Architecture Modulaire avec Spring Modulith: Théorie et Implémentation

Votre Nom

21 juin 2025

Table des matières

Résumé

Ce rapport approfondi présente l'architecture modulaire avec Spring Modulith en se focalisant sur ses annotations clés, la gestion des événements, et le rôle du module commun. Nous expliquons en détail le rôle du fichier `package-info.java` et démontrons l'implémentation complète d'une application e-commerce avec trois modules métiers.

1 Annotations Fondamentales de Spring Modulith

Spring Modulith introduit des annotations spécifiques pour structurer et contrôler l'architecture modulaire :

1.1 @Module

Rôle : Définit un module métier dans l'application
Emplacement : Dans `package-info.java` du module
Paramètres :

- `displayName` : Nom convivial du module
- `allowedDependencies` : Modules autorisés à dépendre de ce module

```
@org.springframework.modulith.Module(
  displayName = "Module Commande",
  allowedDependencies = {"acheteur", "produit"}
)
package com.example.commande;
```

1.2 @ApplicationModuleListener

Rôle : Marque une méthode comme écouteur d'événements inter-modules
Comportement :

- Exécution asynchrone après commit transaction
- Gestion automatique des erreurs
- Découplage complet entre modules

```
@ApplicationModuleListener
public void creerCommande(ProduitAcheteEvent event) {
  // Logique de cr ation de commande
}
```

1.3 @ApplicationModuleTest

Rôle : Permet de tester un module en isolation
Avantages :

- Vérifie les dépendances du module
- Teste le comportement sans démarrer toute l'application

```java
@ApplicationModuleTest
class CommandeModuleTests {
  // Tests sp cifiques au module
}
```

2 Rôle Crucial du package-info.java

Le fichier `package-info.java` est fondamental dans chaque module :

2.1 Structure Typique

```java
/**
 * Module de gestion des commandes
 */
@Module(
  displayName = "Commande",
  allowedDependencies = {"acheteur", "produit"}
)
package com.example.commande;

import org.springframework.modulith.Module;
```

2.2 Fonctions Clés

1. **Déclaration du Module** : Identifie le package comme un module
2. **Contrôle des Dépendances** : Spécifie quels modules peuvent être dépendus
3. **Documentation** : Fournit une description du module
4. **Vérification Architecturale** : Utilisé par Spring Modulith pour valider la structure

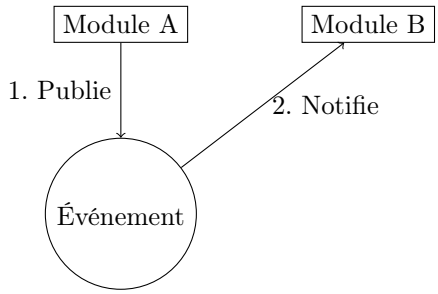
3 Gestion des Événements dans Spring Modulith

3.1 Principe Fondamental

Les événements permettent une communication **découplée** entre modules :
— Émetteur ne connaît pas les consommateurs
— Consommateurs s'abonnent aux événements pertinents

3.2 Workflow Typique

1. Un service publie un événement métier
2. Les écouteurs concernés sont notifiés
3. Traitement asynchrone après commit transaction



3.3 Comparaison : Événements Spring vs Broker Externe

Caractéristique	Événements Spring Modulith	Broker Externe (Kafka/RabbitMQ)
Portée	Application monolithique	Systèmes distribués
Latence	Quasi-instantanée	Dépend du réseau
Fiabilité	Transaction locale	Persistée (durée)
Complexité	Faible	Élevée (infra requise)
Cas d'usage	Communication inter-modules	Communication inter-services

4 Module Commun : Rôle et Implémentation

4.1 Rôle dans l'Architecture

Le module commun fournit des fonctionnalités transversales :
— DTOs partagés entre modules
— Exceptions communes
— Utilitaires génériques
— Configurations centralisées

4.2 Structure Typique

```
src/main/java/
          common
              dto
                    ApiResponse.java
              exception
                    GlobalExceptionHandler.java
              util
                    DateUtils.java
              config
                  WebConfig.java
```

4.3 Contraintes Importantes

— **Ne doit contenir aucune logique métier**

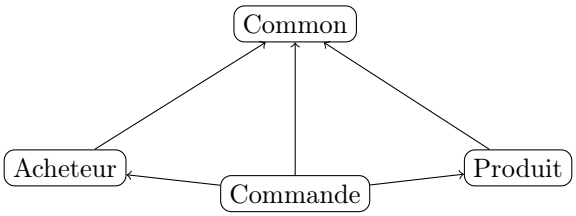
— **Tous les modules peuvent en dépendre**
— **Éviter les dépendances cycliques**

5 Implémentation Complète : E-commerce Modulaire

5.1 Architecture des Modules

```
src/main/java/
          common          // Module commun
          acheteur        // Module m tier
          produit         // Module m tier
          commande        // Module m tier
```

5.2 Dépendances entre Modules



5.3 Séquence Complète : Achat de Produit

1. Client appelle `POST /acheteurs/123/achat` avec ID produit
2. `AcheteurController` valide la requête
3. `AcheteurService` exécute la logique métier
4. Publication d'un `ProduitAcheteEvent`
5. `CommandeService` écoute l'événement
6. Création d'une nouvelle commande
7. Envoi d'une notification (optionnel)

```
// Dans AcheteurService
public void acheterProduit(Long acheteurId, Long produitId) {
  // Validation m tier...
  events.publishEvent(new ProduitAcheteEvent(acheteurId, produitId)
    );
}

// Dans CommandeService
@ApplicationModuleListener
public void creerCommande(ProduitAcheteEvent event) {
  Commande commande = new Commande();
  commande.setAcheteurId(event.acheteurId());
  commande.setProduitId(event.produitId());
```

```
13    commandeRepository.save(commande);
14
15    // Optionnel: Publier un autre    vnement
16    events.publishEvent(new CommandeCreeEvent(commande.getId()));
17  }
```

6 Avantages de l'Approche Modulaire

6.1 Avantages Techniques

Avantage	**Impact**
Maintenabilité	Code organisé par domaine métier
Évolutivité	Modules indépendants modifiables séparément
Testabilité	Tests isolés par module
Documentation auto-générée	Visualisation des dépendances
Transition microservices	Modules prêts à être extraits

6.2 Comparaison Architecturale

Critère	**Monolithe Traditionnel**	**Modular Monolith**
Structure	Couplage fort entre composants	Modules découplés
Évolutivité	Difficile	Facile (par module)
Nouveaux développeurs	Courbe d'apprentissage raide	Compréhension facilitée
Déploiement	Unique	Unique mais structuré

Table 1 – Comparaison architecturale

7 Conclusion

Spring Modulith représente une évolution majeure dans la conception des applications monolithiques. Par son système d'annotations, sa gestion des événements et sa structuration modulaire :

— Il apporte la rigueur du DDD dans les monolithes
— Il offre une alternative pragmatique aux microservices
— Il prépare la migration future vers l'architecture distribuée

Le module commun joue un rôle crucial en centralisant les éléments transversaux tout en maintenant la séparation des préoccupations. Cette approche est particulièrement adaptée pour des applications métier complexes où la simplicité opérationnelle est primordiale.

Références

[1] Documentation officielle Spring Modulith : `https://spring.io/projects/spring-modulith`

[2] Vernon, V. (2013). *Implementing Domain-Driven Design*. Addison-Wesley.

[3] Kamil Grzybek (2020). *Modular Monolith : A Primer*. `https://www.kamilgrzybek.com/blog/modular-monolith-primer/`