EntityManager

L'`EntityManager` est au cœur de Doctrine, agissant comme une façade qui fournit une API haut niveau pour accéder à diverses opérations de base de données, telles que la création, la lecture, la mise à jour et la suppression (CRUD) d'entités.

Voici quelques points clés pour comprendre l'`EntityManager` de Doctrine :

1. Unité de travail

- L'`EntityManager` garde une trace des objets que vous travaillez pendant un cycle de vie de requête.
- Il s'assure que toutes les modifications apportées aux entités soient synchronisées avec la base de données.

2. Gestion des entités

- Vous utilisez l'`EntityManager` pour gérer (sauvegarder, supprimer, rechercher) les entités.
- Par exemple, pour sauvegarder une entité, vous travaillerez avec une méthode comme `persist($entity)` suivie de `flush()`.

3. Le Cycle de Vie

- L'`EntityManager` maintient un contexte de persistance. Il sait quelles entités ont été récupérées et gère leur état.
- Les méthodes telles que `persist()`, `remove()`, et `flush()` sont utilisées pour gérer le cycle de vie des entités.

4. Référentiel

- À l'aide de l'`EntityManager`, vous pouvez accéder aux référentiels qui fournissent des méthodes pour interagir avec une source de données spécifique.
- Les référentiels peuvent être utilisés pour définir des requêtes spécifiques à une entité.

5. Transactions

- L'`EntityManager` gère aussi les transactions, vous permettant de regrouper plusieurs opérations en une seule transaction pour maintenir l'intégrité de la base de données.

L'`EntityManager` dans Doctrine joue un rôle central dans la gestion de la persistance des données. Il agit comme un médiateur entre vos objets en PHP et la base de données. L'une des principales fonctionnalités de l'`EntityManager` est de maintenir un "contexte de persistance". Voici ce que cela signifie :

1. Suivi des Entités

L'`EntityManager` garde une trace des entités que vous avez récupérées de la base de données. Cela signifie qu'il sait quels objets sont actuellement en mémoire et quels changements ont été apportés à ces objets.

2. Gestion des États des Entités

Les entités peuvent avoir différents états dans le contexte de persistance :

- **Managed:** L'entité est suivie par l'`EntityManager`. Tout changement apporté à l'entité sera automatiquement synchronisé avec la base de données lors de l'exécution de `$entityManager->flush()`.
- **Detached:** L'entité n'est pas suivie par l'`EntityManager`. Les modifications apportées à l'entité ne seront pas synchronisées avec la base de données sauf si l'entité est à nouveau associée à l'`EntityManager`.
- **Removed:** L'entité est marquée pour être supprimée de la base de données lors de l'exécution de `$entityManager->flush()`.

3. Flushing

- Quand vous appelez `$entityManager->flush()`, il vérifie toutes les entités dans le contexte de persistance, et toutes les modifications apportées à ces entités (insertion, mise à jour, suppression) sont alors synchronisées avec la base de données.

4. Clearing the EntityManager

- Vous pouvez également "vider" l'`EntityManager` avec `$entityManager->clear()`. Cela détache toutes les entités de l'`EntityManager`, ce qui signifie qu'il arrête de gérer tous les objets.

Exemple :

Considérez ce code simple :

```
$user = $entityManager->find('User', $userId); // Récupérer un utilisateur de la DB

$user->setName('New Name'); // Modifier le nom de l'utilisateur



$entityManager->flush(); // Synchroniser les changements avec la DB
```

Dans cet exemple :

- Lorsque vous récupérez un utilisateur de la base de données, l'`EntityManager` commence à suivre cette entité. Il est maintenant dans un état "managed".
- Vous modifiez ensuite l'entité en mémoire.
- Lorsque `$entityManager->flush()` est appelé, l'`EntityManager` voit que l'entité `User` a été modifiée, et il synchronise ce changement avec la base de données, c'est-à-dire qu'il génère et exécute une requête SQL `UPDATE`.

Conclusion

Le "contexte de persistance" de l'`EntityManager` est comme une session de travail qui garde une trace de vos objets et de leurs changements. Il s'assure que tous les changements sont correctement synchronisés avec la base de données, vous permettant de travailler avec vos objets en PHP de manière plus intuitive et orientée objet.

Une entité est à l'état "**removed**" quand elle est marquée pour suppression de la base de données mais n'a pas encore été effectivement supprimée. Cela se produit lorsque vous utilisez la méthode `remove()` de l'`EntityManager` pour marquer une entité pour suppression, mais que la méthode `flush()` n'a pas encore été appelée pour exécuter la suppression effective dans la base de données.

Voici comment cela fonctionne avec un exemple :

Exemple :

Imaginons que nous ayons une entité `Post` représentant des articles de blog dans une application.

```
$post = $entityManager->find('Post', $postId); // Récupérer un post de la DB


if ($post) {
   $entityManager->remove($post); // Marquer le post pour suppression


   // À ce stade, le post est marqué comme "removed" mais il est toujours dans la DB
   // L'EntityManager sait qu'il doit être supprimé lors du prochain flush()


   $entityManager->flush(); // Exécuter la suppression effective du post dans la DB
}
```

Explication

1. **Récupération de l'Entité :**
   - Nous récupérons un post de la base de données avec la méthode `find()`. L'entité `Post` est maintenant dans un état "managed" car elle est suivie par l'`EntityManager`.
2. **Marquer pour Suppression :**
   - Nous marquons l'entité pour suppression avec la méthode `remove()`. L'entité passe à l'état "removed", mais elle est toujours physiquement présente dans la base de données.
3. **Exécution de la Suppression :**
   - En appelant la méthode `flush()`, nous disons à l'`EntityManager` de synchroniser l'état des entités avec la base de données. Puisque l'entité `Post` est marquée comme "removed", une requête de suppression SQL est générée et exécutée, et l'entité est alors effectivement supprimée de la base de données.

Conclusion

Un entité est à l'état "**removed**" lorsqu'elle est marquée pour suppression, en attente d'être effectivement supprimée lors de la prochaine synchronisation avec la base de données par l'appel de la méthode `flush()` de l'`EntityManager`. Cette approche vous permet de gérer la persistance et la suppression de vos entités de manière plus flexible et orientée objet.

Lorsqu'une entité est passée à la méthode `persist()` de l'`EntityManager`, elle est placée dans le contexte de persistance de Doctrine et son état devient "managed". Cela signifie que l'`EntityManager` est maintenant conscient de cette entité et suivra toutes les modifications qui lui sont apportées.

Voici comment cela fonctionne :

1. Création de l'Entité :

Vous créez une nouvelle instance de l'entité et vous la configurez avec les données nécessaires.

```
$user = new User();

$user->setUsername('JohnDoe');
```

2. Persisting l'Entité :

Vous passez l'entité à la méthode `persist()` de l'`EntityManager`.

```
$entityManager->persist($user);
```

3. Flushing l'Entité :

Vous dites à l'`EntityManager` de synchroniser les objets en mémoire avec la base de données.

```
$entityManager->flush();
```

- Lorsque `flush()` est appelé, Doctrine génère et exécute les requêtes SQL nécessaires pour synchroniser les entités avec la base de données.
- Dans ce cas, une requête `INSERT` serait générée pour insérer la nouvelle entité `User` dans la base de données.

Conclusion

En utilisant `persist()`, vous placez l'entité dans le contexte de persistance et son état devient "managed", ce qui signifie que Doctrine est conscient de cette entité et suivra ses modifications jusqu'à ce que `flush()` soit appelé, moment où les modifications seront synchronisées avec la base de données.

Une entité reste dans le contexte de persistance de l'`EntityManager` jusqu'à ce que l'un des événements suivants se produise :

1. Flush puis Clear

- Lorsque vous exécutez la méthode `flush()`, Doctrine synchronise les entités avec la base de données, effectuant les opérations nécessaires (insertion, mise à jour, suppression).
- Après avoir flushé, vous pouvez exécuter la méthode `clear()` pour détacher toutes les entités du contexte de persistance. Ainsi, elles ne seront plus gérées par l'`EntityManager`.

2. Detach

- Vous pouvez spécifiquement détacher une entité du contexte de persistance en utilisant la méthode `detach($entity)` de l'`EntityManager`. Cette action fait que l'entité n'est plus gérée, et aucune modification apportée à l'entité ne sera synchronisée avec la base de données lors du flush.

3. Remove puis Flush

- Si une entité est marquée pour suppression en utilisant la méthode `remove($entity)` et que `flush()` est ensuite appelé, l'entité sera supprimée de la base de données et automatiquement détachée du contexte de persistance.

4. Fin de la Requête / Réponse HTTP (dans une application web)

- Dans le contexte d'une application web, le contexte de persistance est souvent réinitialisé à la fin de chaque requête/réponse HTTP. Cela signifie que les entités ne sont généralement pas conservées dans le contexte de persistance entre différentes requêtes HTTP, sauf si vous configurez spécifiquement votre application pour le faire (par exemple, en utilisant un pattern Unit of Work).

Conclusion

Le temps pendant lequel une entité reste dans le contexte de persistance dépend de la logique et du flux de travail de votre application. Vous avez le contrôle sur quand et comment les entités sont attachées et détachées du contexte de persistance, et vous pouvez gérer cela en fonction des besoins spécifiques de votre application.

Vous pouvez connaître l'état d'une entité dans le contexte de persistance en utilisant l'API UnitOfWork de Doctrine. L'objet `UnitOfWork` maintient l'état de toutes les entités gérées par l'`EntityManager`.

La méthode `getEntityState()` de l'`UnitOfWork` peut être utilisée pour obtenir l'état actuel d'une entité. Voici comment vous pouvez faire cela :

```
$entityManager = // obtenez votre EntityManager ici
$unitOfWork = $entityManager->getUnitOfWork();

// Charger ou créer une entité
$entity = $entityManager->find(YourEntityClass::class, $entityId);

// Obtenir l'état de l'entité
$entityState = $unitOfWork->getEntityState($entity);

// Les états possibles sont :
// UnitOfWork::STATE_MANAGED
// UnitOfWork::STATE_NEW
// UnitOfWork::STATE_DETACHED
// UnitOfWork::STATE_REMOVED

if ($entityState === UnitOfWork::STATE_MANAGED) {
    echo "L'entité est dans un état managed.";
} elseif ($entityState === UnitOfWork::STATE_NEW) {
    echo "L'entité est dans un état new.";
} elseif ($entityState === UnitOfWork::STATE_DETACHED) {
    echo "L'entité est dans un état detached.";
} elseif ($entityState === UnitOfWork::STATE_REMOVED) {
    echo "L'entité est dans un état removed.";
}
```

Une entité est dans l'état `STATE_NEW` lorsque vous avez créé une instance de l'entité et que vous ne l'avez pas encore persistée en utilisant l'`EntityManager`, ou si vous l'avez détachée après l'avoir créée. Cela signifie que l'entité existe seulement en mémoire et n'est pas encore gérée par l'`EntityManager`, et qu'aucune représentation de cette entité n'existe encore dans la base de données.

Voici un exemple illustrant ceci :

```
$user = new User();

$user->setUsername('JohnDoe');


// À ce stade, $user est dans l'état STATE_NEW

// car il n'est pas encore géré par l'EntityManager et n'existe pas dans la DB


$entityManager->persist($user);


// Maintenant, $user est dans l'état STATE_MANAGED

// car il est maintenant géré par l'EntityManager, mais il n'est toujours pas dans la DB jusqu'à ce que flush() soit appelé


$entityManager->flush();


// Maintenant, $user est toujours dans l'état STATE_MANAGED, mais il a été synchronisé avec la DB
```

Résumé :

- `STATE_NEW` : L'entité a été instanciée mais n'est pas encore gérée par l'`EntityManager`.
- `STATE_MANAGED` : L'entité est gérée par l'`EntityManager`, et les modifications seront synchronisées avec la base de données lors de l'appel à `flush()`.
- `STATE_DETACHED` : L'entité n'est pas gérée par l'`EntityManager`, les modifications ne seront pas synchronisées.
- `STATE_REMOVED` : L'entité est marquée pour suppression de la base de données lors du prochain `flush()`.

Ainsi, l'état `STATE_NEW` est un état temporaire initial dans le cycle de vie de l'entité avant qu'elle ne soit rendue persistante et gérée par l'`EntityManager`.