Locking, Concurrency, and EOF

鈴木鉄也 (suzuki@spice-of-life.net)

Preface

この章のロックとは

- スレッドをロックすること
- eoをスレッドセーフに扱うのが目的
- 他スレッドから同一eoにアクセスされるのを防ぐ？

Locking, Concurrency, and EOF

- ロックと並行性とEOF
- **スレッドをロックしろ**
  - シングルスレッドのJavaアプリケーションなどない（JavaVMは通常マルチスレッドで動作する）
  - WOはスレッドを複数使っている

- WOは次の処理でスレッドを生成する
  - メインスレッドの終了処理
  - セッションタイムアウトの処理
- 実際はこの状況下でEOFにアクセスすることはまずないが、アクセスしないという確証もない
- ロックしないと再現できないバグがでてくるかも
- **ロックしておけ**

What Need to Be Locked?

- eo自身をロックすることはできない
- **EOEditingContextをロックしろ**
  - EOFは次のecのみ面倒を見る
    - セッションのデフォルトec
    - すべてのEOSharedEditingContext
  - shared ec はロックするな

- デフォルトecとshared ecしか使ってないなら**ロックを心配する必要はない**
- 自分で生成したecはロックの面倒を見てやること
- Caution - ecをネストする場合
  - 親も子もロックしろ

- EOAccessレイヤーにアクセスする前にEOObjectStoreCoordinatorをロックしろ
  - 通常EOObjectStoreCoordinatorはアプリケーションに１つ
  - すべてのEODatabaseContextがロックされる
- EODatabaseContext以下のオブジェクト（スナップショットなど）もすべてロックされる

How to Lock Corrently

- NSRecursiveLockを使ってロックする
  - デッドロックせずに複数回ロックできる
  - アンロックしたかどうかを気にせずにロックできる
- ロックとアンロックのバランスを保証
- EOEditingContextのロックに使われる？

NSRecursiveLock

ロックした回数をカウントしている

ロックとアンロックの回数	ロックの処理
アンロックより ロックした回数が多い	ロックを続ける
ロックとアンロックの 回数が同じ	アンロックし、 他のスレッドに解放する

ロックの注意点

1. 早めにロックする
   - ロックなしでアクセスされる
   - ロックする間にデータを改変される
2. ロックしたら必ずアンロックする
   - レスポンスが遅くなる
   - デッドロックの原因になる

P. 85

Locking the Java Way

1. 生成するか参照を取得したらロック

2. try-finally間でロック

- ecの寿命が複数回のrequest-responseループにまたがる場合はロックしないこと

```
EOEditingContext myEC =
    new EOEditingContxt();
try {
    myEC.lock();
    // 処理
} finally {
    myEC.unlock();
    myEC.dispose();
}
```

The Complexity of Locking

- ロックのタイミング
  - request-responseループの始めにロック
  - ループの終わりにアンロック
- ループのawake()とsleep()でロックする？
  - WOApplicationではecを特定できない
  - WOSession, WOComponentはどうか
- request-responseループを調べてみよう
  - １回だけアクセスした場合
  - 連続して同じリンクにアクセスした場合

P. 86 - 補足

Request-Responseループ

1. awake()
2. takeValuesFromRequest()
3. invokeAction()
   - WOActionResults (WOComponent or WOResponse) を返す
   - WOApplication, WOSession, WOComponent
4. WOActionResults.generateResponse()
   - WOComponent.generateResponse()は appeendToResponse()を呼ぶ
5. sleep()

P. 86 - Listing 3-8.

The Order of Calls to `awake()` *and* `sleep()`

awake()/sleep()の呼ばれる順序

スレッド	オブジェクト	メソッド
1	Application	`awake`
1	Session	`awake`
1	Component	`new`
1	Component	`awake`
1	Component	`appendToResponse`
1	Session	`sleep`
1	Application	`sleep`

P. 86 - Listing 3-9.

The Order of Calls to `awake()` *and* `sleep()` *for Concurrent Requests*

スレッド	オブジェクト	メソッド
2	(RequestHandler)	`(handleRequest)`
2	Application	`awake`
2	Session	`awake`
2	Component (previous)	`awake`
2	Component (previous)	`invokeAction`
2	Component (next)	`new`
2	Component (next)	`awake`
3	(RequestHandler)	`(handleRequest)`

P. 86 - Listing 3-9.

The Order of Calls to `awake()` *and* `sleep()` *for Concurrent Requests*

スレッド	オブジェクト	メソッド
3	(RequestHandler)	`(handleRequest)`
2	Component(next)	`appendToResponse`
2	Component(previous)	`sleep`
2	Component(next)	`sleep`
2	Session	`sleep`
2	Application	`sleep`
3	Session	`awake`

⋮

P. 87

“a number of interesting things”

- ループ中でもリクエストを受け付ける
- スレッドを作ってリクエストを処理
- セッションは複数のリクエストを同時に受け付けない

The Impact of Direct Actions on Locking

- Direct Actionではどこで`awake()`, `sleep()`が呼ばれるのか
  - セッションを使う場合
  - セッションを使わない場合

P. 87 - 補足

Direct Action ループ

1. RequestHandler.handleRequest()
2. WODirectAction.performActionNamed()
   - WOActionResults (WOComponent or WOResponse) を返す
3. WOActionResults.generateResponse()
   - WOComponent.generateResponse()は appeendToResponse()を呼ぶ

The Order of Calls to `awake()` *and* `sleep()` *for Direct Actions Not Referencing*

スレッド	オブジェクト	メソッド
0	(Handler with Session ID)	`(handleRequest)`
0	Application	`awake`
0	DirectAction	`performActionNamed`
0	Component	`new`
0	Component	`awake`
0	(DirectAction)	`(end perform...)`
0	Component	`appendToResponse`
0	Component	`sleep`
0	Application	`sleep`

The Order of Calls to `awake()` *and* `sleep()` *for Direct Actions Reference the Session*

スレッド	オブジェクト	メソッド
1	(Handler with Session ID)	`(handleRequest)`
1	Application	`awake`
1	DirectAction	`performActionNamed`
1	Component	`new`
1	Component	`awake`
1	(DirectAction)	`(end perform...)`
1	Component	`appendToResponse`
1	Session	`awake`
1	Component	`sleep`
1	Session	`sleep`

P. 87-88

“Wait a minute!”

- セッションIDが含まれていても、existingSession()を呼ばない限りセッションは準備されない
- Sessionは`appendToResponse()`後に`awake()`される
  - アクションでecをロック処理しても、その後に`appendToResponse()`が呼ばれる

P. 89-90

Towards a Locking Practice

- コンポーネントアクション
  - WOSession, WOComponentのawake(), sleep()でロック
- ダイレクトアクション（セッション有）
  - WOSessionのawake(), sleep()では難しい
- ダイレクトアクション（セッション無）
  - WOSessionは使えない
  - DirectActionでロック

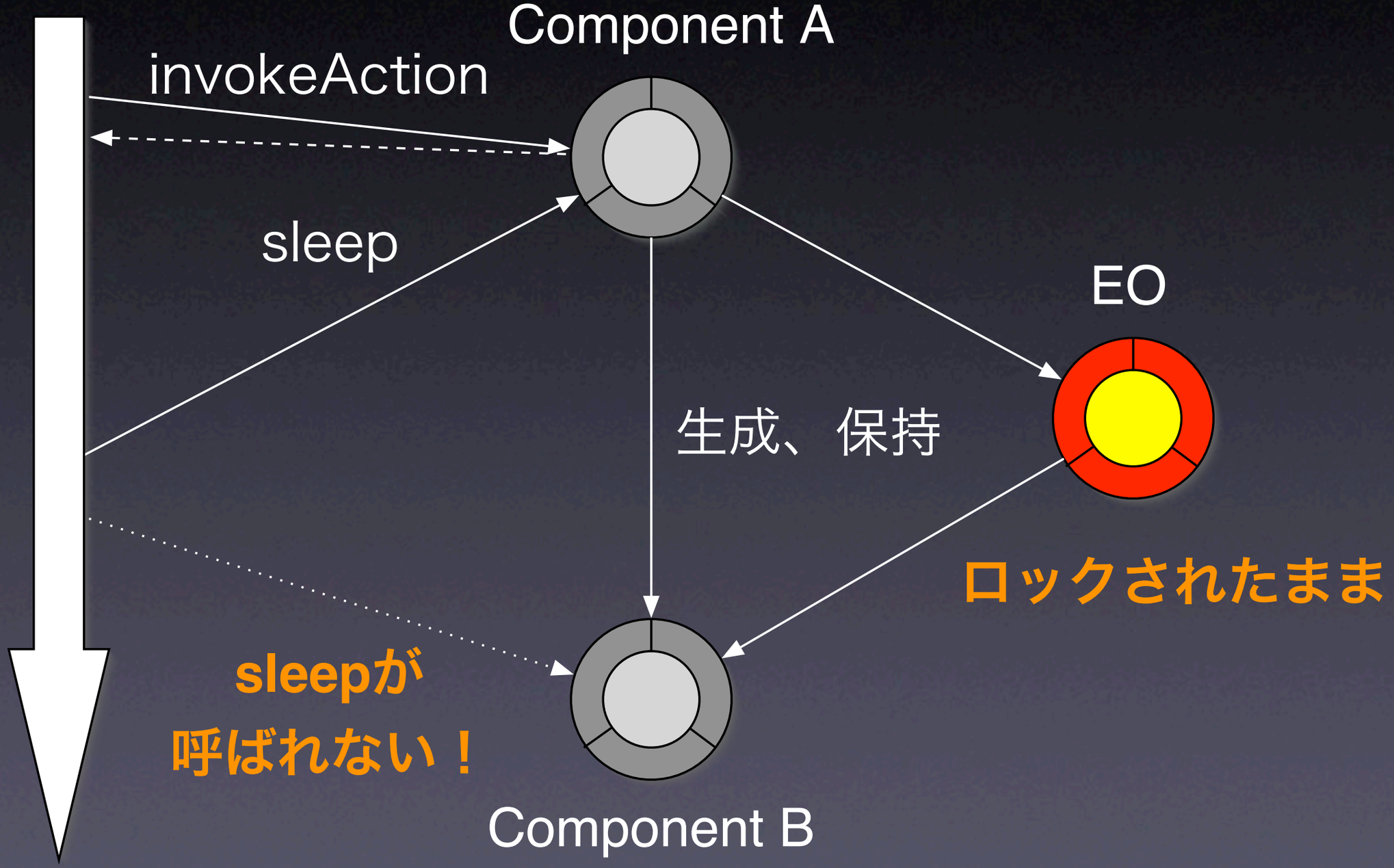
P. 89-90

セッションとEOEditingContext

- デフォルト以外のecも管理してほしい
- WOComponentが実行される前にロック、`sleep()`でアンロック
- ecは弱い参照で保持して欲しいが、セッション終了までゴミ集めされたくない
- 解決するにはループ中のみ強い参照でecを保持し、アンロックを保証する
- MultiECLockManagerを使え

P. 89-90

コンポーネントレベルでのロック



P. 89-90

結局…

- Direct Action では個別に対応する？
- awake-sleepのタイミングに気を使わなければならない
- 銀の弾丸はない

Locking Other EO Objects

- ecと同様にlock()でロックできる
- EOObjectStoreCoordinatorをロックする と関連するオブジェクトストアもロックされる
- EOAccessレイヤーのオブジェクトは直接ロックしても問題ない

補足

EOEditingContext

- ec.lock()のスレッドセーフ
  - **DBに接続するスレッドが１つ**であることを保証する
    - オブジェクトグラフの一貫性を保証するものではない？
- ロック中にフェッチ、保存したら？
  - 他のecに変更が通知される？

感想

感想

- セッションのデフォルトecとshared ecを使え
- スレッドセーフの保証下でDB処理を行うことに力を入れるべき？
- マルチスレッドを完璧に扱える人がどの程度いるのか

Thank you.