USING A JAVA DOMAIN LAYER WITH COLDFUSION

Jaime Metcher

About me

- Programming for 20 yrs
- ColdFusion for 10 yrs (since 4.5)
- CT of Med-E-Serv
- Now moving to U of Q

Introduction

the waterfall process

CF

CF + Java

CWTF + Jawhat?

Java + CF

What is a domain layer?

- Services
- Domain model
- Glue
- Frameworks
- NOT UI
- NOT Controller

What is a domain model?

- Representation in code of domain expertise
- Independent of any one application
- See Fowler PoEAA

Truth in modelling

- Model defines what is true
- Model is the guardian of data integrity
- Every operation that is possible should be valid

Why a domain model?

- Captures abstract ideas in concrete form
- Can be a significant IP asset
- A great API will guide and shape applications built on it

Why Java

- Fast = fewer implementation compromises = a clearer model
- Deployment possibilities
- Tools
- Frameworks

Java domain layer

- Spring for DI
- POJOs for domain model
- Hibernate for ORM

Java domain layer - plumbing

- Hibernate Validator
- Commons collections
- Commons beanutils
- C3P0 connection pooling
- EHCache
- AspectJ

Simplest Hibernate

- Show code

Simplest Hibernate+Spring

- Show code

Hide some plumbing

- Store the SessionFactory in Application
- Cache current Session in Request
- Still explicitly open/close transactions
- OR, use aspects to make selected methods transactional

We're done!!

We now have the full power of Spring and Hibernate at our fingertips.

Or do we...?

Some gotchas

- Page lifecycle problems
- Distributed transactions
- ThreadLocal
- Classloader globals

Page lifecycle

- OnRequestStart() is always called, even after a cflocation
- OnRequestEnd() will NOT be called on a cflocation or a cfabort
- OnError() will reliably be called on any exception, even one from the Java code.

Page lifecycle (ctd)

- Use the underlying servlet life cycle where guaranteed cleanup is required.
- e.g. dangling transactions after a rogue cflocation
- Note that cftransactions ARE cleaned up after cflocation

Distributed transactions

- Transactions are essentially one-per-thread
- If we have more than one SessionFactory we need multiple connections in the same transaction.
- That means JTA with XA datasources
- Complex, slow and actually not very reliable.

Distributed transactions

- Transactions are essentially one-per-thread
- If we have more than one SessionFactory we need multiple connections in the same transaction.
- That means JTA with XA datasources
- Complex, slow and actually not very reliable.

Distributed transactions (ctd)

- Avoid having more than one SessionFactory
- Plug multiple frameworks into a “framework coordinator” which provides the SessionFactory and TransactionManager
- Not completely black box but not too bad.

ThreadLocals

- Variables that are bound to the thread
- Java frameworks use these regularly
- In a thread-pooling environment reliable cleanup is absolutely essential
- CF does not provide reliable cleanup for ColdFusion code

ThreadLocals (ctd)

- Do not initialize Java frameworks from ColdFusion
- Do transaction cleanup etc at the ServletContext level

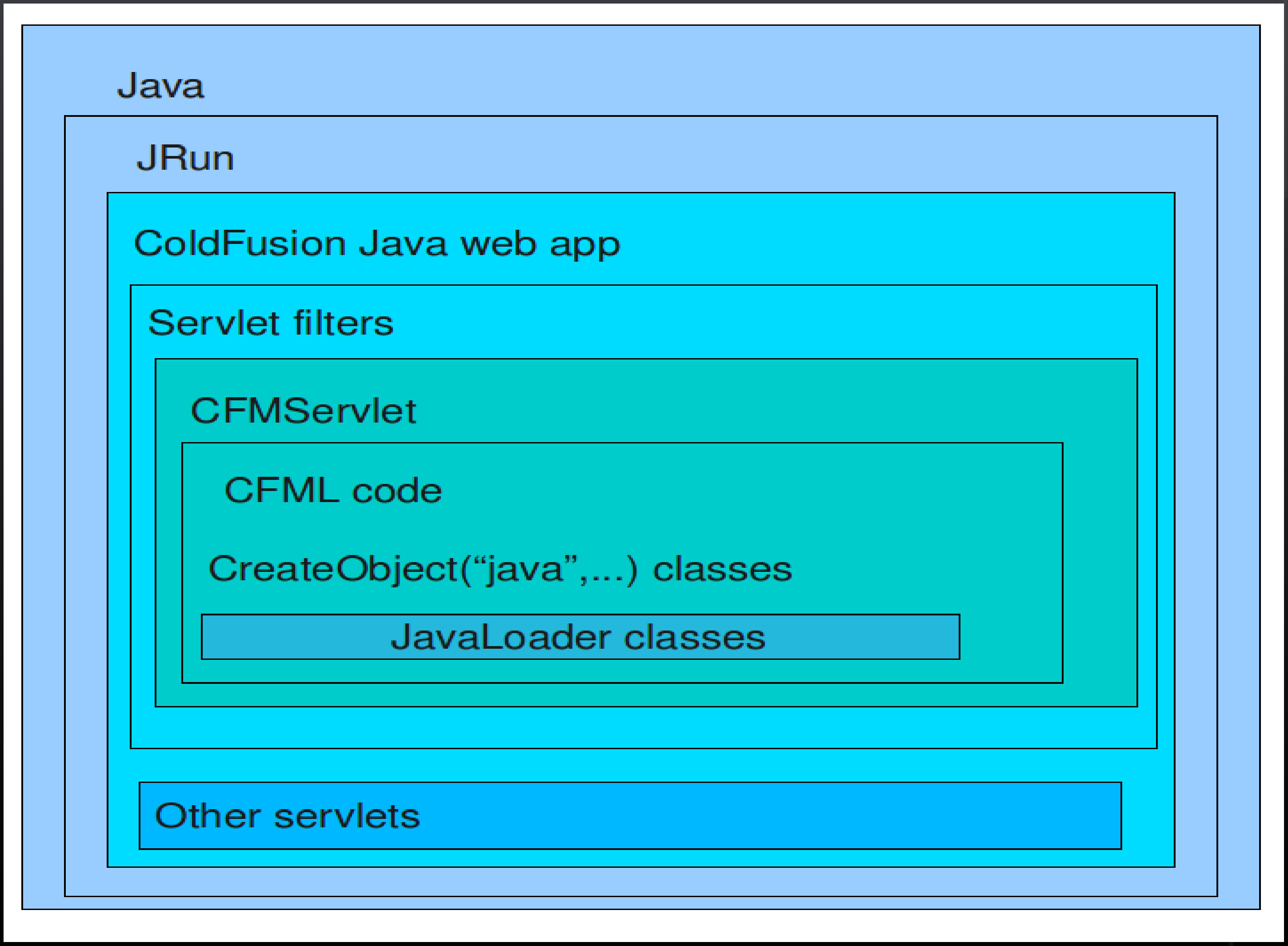
Classloader globals

- Entities that are global within a classloader i.e. within the whole of the CF server
- Static variables and methods
- AspectJ aspects

Classloader globals (ctd)

- Any use of AspectJ AOP within Spring can leak
- Once again, can't “black box” Spring contexts
- Can solve via a custom classloader (e.g. JavaLoader)

CF deployment environment



At the servlet level

- Load the Spring context
- Use the Spring OSIV filter to make a Hibernate session available
- Create a transaction filter to clean up dangling transactions
- Put the Spring context into the Request scope

At the servlet level (ctd)

- Our domain classes must be on the Jrun classpath, NOT the CF classpath
- That means we can't CreateObject() them
- All access to domain is via Request.applicationContext
- Some issues with XML parsers

At the CFML level

- Grab a service object from the Spring context
- Access the domain via transactional service methods
- May also explicitly create a transaction for ad hoc manipulation of domain objects

Show code

- Hibernate's transitive persistence
- Spring AOP DI into transients
- Spring AOP transaction manager
- Spring “open session in view”
- Spring context hierarchy for multiple domain models

Conclusions

- These frameworks come from the JEE world
- Slot them into the JEE stack where they expect to go, and everything is fine
- Ignoring the Java environment underlying ColdFusion is a luxury you can no longer afford.

Questions

jmetcher@gmail.com