Valum micro-framework web

Guillaume Poirier-Morency poirigui@iro.umontreal.ca

Résumé

Valum est un *micro-framework* web écrit en Vala.

- génèse et historique du langage Vala
- présentation du framework Valum
- exemples
- architecture du *middleware*
- architecture du *framework*
- justification
- objectifs
- applications

Historique

À l'origine GIMP, le « GNU Image Manipulator Program » avait sa propre librarie d'interface usager GTK, acronyme de « GIMP Toolkit ».

Le toolkit a été extrait de GIMP pour constituer une librairie complètement indépendante: GTK+.

- l'environnement GNOME est construit sur GTK+
- GTK+ est un projet mature qui fournit un ensemble complet de composantes d'interface utilisateur

De GTK+ à GObject

GTK+ était basé sur GObject, une librairie permettant un style de programmation orienté-objet en C.

Cette librarie a été extraite du projet GTK pour pouvoir être utilisé pour conçevoir d'autres libraries:

- GObject lui-même
- GStreamer pour manipuler des flux audio et vidéo
- GIO, un api de flux asynchrones (*asynchronous stream*)

Elle fournit un ensemble de fonctionnalités adéquates pour programmer des interfaces graphiques:

- propriétés
- signaux
- fermetures et callback
- délégateur

De GObject à Vala

Vala est un langage de programmation moderne orienté-objet qui se compile en C via GObject.

Il permet d'éviter d'écrire la grande quantité de code nécéssaire pour contruire des programmes GObject en C.

En plus de supporter les fonctionnalités de GObject, il donne accès à n'importe quelle librarie écrite en C à travers un modèle de *bindings* basé sur des annotations.

```
[CCode (cname = "mon_objet_t")]
public class MonObjet : Object {
    // implémentation...
}
```

Vala

Vala est très intéressant pour le web, car il repose sur les mêmes principes qui ont rendu JavaScript aussi populaire en programmation serveur:

- traitement asynchrone
- fonctionnalités de haut-niveau (fermetures, signaux, etc...)

Avec quelques avantages considérables:

- haute-performance (compilation et exécution de code machine)
- typage statique, inférence de type et type non-null par défaut
- *bindings* pour les librairies écrites en C

De Vala à Valum

Valum est un projet écrit en Vala été initié il y a environ 4 ans par Antono Vasiljev.

J'ai repris le projet cet hiver dernier lui ai donné un petit coup de jeunesse:

- intégration du protocole FastCGI
- améliorations pour le routage
  - paramètres typés
  - fermeture pour matcher une requête
  - signal pour *handler* une requête
- début de protocole SGI (VSGI)
- améliorations du moteur de vue (*templating engine*)
- sessions et cookies
- intégration de waf
- intégration continue sur Travis CI

Valum, Express.js et Flask

Express.js et Flask sont deux frameworks web qui sont du même niveau architectural que Valum.

Il est donc intéressant de comparer ces plateformes:

	Valum	Express.js	Flask
Langage	Vala	JavaScript	Python
Middleware	VSGI	Node.js	WSGI
Typage	Statique	Dynamique	Dynamique
Traitement asynchrone	Thread Signaux	Event loop Événements	?
Temps d'exécutions	~µs	~ms	~ms

Exemple d'application

Une application écrite sur Valum nécéssite un minimum de code.

```
var app = new Valum.Router ();

// Hello world!
app.get ("", (req, res) => {
    var writer = new DataOutputStream (res);
    writer.put_string ("Hello world!");
});

// sert l'application au port 3003
new SoupServer (app, 3003).listen ();
```

Architecture

Valum est découpé en deux couches: un *middleware* et un *framework*.

Le *middleware* est basé sur libsoup, une librarie qui implémente le protocole HTTP.

Il fournit les abstractions suivante:

Composante	Description
`Request`	représente la la requête du client
`Response`	réponse à la requête du client
`Application`	traite une requête pour produire une réponse
`Server`	sert une application

Diagramme UML de Valum et VSGI.

Architecture du middleware

Les requêtes et les réponses sont basé sur l'api de flux de GIO, ce qui permet de traiter les requêtes de manières itérative (en *chunk*).

```
app.get ("proxy", (req, res) => {
    res.splice (req, OutputStreamSpliceFlags.CLOSE_SOURCE)
});
```

GIO fournit un api pour effectuer des traitement asynchrone sur des flux, il sera donc éventuellement possible d'écrire:

```
app.get ("proxy", (req, res) => {
    res.splice_async (req, OutputStreamSpliceFlags.CLOSE_S
});
```

Architecture du framework

Le *framework* est une librairie de haut-niveau qui facilite l'écriture d'une application web.

- routage
  - *matching* de requête
  - règles avec variable typées
  - scope pour encapsuler des routes
- composantes pour uniformiser les services et faciliter les accès au *middleware*:
  - cache: cache associative pour réduire les besoins en calculs
  - session: donnée serveur propre à chaque utilisateur
  - cookies: données stockés sur le client

Exemple d'application web en Valum.

Pourquoi Valum?

Valum est une étape importante dans l'intégration des technologies web en Vala.

- fournit un *middleware* pour construire d'autres *frameworks*
- constitue un *proof of concept*
- réunit un protocole de communication (HTTP), un langage moderne et la possibilité d'écrire des applications de haute-performance
- déjà opérationnel et documenté! valum.readthedocs.org

Pourquoi Valum?

La source principale de goulot d'étranglement (*bottleneck*) en web est la communication réseau.

- une page qui prends ~200µs à calculer est transmise en 4ms depuis mon ordinateur portable
- une des solution est d'effectuer plus de calcul sur le serveur et de minimiser le transfert des données qui nécéssitent un traitement.

Valum permet d'écrire une application serveur qui effectue une grande quantité de calcul via un langage de haut niveau.

- une application peut passer plus de temps à effectuer des calculs, ce qui permet d'implanter des services web plus coûteux en ressources

Objectifs pour Valum

0.1-beta

- templating CTPL
- tests unitaires
- compiler avec `--enable-experimental-nonnull`

0.1

- spécifications finale du *middleware* (VSGI)
- soumettre le *framework* au Web Framework Benchmarks de TechEmpower

Objectifs pour Valum

0.2

- traitement asynchrone des requêtes
- optimisations (exploiter le traitement asynchrone)
- métriques de performance

0.3

- composantes de haut-niveau
  - cache
  - session
  - authentification

Application

Application web traditionnelles

- codage d'une section critique d'un service web
- les ressources statiques (CSS, JavaScript, templates) peuvent être bundlés dans l'exécutable avec GLib.Resource.

Calcul distribué avec GSL

- librarie pour faire du traitement numérique en C/C++
- les *bindings* existent déjà pour Vala.
- chaque noeud transmet en continu une charge de travail et renvoit le résultat du calcul de la même manière.
- traitement de données asynchrone: reçues et transmises au fur et à mesure sans bloquer le CPU lors de l'attente.