

CDI Update

CDI, JCache CDI Interceptors, and
Spring/CDI Bridge



Outline

- Small overview of CDI
 - High-level comparison of Spring and Guice
- Case Study Spring/CDI bridge
- New features coming in CDI
- Case Study JCache interceptors
- Pretty open to interaction
- (Thanks Kevin Nilson and Van Riper)

Ice Breaker

- Who went to JavaOne?
- Who has worked with CDI? Guice? Spring?
- Who has worked with JSF? GWT? Spring MVC?
- Who likes EJBs?
- Who hates JSF?
- How many people think *Scala* will be the next Java?
- How many people think *Dart* will be the next Java?



About Me

- Currently Working on JSRs for CDI 1.1, JCache, Servlets/WebSocket, EJB, JSR-347 Grid computing, etc.
- InfoQ Editor currently
- Wrote a lot of articles over the years
 - Java, Spring, Cloud, CDI, etc., etc.
- Spoke at some conferences
 - Not in 4+ years
- Independent consultant
 - Currently doing some work for Caucho
- Ran consulting and training company for many years focused on Spring, JSF and Hibernate
- Started working with Java in 1997
- Tonight first public speaking in 3 years
- Wrote several books on SD dev. (not in 6+ years)
- Moved back to bay area Jan 2009
- Director of Development for a while
- Chief Software Architect of medical device company
- (Last Three Years) Recently programmed in Java, PHP, Groovy, Python and C
 - Wrote ECG graphs with C/GTK+ for embedded Linux Medical Device



About Caucho/Resin (only one slide, promise)

- Java application server that has been around since 1998
- Original claim to fame, order of magnitude faster than competitors
 - First time I used was in 2001
 - Used it for speed (online store system, SaaS, fit as many customers on the same instance/server as possible)
- Supports a lot of high traffic customers
 - Stock exchanges
 - Large CRM / SaaS vendors
 - Network appliances
- Cloud support baked in
 - Easy to setup elastic cluster
 - Cloud deployment
 - Cloud aware load balancer
- Support Java EE Web Profile (first to do so)
 - CDI support and the rest
- Health System
 - Can take snapshots of the system (heap, JVM meters, GC Meters, etc.)
 - Just in time profiling
 - Configurable health action
 - Anomaly detection
- Watchdog system
 - Ensures Resin stays up
 - If it goes down for any reason, it is restarted
- Batteries included App server:
 - JTA transactions, EJB light, JMS, HTTP Proxy, HTTP Caching, Faster than Apache HTTPD web server, Distributed Caching (JCahce/ Memcached), GC Less Memory Cache
 - You can setup a scalable cluster in a very short period of time
 - Easy to learn and all of the parts are designed to work together
 - Services are part of the cluster, replication, failover, etc. built in



Context Dependency Injection

- Contexts and Dependency Injection
 - Similar to Spring using `@Autowired/@Inject`
 - Similar to Guice, inspired by Guice
- Was called WebBeans
 - Started out as a standard to glue JSF to EJBs
 - Ideas came from Spring and Guice
 - Bob Lee was on WebBeans at one point (left, heard rumors why, but I was not involved with CDI anyway back then)

Context Dependency Injection

- Started out as a Java EE DI framework only
 - Supports scopes (context)
 - Original purpose to support gluing JSF to EJBs
 - Very Java EE centric
 - You could use CDI in Java SE, but was not standardized
 - Turns out CDI is pretty generic and its core concepts are not really tied to Java EE per se
 - You could use CDI with Tomcat (and some elbow grease)
- Part of Java EE,
 - Resin Caucho, Tom EE, Glassfish, JBoss AS



Why I ignored CDI for a long time

- Big Spring fan, big EJB hater
- Thought that CDI was tied to EJB and JSF
 - I used JSF but not all of the time, didn't like EJB
- Not a fan of EJB 3.0 either (EJB 3.1 ok)
 - JDJ article professing my hatred of EJB 3
 - <http://java.sys-con.com/node/216307>
- Building a JSF framework (Crank) using Spring to do similar but different things than Seam did
 - Not a big fan of Seam
 - Don't always use JSF, a lot of times I was using Spring MVC
- Busy running a consulting business
 - Client and consultant mgmt plus consulting and family

What I like about CDI

- It is a JSR
- It is a standard
- Other JSRs can build on top of it
 - 3 JSRs outside of Java EE are using (more to come)
- It is generic, and becoming more generic
- You can write portable extensions
- <http://cdisource.org/site/>
 - Have not done much with it in a while
 - Turning some attention back to it

CDI Features

- Scope/Context support
 - Built in scopes/context Request, Session, Application, Conversation,
- Dependency Injection
- Events (Observers)
- Interceptors
- Decorators
- Type safe (annotation based) resolution
- Extensions (very extensible)
- Integration with Java EE

Major changes in CDI 1.1

- Not tied to Java EE, EJB or JSF
 - In reality CDI 1.0 wasn't but it was
- CDI for Java SE
 - Standard way to start container (didn't exist before)
 - Standard way to control CDI container
- CDI for Java EE
 - Even Java EE version not tied to JSF
 - Conversation scope could be used by Servlets
 - Supports JSF but not tied to it
- TransactionScoped (assigned to me)

CDI Spring Differences

- From a developer perspective not too much
- De facto Open Source standard versus actual Standard
- Spring is older so it has a lot of evolutionary dead-ends that CDI does not
 - This is getting cleaned up over time
- You could use Spring with `@Inject` in a similar manner to using CDI
 - EJB lite + CDI dev very similar to modern Spring dev
 - Reza Rahmans has a nice slide deck on this
- With CDI concepts of scopes/context is built in from the start
 - With Spring it was added on later (version 2)
- CDI supports type safe resolution of injections, event/observers, and injections (based on Java language constructs)

Which is JavaEE, Spring?

```
@Service
@WebService
public class DefaultBidService implements BidService {
    @PersistenceContext
    private EntityManager entityManager;

    @Transactional
    public void addBid(Bid bid) {
        entityManager.persist(bid);
    }
}

public interface BidService {
    void addBid(Bid bid);
}
```

```
@Stateless
public class DefaultBidService implements BidService {
    @PersistenceContext
    private EntityManager entityManager;

    public void addBid(Bid bid) {
        entityManager.persist(bid);
    }
}

@Remote
@WebService
public interface BidService {
    void addBid(Bid bid);
}
```

How about now?

```
import javax.inject.Inject;

public class AutomatedTellerMachineImpl implements AutomatedTellerMachine {

    private ATMTransport transport;

    @Inject
    public void setTransport(ATMTransport transport) {
        this.transport = transport;
    }

    ...
}
```

```
@Service ("atm")
public class AutomatedTellerMachineImpl implements AutomatedTellerMachine{

    @Autowired (required=true)
    @Qualifier ("default")
    private ATMTransport transport;
```

```
public class AutomatedTellerMachineImpl implements AutomatedTellerMachine {

    @Inject
    private ATMTransport transport;
```

Spring versus CDI versus Guice

- All pretty similar
 - Experts site huge technical advantages of one over the other, I don't see it
 - With CDI Java compiler handles most injections problems
 - With Spring, Spring IDE support handles most injection problems
- One might win out for a specific app
- I am not anti-Spring, I am pro CDI (more on next slide)
- Spring is more than CDI (add on frameworks)
 - Would be good to at least be able to integrate Spring if needed for Open Source frameworks that rely on it or for Spring modules you might need
- In general CDI takes less startup configuration
- Spring is more complex and more powerful but also harder to predict how people will use it
- Don't know Guice well enough to make any comments at all
 - My understanding lighter weight, and type safe like CDI,
 - CDI is more prone to use client proxies than Guice

Spring CDI (similar but different)

- Java Dzone articles (I wrote on Spring and CDI)
- CDI DI tutorial from 2011 based on Spring DI and AOP tutorials
 - <http://java.dzone.com/articles/cdi-di-p1>
 - <http://java.dzone.com/articles/cdi-di-p2>
 - <http://java.dzone.com/articles/cdi-aop>
- Spring tutorials from 07, 08
 - <http://java.dzone.com/articles/dependency-injection-an-introd>
 - <http://java.dzone.com/articles/introduction-spring-aop>

Spring CDI Bridge

- <http://cdisource.org/site/2011/06/spring-and-java-ee-6-cdi-get-it-done/>
- EJB 3.1 and CDI productive convention over configuration model
- People will use EJB 3.1 and CDI because it is the standard
- Even if you decide to adopt EJB 3.1 and CDI, you still may need to integrate with 3rd party libraries that use Spring or Spring modules.
- Spring integration might be a semi-permanent part of your architecture
- Needs work, it works now, but....



Who?

- Just me? No.
 - Cutting my teeth with CDI
 - Unlearning hardwired ideas from 8-9 years of Spring use
- Andy Gibson helped
- Consulted with Caucho Engineers who worked on Candi when I got stuck
 - Got help from Scott Ferguson
 - What is Candi?
- Got help with Mark Struberg
 - He patched OWB
- Initially it only worked with Candi, then Weld or Candi, then Candi and Weld, then Candi, Weld and OWB
- Probably got some help from Peter Muir but not sure



Wanted Spring integration to work both ways

- Inject Spring beans into CDI managed beans
- Inject CDI beans into Spring beans
- Took knowledge from getting JSF, Hibernate and Spring to play nice and applied it to get CDI and Spring to play nice
 - Presto Framework
 - Crank Framework

CDI to Spring injection

```
23 @RequestMapping("/tasks")
24 @Controller
25 public class TaskController {
26
27     @Autowired
28     TaskRepository repo;
29
30     @RequestMapping(method = RequestMethod.POST)
31     public String create(@Valid Task task, BindingResult bindingResult,
32                          Model uiModel, HttpServletRequest httpRequest) {
```

```
17 @Stateless
18 public class TaskRepository {
19
20     @PersistenceContext
21     private EntityManager entityManager;
22
23
24     public void persist(Task task) {
25         this.entityManager.persist(task);
26     }
27
```

Spring to CDI Example

- <http://rick-hightower.blogspot.com/>
- Had some better example but can't find them

```
1 package org.cdisource.springintegration;
2
3 import javax.inject.Inject;
4
5 public class CdiBeanThatHasSpringInjection {
6     @Inject @Spring(name="fooBar")
7     FooSpringBean springBean;
8
9     @Inject @Spring(name="fooBarnotActuallyThere", required=false)
10    FooSpringBean notActuallyThere;
11
12
13    @Inject @Spring(type=FooSpringBean2.class)
14    FooSpringBean2 injectByType;
15
```

Spring to CDI Example

```
1 package org.cdisource.springintegration;
2
3 import javax.inject.Inject;
4
5 public class CdiBeanThatHasSpringLookupInjection {
6     @Inject @SpringLookup("fooBar2")
7     FooSpringBean springBean;
8
9     @Inject @SpringLookup("fooBar2")
10    FooSpringBean springBean2;
```

First problem to overcome

- No standard way to init a container
 - There will be in CDI 1.1
- Created a BeanContainer interface
 - Represents common access to CDI impls (Weld, Candi, OWB)
- BeanContainerManager factory that looks up the BeanContainer
 - Uses Java Service Loader to find right impl
 - Uses Java WeakReference so instance is reloadable (if you reload the war file)
- Wrote an AbstractBeanContainer that has most the low level CDI logic, then wrote specific BeanContainer implementations for Resin Candi, Apache OWB and JBoss Weld
- Come fork me on github <https://github.com/CDISource/cdisource>



Added ability to look up CDI beans from Spring

```
11 public class CdiFactoryBean implements FactoryBean<Object>, InitializingBean {
12
13     private Class<?> beanClass;
14     private boolean singleton = true;
15     private BeanContainer beanContainer;
16     private BeanManager beanManager;
17
18
19     @Override
20     public void afterPropertiesSet() throws Exception {
21         if (beanManager==null) throw new IllegalStateException("BeanManager must be set");
22         beanContainer = new BeanContainerImpl(beanManager);
23     }
24
25     public void setBeanClass(Class<?> beanClass) {
26         this.beanClass = beanClass;
27     }
28
29     @Override
30     public Object getObject() throws Exception {
31         return beanContainer.getBeanByType(beanClass);
32     }
33 }
```

Don't want to tons of XML (1)

```
14 public class CdiBeanFactoryPostProcessor implements BeanFactoryPostProcessor {  
15  
16     private boolean useLongName;  
17  
18     private BeanManagerLocationUtil beanManagerLocationUtil = new BeanManagerLocationUtil();  
19
```

Don't want to tons of XML (2)

```
@Override
public void postProcessBeanFactory(
    ConfigurableListableBeanFactory beanFactory) throws BeansException {

    DefaultListableBeanFactory factory = (DefaultListableBeanFactory) beanFactory;

    Set<Bean<?>> beans = beanManagerLocationUtil.beanManager().getBeans(Object.class);
    for (Bean<?> bean : beans) {
        if (bean instanceof SpringIntegrationExtention.SpringBean) {
            continue;
        }

        if (bean.getName() != null && bean.getName().equals("Spring Injection")){
            continue;
        }
        BeanDefinitionBuilder definition = BeanDefinitionBuilder.rootBeanDefinition(CdiFactoryBean.class)
            .addPropertyValue("beanClass", bean.getBeanClass())
            .addPropertyValue("beanManager", beanManagerLocationUtil.beanManager())
            .setLazyInit(true);

        String name = generateName(bean);
        factory.registerBeanDefinition(name, definition.getBeanDefinition());
    }
}
```

So far we handled using CDI bean in Spring

- Next we want to use Spring beans in CDI

Spring bean injection

- Inject a Spring bean with @Spring of @SpringLookup

```
1 package org.cdisource.springintegration;
2
3 import java.lang.annotation.Retention;
4 import java.lang.annotation.Target;
5 import static java.lang.annotation.ElementType.*;
6 import static java.lang.annotation.RetentionPolicy.*;
7
8 @Retention(RUNTIME) @Target({TYPE, METHOD, FIELD, PARAMETER})
9 public @interface SpringLookup {
10     String value();
11 }
```

More complex / features @Spring

```
1 package org.cdisource.springintegration;
2
3 import java.lang.annotation.Retention;
4 import java.lang.annotation.Target;
5 import static java.lang.annotation.ElementType.*;
6 import static java.lang.annotation.RetentionPolicy.*;
7
8 import javax.enterprise.util.Nonbinding;
9 import javax.inject.Qualifier;
10
11
12 @Qualifier @Retention(RUNTIME) @Target({TYPE, METHOD, FIELD, PARAMETER})
13 public @interface Spring {
14     Class<?> type() default Object.class;
15     String name() default "";
16     @Nonbinding boolean required() default true;
17 }
```

Now the other way, Extension

```
29 public class SpringIntegrationExtention implements Extension {
30     Map <String, SpringBean> springBeans = new HashMap<String, SpringBean>();
31
32
33     public void processInjectionTarget (@Observes ProcessInjectionTarget<?> pit, BeanManager bm) {
34
35         Set<InjectionPoint> injectionPoints = pit.getInjectionTarget().getInjectionPoints();
36
37         synchronized (springBeans) {
38             for (InjectionPoint point: injectionPoints){
39
40                 if (!(point.getType() instanceof Class<?>)) {
41                     continue;
42                 }
43
44                 Class<?> injectionType = (Class<?>) point.getType();
45                 Spring spring = point.getAnnotated().getAnnotation(Spring.class);
46                 if (spring!=null) {
47                     SpringBean springBean = new SpringBean(pit.getAnnotatedType(), spring, injectionType,
48                     springBeans.put(springBean.key(), springBean); //we can do some validation to make sur
49                 } else {
50                     SpringLookup springLookup = point.getAnnotated().getAnnotation(SpringLookup.class);
51                     if (springLookup!=null) {
52                         SpringBean springBean = new SpringBean(springLookup, injectionType, bm);
53                         springBeans.put(springBean.key(), springBean);
54                     }
55                 }
56             }
57         }
58     }
```

Next
Slide
Zooms

Now the other Way part 2

- Process Injection points

```
for (InjectionPoint point: injectionPoints){  
  
    if (!(point.getType() instanceof Class<?>)) {  
        continue;  
    }  
  
    Class<?> injectionType = (Class<?>) point.getType();  
    Spring spring = point.getAnnotated().getAnnotation(Spring.class);  
    if (spring!=null) {  
        SpringBean springBean = new SpringBean(pit.getAnnotatedType(), spring, injectionType,  
        springBeans.put(springBean.key(), springBean); //we can do some validation to make sur  
    } else {  
        SpringLookup springLookup = point.getAnnotated().getAnnotation(SpringLookup.class);  
        if (springLookup!=null) {  
            SpringBean springBean = new SpringBean(springLookup, injectionType, bm);  
            springBeans.put(springBean.key(), springBean);  
        }  
    }  
}
```

Introducing SpringBeans into CDI

```
class SpringBean implements Bean <Object> {
    //InjectionTarget<Object> it;
    Spring spring;
    SpringLookup lookup;
    Class<?> injectionType;
    BeanManager bm;

    AnnotatedType<?> annotatedType;

    SpringBean(AnnotatedType<?> annotatedType, Spring spr
        this.spring = spring;
        this.injectionType = injectionType;
        this.bm = bm;
        this.annotatedType = annotatedType;
        //it = bm.createInjectionTarget(at);
    }
```

SpringIntegrationExtension.java

```
171         @Override
172         public Object create(CreationalContext<Object> ctx) {
173             ApplicationContext applicationContext = ApplicationContextLocatorManager.getInstance().locateApplicati

Object instance = null;
if (spring!=null) {
if (!spring.name().trim().equals("")) {
    if(!spring.required()) {
        if (applicationContext.containsBean(spring.name())) {
            instance = applicationContext.getBean(spring.name(), spring.type());
        }
    } else {
        instance = applicationContext.getBean(spring.name(), spring.type());
    }
} else {
    instance = applicationContext.getBean(spring.type());
}
} else {
    instance = applicationContext.getBean(lookup.value());
}
return instance;
```

NEW CDI 1.1 FEATURES



CDI 1.1 New Features (1)

- CDI 1.1 (JSR-346) is an update to CDI 1.0 (JSR-299)
- @Disposes methods for producer fields
- CDI class,
 - programmatic access to CDI facilities from outside a managed bean
- Pass qualifiers event was fired with to the ObserverMethod
- Ability to veto beans declaratively using @Veto and @Requires
- Ability to access the BeanManager from the ServletContext
 - Bean Manager
 - Look up bean by type
 - Look up bean by name
 - Do all sorts of utility stuff
 - CDI
 - Access to bean manager
 - Global Instance
 - CDIProvider
 - Access to CDI
- Conversations in Servlet requests

CDI 1.1 New Features (2)

- Application lifecycle events in Java EE
- Injection of Bean metadata into bean instances
- Programmatic access to a container provided Producer, InjectionTarget, AnnotatedType
- AnnotationLiteral and TypeLiteral Helper classes
- Override attributes of a Bean via BeanAttributes
- Process modules via ProcessModule
- Wrap the InjectionPoint
- Extension instances from BeanManager
- Injection of the ServletContext
- Access to beans.xml in ProcessModule
- Injection into enums

CDI 1.1 New Features (3)

- **@RequestScoped** now applies
 - EJB business method calls
 - EJB MDB message handler calls
 - WebService
 - Timed EJB calls
- **@ApplicationScoped** now applies
 - All types of Java EE applications
- **@TransactionScoped**

CDI 1.1 New Features (4)

- XML support like Candi and Seam XML
- Provide implementation for any interface
 - Service Handlers
- Global ordering for interceptors, decorators and events
- Global enablement of interceptors, decorators and alternatives
- Split specification into core and Java EE integration
- Bootstrap support
- Java SE context definition
- Transaction scope

JCache CDI support

How to support CDI and not be a tool



JSR 107 JCache

- Added annotation driven caching that uses CDI but does not preclude the use of Spring
- RI ships with CDI support and Spring support
 - Only CDI support is required
- Uses CDI but does not force Spring users to include CDI libraries
 - Be standard, but be cool to others
- Code from CDI Source cut/paste into TCK by me
 - Used what I learned. 😊
- How I got involved



CDI Annotation Usage

```
@CacheResult(cacheName="blogManager")  
public Blog getBlogEntry(String title) {...}
```

```
@CacheRemoveEntry(cacheName="blogManager")  
public void removeBlogEntry(String title) {...}
```

```
@CacheRemoveAll(cacheName="blogManager")  
public void removeAllBlogs() {...}
```

```
@CachePut(cacheName="blogManager")  
public void createEntry(@CacheKeyParam String title,
```

```
@CacheValue Blog blog) {...}  
@CacheResult(cacheName="blogManager")  
public Blog getEntryCached(String randomArg,  
@CacheKeyParam String title){...}
```

CDI Interceptor in RI

```
28  /**
29   * Interceptor for {@link CacheResult}
30   *
31   * @author Rick Hightower
32   * @author Eric Dalquist
33   */
34  @CacheResult @Interceptor
35  public class CacheResultInterceptor extends AbstractCacheResultInterceptor<InvocationContext> {
36
37      @Inject
38      private CacheLookupUtil lookup;
39
40      /**
41       * @param invocationContext The intercepted invocation
42       * @return The result from {@link InvocationContext#proceed()}
43       * @throws Throwable likely {@link InvocationContext#proceed()} threw an exception
44       */
45      @AroundInvoke
46      public Object cacheResult(InvocationContext invocationContext) throws Throwable {
47          return this.cacheResult(this.lookup, invocationContext);
48      }
49  }
```

Annotation from JCache

```
51  *
52  * @author Eric Dalquist
53  * @author Rick Hightower
54  * @since 1.0
55  */
56  @Target( {ElementType.METHOD, ElementType.TYPE} )
57  @Retention(RetentionPolicy.RUNTIME)
58  public @interface CacheResult {
59
60      /**
61       * (Optional) name of the cache.
62       * <p/>
63       * If not specified defaults first to {@link Ca
64       * defaults to: package.name.ClassName.methodName
65       */
66      @Nonbinding
67      String cacheName() default "";
68
```

Use Extension to make annotation neutral

```
32 public class InterceptorExtension implements Extension {
33
34     /**
35      * Service interface implemented by extensions. An extension is a service provider declared
36      *
37      * @param beforeBeanDiscoveryEvent the event to register
38      */
39     void discoverInterceptorBindings(@Observes BeforeBeanDiscovery beforeBeanDiscoveryEvent) {
40         beforeBeanDiscoveryEvent.addInterceptorBinding(CachePut.class);
41         beforeBeanDiscoveryEvent.addInterceptorBinding(CacheResult.class);
42         beforeBeanDiscoveryEvent.addInterceptorBinding(CacheRemoveEntry.class);
43         beforeBeanDiscoveryEvent.addInterceptorBinding(CacheRemoveAll.class);
44     }
```