

Acegi Security System for Spring Framework



Acegi
**SECURITY
SYSTEM**
FOR SPRING

Part 2
Securing
Method Invocation

Outline

- Intro
- Case Study
- Business Interface
- Business Implementation
- Servlet Controller
- Protecting Business Method
- Proxying Business Object
- Conclusion

Introduction to Acegi



Acegi ++

- ❑ Acegi Security System for Spring
 - <http://acegisecurity.sourceforge.net>
- ❑ Security control in various granularity
 - Securing URL
 - Securing Method Invocation
 - Securing Object Instance
- ❑ Non-invasive architecture
 - Servlet Filter based
- ❑ Modular Architecture
 - Pluggable ACL backend
 - Pluggable authentication mechanism
 - Integration with Application Server

Alternatives

- ❑ Seraph: <http://opensource.atlassian.com/seraph/>
- ❑ jSai - Servlet Security : <http://oss.ipov.org/jsai/>
- ❑ Gabriel : <http://gabriel.codehaus.org/>
- ❑ JOSSO : <http://www.josso.org/>
- ❑ Kasai: <http://www.manentiasoftware.com/kasai/goToHome.action>
- ❑ jPAM : <http://jpam.sourceforge.net/>
- ❑ OpenSAML : <http://www.opensaml.org/>
- ❑ JAAS
- ❑ Custom Implementation

Acegi and Spring

- ❑ Acegi use Spring's Bean Proxy to intercept method invocation
- ❑ Acegi supports AspectJ's Join Point (requires bytecode manipulation by AspectJ compiler)
- ❑ AspectJ's Join Point can be configured via Spring as well

Security Granularity

- URL authorization
 - <http://apps:8080/index.htm> -> for public
 - <http://apps:8080/user.htm> -> for authorized user
- Method Invocation
 - public void getData() -> all user
 - public void modifyData() -> supervisor only
- Object instance
 - order.getValue() < \$100 -> all user
 - order.getValue() > \$100 -> supervisor only
 - customer.getName() == endy -> account mgr only

Non-invasive architecture

- ❑ Use proxy to intercept method invocation and apply security constraints
- ❑ Business interface can be clean, not cluttered with security logic
- ❑ Users of Spring's declarative transaction should find security interception familiar

Convolutated Business Interface

□ Avoid this :

```
public void approve(Order o, User u)
throws IllegalAccessException
{
    if (o.getValue() > 100) {
        if (!"supervisor".equals(u.getRole())) {
            throw new IllegalAccessException();
        }
    }
    o.setStatus("approved");
}
```

Clean Business Interface

- Should be like this :

```
public void approve(Order o)
{
    o.setStatus("approved");
}
```

Case Study

- We will secure method invocation
- User can create Purchase Order
 - public void create(PurchaseOrder po)
- Supervisor can approve Purchase Order
 - public void approve(PurchaseOrder po)
- These requirement to be implemented declaratively

Business Classes

- PurchaseOrder
 - Simple JavaBean
- BusinessFacade
 - Interface for business methods
- BusinessFacadeInMemoryImpl
 - Simple implementation of BusinessFacade
 - Store PurchaseOrder objects in HashMap

PurchaseOrder.java

```
public class PurchaseOrder {  
    private String description;  
    private String type;  
    private double value;  
    private String status = "posted";  
    // getter and setter code  
    // not displayed  
}
```

BusinessFacade.java

```
public interface BusinessFacade {  
    public void createPurchaseOrder  
        (PurchaseOrder po);  
    public void approvePurchaseOrder  
        (PurchaseOrder po);  
    public PurchaseOrder getPurchaseOrder  
        (String description);  
    public List getAllPurchaseOrder ();  
}
```

BusinessFacadeInMemoryImpl.java

```
public class BusinessFacadeInMemoryImpl implements BusinessFacade {
    private Map purchaseOrders = new HashMap();
    public BusinessFacadeInMemoryImpl() {
        // initialize some sample data
    }
    public void createPurchaseOrder(PurchaseOrder po) {
        purchaseOrders.put(po.getDescription(), po);
    }
    public void approvePurchaseOrder(PurchaseOrder po) {
        Assert.notNull(po);
        PurchaseOrder px = (PurchaseOrder)
            purchaseOrders.get(po.getDescription());
        if (px == null) return;
        px.setStatus("approved");
    }
    public List getAllPurchaseOrder() {
        Iterator it = purchaseOrders.keySet().iterator();
        ArrayList result = new ArrayList();
        while(it.hasNext()) {
            result.add(purchaseOrders.get(it.next()));
        }
        return result;
    }
    public PurchaseOrder getPurchaseOrder(String description) {
        return (PurchaseOrder) purchaseOrders.get(description);
    }
}
```

Servlet Controller

- ❑ LoginController
 - Display login form
 - Do not check username and password, as it has been handled by Acegi
- ❑ LogoutController
 - Invalidate session and redirect to default page
- ❑ ListController
 - Invoke BusinessFacade.getAllPurchaseOrders
 - Render template (list.htm)
- ❑ Create Controller
 - Display create.htm page (default)
 - Invoke BusinessFacade.createPurchaseOrder (upon form submission)
- ❑ ApproveController
 - Display approve.htm
- ❑ ApproveProcessController
 - Invoke BusinessFacade.approvePurchaseOrder

ListController.java

```
public class ListController implements Controller {

    private BusinessFacade businessFacade;

    public ModelAndView handleRequest(HttpServletRequest req,
        HttpServletResponse res) throws Exception {
        List allPurchaseOrders =
            businessFacade.getAllPurchaseOrder();

        Map model = new HashMap();
        model.put("allOrders", allPurchaseOrders);

        return
            new ModelAndView("list", "allOrders", allPurchaseOrders);
    }
    public void setBusinessFacade(BusinessFacade businessFacade) {
        this.businessFacade = businessFacade;
    }
}
```

CreateController.java

```
String desc = req.getParameter("description");  
String value = req.getParameter("value");  
String type = req.getParameter("type");
```

```
PurchaseOrder po = new PurchaseOrder();  
po.setDescription(desc);  
po.setValue(Double.parseDouble(value));  
po.setType(type);
```

```
businessFacade.createPurchaseOrder(po);
```

ApproveProcessController.java

```
PurchaseOrder po =
    businessFacade.getPurchaseOrder(desc);
if (po == null) {
    res.sendRedirect("list.htm");
    return null;
}
businessFacade.approvePurchaseOrder(po);
```

Protecting Business Methods

```
<bean id="businessMethodSecurity"  
      class="net.sf.acegisecurity.intercept.method.  
            aopalliance.MethodSecurityInterceptor">  
  <property name="authenticationManager">  
    <ref bean="authenticationManager" />  
  </property>  
  <property name="accessDecisionManager">  
    <ref local="businessAccessDecisionManager" />  
  </property>  
  <property name="objectDefinitionSource">  
    <value>  
      BusinessFacade.approve*=ROLE_SUPERVISOR  
      BusinessFacade.create*=ROLE_USER  
    </value>  
  </property>  
</bean>
```

businessAccessDecisionManager

```
<bean id="businessAccessDecisionManager"  
      class="net.sf.acegisecurity.  
          vote.AffirmativeBased">  
  
    <property name="allowIfAllAbstainDecisions">  
      <value>>false</value>  
    </property>  
    <property name="decisionVoters">  
      <list>  
        <ref bean="roleVoter"/>  
      </list>  
    </property>  
</bean>
```

Proxying Business Object

```
<bean id="businessFacade"  
      class="org.springframework.aop.  
            framework.ProxyFactoryBean">  
  <property name="proxyInterfaces">  
    <value>  
      tutorial.acegi.business.BusinessFacade  
    </value>  
  </property>  
  <property name="interceptorNames">  
    <list>  
      <idref bean="businessMethodSecurity"/>  
      <idref local="businessFacadeTarget"/>  
    </list>  
  </property>  
</bean>  
<bean id="businessFacadeTarget"  
      class="BusinessFacadeInMemoryImpl"/>
```

Conclusion

- ❑ Aop can protect method invocation transparently (no modification in business code)
- ❑ Method interception is made possible using Spring's bean proxy mechanism (similar to transaction proxy)
- ❑ As alternative to Spring's proxy, we can use AspectJ join point (need AspectJ compiler)

Thank you



Endy Muhardin

<http://endy.artivisi.com>

last updated : 20050521