J2EE Technologies Research Project

CS526 Advanced Internet and Web Systems
University of Colorado, Colorado Springs

Enoch Leung
15th May, 2006

Abstract

J2EE had became a major enterprise application platform in 6 years since its initial release. This paper discuss the basic of current J2EE technologies and the security features of the main functionalities it provides.

Introduction

Servlet was introduced in June 1997 and J2EE was announced in June 1999 with the first version shipped in December 1999. Servlet and EJB as the core components of the proposed J2EE architecture. This relatively new architecture had been adopt in many enterprises and is widely used today. It provides enterprises a solution to introduce services using the Internet in a fast an efficient way. Meanwhile, broadband connection adoption is growing up in th world. Broadband penetration as of Q3 2005 is 73.03% in Hong Kong and 67.32% in South Korea [1]. This growth trend reflects the adoption of web technologies like with J2EE from Sun Microsystem and .NET from Microsoft, and it also make computer virus, worms and hacking activities grow well. Network computing security became an increasing to the survival of these web technologies. This paper discuss the basic structure of J2EE, special features in commercial offerings of J2EE application servers and the security aspects of J2EE technologies.

J2EE Components

J2EE application servers are called web application servers also. A J2EE compliance server is a collection of J2EE components. The major components of the latest J2EE 1.4 are Enterprise JavaBeans (EJB), Java Servlet, Java Messaging Service (JMS), Java API for XML-RPC (JAX-RPC), Java

Connector Architecture (JCA), Java Authorization Contracts for Container (JACC), Java Transaction API (JTA) and Java Management Extensions (JMX). Many of them create resources like database connection and are provided to the developers through the JNDI interfaces.

EJB is the collection of entity beans, session beans and message-driven beans. Basic entity bean can be viewed as the instance of a row of table in a database. However, it is possible to create an entity bean as a result of join query to the database across tables. There are two types of entity beans known as bean-managed persistence (BMP) and container-managed persistence (CMP). Implementation of BMP requires the developer to implement code for load and store of the beans. CMP delegates the task of loading and storing the bean to the container, and possibly caching the beans as well. CMP requires the use of the EJB-QL, which is similar to SQL. With CMP we also have container-managed relationship (CMR), which defines how entity beans are related to each other without managing it in code [2].

Session beans is where business logic resides as defined in the J2EE architecture. There are also two type of session beans, namely stateful and stateless. The stateful session beans keep conversational information between the client and the bean while stateless session beans simply provide a service exposed to the client. Stateless session beans provides an alternative to HTTP sessions provided by the Servlet API one is above to skip the web interface an develop a desktop application that use EJB. Message-driven bean is introduced in EJB 2.0 and it is based on JMS. The bean construction is similar to a stateless session bean with the difference that its creation is automatic on message received, while entity and session beans have to be created explicitly though a stub.

JMS is another important component in the J2EE specification. Enterprise applications normally rely on message-oriented middleware (MOM) to form communication between components, new and old. JMS works as an interface to these MOM and form the bridge between J2EE applications and legacy components. JAX-RPC provide an interface to Java XML processing as remote procedure calls, and it form the basis of web services on J2EE platforms. JCA and JTA are database related technologies, and JCA is also used sometime for basic authentication in spite of the availability of Java Authentication and Authorization Service (JAAS) with J2SE. JACC is a new component in J2EE 1.4 specification as an interface to external authorization policy management components that exists in the enterprise infrastructure already. At last, JMX, even though it is J2SE based, is the technology used in J2EE for managing J2EE specific resources.

J2EE Application Deployment

J2EE application is packaged into different Java Application Archive (JAR) files. Servlet and JSP are packaged into a specific JAR file called Web Application Archive. EJB resources are packaged into regular JAR files. An Enterprise Application Archive (EAR) is created for application deployment, which contains one or more of the fore mentioned archives. In each of these archives, a specific deployment descriptor is included. A deployment descriptor is an XML file describing the content of the archive. For instance, in an EJB-JAR file the deployment descriptor will specify, when appropriate, the CMP and CMR properties of entity beans. The format of deployment descriptor to different archive format is also different, for they serve different purpose.

A basic structure of an application server consist of the web container that deploy the WAR files and an EJB container that deploy the EJB-JAR files in an EAR file. The stub for the EJB are loaded into the application server's JNDI registry for lookup during deployment. Other application resources like database connections are provided by the container and is loaded to the JNDI registry by the application server independently.

Web application servers support hot deploy. This feature allow an web application to be updated on the fly without restarting the server. Web application servers also support multiple web applications in runtime. These functionalities are achieved by using specialized class loaders. Class loader is a basis and important component in a Java Virtual Machine (JVM) and is known as java.lang.ClassLoader. When a JVM starts, it will create a system class loader. When a class is being used by instantiation or static method call, this class is loaded to memory by the class loader and it will keep an reference to it. Be default, there is no way to remove a class from memory until the class loader is destroyed. Class loader is also hierarchical. The base class loader of JVM is called a system class loader, and subsequence class loader can be loaded on top of it. J2EE servers take advantage of this functionality by deploying each J2EE applications with an independent class loader. When the application is removed or require a redeployment, the class loader associated with this application is also removed. This is important because most of the time redeployed application features a new version of the same class that is loaded into memory already. Without unloading the class, the JVM will throw an exception, and the application will not be deployed.

Web Application Server on the market

Commercial web application servers are available, and the widely used ones are IBM WebSphere and BEA WebLogic. There is an open source freeware alternative to the two offerings called JBoss. These three are full featured web application servers that supports the complete set of J2EE 1.4 specification plus additions and enhancement.

IBM WebSphere features a strong integration with the Eclipse IDE and formed a complete developer solution. It also support hot and delta deploy within Eclipse IDE with easy to use interfaces. In addition, it also support runtime profiling of web application through the Java Virtual Machine Tool Interface (JVMDI) and embedded Tivoli Access Lite. It also have multiple user registries though Tivoli Access Manager. At last, it also features JSR-168 support through by including an extension to the open-source Servlet extensions called Apache Struts.

BEA WebLogic also features integration with the Eclipse IDE and support JSR-168 with its proprietary technologies. In addition, it also support Java Server Faces (JSF) and support multiple user registries right out of box. JBoss also features Eclipse IDE integration with their JBoss Eclipse-IDE, a plug-in package to Eclipse. Off all features, Aspect-Oriented Programming package is the significant additions to JBoss where the other two commercial offerings do not provide.

Servlet and EJB security

J2EE application security is managed by the application server through JAAS. Security is broken down into the tasks of authentication and authorization. Authentication is an action that verify the identify of the requester, and authorization is an action that verify the permission of the request at the calling time.

JAAS specify the structure and API of authentication and authorization in Java. Authentication is carried out by a class implementing the javax.security.auth.login.LoginModule interface [3]. Developer can specify different callbacks that let the end user input their login informations, which is not restricted to username and password only [4]. When login is successful, a Subject object is created and it retrievable by the application. This Subject object is associated with zero or more Principal object, which represent the identities of the caller. These Principal objects are used in role-based authorization within an J2EE application. In simplified terms, one can view the relationship as if Subject is the identity of the caller, and Principals of the Subject are the roles of the caller. It is also possible to chain login modules such that when a caller tried to be authenticated, the login information is passed to different login modules and be authenticated at the same time. There are four JAAS authentication level being known as required, sufficient, requisite and optional [5]. With difficult authentication level and multiple login modules, it is possible to require different level of authentication for different resources. For instance, with one resource administrator can define that authentication with LDAP as required but optional with ActiveDirectory. With another resource, it can be defined as required to pass both authentication. Code snippet for JBoss JAAS module can be found in appendix.

Servlet security is developed based on the HTTP protocol. There are four forms of them, namely BASIC, DIGEST, Form and credential. BASIC, DIGEST and credential forms of authentication and authorization are the same as defined in HTTP. Form based authentication and authorization is done through an HTTP POST request to the default action j_security_check with the parameter j_username and j_password. Since HTTP POST is inherently insecure, it is necessary to provide the login form with HTTPS. In association with JAAS as mentioned above, one can pass the username and password retrieved from the HTTP POST request and forward it to the JAAS layer for authentication, and it is tied with HTTP session [5]. Once authenticated, it is possible to verify the user by using the method calls isUserInRole() and getUserPrincipal() defined in the HttpServletRequest interface.

EJB security is implemented with JAAS as mentioned above. With Servlet working as a front-end to EJB, security is handled by Servlet as mentioned above. Security information will be passed from Servlet to EJB, and it is retrievable similarly with the method calls isCallerInRole() and getCallerPrincipal() defined in the EJBContext interface. When there is no Servlet serving as a front-end to EJB like traditional desktop applications, EJB security relies on proprietary implementations that work over RMI/IIOP and specific JAAS modules plus callbacks [6]. For instance, IBM WebSphere use login tokens that send though RMI/IIOP for authentication and authorization when it is necessary. In the Java standalone application, developers can use IBM WebSphere library to do authentication. Role definition for authorization are defined in the deployment descriptors of EJB-JAR and WAR respectively.

As an alternative to JAAS, it is possible to create authentication and authorization using JCA, which was introduced in JDBC 3.0. The basic idea is that authentication and authorization with database is also based on username and password, so it is possible to treat resource as database resource and adopt a common J2EE interface instead of relying on JAAS, which is a J2SE interface.

JACC is a new addition to J2EE 1.4 specification. There are time when resource access policy is managed currently by external application like IBM Tivoli Access Manager. JACC provide an interface to the external application such that J2EE resource can be authorized against the policy defined in these external applications [7]. There is no code modification needed, for the new security information provided is handled by the application server and is available to the J2EE resource by using the same method calls mentioned above.

There are security limitation with J2EE. When EJB is used alone, it is necessary to expose the JNDI directory through RMI such that standalone application can do lookup on resources. This pose a security risk just the same as directory listing not disabled with Apache web server. The use of EJB without Servlet also requires the use of proprietary technologies like client-container which help authenticate the user with the application server. This put the application to vendor lock-in status. Performance degradation is another issue. In a WebSphere pre-sales session with IBM, it is known that enabling security will bring a 30% performance hit on WebSphere 6.0 server. Other application server, while not known, should bear similar performance hit when security is enabled. It is by design that it is not allowed for developers to insert Principal to a Servlet or EJB session, as it is managed by the application server. This limitation forces authentication and authorization to be done in the specific ways defined with Servlet and EJB as mentioned above.

Web Services and J2EE

Web services had became a phenomenon since 2002 when SOAP is finalized in XML syntax. Web services provide a mean for one to access resource in a messaging style using normally HTTP as the transport. SOAP, following the command design pattern described in the classic Design Pattern book, allows SOAP message to be embedded into another SOAP message as attachment. This functionality provide a mean to integrate web services into one. Web services is essentially remote procedure calls in a message based format.

SOAP in J2EE is an evolution from the original XML remote procedure call solution and is supported by the J2EE SOAP with Attachment API for Java (SAAJ) [8]. While it is a J2EE component, SAAJ is available separately as a library in the Java Web Service Developer Pack and it can be used with J2SE. Building on top of XML, SOAP object in Java is an extension to the Document Object Model (DOM) of XML in Java. In turn, manipulation of SOAP message is similar to XML DOM tree in Java. SOAP message consist of an envelop (SOAPEnvelop), an optional header (SOAPHeader), a body (SOAPBody) and optionally attachment (AttachmentPart). On error, a fault element is created, insert to the body and send back to the requester. However, SOAP message should not contain any XML processing instruction [9].

SOAP message top level element is the envelop. The schema of SOAP envelop is defined in W3C, and it should adhere to the soap namespace. SOAP message should also adhere to soap encoding namespace. These are two attributes normally defined with the Envelop tag. SOAP header, when present, is the immediate child node of Envelope that contains specific processing instruction for endpoints. SOAP body represent the actual message of the web service. An error can occur at any end point, including the request sender. When it happens, a Fault node is added to the Body and the SOAP response is return immediately. Response of SOAP is also a SOAP message [9].

For EJB, it relies on the use of JAX-RPC and requires modification to EJB descriptor, mapping.xml for JAX-RPC, webservices.xml of EJB-JAR that describe web services in the EJB-JAR and creation of Web Service Description Language (WSDL) file for the EJB to be exposed. Service endpoint and address are not specified in the web service deployment descriptor (webservices.xml) and is to be defined in application server specific ways. Web services are inherently stateless, so one should use stateless session beans only as web services [10]. For Servlet, it follows the same procedure as of EJB except that it is needed to create a remote interface for the Servlet, while remote interface is part of EJB specification.

XML / WS-security

XML had became the standard in message interchange, and it is an integral part to many applications like J2EE. The increasing security risk created the need to make XML secure. To make interchanging XML secure, it is necessary to ensure that an XML file is transferred securely. In addition, one need a mean to tell the integrity of the XML file as well as the validity of the sender. The transport aspect of these concern is solved by HTTPS, which provides a secure point-to-point transport channel for data interchange. However, it does not provide a mean to tell the identity of the sender, and there is no way to tell if the content is modified at the end points. Consider HTTPS as a postal service which will not inspect mail. However, one cannot tell the sender is really the sender, and one cannot tell if the mail is read and put back to the mailbox before the intended receiver get the mail. There is also a need in using an insecure channel like HTTP to transfer XML with encrypted information such that one can avoid the overhead of encrypting the complete XML message with HTTPS, providing that the information needed encryption is a small portion compare to the complete XML message.

XML digital signature is the solution to the user identity and message integrity [11]. However, the first step is to create a canonicalized XML. XML canonicalization is a process that convert an XML into a standard content-wise identical form. It is necessary because whitespace and indent are common with XML they should serve no meaning to a XML message. While it is true content-wise, encryption will treat them as differences. It will make the signing and encryption process complicated, if not impossible.

XML digital signature specify the Signature tag that contains the signed information. The signed information can either be referenced in the tag, a sibling to the tag, a child to the tag or a parent to the tag. Inside the Signature tag, a SignedInfo tag exists. This tag contains information about how the XML signature is created. For instance, a CanonicalizationMethod tag in SignedInfo is defined and tell the algorithm used to canonicalize the XML resource. Other information like signature method, digest method and value can also be found in the tag. A KeyInfo tag contains the key used to validate the signature [11].

In order to provide a complete security solution to XML, XML encryption is a standard introduced that defines how encrypted information is represent in an XML file in addition to XML digital signature. In the previous postal example, it may be necessary to protect the content in the mail such that even if the mail is tampered, it is impossible to get anything useful out of it. XML encryption allows one to encrypt the whole or only part of a XML message. Following the Public Key Infrastructure (PKI), it is needed to do key exchange. In asymmetric key exchange, sender send his public key to receiver, receiver encrypt his private key using the public key of the sender, send it back to the sender, and the sender decrypt it with his private key. This is done once and later it is possible to refer to the exchanged key in XML encryption [12].

WS-security is the web service security standard defined in 2004 describing the standard to make web services secure. It is build on top of the XML digital signature and the XML encryption standards with the addition of UserName Token Profile 1.0 and X.509 Certificate Token Profile 1.0 of the OASIS group for key exchange [13]. The WS-security standard is also managed by the OASIS group. XML signature is included in the SOAP header section inside a Security tag adhering to the namespace and is normally named wsse. The tag is also defined with the actor and mustUnderstand attributes. The actor attribute refer to the expected handler of the request, and the mustUnderstand attribute specify that the receiving end point must understand SOAP header for processing, which is a requirement since the XML digital signature is embedded in it. The content in SOAP body is encrypted in XML encryption standard, with the key referenced in the XML digital signature which in turn is part of the SOAP header [14].

In J2EE context, WS-security is done though a collection of API. A standalone version is also available for J2SE in the JWSDP called XML and Web Services Security. This collection of API conforms to the XML digital signature and XML encryption standards defined in W3C, which are the Java XML Digital Signature API and the Apache XML-Enc API. However, This collection of API is in Early Access (EA) status and is not frozen for changes. In turn, we can conclude that there is no direct support of web services security in J2EE as of today.

JMS

Java Messaging Service is another import technology in the J2EE specification that is widely used because of its implication to legacy applications. JMS serves as a bridge between J2EE application server and the existing MOM of an enterprise. A MOM is a platform that provide a mean for legacy resources like mainframe applications to communication with newer resources like J2EE applications. It forms the basis of client – server computing in the 1980s to 1990s.

JMS supports two forms of communication with the MOM: queues and topics. A queue is a point-to-point communication channel and a topic is a publisher – subscriber bus. Using a queue, message is send and received by two entities only. With a topic, a message is broadcast to all the subscriber to the topic by the publisher [15]. The main reason of using MOM is that they provide guarantee of message delivery. Traditional ways of communication like TCP/IP and UDP do not have this guarantee because there is no persistence of messages, which is handled by the MOM software. In case of server crash, message could be lost. MOM persist the messages and will attempt redelivery when the server is up and running again. Because of the widespread adoption of MOM in the enterprise, J2EE comes out with JMS and provide a mean to integrate legacy resource into session beans and provide a smooth migration path.

However, the nature of JMS makes it having no security functionality defined with its specification. In turn, JMS is inherently insecure, and security with JMS relies on proprietary implementations. Commercial J2EE application servers like WebLogic provides means to protect all kind of J2EE resources, including JMS queue and topics. As an alternative, one can use Java 2 security to protect JMS as well. One can protect access to JMS resource by protecting the JNDI directory for JMS resource lookup. This is also recommended as JNDI directory allow resolution to resources more than just JMS. On message transportation, it is possible use to HTTPS for security. MOM can be set up such that it requires username and password for login. However, since JMS does not have security defined, it is not possible to integrate it directly with JAAS. Another problem is that even though authentication can be solved by using username and password, role-based authorization is not possible. Looking at message-drive bean which use JMS, a MDB is created upon receive of message. However, the creation of MDB is event (message) driven and the caller identities are not necessary sent along with the message. One programmatic solution to this is to embed authenticated Principal objects with the message as Principal objects are serializable. However, as mentioned in EJB security limitation that it is not possible to associate new Principal objects to the EJBContext, authorization has to be done manually in source code in this case.

Enterprise Service Bus

Enterprise Service Bus (ESB) was a term claimed to be coined by Sonics Software, while other claims that it is coined by analysts at Gartner [16]. ESB propose the use of Service Oriented Architecture (SOA) and provide a new backbone for integration between J2EE applications and other systems though various means. SOA is an architecture style with the goal of achieving loose coupling between interacting software agents. A service is a unit of work done by a service provider to achieve end result desirable to the service consumer. An ESB, as defined by Sonic Software, is the collection of minimally MOM, web services, XML and transformation with schema plus content-based routing. It is an architecture rather than a product specification, so offerings on the market may call themselves an ESB with different features and capabilities. Interaction of J2EE and other resources on the ESB is serviced by MOM, and JMS acts as the adapter of J2EE. Other proprietary resources will have their own messaging adapters such that they can participate in ESB communication. In addition to use messaging as the communication mean, web services is a communication alternative that components use to participate on the ESB.

ESB, while not clearly defined as of now, is a feature that many commercial application servers claims to support. IBM WebSphere supports ESB by their components WebSphere Service Bus and WebSphere Message Broker. BEA WebLogic and other commercial J2EE application servers also supports ESB. Most of the application servers work around on web services as the communication mean such that they can reach the broadest number of clients. However, it is necessary to know that web services, while serving as the communication mean in an ESB, is not recommended for use in communication between applications residing on the same application server because of the high cost of marshaling and unmarshaling of XML files associated with processing web services.

Summary

The advance in web technologies is pushing new features to be implemented in J2EE application servers now and then. J2EE introduced practical application hot deployment that allow dynamic update of service without significant interrupt to the service provided. The introduction of EJB brings the n-tier architecture to the web applications that moved away from the traditional client – server architecture with a finer dissection on presentation layer that the user view, the business logic layer that model the business process and the persistence layer that act as the business data store. Web services created a new way of communication and integration between new and legacy systems. Security, while becoming the main concern in web applications, received its due respect and have been specified in J2EE components and web services to various degree. JMS, web services and ESB also created a new path of enterprise integration based on a communication backbone that is not available before.

References

[1] Worldwide Broadband Survey, http://www.websiteoptimization.com/bw/0601/

[2] Container-managed Rations in 21st Century, http://www.javaworld.com/javaworld/jw-04-2002/jw-0419-cmp.html

[3] Using JAAS for Authentication and Authorization, http://www.mooreds.com/jaas.html

[4] All That JAAS, http://www.javaworld.com/javaworld/jw-09-2002/jw-0913-jaas.html

[5] Security Fundamental, http://edocs.bea.com/wls/docs90/secintro/concepts.html#1122439

[6] Java RMI over IIOP, http://java.sun.com/j2se/1.4.2/docs/guide/rmi-iiop/

[7] JSR 115, http://www.jcp.org/en/jsr/detail?id=115

[8] SAAJ FAQ, http://java.sun.com/webservices/saaj/faq.html

[9] Simple Object Access Protocol 1.1, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[10] Turn EJB components into Web Services, http://www.javaworld.com/javaworld/jw-08-2004/jw-0802-ejbws.html

[11] An Introduction to XML Digital Signature, http://www.xml.com/pub/a/2001/08/08/xmldsig.html

[12] Exploring XML Encryption, http://www-128.ibm.com/developerworks/xml/library/x-encrypt

[13] Web Service Security, http://www-128.ibm.com/developerworks/library/ws-secure/

[14] Secure Web Services, http://www.javaworld.com/javaworld/jw-03-2003/jw-0321-wssecurity.html

[15] A Java Message Service Primer, http://www.onjava.com/pub/a/onjava/2001/05/03/jms_primer.html

[16] Take the Enterprise Service Bus, http://www.ftponline.com/ea/magazine/spring/columns/architectstoolbox/default_pf.aspx

Appendices

JBoss JAAS LoginModule code snippet

```
public final class JBossLoginModule extends AbstractServerLoginModule {

        private static final String EMPTY = " ";
        public void initialize(Subject s, CallbackHandler c, Map t, Map o) {
                super.initialize(s, c, t, o);
        }
        protected Principal getIdentity() {
                // return new SimplePrincipal(username);
                System.out.println("called getIdentity()");
                return new SimplePrincipal("hardcode_moron");
        }
        protected Group[] getRoleSets() throws LoginException {
                System.out.println("called getRoleSets()");
                // decode group by username
                String roleNames = "Web,MyCompany,Echo";
                SimpleGroup roles = new SimpleGroup("Roles");
                SimpleGroup callerPrincipal = new SimpleGroup("CallerPrincipal");
                Group[] roleSets = {roles,callerPrincipal};
                if( roleNames != null )
                {
                        StringTokenizer tokenizer = new StringTokenizer(roleNames, ",");
                        while (tokenizer.hasMoreTokens()) {
                                String roleName = tokenizer.nextToken();
                                roles.addMember(new SimplePrincipal(roleName));
                        }
                }
                callerPrincipal.addMember(new SimplePrincipal("hardcoded!!!"));
                System.out.println("caller principals = hardcoded!!!");
                return roleSets;
        }
        public boolean login() throws LoginException {
                if (super.login()) {
                        // do proper name resolution here
                        return true;
                }
                NameCallback nc = new NameCallback(EMPTY,EMPTY);
                PasswordCallback pc = new PasswordCallback(EMPTY,false);
                Callback[] callbacks = {nc, pc};
                try {
                        callbackHandler.handle(callbacks);
                        String username = nc.getName();
                        char[] password = pc.getPassword();
                        // get username & password okay, now what?
                        NtlmPasswordAuthentication ntpa = new
NtlmPasswordAuthentication("NTLM_DOMAIN","MyUsername","MyPassword");
                        UniAddress addr = new
UniAddress(NbtAddress.getByName("NTLM_DOMAIN"));
                        SmbSession.logon(addr,ntpa);
                        System.out.println("NTLM login success");
                        return true;
                } catch (Exception e) {
                        // do nothing
                }
                return false;
        }
}
```