

# Annotation pour les g33ks

THE ENTERPRISE SOCIAL PLATFORM

Julien Viet

FEBRUARY 2013



# JULIEN VIET

- Contact
  - [julien@julienviet.com](mailto:julien@julienviet.com)
  - @julienviet
  - <http://github.com/vietj>
- Open source depuis + de 10 ans (déjà)
  - Avec l'opportunité de pouvoir en vivre
    - 2003 → 2008 JBoss
    - 2008 → 2013 Exoplatform
  - Mission officielle: le portail pour Java EE
    - Le portail pour Java EE (officiel)
- Marsjug Leader

# MISSION OFFICIEUSE

- CRaSH : le shell pour la JVM → Tools In Action à Devvxx !!!!
- Wikbook : écrire de la doc au format wiki pour docbook dans des projets Java en incluant du source code
- Juzu : que nous allons voir

# EXOPLATFORM

- The Enterprise Social Platform
  - Social collaboration Solution
  - Open Source and Enterprise Ready
  - Highly extensible platform
- Notre blog
  - <http://blog.exoplatform.com>

# FEATURES



**All in a Single Platform**

# INTRO A JUZU

# JUZU

- Framework MVC
  - Apprentissage rapide
  - Simplicité et liberté
- Plusieurs runtimes
  - Servlet
  - Portlet
  - Vert.x (poc)
- Motivation
  - Besoin d'un framework simple et puissant pour portlet/servlet

The screenshot shows an IDE interface with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'myapp' with a structure including '.idea', 'src', 'main', 'java', 'org.example', and 'templates'. The 'templates' folder contains 'index.html', 'Controller', and 'package-info.java'. The code editor displays the contents of 'package-info.java', 'Controller.java', and 'index.html'. The 'package-info.java' file contains the following code:

```
@juzu.Application
@juzu.plugin.servlet.Servlet(value = "/")
package org.example;
```

The 'Controller.java' file contains the following code:

```
public class Controller {
    @Inject
    @juzu.Path("index.html")
    Template index;

    @juzu.View @Route("/")
    public Response index() throws IOException {
        return index.ok();
    }
}
```

The 'index.html' file contains the following text:

```
Hello World
```

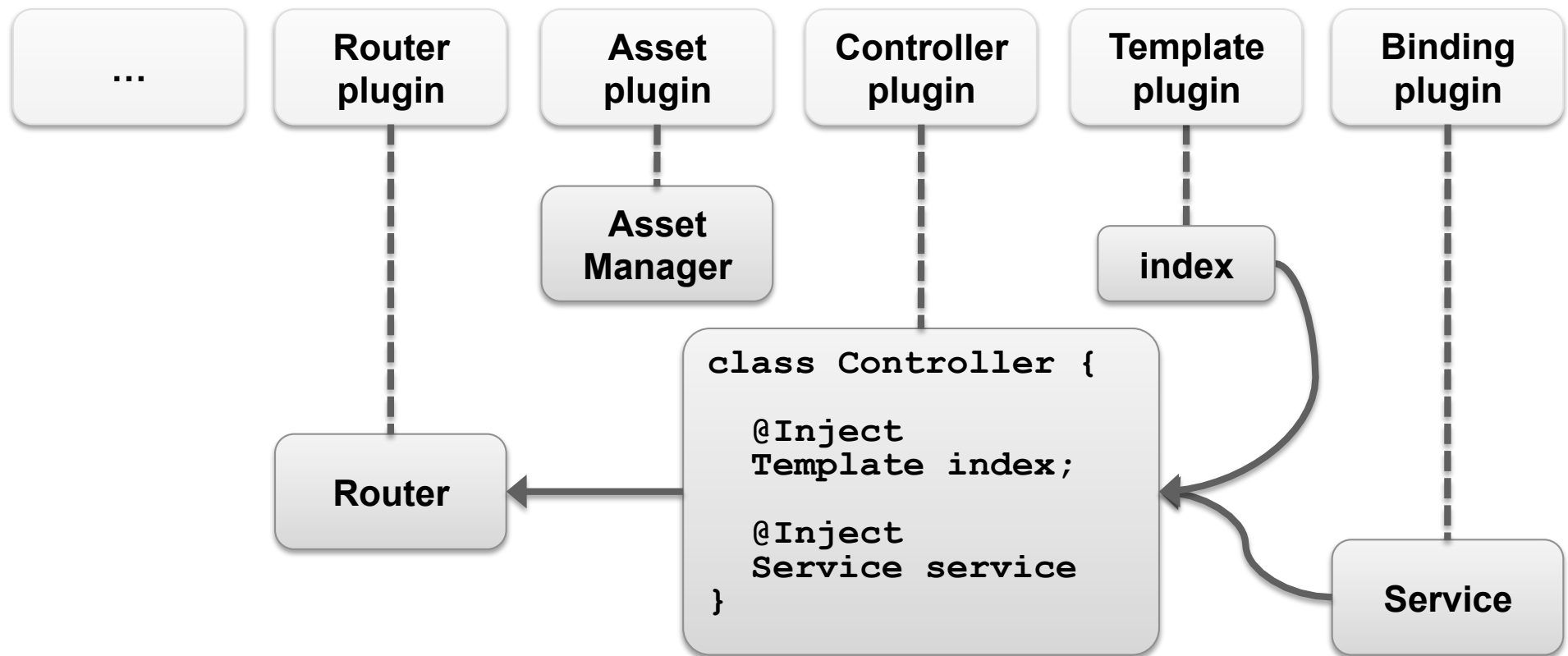


# USINE A PLUGIN

- A la compilation
  - Generer de la configuration, du code source, des ressources
  - Faire échouer la compilation
- A l'execution
  - Interception des requêtes
  - Déclarer des beans dans le container d'injection

# INJECTION: JSR-330

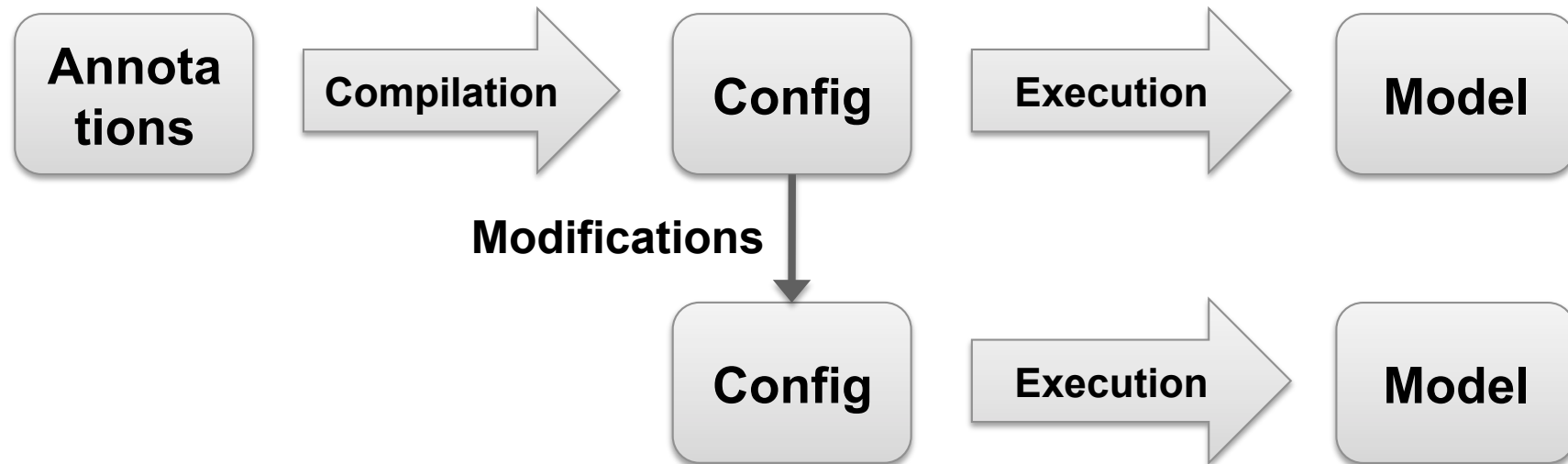
- Supporte Spring, Guice 3.0 ou CDI



# GESTION DE LA CONFIGURATION

Notre approche: générer la configuration à partir des annotations

- Les annotations sont puissantes
- On peut lire la configuration effective
- Et la surcharger!



# TRAITEMENT D'ANNOTATION

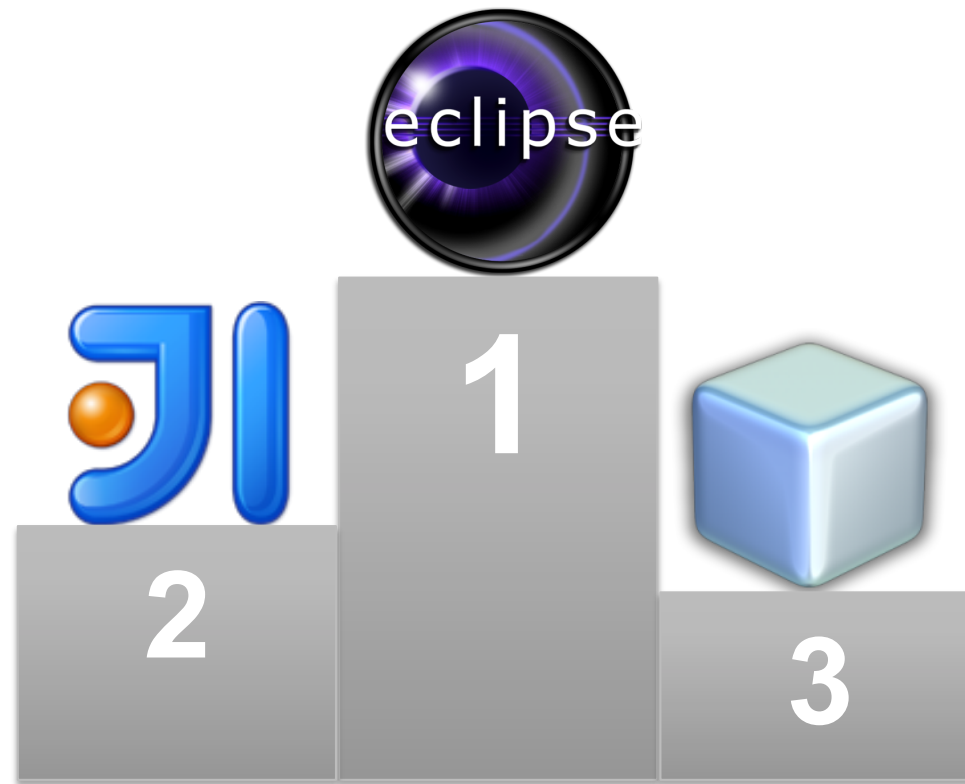
# **MAIS AVANT UN PETIT SONDAGE**

# LES OUTILS

# SUPPORT DES OUTILS

- Basé sur des standards depuis J2SE 6
  - javax.annotation : Pluggable Annotation Processing API (JSR 269)
  - javax.lang.model
  - javax.tools
- Deux compilateurs
  - Oracle : javac
  - Eclipse : ecj
- Et les IDEs ?

# ET LES IDEES ?








**AVANT TOUT, QUELQUES  
BONNES PRATIQUES 😊**

# LOGGING

- Comprendre ce qu'il se passe chez l'utilisateur
- SOURCE\_OUTPUT/my.log
  - Facile à trouver
  - Séparé des classes
- Attention aux compilation successives! → append

# REPORTER UNE ERREUR

- Messenger
  -  WARNING / MANDATORY\_WARNING
  -  ERROR: fait échouer la compilation
  -  NOTE
  - ? OTHER
- Avec un élément ou une annotation pour contextualiser l'erreur

# GESTION DE VOS BUGS

```
try {  
    doProcess(annotations, roundEnv);  
} catch (Exception e) {  
    processingEnv.getMessager().printMessage(  
        Diagnostic.Kind.ERROR,  
        e.getMessage());  
}
```

- **Sinon erreur compilateur (selon les versions)**

# GENERATION

- Indiquer un element ou plusieurs elements origines lors de la creation d'un fichier
  - Etabli une relation de dépendance entre l'origine et le fichier
  - Utile à l'IDE pour supprimer les fichiers en cascade

# GENERATION

# GENERATION

- De quoi?
  - Une source java qui sera compilé
  - Une classe java (bytecode)
  - Une resource
- Et où?
  - CLASS\_OUTPUT
  - SOURCE\_OUTPUT

# CYCLE DE VIE

- Maximum une fois par round → bufferiser les fichiers et les écrire à la fin du round
- Ecrase le fichier existant issu d'un round précédent ou d'une précédente compilation
- Invoker `close ()` sur le stream



# GENERATION DE SOURCE JAVA

- Avant le dernier round
- Annoter le source code généré avec `javax.annotation.Generated`
- Indiquer les éléments à l'origine du fichier source crée
- ⓘ votre code peut utiliser le code généré

# DANS JUZU: TYPE SAFE URL

```
public class Controller {  
  
    @View @Route("/{id}")  
    public Response show(String id) {  
        return Response.ok("<a href='" +  
            Controller_.update(id) + "'>update</a>");  
    }  
  
    @Action @Route("/{id}")  
    public void update(String id) {  
        // Do the update  
    }  
}
```

# GENERATION DE RESOURCE

```
filer.createResource(  
    JavaFileManager.Location location,  
    CharSequence pkg,  
    CharSequence relativeName,  
    Element... originatingElements)
```

- **①**écriture possible dans META-INF
  - `filer.createResource(CLASS_OUTPUT, "", "META-INF/resource.txt")`

# DANS JUZU: COMPILATION *LESS*

```
@Application
@Less (
    "assets/bootstrap.less",
    minify = true)
package my.app;
```

# AUTRES CAS D'UTILISATION

# AUTRES CAS D'UTILISATION

- *Etendre* une classe
  - Par une super classe
  - Par une sous classe → requiert une factory
  - Cas d'utilisation
    - static proxy
    - generation de JavaBean
- Classe compagnon
  - Foo → Foo\_
  - Cas d'utilisation
    - Generateur de builder pattern
    - JPA meta model

# AUTRES CAS D'UTILISATION

- Remplacer le scan d'annotations par un descripteur centralisé
- Etc...

# FRAMEWORKS BASÉS SUR APT

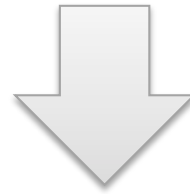
- AndroidAnnotations
- Dagger : l'IOC de Square
- APTVir : un virus APT (sur mon GitHub)
- Type safe queries
  - JPA2 MetaModel
  - QueryDSL
- JBoss Logging Tooling: logger typé
- Storm-gen: generateur de DAO



# ANDROID ANNOTATIONS

```
@Background
```

```
void searchAsync(String searchString) {  
    Bookmarks bm = client.getBookmarks(searchString);  
    updateBookmarks(bm);  
}
```



```
@Override
```

```
void searchAsync(String searchString) {  
    backgroundExecutor.execute(  
        () -> { super.searchAsync(searchString); }  
    );  
}
```

# L'ARBRE SYNTAXIQUE

- L'arme absolue
- Complicé et non portable
- Pour javac deux niveaux
  - Compiler Tree API (javac) : lecture seule
  - Implementation de la Tree API : écriture
- Utilisé par Lombok

# COMPILATION INCREMENTALE

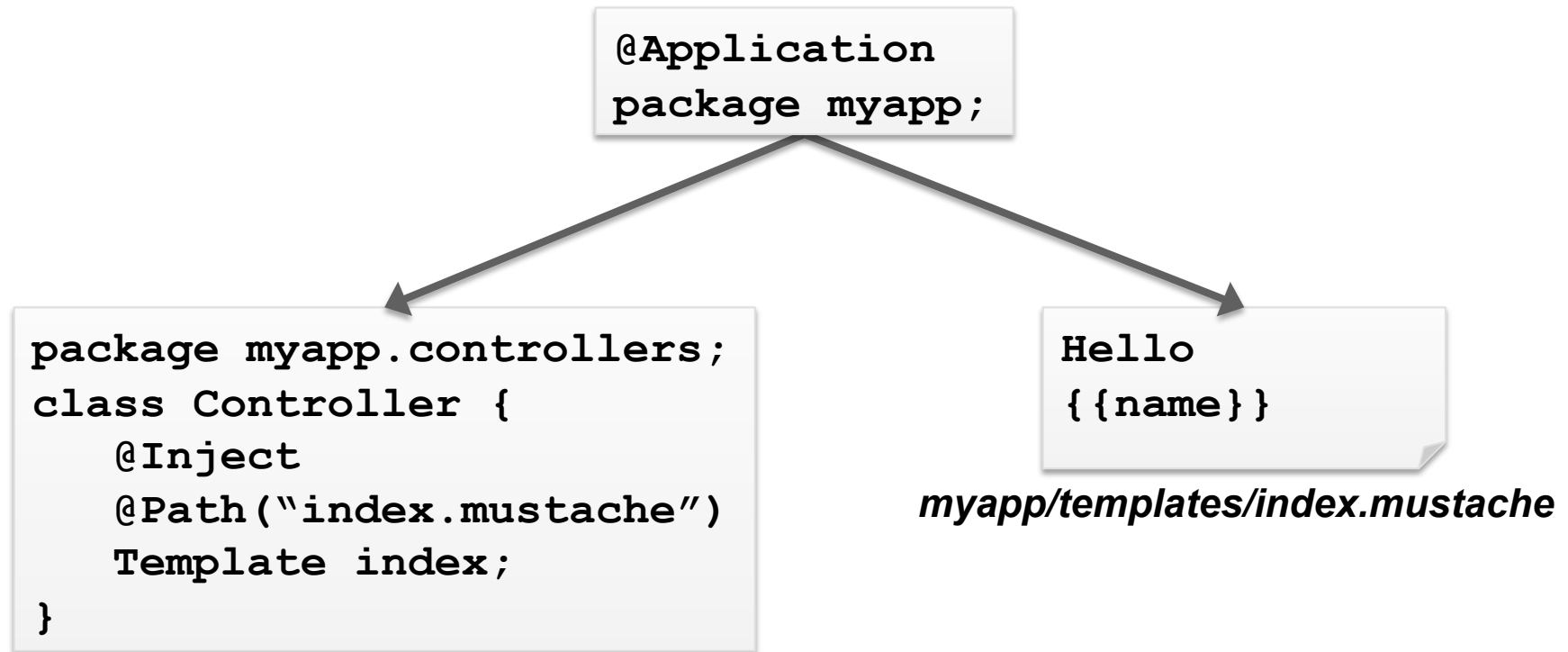
# COMPILATION INCREMENTALE

- Dans Eclipse la compilation est incrémentale
- Ergonomie accrue pour l'utilisateur
- Tous les unités de compilation (fichiers) ne sont pas disponibles en même temps

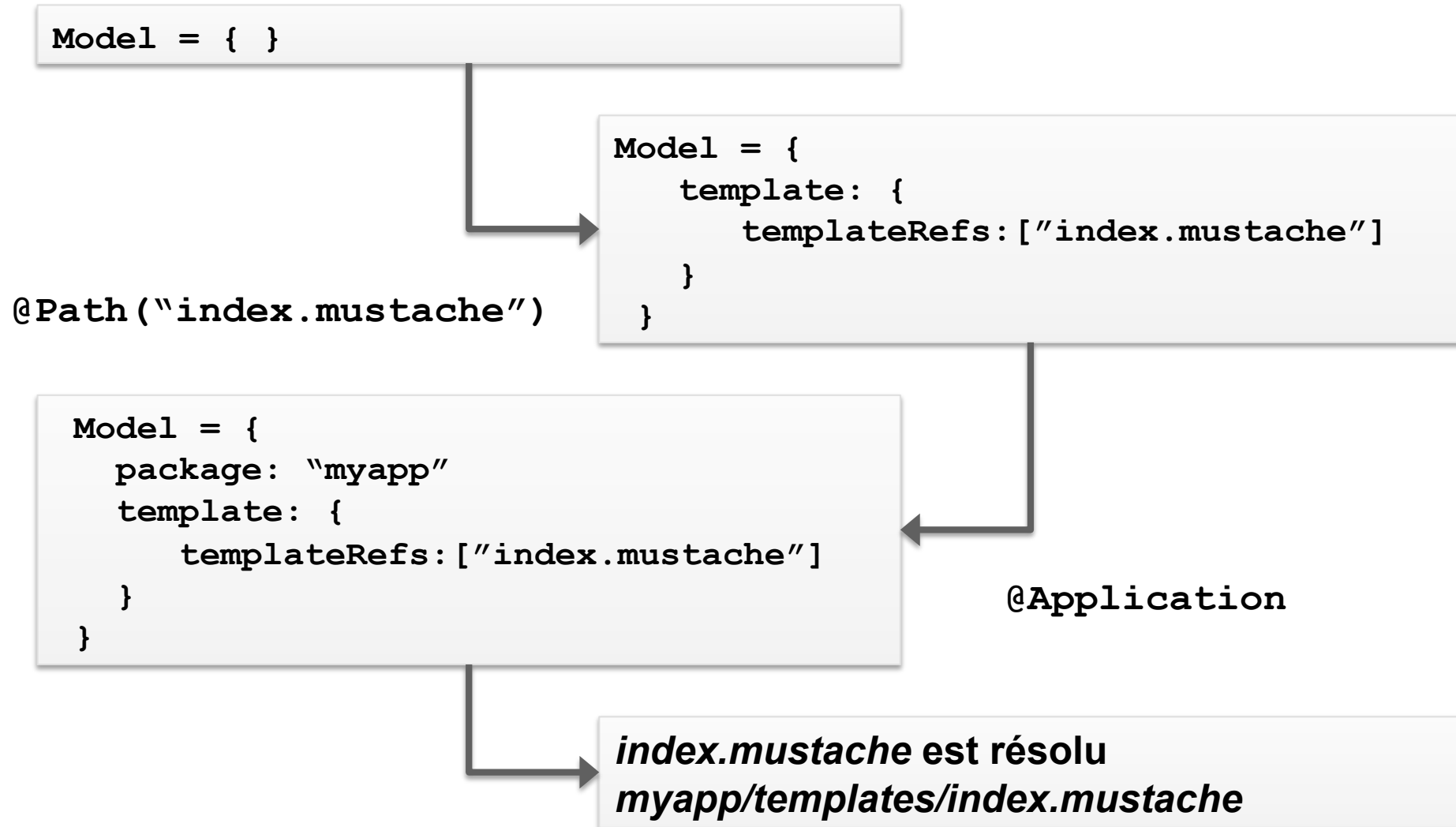
# DESIGN DU PROCESSEUR

- Peut devoir être adapté quand il existe des relations entre les éléments traités
- Regardons ensemble un exemple...

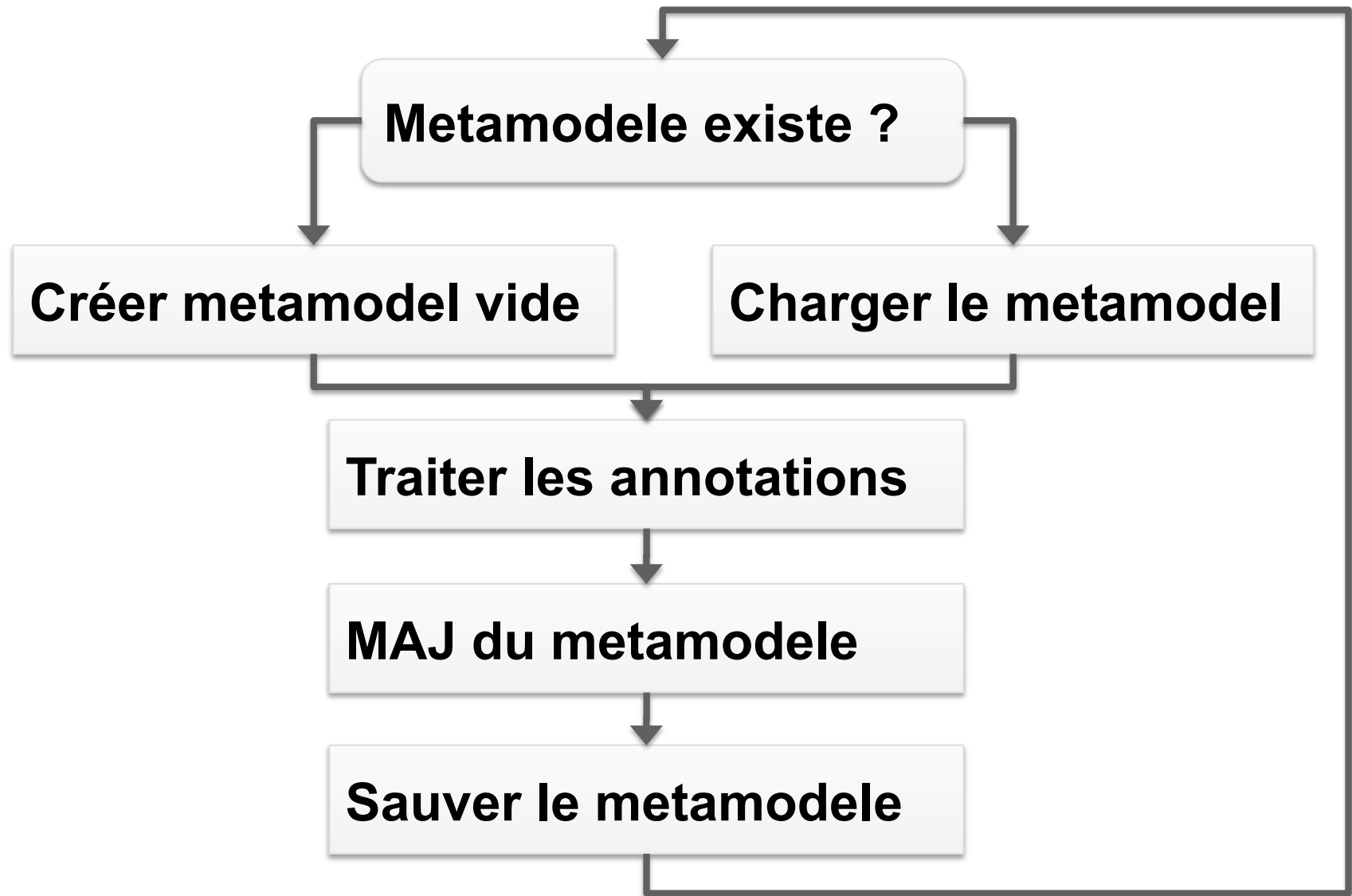
# EXAMPLE



# LE METAMODELE



# CYCLE D'UNE COMPILATION





# MAIS AU FAIT

- Pourquoi analyser un template à la compilation ?
  - Vérifier les erreurs
  - Précalculer
  - Pour Juzu : générer une sous classe pour l'injection de dépendance

```
@Path("index.gtpl")  
Public class index extends  
juzu.Template {  
}
```

- Question bonus : quid de @Path ?

# TESTER SON PROCESSEUR

# AVEC L'API JAVAX.TOOLS

1

**Obtenir le compilateur et file manager**

```
ToolProvider.getSystemJavaCompiler()  
compiler.getStandardFileManager(...)
```

2

**Configurer le file manager**

```
mgr.setLocation(SOURCE_OUTPUT, sourceOut);  
mgr.setLocation(CLASS_OUTPUT, classOut);
```

3

**Créer et configurer une tâche de compilation**

```
CompilationTask task = compiler.getTask(...)  
task.setProcessors(processors)
```

4

**Compiler**

```
assertEquals(Boolean.TRUE, task.call());
```

# VERIFICATION D'UN ECHEC

```
interface Diagnostic<S> {  
    Kind getKind();  
    S getSource();  
    long getPosition();  
    long getStartPosition();  
    long getEndPosition();  
    long getLineNumber();  
    long getColumnNumber();  
    String getMessage(Locale locale);  
}
```

# COMPILATION INCRÉMENTALE

- Abordable mais un peu plus complexe
- Simulation d'un environnement incrémental
- Exemple
  1. Compiler Foo.java → Foo.class
  2. Supprimer Foo.java et déplacer Foo.class dans le classpath
  3. Compiler Bar.java

**WHAT ELSE ?**

# SUPPRESSION DE FICHER

- Suppression de fichier
  - Serait utile pour l'incremental
  - Fichier Java: utiliser les dépendances
  - Fichier resource: opération existe mais non supporté Eclipse
- Workaround
  - Remplacer par un fichier vide...

# ACCESS AU SOURCE

- Noms des paramètres de méthode
  - Sinon mode debug
  - Ou Java 8
- JavaDoc
  - **Elements#getDocComment (Element)**
  - Contenu brut



# MESSAGER

- Eclipse `Message#printMessage`
  - Sans element → NPE interne
  - Inopérant pour un package → dirty hack

# SUPPORT DU SOURCE\_PATH

- Indispensable pour lire les ressources
- Javac
  - Doit être fourni par l'environnement
  - Mais possible à partir d'un élément avec Compiler Tree API
- Eclipse
  - Non fourni (bug)
  - Peut être obtenu avec l'API interne

# LES RESSOURCES

- Javac

- getResource pour lire et createResource pour écrire

- ECJ

- Même resource pour lire / écrire → cacher la resource

# COMPLETION

- Pour les IDE, semble supporté uniquement par Netbeans

# MA CONCLUSION Q&A

# MA CONCLUSION

- Ecrire un processeur est simple
- Mais peut devenir compliqué
- Très bon support du compilateur
- Support des IDE à améliorer
  - Eclipse sauve les honneurs
  - IntelliJ et netbeans mention passable