The ApplicationContext

Summary

Description

The ApplicationContext is the extension of BeanFactory which provides following functions:

- MessageSource :. Supports to access messages by i18n-sytle
- Access to resources : Supports easy access to resources such as URL and files
- Event propagation : Conveys events to the bean that realized the ApplicationListener interface
- Loading of multiple (hierarchical) contexts : Supports the hierarchical contents to write context to certain layers such as web layers of the application

BeanFactory or ApplicationContext?

It is better to use ApplicationContext over BeanFactory. Find below a feature matrix that lists what features are provided by the BeanFactory and ApplicationContext interfaces

Feature	BeanFactory	ApplicationContext
Bean instantiation/wiring	Yes	Yes
Automatic BeanPostProcessor registration	No	Yes
Automatic BeanFactoryPostProcessor registration	No	Yes
Convenient MessageSource access (for i18n)	No	Yes
ApplicationEvent publication	No	Yes

Internationalization using MessageSources

Refer to Resource.

Event

Event handling in the ApplicationContext is provided through the ApplicationEvent class and ApplicationListener interface. If a bean which implements the ApplicationListener interface is deployed into the context, every time an ApplicationEvent gets published to the ApplicationContext, that bean will be notified. Spring provides the following standard events:

Event	Description
ContextRefreshedEvent	Published when the ApplicationContext is initialized or refreshed, e.g. using the refresh() method on the ConfigurableApplicationContext interface. "Initialized" here means that all beans are loaded, post-processor beans are detected and activated, singletons are pre-instantiated, and the ApplicationContext object is ready for use. A refresh may be triggered multiple times, as long as the context hasn't been closed - provided that the chosen ApplicationContext actually supports such "hot" refreshes (which e.g. XmlWebApplicationContext does but GenericApplicationContext doesn't).
ContextStartedEvent	Published when the ApplicationContext is started, using the start() method on the ConfigurableApplicationContext interface. "Started" here means that all Lifecycle beans will receive an explicit start signal. This will typically be used for restarting after an explicit stop, but may also be used for starting components that haven't been configured for autostart (e.g. haven't started on initialization already).
ContextStoppedEvent	Published when the ApplicationContext is stopped, using the stop() method on the ConfigurableApplicationContext interface. "Stopped" here means that all

	Lifecycle beans will receive an explicit stop signal. A stopped context may be restarted through a start() call.
ContextClosedEvent	Published when the ApplicationContext is closed, using the close() method on the ConfigurableApplicationContext interface. "Closed" here means that all singleton beans are destroyed. A closed context has reached its end of life; it cannot be refreshed or restarted.
RequestHandledEvent	A web-specific event telling all beans that an HTTP request has been serviced (this will be published *after* the request has been finished). Note that this event is only applicable for web applications using Spring's DispatcherServlet.

Implementing custom events can be done as well. Simply call the publishEvent() method on the ApplicationContext, specifying a parameter which is an instance of your custom event class implementing ApplicationEvent. Event listeners receive events synchronously. This means the publishEvent() method blocks until all listeners have finished processing the event (it is possible to supply an alternate event publishing strategy via a ApplicationEventMulticaster implementation). Furthermore, when a listener receives an event it operates inside the transaction context of the publisher, if a transaction context is available.

Example:

```
<bean id="emailer" class="example.EmailBean">
  <property name="blackList">
    <list>
      <value>black@list.org</value>
      <value>white@list.org</value>
      <value>john@doe.org</value>
    </list>
  </property>
</bean>

<bean id="blackListListener" class="example.BlackListNotifier">
  <property name="notificationAddress" value="spam@list.org"/>
</bean>
public class EmailBean implements ApplicationContextAware {

    private List blackList;
    private ApplicationContext ctx;

    public void setBlackList(List blackList) {
        this.blackList = blackList;
    }

    public void setApplicationContext(ApplicationContext ctx) {
        this.ctx = ctx;
    }

    public void sendEmail(String address, String text) {
        if (blackList.contains(address)) {
            BlackListEvent event = new BlackListEvent(address, text);
            ctx.publishEvent(event);
            return;
        }
        // send email...
    }
}
public class BlackListNotifier implements ApplicationListener {

    private String notificationAddress;

    public void setNotificationAddress(String notificationAddress) {
```

```
        this.notificationAddress = notificationAddress;
    }

    public void onApplicationEvent(ApplicationEvent event) {
        if (event instanceof BlackListEvent) {
            // notify appropriate person...
        }
    }
}
```

Convenient ApplicationContext instantiation for web applications

As opposed to the BeanFactory based on the program, ApplicationContext instances can be created declaratively using a ContextLoader.

The ContextLoader mechanism comes in two flavors: the ContextLoaderListener and the ContextLoaderServlet. They both have the same functionality but differ in that the listener version cannot be reliably used in Servlet 2.3 containers. Since the Servlet 2.4 specification, servlet context listeners are required to execute immediately after the servlet context for the web application has been created and is available to service the first request (and also when the servlet context is about to be shut down): as such a servlet context listener is an ideal place to initialize the Spring ApplicationContext.

You can register an ApplicationContext using the ContextLoaderListener as follows:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/daoContext.xml /WEB-INF/applicationContext.xml</param-value>
</context-param>

<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- or use the ContextLoaderServlet instead of the above listener
<servlet>
  <servlet-name>context</servlet-name>
  <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
-->
```

The listener inspects the 'contextConfigLocation' parameter. If the parameter does not exist, the listener will use /WEB-INF/applicationContext.xml as a default. When it *does* exist, it will separate the String using predefined delimiters (comma, semicolon and whitespace) and use the values as locations where application contexts will be searched for. Ant-style path patterns are supported as well: e.g. /WEB-INF/*Context.xml (for all files whose name ends with "Context.xml", residing in the "WEB-INF" directory) or /WEB-INF/**/*Context.xml (for all such files in any subdirectory of "WEB-INF").

The ContextLoaderServlet can be used instead of the ContextLoaderListener. The servlet will use the 'contextConfigLocation' parameter just as the listener does.

Reference

- Spring Framework - Reference Document / 3.8. The ApplicationContext