L'adaptation d'Antidote pour le Web

Gabriel Aubut-Lussier

Gabriel Aubut-Lussier

Druide

L'adaptation d'Antidote pour le Web (1/n)

Avertissement

- La présentation qui va suivre est inspirée de la réalité
- Le code et les concepts qui seront présentés peuvent être :
  - simplifiés par rapport à la réalité
  - modernisés par rapport à la réalité

Postulats

Définitions de ***postulat***, nom masculin

Masquer les traductions ⇄

- MATHÉMATIQUES, LOGIQUE – Principe que l'on demande d'admettre comme vrai sans démonstration.

  ⇄ FORMAL *postulate*

- RELIGION – Période qui précède le noviciat dans un monastère ou une communauté religieuse.

  ⇄ *postulate* […]

Précisions

De *postulatum*; du latin *postulatum*, 'demande'… »

IXe — XXIe s.
XVIIIe s.

0 — 100
47 (fréquent)

Afficher l'API

m. s. un postulat [pòstylà]

Variantes régionales

	postulat
QUÉBEC	[pòstylà]
FRANCE	[pòstyla]

[a] amour, bateau, lac
[à] † pâte, rat, là-bas

† *Surtout au Québec.*

Postulats

- L'état de l'application n'est pas connu côté serveur
- La couche linguistique n'est pas connue côté client
- JSON est un format adéquat pour échanger des données
- La librairie RapidJSON est adéquate pour traiter du JSON
- La librairie Beast est adéquate pour un serveur HTTP
    - Mise à jour par rapport à ma présentation de mars 2017

Sommaire

1. REST
2. Multifil
3. Tests asynchrones
4. Conclusion
5. Bonus : Anecdotes de problèmes en production
6. Bonus : États globaux

1. REST

REST

- Il faut concevoir une interface REST composée de plusieurs *routes* (déterminées par le *path* d'un URL)
- Les intrants d'une *route* proviennent de plusieurs *sources*
- Une *route* produit, à titre d'extrant, une *réponse* HTTP
- Chaque *route* est associée à un *gestionnaire*; c'est-à-dire : une fonction

Intrants

- Représentent un danger; tout doit être validé
- Différentes sources d'intrants :
  - Le verbe HTTP (GET, POST, etc.)
  - À même la *route* (ex. : GET /utilisateur/42)
  - Le *query string* (ex. : GET /search?q=isocpp)
  - Le *body* de la requête (ex. : POST /submit)
  - L'entête HTTP *Cookie* (ex. : Cookie: a=b)
  - Un entête HTTP quelconque (ex. : Referrer: https://isocpp.org)

Extrant

- La *réponse* HTTP est composée :
  - du code de *statut* HTTP
  - d'*entêtes* HTTP
  - d'un *body* qui est optionnel

Gestionnaire

- Implémenté avec une *fonction*
- Peut consommer tous les *intrants* fournis par la *requête*
- Doit produire une *réponse*
- N'est jamais invoqué s'il y a un problème de validation

Signatures

```
template <typename T,
          typename Req,
          typename R,
          typename ... Params>
concept Gestionnaire = Extrant<R>
                       && Intrant<Params, Req>...
                       && std::is_invocable_r<
                            R,
                            T,
                            Req
                          >::value;

template <typename T>
concept Extrant = Serialiseur<T::serialiseur_type,
                              T::value_type>;

template <typename T, typename Req>
concept Intrant = Validateur<T::validateur_type, T::value_type>
                  && Source<T::source_type, T::key_type, Req>;
```

Signatures

```
template <typename T, typename V>
concept Validateur = std::is_invocable_r<std::optional<V>,
                                         T,
                                         std::string
                                        >::value;

template <typename T, typename Key, typename Request>
concept Source = std::is_invocable_r<std::string,
                                     T,
                                     Key,
                                     Request
                                    >::value;
```

Avantages

- La structure du JSON contenu dans une *réponse* est maintenant typée et vérifiée par le compilateur
- Impossible de faire varier la structure de la *réponse* pour deux invocations différentes du même *gestionnaire*
- Chaque paramètre doit être validé
- Les validations ne font plus partie de la logique pour gérer les requêtes valides

En résumé

- Le système de type de C++ peut être utilisé afin d'imposer des principes de sécurité informatique
- Un routeur de requêtes d'API REST n'est pas suffisant : la validation des intrants doit être au coeur du design

2. Multifil

Multifil

- Ne pas sous-estimer le problème
- Il n'est pas suffisant de croire qu'on a la bonne solution
- Chaque bogue lié au multifil coûte cher à analyser
- Chaque patch complexifie le problème

Catégories de types

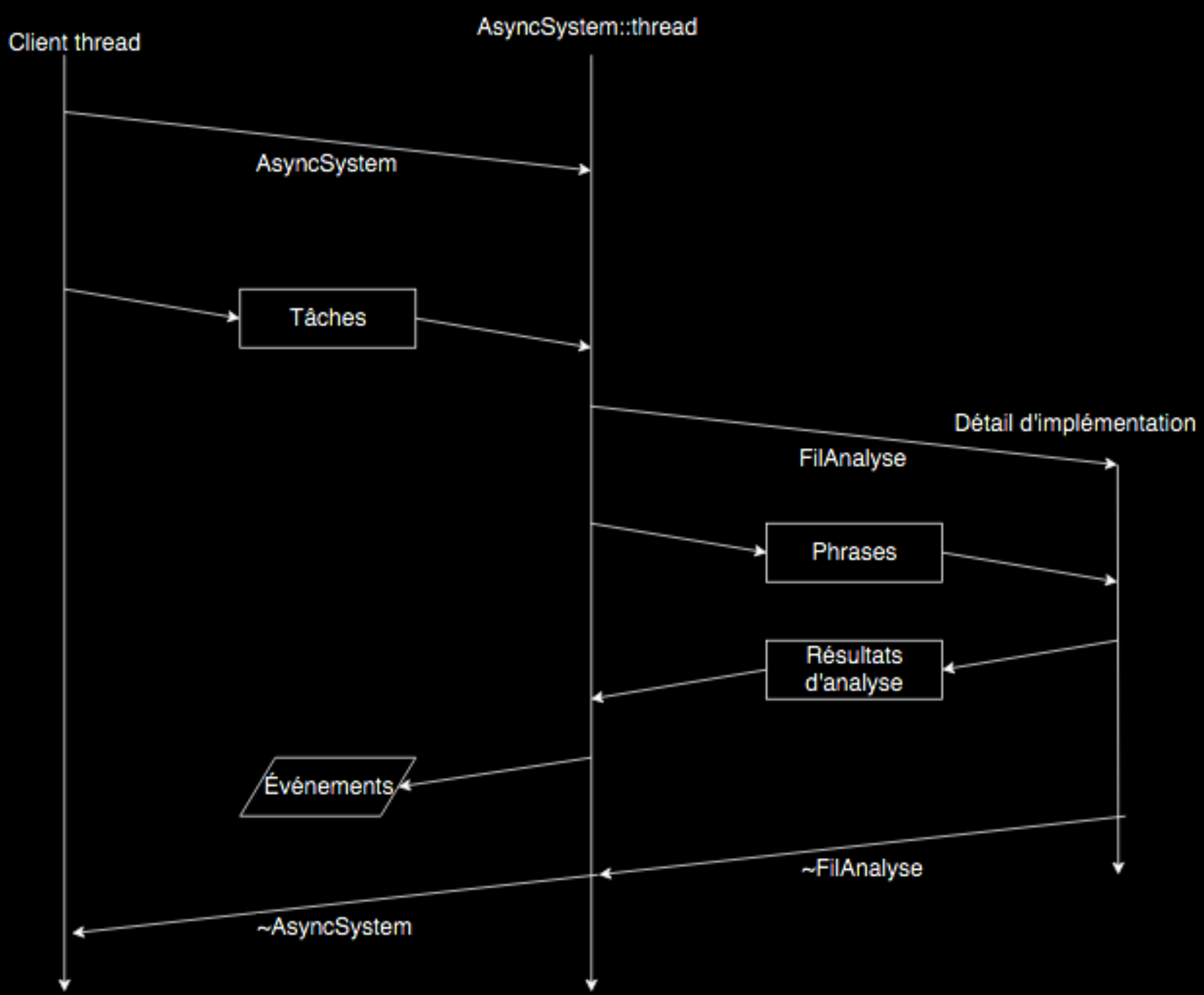
- Thread-safe
- Thread-compatible (*const* signifie aussi race-free)
- Thread-unsafe

Outils

- ThreadGuard<T>
- boost::synchronized_value<T>
- SynchronizedConditionVariable<T>
- etc.

Données partagées

- Identifier et circonscrire les données partagées
- Les encapsuler avec le bon outil
- Minimiser la quantité de données partagées
- Minimiser le temps passé à manipuler les données partagées


Client thread
AsyncSystem::thread
AsyncSystem
Tâches
Détail d'implémentation
FilAnalyse
Phrases
Résultats
d'analyse
Événements
~FilAnalyse
~AsyncSystem

Données partagées

- Identifier et circonscrire les données partagées
- Les encapsuler avec le bon outil
- Minimiser la quantité de données partagées
- Minimiser le temps passé à manipuler les données partagées

En résumé

- Utiliser des primitives de synchronisation existantes pour :
  - prévenir les erreurs
  - communiquer l'intention
- Concevoir des types Thread-compatible
- Modéliser et minimiser les données partagées

4. Conclusion

Conclusion

- 1. REST
  - Imposer des principes de sécurité via le système de type
  - Mettre la validation des intrants au coeur du design
- 2. Multifil
  - Utiliser des primitives de synchronisation existantes
  - Concevoir des types Thread-compatible
  - Modéliser et minimiser les données partagées

Merci !

- Questions?
- Commentaires!
- Bonus…