JAVAWORLD

NEWS

Use a distributed cache to cluster your Spring remoting services

Add automatic discovery and clustering to Spring remoting

By Mikhail Garber

JavaWorld | Oct 31, 2005 12:00 AM PT


```
  public void put(String path, Object key, Object value) throws Exception;
   public Object get(String path, Object key) throws Exception;
```

JBoss Cache is a tree-like structure, which is why we need the `path` parameter.

This interface's server-side implementation references the cache MBean as follows:

```
 <bean id="CacheService" class="app.service.JBossCacheServiceImpl">
   <property name="cacheMBean" ref="CustomTreeCacheMBean"/>
</bean>
```

The `JBossCacheServiceImpl` bean simply delegates calls from the cache service to the cache MBean.

And, as the last of the server-side Spring beans, we need a `ServicePublisher` that watches the Spring container's lifecycle, and publishes or removes the service definitions (URLs) to and from our cache:

```
 <bean id="ServicePublisher" class="app.service.ServicePublisher">
   <property name="cache" ref="CacheService"/>
</bean>
```

This code illustrates how `ServicePublisher` would react if the Spring context were refreshed (when, for example, an application has just deployed):

```
private void contextRefreshed() throws Exception {
    logger.info("context refreshed");

    String[] names = context
        .getBeanNamesForType(AutoDiscoveredServiceExporter.class);
    logger.info("exporting services:" + names.length);
    for (int i = 0; i < names.length; i++) {
      String serviceUrl = makeUrl(names[i]);
      try {
        Set services = (Set) cache.get(SERVICE_PREFIX + names[i],
            SERVICE_KEY);
        if (services == null)
          services = new HashSet();
        services.add(serviceUrl);
        cache.put(SERVICE_PREFIX + names[i], SERVICE_KEY, services);
        logger.info("added:" + serviceUrl);
      } catch (Exception ex) {
        logger.error("exception adding service:", ex);
      }
    }
```

As you can see, the publisher simply iterates over the list of services exported via cached service descriptions and adds the definitions (URLs) to the cache. Our cache is designed such that the path (or cache region) contains the name of the service, whose URL list is stored as a `Set` object under some static key in this region. Making the service name part of the path is important for the JBoss Cache implementation because it creates and releases the transactional locks based on the path. This way, the updates made for Service A will not interfere with updates made for Service B because they are mapped to different paths: */some/prefix/serviceA/key=(list of URLs)* and */some/prefix/serviceB/key=(list of URLs)*.

The code for removing service definitions (when context is closed) is similar.

Now, let's move to the client side. We need to have a cache implementation to share with the server-side one:

```
 <bean id="LocalCacheService" class="app.auto.LocalJBossCacheServiceImpl">
</bean>
```

The `LocalJBossCacheServiceImpl` holds a reference to a standalone copy of JBoss Cache configured from the same custom-cache-service.xml file we used on the server:

```
public LocalJBossCacheServiceImpl() throws Exception {
    super();
    cache = new TreeCache();
    PropertyConfigurator config = new PropertyConfigurator();
    config.configure(cache, "app/context/custom-cache-service.xml");
 }
```

This cache definition file includes the configuration for the JGroups layer, allowing all the cache members to find each other via UDP multicast, both clients and servers.

The `LocalJBossCacheServiceImpl` also implements our `CacheServiceInterface` and provides caching services to our last Spring bean, `AutoDiscoveredService`. This bean extends the standard Spring `HttpInvokerProxyFactoryBean` but is configured differently:

```
<bean id="TestService"
    class="app.auto.AutoDiscoveredService">
    <property name="serviceInterface"
       value="app.service.TestServiceInterface" />
    <property name="cache" ref="LocalCacheService"/>
 </bean>
```

First and foremost, a URL is no longer present. The `AutoDiscoveredService` automatically discovers any of our special Spring remoting services on our network exposed under the `TestService` name. For that discovery to happen, this bean obtains the list of URLs from our distributed cache:

```
private List getServiceUrls() throws Exception {
    Set services = (Set) cache.get(ServicePublisher.SERVICE_PREFIX
          + beanName, ServicePublisher.SERVICE_KEY);
    if (services == null)
       return null;
    ArrayList results = new ArrayList(services);
    Collections.shuffle(results);
    logger.info("shuffled:" + results);
    return results;
 }
```

The `Collections.shuffle` call randomly rearranges the list of URLs associated with this service so the client method invocation is load-balanced between them. The actual remote call is as follows:

```
public Object invoke(MethodInvocation arg0) throws Throwable {

    List urls = getServiceUrls();
    if (urls != null)
      for (Iterator allUrls = urls.iterator(); allUrls.hasNext();) {
        String serviceUrl = null;
        try {
          serviceUrl = (String) allUrls.next();
          super.setServiceUrl(serviceUrl);
          logger.info("going to:" + serviceUrl);
          return super.invoke(arg0);
        } catch (Throwable problem) {
          if (problem instanceof IOException
              || problem instanceof RemoteAccessException) {
            logger.warn("got error accessing:"
                + super.getServiceUrl(), problem);
            removeFailedService(serviceUrl);
          } else {
            throw problem;
          }
        }
      }
    throw new IllegalStateException("No services configured for name:"
        + beanName);
  }
```

As you can see, if a remote call results in a thrown exception, (`RemoteAccessException` for Spring problems or `IOException` for general IO problems like network outage), the client code will deal with the problem and then try the next URL from the list, thus providing transparent failover of this service. If a call failed because of some other kind of exception, it is rethrown to be handled by the client.

A `removeFailedService()` method below simply removes the failed URL from the list and updates the distributed cache, making this information available to all other clients simultaneously:

```
  private void removeFailedService(String url) {
      try {
         logger.info("removing failed service:" + url);
         Set services = (Set) cache.get(ServicePublisher.SERVICE_PREFIX
               + beanName, ServicePublisher.SERVICE_KEY);
         if (services != null) {
            services.remove(url);
            cache.put(ServicePublisher.SERVICE_PREFIX + beanName, ServicePublisher.SER
VICE_KEY,
                  services);
            logger.info("removed failed service at:" + url);
         }
      } catch (Exception e) {
         logger.warn("failed to remove failed service:" + url, e);
      }
   }
```

If you build and deploy the example Web application to more than one JBoss server and run the provided `LoopingAutoDiscoveredRemoteServiceTest`, you can see how the incoming requests are load-balanced between the members of your Spring cluster. You can also stop and restart any of the servers, and the calls dynamically rout to the remaining server(s). If you crash a server (by pulling the network cable out or sending kill -9 on Linux), you will see an exception logged on the client console, but all the requests will be served uninterruptedly by failing over to the other server.

Conclusion

In this article, we looked at how to cluster network services provided by Spring remoting. In addition, you learned how to simplify the deployment of complex multitier applications by defining the individual services by their names only and relying on automatic discovery to bind every service to the appropriate URL.

Mikhail Garber is a Dallas-based information technology professional with more than 14 years of experience in enterprise software development. Garber specializes in Java/J2EE, databases, messaging, and open source solutions. His services have been employed by such organizations as Mary Kay Cosmetics, Boeing Defense and Space, Verizon Wireless, the US government, Lockheed Martin, Sabre/Travelocity, and many others.

Learn more about this topic

- Download the code that accompanies this article http://images.techhive.com/downloads/idge/imported/article/jvw/2005/10/jw-1031-spring.zip

(http://images.techhive.com/downloads/idge/imported/article/jvw/2005/10/jw-1031-spring.zip)

- Spring Framework and Spring remoting
  http://www.springframework.org (http://www.springframework.org)
- JBoss Cache
  http://www.jboss.org/products/jbosscache (http://www.jboss.org/products/jbosscache)
- For more articles on Spring, browse these recent *JavaWorld* articles:
- "Use Spring to Create a Simple Workflow Engine," Steve Dodge (April 2005)
  http://www.javaworld.com/javaworld/jw-04-2005/jw-0411-spring.html
  (http://www.javaworld.com/javaworld/jw-04-2005/jw-0411-spring.html)
- "Pro SpringSpring and EJB," Rob Harrop and Jan Machacek (February 2005)
  http://www.javaworld.com/javaworld/jw-02-2005/jw-0214-springejb.html
  (http://www.javaworld.com/javaworld/jw-02-2005/jw-0214-springejb.html)
- "Let Your Ant Enjoy Spring," Josef Betancourt (February 2005)
  http://www.javaworld.com/javaworld/jw-02-2005/jw-0214-antspring.html
  (http://www.javaworld.com/javaworld/jw-02-2005/jw-0214-antspring.html)
- Leverage Spring, Struts, Hibernate, and Axis to "Design a Simple Service-Oriented J2EE Application Framework," Fangjian Wu (October 2004)
  http://www.javaworld.com/javaworld/jw-10-2004/jw-1004-soa.html
  (http://www.javaworld.com/javaworld/jw-10-2004/jw-1004-soa.html)
- Learn how to build a Web application with JavaServer Faces, Spring, and Hibernate in "Put JSF to Work," Derek Yang Shen (July 2004)
  http://www.javaworld.com/javaworld/jw-07-2004/jw-0719-jsf.html
  (http://www.javaworld.com/javaworld/jw-07-2004/jw-0719-jsf.html)

- For more articles on Java development tools, browse the **Development Tools** section of *JavaWorld*'s Topical Index
  http://www.javaworld.com/channel_content/jw-tools-index.shtml
  (http://www.javaworld.com/channel_content/jw-tools-index.shtml)
- For more articles on application servers, browse the **Java Application Servers** section of *JavaWorld*'s Topical Index
  http://www.javaworld.com/channel_content/jw-appserv-index.shtml
  (http://www.javaworld.com/channel_content/jw-appserv-index.shtml)