Perspectives types als Purescript types

|  |  |  |
| --- | --- | --- |
| Joop Ringelberg | 03-02-19 | Versie: 1 |

# Introductie

In de ‘hogere’ modules van de Perspectives implementatie komen we queries tegen. Deze queries zijn gedefinieerd als combinaties van ObjectGetters, of van TripleGetters (deze laatste memoriseren resultaten in de dependency tracking cache). Een (basis) ObjectGetter beeldt een Context af op Rollen, of een Rol op Propertywaarden. Overigens, een Getter heeft met de *identifiers* van Contexten en Rollen als parameter of resultaat; nooit de representaties zelf.

Bij het combineren van ObjectGetters moeten we goed letten op de Perspectives-types van de zaken waarnaar die identifiers verwijzen. Zo moeten we fouten voorkomen, zoals deze: *neem van een Context zijn buitenrol en daar dan een rol in context van*.

Echter, de ‘lagere’ modules van Perspectives representeren die identifiers als niet-getypeerde Purescript Strings. Of een identifier nu een Context, Rol of waarde representeert – het is en blijft een String voor Purescript. Dat betekent dat de Purescript compiler ons niet helpt bij de compositie van ObjectGetters tot queries.

Bij complexere queries blijkt dat je snel type-fouten maakt. Daarom representeren we Perspectives types tot op zekere hoogte als Purescript types.

# Wat we van elkaar onderscheiden

Perspectives is een modelleertaal waarin we onbeperkt veel types kunnen beschrijven. De Purescript implementatie van Perspectives daarentegen is gesloten en eindig: we kunnen er slechts een beperkt aantal Perspectives types in representeren. De keuze is pragmatisch: het gaat om de ondersteuning van de programmeur(s) van Perspectives.

In Perspectives komen we alleen queries tegen die zijn opgesteld in termen van één de volgende basismodellen:

* model:Perspectives
* model:QueryAst

We representeren dus alleen de types die we in deze modellen tegenkomen.

## Contexten

Elke Context die we in Purescript code kunnen tegenkomen, heeft dus een type dat gemodelleerd is in één van de basismodellen. In termen van de data structuur PerspectContext: bij pspType mogen we de identifiers verwachten van alle contexten[[1]](#footnote-1) waarvan het type psp:Context is, of psp:Context als Aspect heeft[[2]](#footnote-2). Voor elk van de gedefinieerde typen Contexten in de basismodellen hebben we een Purescript newtype: View, Actie, Zaak, enz., maar ook TrustedCluster (type Zaak) en PBool (type SimpleValue). Eén van deze types is psp:Context zelf. In Purescript is psp:Context afgebeeld op ContextDef (‘de definitie van Context’). Het type ‘Context’ bestaat echter niet.

Bovendien hebben we een Class waarvan alle Purescript typen die een type Perspectives Context uit één van de basismodellen representeren, een instantie zijn. Een functie die werkt op een willekeurig Purescript Context type, geven we een type parameter die onderhevig is aan de beperking dat het een instantie van de Class ContextType moet zijn.

Maar wat typeren we met deze Purescript types? Welnu, de identifiers waarmee we Contexten identificeren. Oftewel: het id\_ veld van PerspectContext (maar zie “Waar maken we onderscheid?”).

## Rollen

Er zijn veel meer rollen gedefinieerd in de basismodellen, dan contexten. Bovendien hebben deze rollen lange namen omdat ze gescoped zijn in hun (vaak geneste) contexten. We onderscheiden deze typen rollen niet van elkaar, maar groeperen ze allemaal onder het type RolDef.

Dus: waar we voor elk type context in de basismodellen een Purescript type hebben, hebben we voor alle types rollen hetzelfde Purescript type: RolDef. Wat ook het type rol, het is altijd een RolDef. Meestal gebruiken we RolDef als argument voor een functie die een getter construeert om instanties van die rol bij een context op te halen (en zij zijn dan getypeerd als RolInContext, zie hieronder!).

We onderscheiden namelijk ook verschillende soorten *instanties van rollen*. We moeten namelijk in de Purescript code rekening houden met de *verschillende verhoudingen die rollen hebben tot contexten*. Is het een RolInContext, een BuitenRol of een BinnenRol?

We hebben daarom precies deze drie types in Purescript. Merk het verschil in omgang op met Context! Daar representeerden we de types die voorkomen als waarde van pspType van PerspectContext, als Purescript types. *Dat doen we dus niet voor rollen*. Nog een keer:

1. geen van de rol-types in de basismodellen (gerepresenteerd met contexten waarvan het type psp:Rol is) is gerepresenteerd in Purescript;
2. in plaats daarvan zijn alle rol-types getypeerd als RolDef;
3. RolInContext, BinnenRol en BuitenRol zijn **niet** gemodelleerd in de basismodellen!

We hebben wel weer een type Class die abstraheert over deze drie soorten rollen: RolKind. We hebben nòg een type Class voor rollen: Binding, met leden RolInContext en BuitenRol. Alleen instanties van Binding komen voor als de binding van een rol.

Elke identifier die een Rol-instantie representeert, typeren we in Purescript dus met een instantie van RolKind. Een identifier die een Rol-type representeert, typeren we met RolDef. Meer hebben we niet nodig.

# Waar maken we onderscheid?

Boven schreef ik dat de member “\_id” van PerspectContext (en PerspectRol) met een Purescript-representatie van Perspectives-types getypeerd wordt. Maar dat is onjuist. Het is contraproductief om op dit lage niveau types te onderscheiden. De typering in Perspectives termen wordt pas nuttig op het niveau van ObjectsGetters.

Sterker nog, als we b.v. PerspectContext typeren, komen we in problemen met de toestand van MonadPerspectives. Daarin houden we namelijk een cache bij van al uit de database gehaalde PerspectContexen. Die cache is een StrMap. Maar een StrMap heeft weliswaar een type-parameter, maar die kan – voor een bepaalde instantie – maar aan één type gebonden worden. Dus als we PerspectContext parametriseren met een type variabele, bindt de compiler daar op enig moment één bepaalde Purescript afbeelding van een Perspectives aan – en andere soorten PerspectContexten komen er niet meer in.

Kortom, op laag niveau houden we het op Strings.

Pas een ObjectsGetter zelf gaan we typeren – en maken er daarmee een TypedObjectsGetter van. We noteren dat met een infix-operator op type niveau:

contextType :: forall t e. ContextType t => (Context ~~> t) e

Maar hoe krijgen we dat voor elkaar? Met unsafeCoerce. Dat is simpelweg een opdracht aan de Purescript compiler om maar aan te nemen dat ons type klopt. We gebruiken daar een alias voor: typeWithPerspectivesTypes.

## De compiler heeft het soms moeilijk

We combineren TypedObjectsGetters met een infix operator: /~/. Deze operator werkt met een ‘tussentype’: gegeven twee getters f :: (a ~~> b) en g :: (b ~~> c) construeren we een getter (a ~~> c). Maar als we dat type geven voor de expressie f /~/ g, klaagt de compiler over een type dat hij gedetecteerd heeft maar wat niet in de body van het type voorkomt. En over dat type wil hij meer weten. Dat kan. Hieronder een voorbeeld:

rolBindingDef :: forall c b rt e. Binding b => RolKind rt => ContextType c => (rt ~~> c) e

rolBindingDef = (binding :: (rt ~~> b) e) /-/ context

Als we de typering van de toepassing van binding weglaten, klaagt de compiler. Het type b komt niet voor in de body van het type ((rt ~~> Context) e). We moeten het declareren, zo nodig beperken met een type class, en gebruiken in de body van de functie.

1. Een type is in Perspectives beschreven met een Context (óók Rol- en Property types!). [↑](#footnote-ref-1)
2. Dit zijn dus Context-typen of Context-beschrijvingen of Context-definities. Verwar ContextDef niet met ContextType. ContextDef is de definitie van Context; ContextType is de Class waarvan alle context-typen lid zijn. [↑](#footnote-ref-2)