Query combinatoren

|  |  |  |
| --- | --- | --- |
| Joop Ringelberg | 26-09-17 | Versie: 1 |

# Introductie

Queries worden geschreven met propertygetters en combinatoren. Een combinator neemt één of meerdere getters als argument en produceert een functie met de signatuur van een getter (single of plural). Dankzij deze signatuur kunnen combinaties weer onderdeel zijn van andere combinaties.

Combinatoren en getters bewaren resultaten in Locations en verbinden die onderling. Hiermee bereiken we vier doelen:

1. dependency tracking (wat geen doel op zich is maar de andere drie mogelijk maakt);
2. resultaten van berekeningen worden bewaard, zodat herberekenen niet nodig is;
3. deelresultaten kunnen soms worden hergebruikt;
4. een theorieverandering kunnen we verwerken door alléén de strikt noodzakelijke berekeningen opnieuw uit te voeren.

Voordat een query-expressie wordt uitgerekend, moeten we kunnen nagaan of dat al eerder is gebeurd. Onder een query-expressie verstaan we de applicatie van een query (=functie) op een locatie die een argument voor die functie bevat. Omdat op dezelfde locatie willekeurig veel queries kunnen worden toegepast, en we al die resultaten willen bewaren, moeten we hen opslaan met een index. Wanneer we een query toepassen, leiden we uit die query een identificatie (=naam) af en gaan na of er al een resultaat is bewaard in de betreffende locatie met die identificatie.

De naamgeving van getters is eenvoudig: we kiezen de naam van de property waarmee de getter is geconstrueerd. Voor queries construeren we een naam die is samengesteld uit de naam van de toegepaste combinator en zijn componenten.

In dit stuk beschrijf ik de naamgeving van queries en laat zien welke structuur van locaties ontstaat.

# Functienamen in Purescript

De implementatie van Location.purs maakt gebruik van de *name* property van javascript fucnties.

Een functie die op top level in een module is gedefineerd, resulteert in een javascript functie die ook daadwerkelijk die naam draagt, zoals de functie f hieronder:

f : Integer -> Integer

f = a + 1

Ook een lokaal gebonden functie (met let of where) resulteert in een gelijknamige javascript functie, zoals de functie g hieronder:

f :: Integer -> Integer

f x = g x + g x where

g y = y + 1

## Functionele resultaten hebben geen naam

Pas op met de onderstaande situatie:

f = (\x -> x + (g 1)) where g y = y \* 2

Hier heeft de javascript functie die gebonden is aan f, geen naam (d.w.z. heeft geen name property).

## Wanneer wordt de name property gebruikt?

De functionele argumenten van map, apply en bind (van Location en van LocationT e a) dienen een name property te hebben (dit geldt voor de javascript functies die door de implementatie van de genoemde functies worden geconsumeerd). Hetzelfde geldt voor de functie nestLocationInMonad.

# Namen voor combinatoren

Om misverstanden te voorkomen: het gaat niet zozeer om de naam van de combinator zelf (dus de functie die componenten combineert), als wel om de naam van de samengestelde functie die het resultáát is van de combinator. Oftewel, om de naam van de query die door de combinator wordt geproduceerd. Een voorbeeld:

f >-> g

de beginlocatie. Nemen we aan dat f en g getters zijn, dan is de naam van de geconstrueerde query: f>->g. Een echt voorbeeld:

rol\_RolBinding >-> label

De naam hiervan is: model:SysteemDomein#rol\_RolBinding>->rdfs:label (merk op dat de bijdrage van de getters de curies van de properties zijn!).

Om een tweede misverstand te voorkomen: de naam van deze query wordt niet gebruikt als alléén deze query wordt toegepast op een locatie. Pas als deze query een onderdeel is van een grotere query, wordt de naam gebruikt (als onderdeel van de naam van die grotere query).

## Single to Single compositie operator (>->)

Deze combinator bestaat simpelweg uit twee single getters. De naam is:

{naam van linker operand}>->{naam van rechter operand}

f

g

## Single to Plural compositie operator (>->>)

Deze combinator bestaat simpelweg uit een single getter en een plural getter. De naam is:

{naam van linker operand}>->>{naam van rechter operand}

f

g

## Plural to Single compositie operator (>>->)

Deze combinator is complexer. De naam volgt hetzelfde patroon. Maar deze combinator legt een interne structuur aan. Uit het resultaat van de eerste operand f (een array met resources) construeert hij een array met (maybe) resources (dus: Maybe Resource). De tekening bevat óók de locaties met resources en hun g-waarde. De ovalen binnen de tweede locatie van boven zijn gestippeld getekend om aan te geven dat de *inhoud* van deze locaties in het array zit). Dit array wordt omgezet naar een locatie met een array van resources. Deze laatste locatie wordt verbonden met de locatie die het resultaat van de eerste operand bevat, onder de naam van de tweede operand. Merk op dat deze naam óók gebruikt wordt om elke individuele locatie met een resource s te verbinden met zijn functiewaarde t (t = g(s)).

f

g

g

g

## Plural to Plural compositie operator (>>->>)

De naamgeving volgt het inmiddels bekende patroon. Ook deze combinator maakt een interne structuur. Hier wordt echter een tussenresultaat berekend dat een array van arrays van resources is. Deze arrays worden samengevoegd in het eindresultaat, dat met de naam van g verbonden wordt met het resultaat van de eerste operand.

f

g

g

g

## concat combinator

Deze combinator (geen infix operator) concateneert de meervoudige resultaten van zijn twee operandi (beiden plural getters). De naam van met deze combinator gegeneerde queries volgt deze structuur:

concat {naam van eerste operand} {naam van tweede operand}

Deze combinator wordt gemaakt door gewone array concatenatie te liften naar de monad stack van Aff en Locatie. Dit betekent dat alle locaties en verbindingen automatisch gelegd worden door de map en apply functies van LocationT. In de figuur staan nu ook de namen van locaties weergegeven. We zien dat de map (<$>) operator een tussenlocatie aanmaakt. Deze tussenlocatie bevat een anonieme functie. Het uiteindelijke resultaat heeft twee supports, conform de werking van apply. Beide verbindingen dragen de naam die de apply functie construeert.

f

g

f r

g r

concat<$>

concat<$>f r

concat<$>f r<\*>

concat<$>f r<\*>

concat<$>f r<\*>g r

r

## addTo combinator

Deze combinator volgt hetzelfde patroon als de concat combinator, maar de eerste operand is een single getter. Het resultaat van deze getter wordt toegevoegd aan het resultaat van de tweede operand, een plural getter. De naam van met deze combinator gegeneerde queries volgt deze structuur:

addTo {naam van eerste operand} {naam van tweede operand}

De opgezette structuur van locaties lijkt sterk op die van concat.

f

g

f r

g r

cons<$>

cons<$>f r

cons<$>f r<\*>

cons<$>f r<\*>

cons<$>f r<\*>g r

r

## isNothing query

Deze query is een singleGetter. Indien die de locatie waarop hij wordt toegepast, een Nothing waarde bevat, is het resultaat van de query false; anders is het true.

isNothing :: forall e. Location (Maybe Resource) -> NestedLocation e (Maybe Boolean)

Hoewel deze query geen combinator is, is hij nodig voor de combinator hasValue en is daarom opgenomen in de file combinators.purs.

## hasValue combinator

Deze combinator neemt één singleGetter operand en combineert die met de isNothing singleGetter. De definitie luidt:

f >-> isNothing

De structuur van locaties die gegenereerd wordt, is dat van single to single. De naam van met deze combinator gegenereerde queries volgt dit patroon:

hasValue {naam van operand}

## filter combinator

De filter combinator neemt twee operandi als argumenten: één SingleGetter Boolean (het *criterium*) en één PluralGetter Resource. De resulterende query is ook een PluralGetter Resource. Het resultaat van de query, wanneer toegepast op een locatie met een resource, is een (locatie met) een array van resources die allemaal aan het criterium voldoen. De naamgeving is volgens dit patroon:

filter {naam van criterium} {naam van tweede operand}

<s1;true>, <s2;false>…

f

criterium

criterium

Filter criterium

filter criterium f r

f r

De drie locaties die worden gemaakt, zijn op twee manieren met elkaar verbonden. De belangrijke afhankelijkheid is die tussen f r (de bovenste locatie) en filter criterium f r (de onderste locatie). De verbinding tussen de middelste en onderste locatie is niet belangrijk voor dependency tracking of herberekening.

## mclosure combinator

Deze combinator is als het ware een Y-combinator voor de querytaal. Hij introduceert de recursie van een singlegetter. Hij neemt één operand, de singlegetter (SingleGetter Resource), en past die herhaaldelijk toe. Daarbij verzamelt hij de tussenresultaten. Het proces stopt zodra de singlegetter geen resultaat meer geeft (Nothing teruggeeft) of als een resultaat wordt teruggegeven dat al eerder is geproduceerd. De query die ontstaat berekent, kortom, het *fixpoint* van de singlegetter. De naam van de query wordt samengesteld volgens dit patroon:

mclosure {naam van singlegetter}

De structuur van locaties die gegenereerd wordt, hangt af van het aantal recursieve stappen. Voor elke stap – behalve de laatste – wordt hetzelfde patroon gegenereerd. Daarbij worden de verbindingen zodanig gelegd, dat als de theorie verandert en de singlegetter op een bepaalde resource een ànder resultaat produceert, de structuur automatisch aangepast wordt. Dit kan twee kanten op gaan:

* als het nieuwe resultaat Nothing is, wordt de structuur op de betreffende plek afgekapt;
* als het nieuwe resultaat niet Nothing is, wordt de structuur aan het eind voortgezet.

Dit is de structuur van locaties.

a

mcons<$>

mcons<$>a

mcons<$>f a<\*>

mcons<$>a<\*>

concat<$>f a<\*>f a

f a

mcons<$>

mcons<$>f a

mcons<$>f a<\*>

f

Nothing

mcons<$>

mcons<$>Nothing

mcons<$>Nothing<\*>

f

mcons<$>f a<\*>

mcons<$>Nothing<\*>

a >>= recur

f a >>= recur

Nothing >>= recur

Toelichting.