**Spring的ApplicationEvent的使用**

    Spring 3.0中提供了很多类似\*Aware的类，其中ApplicationContextAware接口可以实现我们在初始化bean的时候给bean注入ApplicationConxt（Spring上下文对象）对象。ApplicationContextAware接口提供了publishEvent方法，实现了Observe（观察者）设计模式的传播机制，实现了对bean的传播。通过ApplicationContextAware我们可以把系统中所有ApplicationEvent传播给系统中所有的ApplicationListener。因此，我们只需要构造好我们自己的ApplicationEvent和ApplicationListener，就可以在系统中实现相应的监听器。

   下面以增加学生的示例来演示如何构造Spring的监听器，StudentAddEvent是监听的事件对象，StudentAddListener是事件的监听器（负责处理接收到的监听事件），StudentAddBean负责触发StudentAddEvent事件。具体步骤如下：

**1.  定义StudentAddEvent监听事件**

新建StudentAddEvent类，实现抽象类

org.springframework.context.ApplicationEvent

StudentAddEvent类中需要实现自己的构造函数，具体代码如下:

**[java]** [view plaincopy](http://blog.csdn.net/wgw335363240/article/details/7202320)

1. **package** com.trs.spring.event;
3. **import** org.springframework.context.ApplicationEvent;
5. /\*\*
6. \* 增加学生的监听事件
7. \*/
8. **public** **class** StudentAddEvent **extends** ApplicationEvent {
10. /\*\*
11. \*
12. \*/
13. **private** **static** **final** **long** serialVersionUID = 20L;
15. /\*\*
16. \* 学生姓名
17. \*/
18. **private** String m\_sStudentName;
20. /\*\*
21. \* @param source
22. \*/
23. **public** StudentAddEvent(Object source, String \_sStudentName) {
24. **super**(source);
25. **this**.m\_sStudentName = \_sStudentName;
26. }
28. /\*\*
29. \* 获取学生姓名
30. \*
31. \* @return
32. \*/
33. **public** String getStudentName() {
34. **return** m\_sStudentName;
35. }
37. }

**[java]** [view plaincopy](http://blog.csdn.net/xingtianyiyun/article/details/7388098)

1. **package** com.trs.spring.event;
3. **import** org.springframework.context.ApplicationEvent;
5. /\*\*
6. \* 增加学生的监听事件
7. \*/
8. **public** **class** StudentAddEvent **extends** ApplicationEvent {
10. /\*\*
11. \*
12. \*/
13. **private** **static** **final** **long** serialVersionUID = 20L;
15. /\*\*
16. \* 学生姓名
17. \*/
18. **private** String m\_sStudentName;
20. /\*\*
21. \* @param source
22. \*/
23. **public** StudentAddEvent(Object source, String \_sStudentName) {
24. **super**(source);
25. **this**.m\_sStudentName = \_sStudentName;
26. }
28. /\*\*
29. \* 获取学生姓名
30. \*
31. \* @return
32. \*/
33. **public** String getStudentName() {
34. **return** m\_sStudentName;
35. }
37. }

**2.  定义StudentAddListener监听器**

新建StudentAddListener类，实现接口

org.springframework.context.ApplicationListener

中的onApplicationEvent方法，在该方法中只处理StudentAddEvent类型的ApplicationEvent事件，代码如下：

**[java]** [view plaincopy](http://blog.csdn.net/wgw335363240/article/details/7202320)

1. **package** com.trs.spring.event;
3. **import** org.springframework.context.ApplicationEvent;
4. **import** org.springframework.context.ApplicationListener;
6. **public** **class** StudentAddListener **implements** ApplicationListener {
8. /\*
9. \* (non-Javadoc)
10. \*
11. \* @see
12. \* org.springframework.context.ApplicationListener#onApplicationEvent(org
13. \* .springframework.context.ApplicationEvent)
14. \*/
15. **public** **void** onApplicationEvent(ApplicationEvent \_event) {
16. // 1.判断是否是增加学生对象的事件
17. **if** (!(\_event **instanceof** StudentAddEvent)) {
18. **return**;
19. }
21. // 2.是增加学生事件的对象，进行逻辑处理，比如记日志、积分等
22. StudentAddEvent studentAddEvent = (StudentAddEvent) \_event;
23. System.out.println("增加了学生:::" + studentAddEvent.getStudentName());
24. }
26. }

**[java]** [view plaincopy](http://blog.csdn.net/xingtianyiyun/article/details/7388098)

1. **package** com.trs.spring.event;
3. **import** org.springframework.context.ApplicationEvent;
4. **import** org.springframework.context.ApplicationListener;
6. **public** **class** StudentAddListener **implements** ApplicationListener {
8. /\*
9. \* (non-Javadoc)
10. \*
11. \* @see
12. \* org.springframework.context.ApplicationListener#onApplicationEvent(org
13. \* .springframework.context.ApplicationEvent)
14. \*/
15. **public** **void** onApplicationEvent(ApplicationEvent \_event) {
16. // 1.判断是否是增加学生对象的事件
17. **if** (!(\_event **instanceof** StudentAddEvent)) {
18. **return**;
19. }
21. // 2.是增加学生事件的对象，进行逻辑处理，比如记日志、积分等
22. StudentAddEvent studentAddEvent = (StudentAddEvent) \_event;
23. System.out.println("增加了学生:::" + studentAddEvent.getStudentName());
24. }
26. }

**3.  定义StudentAddBean触发StudentAddEvent事件**

新建StudentAddBean类，实现接口

org.springframework.context.ApplicationContextAware

中的setApplicationContext方法，在构造bean的时候注入Spring的上下文对象，以便通过Spring上下文对象的publishEvent方法来触发StudentAddEvent事件，具体代码如下：

**[java]** [view plaincopy](http://blog.csdn.net/wgw335363240/article/details/7202320)

1. **package** com.trs.spring.event;
3. **import** org.springframework.beans.BeansException;
4. **import** org.springframework.context.ApplicationContext;
5. **import** org.springframework.context.ApplicationContextAware;
6. **import** org.springframework.context.support.ClassPathXmlApplicationContext;
8. **public** **class** StudentAddBean **implements** ApplicationContextAware {
9. /\*\*
10. \* 定义Spring上下文对象
11. \*/
12. **private** ApplicationContext m\_applicationContext = **null**;
14. /\*
15. \* (non-Javadoc)
16. \*
17. \* @see
18. \* org.springframework.context.ApplicationContextAware#setApplicationContext
19. \* (org.springframework.context.ApplicationContext)
20. \*/
21. **public** **void** setApplicationContext(ApplicationContext \_applicationContext)
22. **throws** BeansException {
23. **this**.m\_applicationContext = \_applicationContext;
25. }
27. /\*\*
28. \* 增加一个学生
29. \*
30. \* @param \_sStudentName
31. \*/
32. **public** **void** addStudent(String \_sStudentName) {
33. // 1.构造一个增加学生的事件
34. StudentAddEvent aStudentEvent = **new** StudentAddEvent(
35. m\_applicationContext, \_sStudentName);
36. // 2.触发增加学生事件
37. m\_applicationContext.publishEvent(aStudentEvent);
38. }
40. /\*\*
41. \* @param args
42. \*/
43. **public** **static** **void** main(String[] args) {
44. String[] xmlConfig = **new** String[] { "applicationContext.xml" };
45. // 使用ApplicationContext来初始化系统
46. ApplicationContext context = **new** ClassPathXmlApplicationContext(
47. xmlConfig);
48. StudentAddBean studentBean = (StudentAddBean) context
49. .getBean("StudentAddBean");
50. studentBean.addStudent("我是第一个学生");
51. studentBean.addStudent("第二个学生已经添加");
53. }
55. }

**[java]** [view plaincopy](http://blog.csdn.net/xingtianyiyun/article/details/7388098)

1. **package** com.trs.spring.event;
3. **import** org.springframework.beans.BeansException;
4. **import** org.springframework.context.ApplicationContext;
5. **import** org.springframework.context.ApplicationContextAware;
6. **import** org.springframework.context.support.ClassPathXmlApplicationContext;
8. **public** **class** StudentAddBean **implements** ApplicationContextAware {
9. /\*\*
10. \* 定义Spring上下文对象
11. \*/
12. **private** ApplicationContext m\_applicationContext = **null**;
14. /\*
15. \* (non-Javadoc)
16. \*
17. \* @see
18. \* org.springframework.context.ApplicationContextAware#setApplicationContext
19. \* (org.springframework.context.ApplicationContext)
20. \*/
21. **public** **void** setApplicationContext(ApplicationContext \_applicationContext)
22. **throws** BeansException {
23. **this**.m\_applicationContext = \_applicationContext;
25. }
27. /\*\*
28. \* 增加一个学生
29. \*
30. \* @param \_sStudentName
31. \*/
32. **public** **void** addStudent(String \_sStudentName) {
33. // 1.构造一个增加学生的事件
34. StudentAddEvent aStudentEvent = **new** StudentAddEvent(
35. m\_applicationContext, \_sStudentName);
36. // 2.触发增加学生事件
37. m\_applicationContext.publishEvent(aStudentEvent);
38. }
40. /\*\*
41. \* @param args
42. \*/
43. **public** **static** **void** main(String[] args) {
44. String[] xmlConfig = **new** String[] { "applicationContext.xml" };
45. // 使用ApplicationContext来初始化系统
46. ApplicationContext context = **new** ClassPathXmlApplicationContext(
47. xmlConfig);
48. StudentAddBean studentBean = (StudentAddBean) context
49. .getBean("StudentAddBean");
50. studentBean.addStudent("我是第一个学生");
51. studentBean.addStudent("第二个学生已经添加");
53. }
55. }

**4.  applicationContext.xml配置文件**

|  |
| --- |
| <bean id="StudentAddBean" class="com.trs.spring.event.StudentAddBean"></bean>  <bean id="StudentAddListener" class="com.trs.spring.event.StudentAddListener"></bean> |

**5.  说明**

ApplicationContext在运行期会自动检测到所有实现了ApplicationListener的bean对象，并将其作为事件接收对象。当ApplicationContext的publishEvent方法被触发时，每个实现了ApplicationListener接口的bean都会收到ApplicationEvent对象，每个ApplicationListener可根据事件类型只接收处理自己感兴趣的事件，比如上面的StudentAddListener只接收StudentAddEvent事件。

**6.  执行StudentAddBean的main函数，结果如下：**

|  |
| --- |
| 增加了学生:::我是第一个学生  增加了学生:::第二个学生已经添加 |