/total == 3 a/x: 100 a/total == 102

reactor [a: 1 b: is [[none] c] c: is [a < 4]] == make object! [ a: 1 b: true c: true ]

=== react?

*\*Syntaxe\**

react? <obj> <field> react?/target <obj> <field>

<obj> : kontrolovaný objekt (object!). <field> : pole kontrolovaného objektu (word!).

Vrací : reakci (block! function!) nebo hodnotu none!

*\*Popis\**

Funkce `react?` kontroluje, zda je pole objektu reaktivním zdrojem. Pokud ano, vrátí se první reakce, nalezená jako zdroj v poli objektu; pokud ne, vrací se hodnota `none`. Upřesnění `/target` kontroluje, zda je pole cílem místo zdrojem a vrátí první reakci, zacílenou na toto pole nebo vrátí `none` při absenci shody.

=== clear-reactions

*\*Syntaxe\**

clear-reactions

*\*Popis\**

Odstraní bezpodmínečně všechny definované reakce.

=== dump-reactions

*\*Syntaxe\**

dump-reactions

### **\*Popis\***

Vytvoří seznam registrovaných reakcí pro ladící účely.

**==** Reaktivní objekty

Obyčejné objekty v Redu nevykazují reaktivní chování. Aby se objekt stal reaktivním zdrojem, musí být vytvořen z jednoho z následujících prototypů:

**===** reactor!

### **\*Syntaxe\***

**make reactor!** <body>

<body> : blok s tělem objektu (block!).

Vrací : reaktivní objekt.

### **\*Popis\***

Vytvoří nový reaktivní objekt z těla bloku. Nastavení pole vráceného objektu na novou hodnotu spustí reakce pro toto pole definované.

NOTE: Tělo může obsahovat výrazy s `is` .

**===** deep-reactor!

### **\*Syntaxe\***

**make deep-reactor!** <body>

<body> : blok s tělem objektu (block!).

Vrací : reaktivní objekt.

### **\*Popis\***

Vytvoří nový reaktivní objekt z těla bloku. Nastavení pole vráceného objektu na novou hodnotu nebo změna řady (series), na kterou pole odkazuje (včetně vnořených řad), spustí reakce pro toto pole definované.

NOTE: Tělo může obsahovat výrazy s `is`.

### **\*Příklad\***

Ukazuje, jak změna řady, dokonce i vnořené, spustí reakci.

NOTE: Je na uživateli aby v tomto případě zabránil cyklování. Když například `deep-reactor!` mění ve formuli reaktoru (e.g. `is` hodnoty řad, může vytvořit nekonečné reaktivní cyckly.

```
r: deep-reactor [ x: [1 2 3] y: [[a b] [c d]] total: is [append copy x copy y] ] append r/y/2 'e print mold  
r/total
```