TIKAL Open Source Solutions for Software Development

By : Zvika Markfeld
Tikal Knowledge

Spring Tips & Tricks

"It's Easy to Hit a Fly With A Gun..."

10/31/2010

Session Context

- Many companies are using Spring...
- But do they know what they are doing?
- Have you ever duplicated XML definitions?
- Did you ever need to systematically inject mock-ups?
- Did you ever need to do transactional work at application init'?
- Did you ever wonder about using multiple XML files for modularity and environment separation?
- Session Goal: getting familiar with (my) Spring favorite practices
  - Open for discussion

Context Initialization

There's 50 Way to Bootstrap an Application...

common-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>

    <beans>
        <context:property-placeholder location="classpath:my-app.properties"/>

     <context:annotation-config />

     <context:component-scan base-package="com.mycompany.myproduct" />

     <tx:annotation-driven />

     <aop:aspectj-autoproxy proxy-target-class="true" />

     <bean id="mbeanServer"
         class="org.springframework.jmx.support.MBeanServerFactoryBean">
       <property name="locateExistingServerIfPossible" value="true"/>
        </bean>

     <context:mbean-export server="mbeanServer"
         default-domain="com.mycompany.myproduct"
         registration="replaceExisting"/>
    </beans>
```

All other *-context.xml

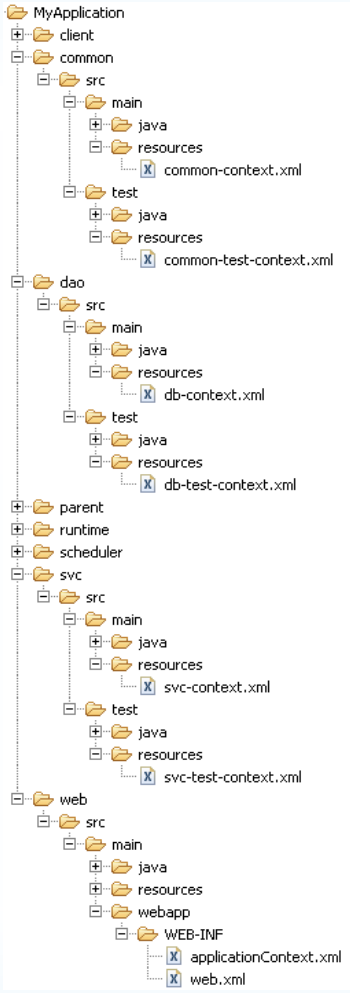
- **src/main/resources/db-context:**
  Datasource, hibernate session factory, tx, etc.
- **src/test/resources/db-test-context:**
  Same, with in-memory implementations
- **src/main/resources/svc-context:**
  Factory beans: aspects, scheduling, manual proxies, etc.
- **src/test/resources/svc-test-context:**
  Same, with mocks
- **src/main/webapp/applicationContext:**
  Together with web.xml defining the web application:
  security, web services, filters, interceptors, proxies...

Web Module's applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>

	<import resource="classpath:common-context.xml"/>

	<import resource="classpath:db-context.xml"/>

	<import resource="classpath:svc-context.xml"/>

	<!-- other web definitions: web services, flex, etc. -->
</beans>
```

Context Aggregation

- Majority of beans are loaded via annotations
  - Annotation scanning defined in `common-context.xml`
- XML context files typically define pre-written beans - data Sources, scheduler triggers, etc.
  - db-context.xml defines datasources, session factory, tx manager, etc.
- Each execution context defines its list of xmls used for starting up the context
  - Test: @ContextConfiguration
  - Web: WEB-INF/applicationContext.xml
- Best way I've found, so far

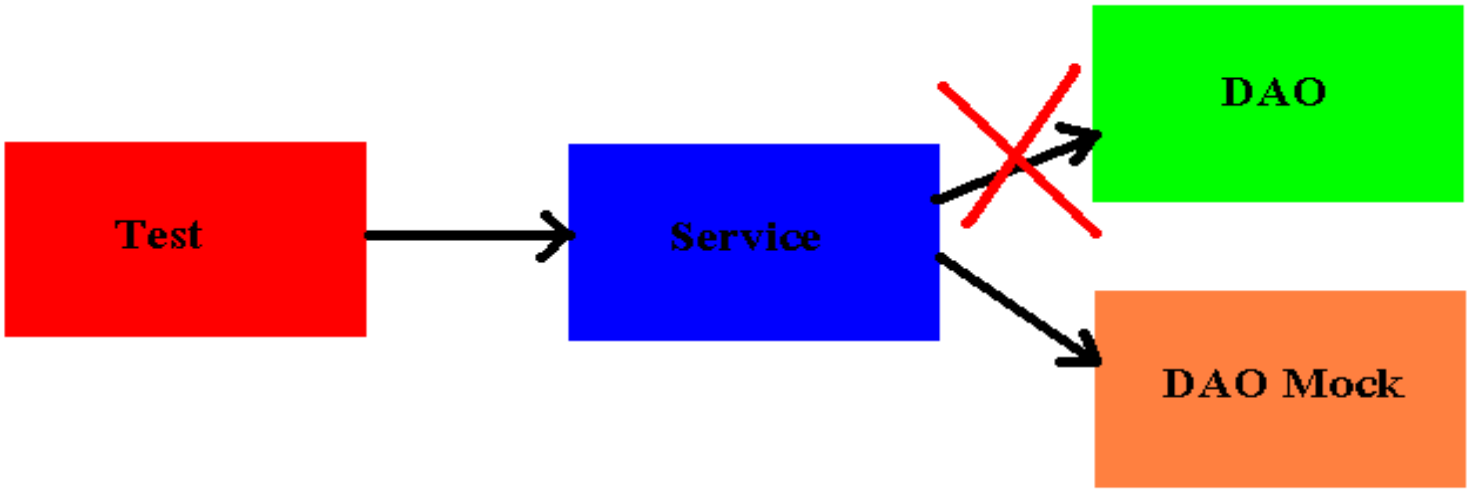


Testing

Easy to do,
if you put your mind to it?

Stubbin'

- Let's say that while testing the Service layer, you don't want DAOs to access the database...
- So you've re-implemented your DAO interfaces and now you need to autowire them into the tested service...
- Problem: using annotations, from test code you cannot modify 2nd level injections



Solution: Tweak Component Scanner

common-test-context.xml

```
<beans >

 <context:component-scan base-package="org.mycompany">

    <!-- Exclude real DAos -->
    <context:exclude-filter type="aspectj"
          expression="com.mycompany.myproduct..*Hibernate*Dao"/>

    <!-- OR: -->
    <context:exclude-filter type="annotation"
           expression="org.springframework.stereotype.Repository"/>

    <!-- Include the DAO Stubs -->
       <context:include-filter type="annotation"
           expression="com.mycompany.myproduct.util.spring.stereotype.Stub"/>
 </context:component-scan>

</beans>
```

RestTemplate & XPathTemplate

Cracking Up Web Services

RestTemplate & XPathTemplate

- Loyal to Spring's [YouNameIt]Template tradition
- Simplify REST access and XPath operations
- For example, accessing a REST endpoint via GET:

```
String result = restTemplate.getForObject(
    "http://example.com/hotels/{hotel}/bookings/{booking}",
     String.class, "42", "21");
```

OR:

```
Map<String, String> vars = new HashMap<String, String>();
vars.put("hotel", "42");
vars.put("booking", "21");
String result = restTemplate.getForObject(
    "http://example.com/hotels/{hotel}/bookings/{booking}",
     String.class, vars);
```

Other RestTemplate Methods

- `void delete(String url, Map<String,?> vars)`
- `HttpHeaders headForHeaders(String url, Map<String,?> vars)`
- `Set<HttpMethod> optionsForAllow(String url, Map<String,?> vars)`
- `<T> T postForObject(String url, Object request, Class<T> responseType, Map<String,?> vars)`
- `void put(String url, Object request, Map<String,?> urlVariables)`
- `void setMessageConverters(List<HttpMessageConverter<?>> converters)`

Real World Example... Sort of

When asking for the following URL:

http://www.flickr.com/services/rest?method=flickr.photos.search&api+key=xxx&tags=penguins

Document retrieved:

```
<photos page="2" pages="89" perpage="10" total="881">
      <photo id="2636" owner="47058503995@N01"
          secret="a123456" server="2" title="test_04"
          ispublic="1" isfriend="0" isfamily="0" />
      <photo id="2633" owner="47058503995@N01"
          secret="c123456" server="2" title="test_01"
          ispublic="1" isfriend="0" isfamily="0" />
      <photo id="2610" owner="12037949754@N01"
          secret="d123456" server="2" title="00_tall"
          ispublic="1" isfriend="0" isfamily="0" />
</photos>
```

Doing the Same With RestTemplate

```
final String PHOTO_SEARCH_URL =
    "http://www.flickr.com/services/rest" +
    "?method=flickr.photos.search&api+key={api-key}&" +
    "tags={tag}&per_page=10";

  javax.xml.transform.Source photos =
    restTemplate.getForObject(
       PHOTO_SEARCH_URL,
       Source.class,
       apiKey,
       searchTerm);
```

Flickin' Through The XMLs

Accessing The retrieved XML could not be easier...

```
List<BufferedImage> imageList = xpathTemplate.evaluate(
   "//photo", photos, new NodeMapper() {

        public Object mapNode(Node node, int i) throws DOMException {

           Element photo = (Element) node;

           Map<String, String> variables = new HashMap<String, String>(3);
           variables.put("server", photo.getAttribute("server"));
           variables.put("id", photo.getAttribute("id"));
           variables.put("secret", photo.getAttribute("secret"));

           String photoUrl =
            "http://static.flickr.com/{server}/{id}_{secret}_m.jpg";

           return restTemplate.getForObject(
            photoUrl, BufferedImage.class, variables);
        }
   });
```

Configuration

```
<beans>

     <bean id="restTemplate"
    class="org.sf.web.client.RestTemplate">

    <property name="messageConverters">
        <list>
          <bean class="org.sf...SourceHttpMessageConverter"/>
          <bean class="com.mycomp...BufferedImageHttpMessageConverter"/>
        </list>
      </property>
     </bean>

     <bean id="xpathTemplate" class="org.sf.xml.xpath.Jaxp13XPathTemplate"/>

   </beans>
```

Converting The Images

```
public class BufferedImageHttpMessageConverter implements
        HttpMessageConverter<BufferedImage> {

        public List<MediaType> getSupportedMediaTypes() {
            return Collections.singletonList(new MediaType("image", "jpeg"));
        }

        public boolean supports(Class<? extends BufferedImage> clazz) {
            return BufferedImage.class.equals(clazz);
        }

        public BufferedImage read(Class<BufferedImage> c, HttpInputMessage input)
             throws IOException {

            return ImageIO.read(inputMessage.getBody());
        }

        public void write(BufferedImage image, HttpOutputMessage message)
             throws IOException {

         throw new UnsupportedOperationException("Not implemented");
        }
    }
```

Spring Expression Language

SpEL-led incorrectly

@Value Injection

```
@Component
Class MyBean {

        // Assuming <context:property-placeholder
        //             location="classpath:app-${my.env}.properties"/>
    @Value("#{hostname}")
        private String hostname;

        // Using SPEL to access other beans' properties
        @Value("#{otherBean.someProperty}")
        private String someProperty;

        // ...or system properties
        @Value("#{systemProperties['user.region']}")
        private String userRegion;

        // ...anybody heard of Elvis, the Operator?
        @Value("#{systemProperties['user.home'] ?: '.'}")
        private String userHome;

        // ... or activate methods on arbitrary objects
        @Value('#{T(java.lang.Math).random() * 100.0}')
        private int random;

    // ...
    }
```

JMX

Kickass Protocol

Spring JMX

- Your service beans can be exposed as JMX MBean quite easily:

```
<bean id="mbeanServer" class="org...jmx.support.MBeanServerFactoryBean">
        <property name="locateExistingServerIfPossible" value="true"></property>
</bean>

<context:mbean-export
     server="mbeanServer" default-domain="com.mycompany.myprod" />
```

```
@ManagedResource(objectName = "com.mycompany.myprod:name=MyService", ...)
public class MyServiceImpl implements MyService {

   @ManagedAttribute(...)
   public void setSomethingToTuneInRuntime(long value) { ... }

   @ManagedOperation(...)
   public Collection<Dinosaur> getPrehistoricCreatures(Era era) { ... }
}
```

Interacting with JMX Layer

- JBoss provides a web-application names jmx-console...
- Nice (and free), but not fully functional
  - Meet jmx-admin!
  - Runs on any server
  - Complex return objects rendered as html trees
  - Extensible argument subsystem
    - XML
    - Enumerated values
    - Classpath resources
    - A few others...
  - Developed by yours truely for Tikal clients
  - Will be available on site with this post

JMX Admin - MBean List View

JMX Admin - MBean View

MBean: com.vmware.am.apm.app:name=FlexService

<< back to admin view

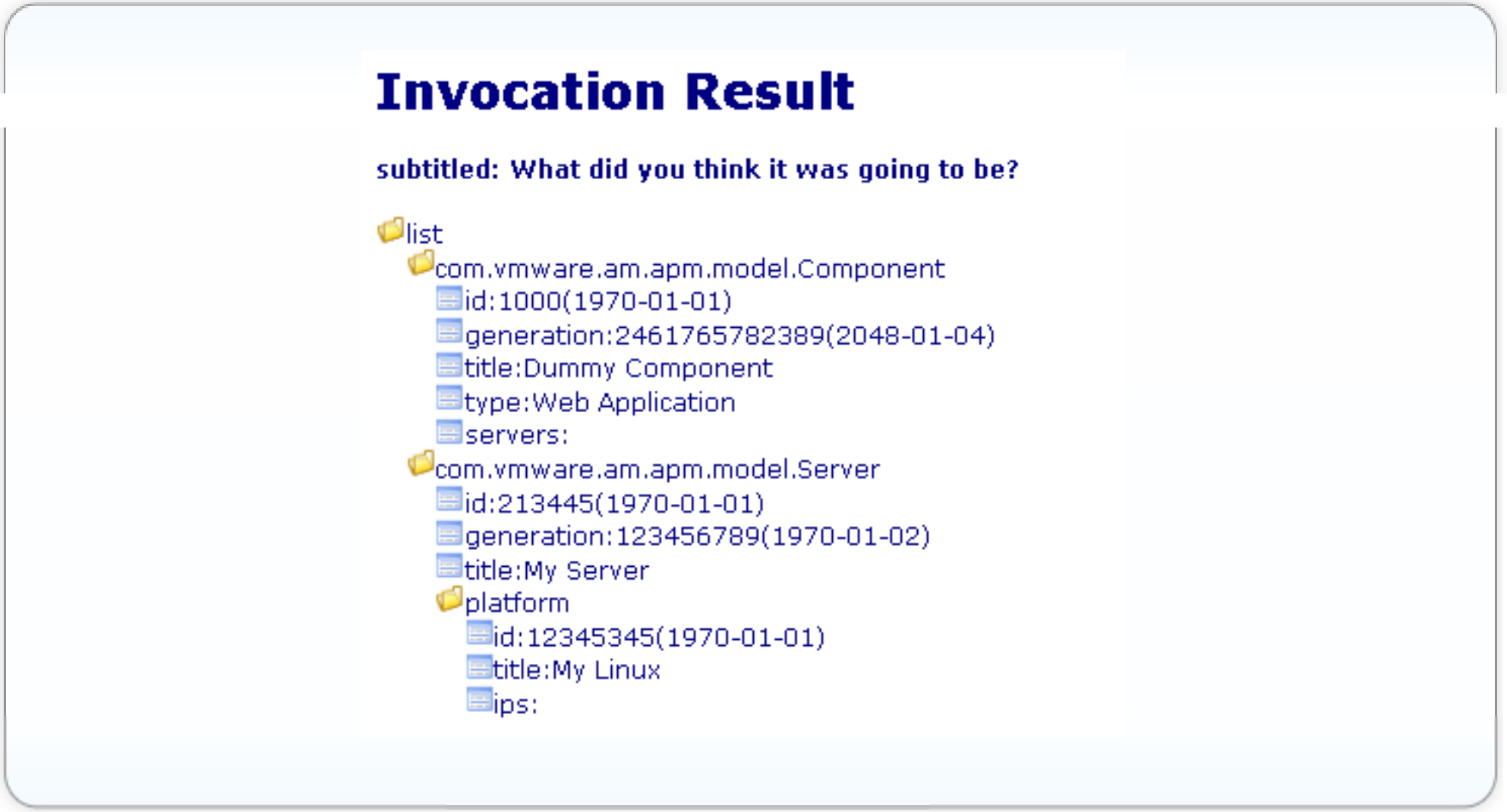
Attributes

Name	Value

Operations

Operation Name	Return Type	Description	Parameters	Invoke
getNotifications	java.util.Collection	getNotifications	p1 com.vmware.am.apm.model.TopologyActor	Hit Me!
getMonitoredApplications	java.util.Collection	getMonitoredApplications		Hit Me!
getMonitoredTiers	java.util.Collection	getMonitoredTiers	p1 com.vmware.am.apm.model.Application	Hit Me!
registerAdapter	boolean	registerAdapter	p1 com.vmware.am.apm.adapter.model.DataSourceConnectionDetails	Hit Me!
registerHypericAdapter	void	registerHypericAdapter	p1 java.lang.String p2 int p3 boolean p4 java.lang.String p5 java.lang.String	Hit Me!
getManagedAdapters	java.util.Collection	getManagedAdapters		Hit Me!
getMonitoredKPIs	java.util.Collection	getMonitoredKPIs	p1 com.vmware.am.apm.model.TopologyObject	Hit Me!
getIndicatorOverTime	java.util.Collection	getIndicatorOverTime	p1 com.vmware.am.apm.model.TopologyObject p2 java.lang.String	Hit Me!
getIndicators	java.util.Collection	getIndicators	p1 com.vmware.am.apm.model.TopologyObject	Hit Me!
getTransactionComponents	java.util.Collection	getTransactionComponents	p1 com.vmware.am.apm.model.Transaction	Hit Me!
getMonitoredTransactions	java.util.Collection	getMonitoredTransactions	p1 com.vmware.am.apm.model.Application	Hit Me!
getTopologyObjectHealth	double	getTopologyObjectHealth	p1 com.vmware.am.apm.model.TopologyObject	Hit Me!
getMonitoredTransactionElements	java.util.Collection	getMonitoredTransactionElements	p1 com.vmware.am.apm.model.Transaction	Hit Me!
getComponentMonitoredTransactionElement	java.util.Collection	getComponentMonitoredTransactionElement	p1 com.vmware.am.apm.model.Component	Hit Me!
restartPlatform	void	restartPlatform	p1 com.vmware.am.apm.model.Platform	Hit Me!
getMonitoredComponent	com.vmware.am.apm.flex.model.MonitoredComponent	getMonitoredComponent	p1 com.vmware.am.apm.model.Component	Hit Me!

JMX Admin - Invocation Result View



Testing Services via JMX Layer

```
import javax.management.remote.JMXConnectorFactory as JmxFactory
    import javax.management.remote.JMXServiceURL as JmxUrl
    import javax.management.*;
    import javax.management.remote.*;
    // ...

def env = [(JMXConnector.CREDENTIALS): (String[])["admin", "springsource"]]

def jmxUrl = "service:jmx:rmi:///jndi/rmi://localhost:6969/jmxrmi";

def server = JmxFactory.connect(new JMXServiceURL(jmxUrl), env)
    .getMBeanServerConnection();

def myService = new GroovyMBean(server, "com.mycompany.myprod:name=MyService")
myService.setSomethingToTuneInRuntime(1023);

def dinos = myService.getPrehistoricCreatures(Era.MESOZOIC)

dinos.each { dino ->
    println "name: ${dino.name}"
}
```

Scripting Initialization Data

- On server startup, in development environments
  - using a ServletContextListener, ApplicationContextListener, or @PostConstruct method
- On system installation, in production environments
  - Creating script directory and directly executing scripts under it
  - Packaging groovy scripts and runtime in the application installer

Writing JMeter Scenarios

- Using jmeter-groovy-sampler, an opensource project, you can write JMeter stress tests in groovy
- Useful in scenarios where no other protocol is available (BlazeDS, binary http invoker, ...)
- Can be executed in various ways
  - Development environment: Eclipse, JMeter
  - Build environment: Maven, Hudson

Placing Properties

Ancient as the Moon, but still...

- Environment-dependent configuration

```
<context:property-placeholder location="classpath:db-${my.env}.properties"/>
```

- External properties file

```
<context:property-placeholder location="file:///${ext.prop.dir}db.properties"/>
```

- System properties override

```
<bean class="org.sf.beans.factory.config.PropertyPlaceholderConfigurer">
  <property name="systemPropertiesModeName" value="SYSTEM_PROPERTIES_MODE_OVERRIDE"/>
  <property name="location" value="classpath:application.properties"/>
</bean>
```

- Local properties override

```
<context:property-placeholder
    location="classpath:my-app.properties,classpath:myapp.local.properties,${config}"
    ignoreResourcesNotFound="true"/>
```

Thank You!