Wt - Bug #3377

Reading Wt::Dbo::ptr concurrently

06/24/2014 09:28 PM - Saif Rehman

Status:	Feedback	**Start date:**	06/24/2014
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

Creating a copy of the same Wt::Dbo::ptr concurrently(using copy constructor) causes an unhandled exception

http://redmine.webtoolkit.eu/boards/2/topics/9254

History

#1 - 06/26/2014 12:16 AM - Koen Deforche

- Status changed from New to Feedback

Hey,

A Wt::Dbo::Session (and all associated Wt::Dbo::ptr's) is not thread-safe. If you share one then you need to make sure you lock properly.

Regards,

koen

#2 - 06/26/2014 01:36 AM - Saif Rehman

I use a read/write lock. And the problem is occurring on reading the same ptr concurrently(but different Wt::Dbo::ptr object).

Most smart pointers are safe to read concurrently, do they implement locking or their algorithm is thread safe? Is there a way to fix this problem without switching to using simple locks on both read and write functions.

If Wt::Dbo::Session is causing the problem, would disconnecting the ptr from the session as we discussed in an old issue fix this issue? I'll try and let you know.

#3 - 06/26/2014 10:31 AM - Koen Deforche

Hey,

If you have two pointers referencing the same object from a single session, then it's not safe to manipulate (read or write) those pointers from two threads concurrently. A read operation may trigger the loading of the object for example, while a write operation needs to mark it as dirty. Those operations access the session object.

The thread-safety model of Wt::Dbo is plain simple: you cannot share a session or any of its associated pointers.

If you find that restrictive than just think of how transactions would be handled if you tried to do that? There's a semantic reason why thread-safe dbo pointers do not make sense (in contrast to std::shared_ptr which is actually useful in multi-threaded environments).

If you can uncouple the ptr from the session, then there is no longer a big risk, but nevertheless it would still be unsafe since we need to make sure concurrent modifications to the ptr (meta-)data does not break things.

#4 - 06/26/2014 12:57 PM - Saif Rehman

I really appreciate your reply, however, I wasn't clear. I have the following aspects in a database that stores Wt::Dbo::ptr in a map:

- All write functions will grab boost::lock_guardboost::shared_mutex lock(mutex) which will prevent any concurrency problem while writing
- All the Wt::Dbo::ptrs are fully loaded(there will be no lazy loading) and it is assumed that no database backend lookup function will be called
- No data will be modified inside any Wt::Dbo::ptr(I would use Wt::Dbo::ptr when available) from outside the database
- All read functions(ptr getter functions) will grab a shared read lock

So I know that from outside the database, no write/modification/Dbo::Session function can be called. The Wt::Dbo::ptrs are used to share data between sessions to prevent unnecessary copying of data.

If I switch to a simple exclusive ownership lock, it would potentially solve the problem with concurrent calls to getter functions, however, the problem would still be there since the copy constructor(take and release functions) are thread unsafe.

Can you please suggest how can I make the meta-data class thread safe to serve the purpose above?

Maybe a derived Dbo::ptr class that makes reading thread safe using locking or atomic synchronization as in std::shared_ptr.

Or maybe I should store the Dbo object in a shared ptr, however, it would still have mapped Wt::Dbo::ptr references.

#5 - 06/26/2014 03:12 PM - Koen Deforche

Hey,

You we would then only need to make the reference counting property thread-safe, which could perhaps be done with some atomic instruction?

Regards,

koen

#6 - 06/26/2014 03:45 PM - Saif Rehman

Ok, I'll look it up