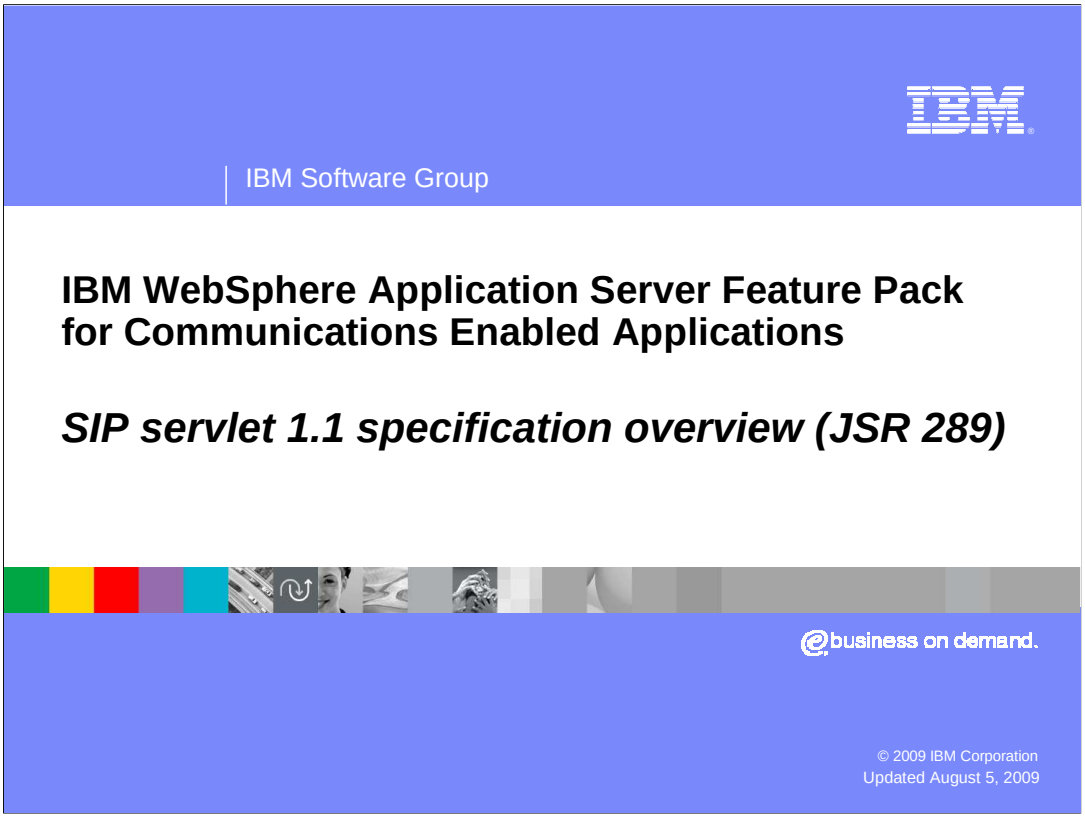


This presentation provides an overview of new features in the JSR 289 specification for SIP servlet 1.1. The IBM WebSphere® Application Server Feature Pack for Communications Enabled Applications (CEA) is compliant with the JSR 289 specification.

Agenda

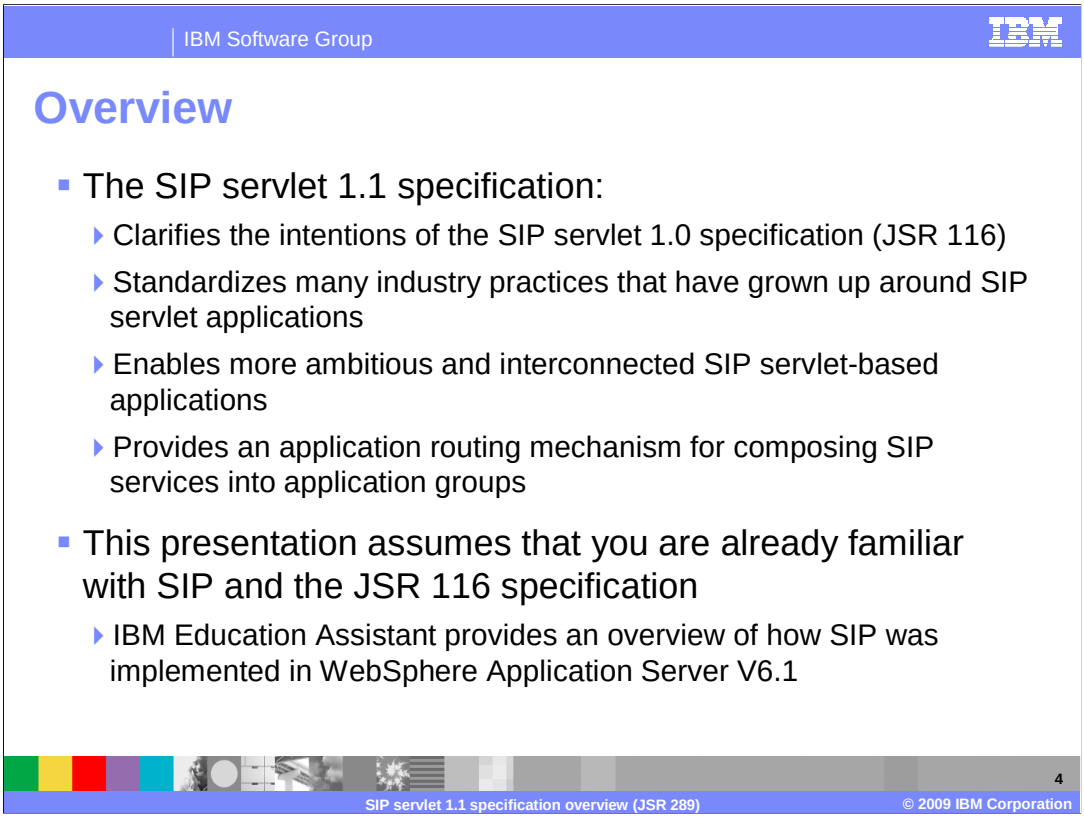
- Overview
- Application router
- Annotation-based development
- Other specification changes

This presentation begins by providing an overview of the goals of the SIP servlet 1.1 specification, and then discusses some of the key new features included in the specification, starting with the application router. The application router provides a flexible mechanism for grouping SIP application components together to provide end-to-end services. The programming model for SIP servlet 1.1, like many other current Java™ specifications, includes annotations to speed up development and simplify application structure and packaging. Other updates in the JSR 289 specification include improved support for converged applications that contain SIP components and other Java EE components, and a B2buaHelper API that simplifies the process for developing the common B2BUA model for SIP applications.

Section

Overview

This section provides an overview of the goals of the SIP servlet 1.1 specification.



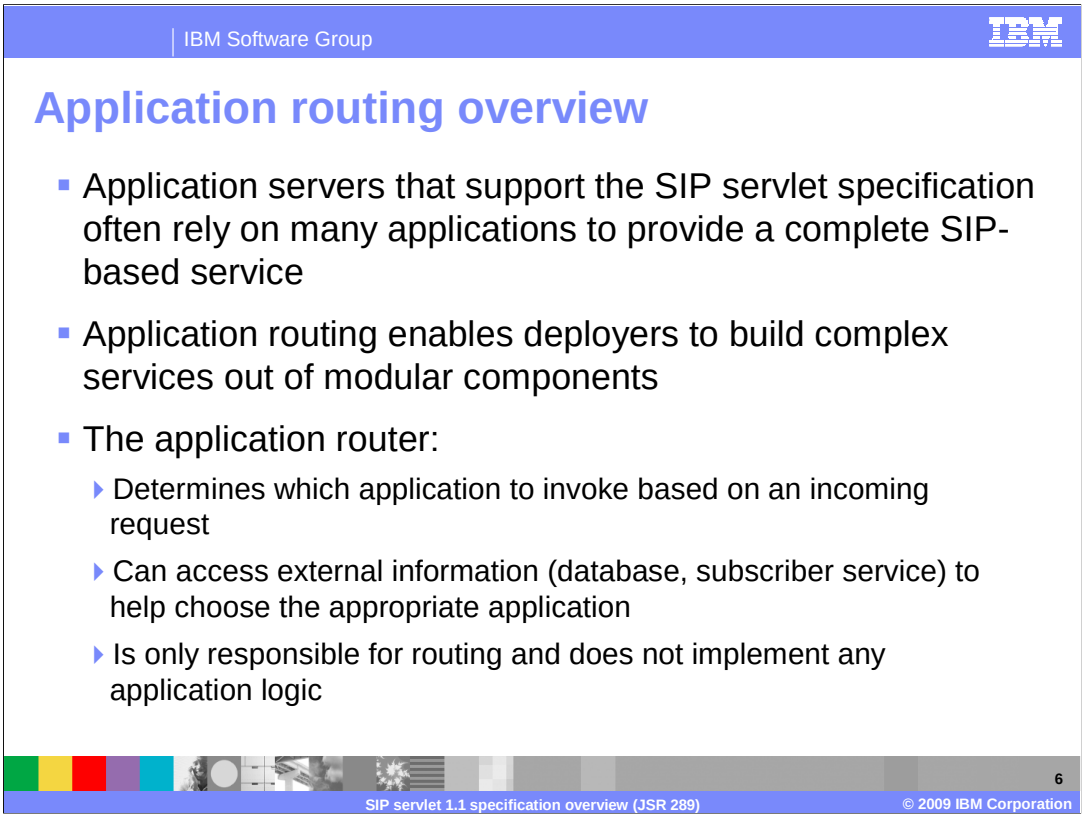
The SIP servlet 1.1 specification builds on the previous 1.0 specification by providing several important new features. In the initial specification, there were several behaviors that were not clearly defined. These intended behaviors are more carefully described in the current specification. Many industry best practices grew up around SIP servlet applications beyond what was covered in the 1.0 specification, and the 1.1 specification attempts to codify many of these best practices. The SIP servlet 1.1 specification also enables developers to create more ambitious and interconnected SIP servlet-based applications, including applications that incorporate both SIP components and other Java EE components, like HTTP servlets and Enterprise JavaBeans.

This presentation does not cover the session initiation protocol (SIP) or the SIP servlet 1.0 specification and assumes that the student already has a basic understanding of SIP. For those interested in a SIP refresher, see the Reference section at the end of this module for a link to SIP overview presentation that describes how the SIP container was initially implemented in the application server.

Section

Application router

This section of the presentation provides an overview of the new application router component that is a part of the JSR 289 specification.



SIP servlet application servers are typically provisioned with many different applications. Each application provides specific functionality, but, by invoking multiple applications to service a call, the deployer can build a complex and complete service. This modular and compositional approach makes it easier for application developers to develop new applications and for the deployer to combine applications from different sources and manage feature interaction. A typical example from traditional telephony is a call-screening application and a call-forwarding application. If the application server receives an incoming INVITE destined to a callee who subscribes to both services, both applications should be invoked.

The application router is a separate component, outside of the SIP container. The container receives initial requests, calls the application router to determine which application to invoke, and then the container calls that application. Once the container has called into an application, that application calls into the appropriate servlet to handle the request, based on the application's configuration – for example, using mappings defined in the application's deployment descriptor. By default, WebSphere Application Server uses application start-up weights to define the routing order. The JSR 289 specification also defines a Default Application Router (DAR) properties file format and a custom application router application format to describe application routing.

Benefits of using application routing

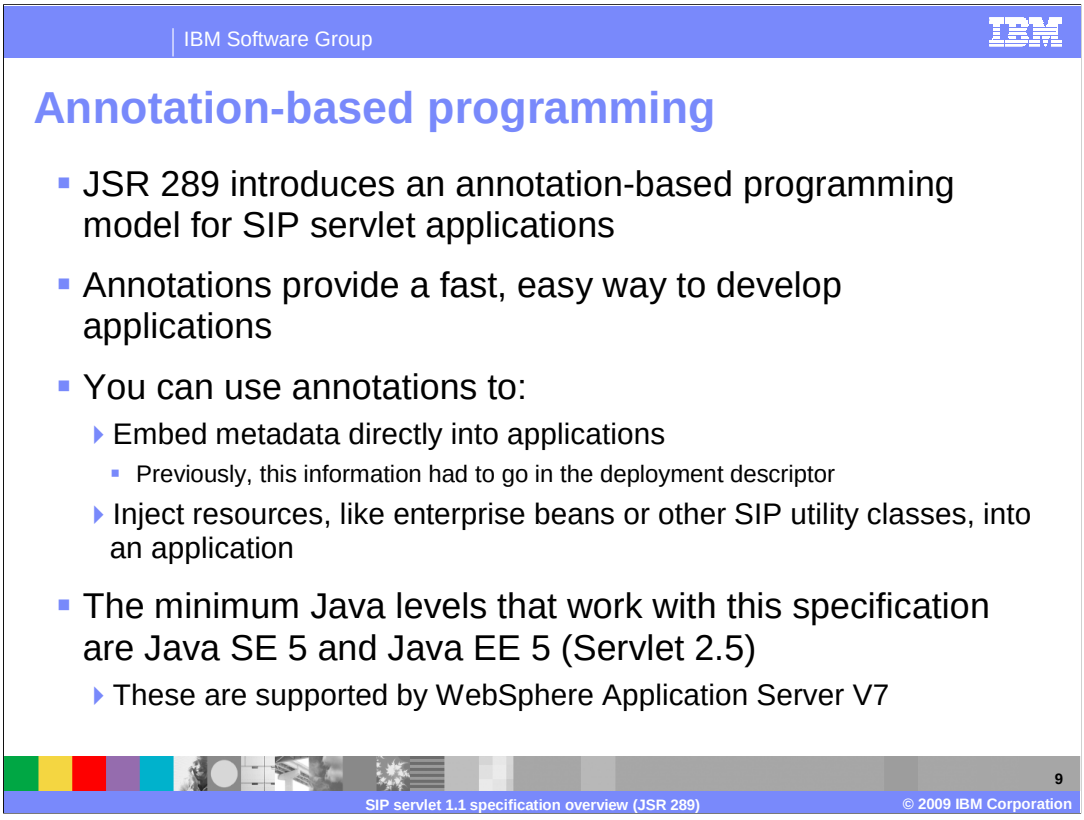
- Simplifies the process of integrating applications to provide rich services
  - Without needing to develop customized wrappers for application components
- Gives the deployer control over application composition
  - Deployer is responsible for end-to-end services
  - Deployer maintains subscriber information that the application router can access

The application router makes it easier to buy a vendor application and invoke its services, without having to write custom wrapper code. This gives the deployer control over how the services behave, rather than leaving integration decisions in the hands of the application developer. Say, for example, you provide telephone service to a large number of subscribers, and a law enforcement agency comes to you with a call tracing and monitoring application that you need to run on a specific subset of your subscribers. Previously, this application was invoked for all subscribers and had to include logic to run only on the required subscribers, or you needed to write an application wrapper to determine whether to invoke the application for a particular user. Now, under the SIP servlet 1.1 specification, all of the logic for determining which users require which application services can be moved outside the scope of the application itself and into the application router.

Section

Annotation-based development

This section provides an overview of the annotation-based programming model introduced in the SIP servlet 1.1 specification.



The SIP servlet 1.1 specification introduces an annotation-based programming model for SIP servlet applications, similar to how annotations are used throughout the Java EE 5 specification. Annotations improve the development experience by simplifying the code being created. Annotations allow you to embed metadata directly into applications, rather than having to use deployment descriptors. Deployment descriptors are still an option, and will override settings described in the annotations, but they are not required.

Resource injection is a simplified model for pulling resources, like SIP utility classes or Enterprise JavaBeans, into an enterprise application, and the new SIP servlet annotations in JSR 289 support resource injection. Because of the use of annotations in the SIP servlet 1.1 specification, to use this specification, you need to be using versions of Java that support annotations – Java SE 5 and Java EE 5, both of which are supported in WebSphere Application Server Version 7.

Section

Other specification changes

The last section of this presentation covers other JSR 289 updates, including improved support for converged applications and the B2buaHelper APIs.

Converged applications

- JSR 289 provides a new, standardized mechanism for building converged applications
  - A converged application contains SIP servlet components and other Java EE components, like HTTP servlets and enterprise beans
  - Resource injection of the SipFactory class allows non-servlet components to access servlet context information
  - Converged applications can be packaged as EAR files
- The specification includes two new classes to support convergence:
  - ConvergedHttpSession: extension to HttpSession for converged applications
  - SipSessionUtil: session management for converged applications

Converged applications contain both SIP servlet components and other Java EE components, like HTTP servlets and Enterprise JavaBeans. The ability to use annotations for resource injection rather than relying purely on ServletContext lookup allows non-servlet components to access information from the SipFactory. IBM supported application convergence in WebSphere Application Server V6.1 using proprietary APIs, and now this convergence model has become the standard in JSR 289. The IBM APIs are still supported, but the recommendation is to move to the new standardized APIs that are a part of the SIP servlet 1.1 specification. The two new classes to support convergence in JSR 289 are the ConvergedHttpSession, which is an extension to HttpSession for converged applications, and the SipSessionUtil class, which provides session management capability for converged applications.

B2buaHelper APIs

- A back-to-back user agent (B2BUA) is a SIP application that sits in the middle of a call, transforming and proxying requests
  - Common pattern in SIP applications
  - Implementations were error-prone
- The new B2buaHelper class makes the B2BUA pattern very easy to implement
  - Ability to create a copy of an incoming request
  - Automatically maintains links between sessions on both sides of the B2BUA

A back-to-back user agent, or B2BUA, is a common pattern in SIP applications. The B2BUA inserts itself into the path of the request by taking in the request, then acting as a user agent server, or UAS, to perform some operation or transformation on the request, and then acting as a user agent client, or UAC, and sending the request on. Previously, the B2BUA had to clone many requests and responses passing through it and make sure that the requests and responses got mapped appropriately back and forth across the call. Implementations of the request mappings were often complicated and error prone. The new B2buaHelper class makes the B2BUA pattern very easy to implement by providing a mechanism to create a copy of an incoming request and automatically maintaining links between sessions on both sides of the call.

B2buaHelper example

- Retrieve the B2BUA helper instance

```
B2buaHelper helper = originalRequest.getB2buaHelper();
```

- Create a new request, based on the original request, and link the requests and their SipSessions together

```
SipServletRequest newRequest =
  helper.createRequest ( originalRequest, true );
```

- Later, access the linked session information

```
doSuccessResponse( SipServletResponse response ) {
    ...
    SipSession otherSession =
    B2buaHelper.getLinkedSession( response.getSession() );
    ...  }
```


The B2buaHelper class instance can be retrieved from a SipServletRequest by invoking the getB2buaHelper() method on it. By making that method call, that indicates to the container that the application is acting as a B2BUA. From that point on, any user agent operation is permitted by the application, but the application can no longer act as a Proxy.

When an application receives an initial request for which it wants to act as a B2BUA, it can invoke the createRequest() method on the B2buaHelper class. This method returns a request that is identical to the one provided as an argument, with the appropriate header fields copied across. By passing in the second argument to the createRequest method as true, the SipSessions are linked together for the original and new SipServletRequests. By linking the sessions together, you might be able to navigate from one to the other. One common function of a B2BUA is to forward requests and responses from one SipSession to another, after performing some transformation or application of business logic. Using linked sessions under the B2buaHelper API, as shown here, simplifies that pattern.

Session key based targeting

- Typically, when an application is invoked, a SipApplicationSession object associated with that application gets created
- Sometimes requests from multiple application calls need to be routed through a single SipApplicationSession instance
  - Session key based targeting allows this behavior
- The @SipApplicationKey annotation identifies the method that generates the session key
  - An application (packaged as a SAR or WAR file) can only contain one @SipApplicationKey annotation

Typically, when an application is invoked, a new SipAplicationSession object gets created and associated with that application. However, sometimes it is required to route all requests for a subscriber, application, or some other combination of factors to a single SipApplicationSession instance. For example, consider a call waiting application. Say that Alice subscribes to the call waiting service, and she's on the telephone, talking with Bob. During the call, Alice's mother tries to call her, so the call waiting application is invoked to handle the request. The call waiting application should have a way to indicate its need to associate with the existing SipApplicationSession for Alice's current call. It's possible to create such an association using a SipApplicationKey. For an application to use session key based targeting, it needs to have one method identified by the @SipApplicationKey annotation that it responsible for generating the session key. Each SipApplicationSession can only be referred to by a single key.

Finding an application session

- When processing an initial request, the container will call the @SipApplicationKey method in an application, if it exists
  - If a SipApplicationSession associated with the key already exists, then that session is used in processing the incoming request

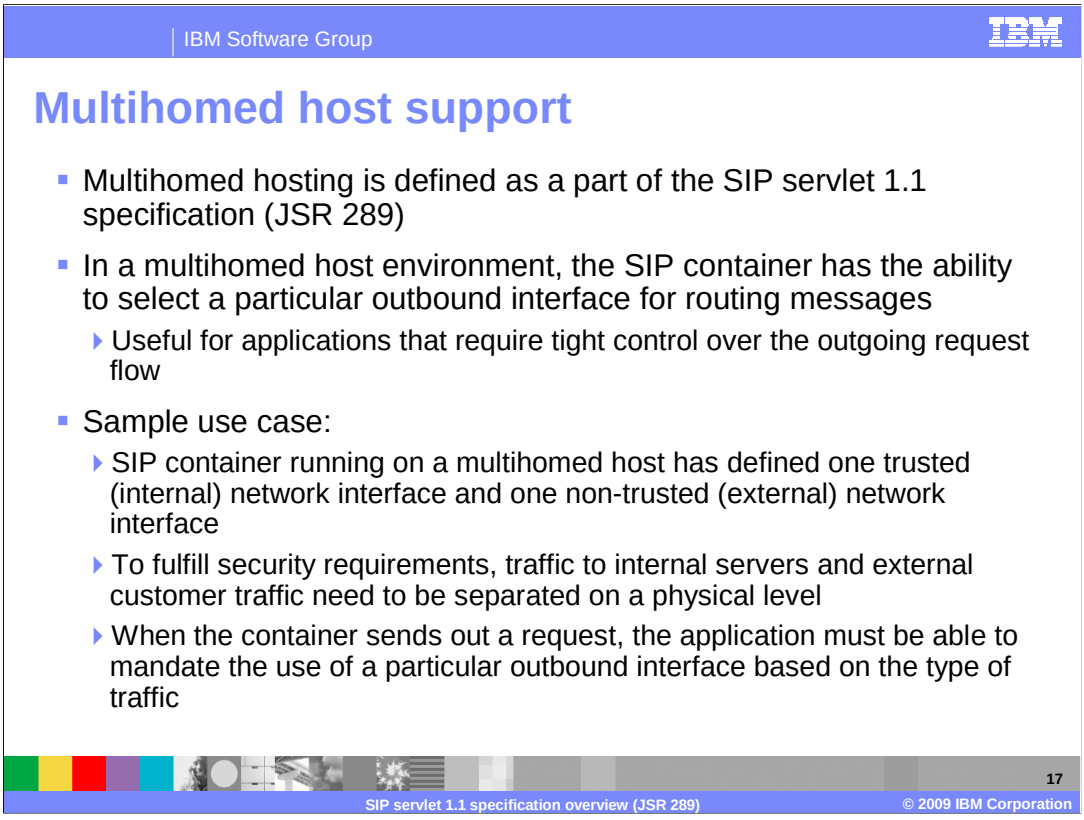
```
@javax.servlet.sip.annotation.SipApplication
package com.ibm.example;
import javax.servlet.sip.annotation.SipApplicationKey;
import javax.servlet.sip.SipServletRequest;
public class ChatRoomMapper {
   @SipApplicationKey
   public static String sessionKey(SipServletRequest req){
      return hash(req.getRequestURI +
                  getDomain(req.getFrom()); }
}
```


When processing an initial request, the container will call the @SipApplicationKey method in an application, if such a method exists. This method takes as a parameter the incoming SipServletRequest, which is used to generate the key. The example here shows a method that has been defined to create an application session key. The method must be a public static method, returning a String, and it cannot modify the incoming SipServletRequest. If the container finds an application session already associated with a particular key, then that session is used in processing the incoming request.

Parameterable interface

- The Parameterable interface allows a SIP header field value to be represented as a parameter holder, rather than a String
  - Modifications to a Parameterable object cause the corresponding header field in the underlying message to be modified
- The Address class now implements the Parameterable interface
- The SipServletMessage class added new methods to support Parameterable header types
- The SipFactory class has a new method to parse Parameterable header types

The Parameterable interface allows a SIP header field value to be represented as a parameter, rather than as a String. Having the ability to access Parameterable fields in a parsed form is more convenient and allows for better performance than accessing those header fields directly as Strings. Modifying a Parameterable object causes the corresponding header field in the underlying object to be modified. The Address class now implements the Parameterable interface, and the SipServletMessage and SipFactory classes have new methods to support Parameterable types.



Multihomed hosting is defined as a part of the SIP servlet 1.1 specification, JSR 289. In a multihomed host environment, the SIP container has the ability to select a particular outbound interface for routing messages. This is useful for applications that require tight control over the outgoing request flow. For example, consider a topology in which the SIP container running on a multihomed host has defined one trusted network interface and one non-trusted network interface. The trusted interface is for the internal network, and the non-trusted interface is for the external, or customer-facing, network. To fulfill security requirements, traffic to internal servers must be separated on a physical level from external customer traffic. In this context, when the SIP container sends out a request, the application must be able to mandate the use of a particular outbound interface based on the type of traffic. Using the new multihomed hosting APIs, the application can be written to do just that.

Using multihomed hosting

- To use multihomed hosting:
  - The SIP servlet application must implement the APIs for multihomed support in JSR 289
    - List of outbound interfaces is maintained by the SIP container and available to applications through a context attribute
    - The application must set the interface on the Proxy, the ProxyBranch, or the SipSession object before sending any outbound requests
    - The container sees the interface attribute and notifies the proxy which outbound interface to send the outbound request on
  - The WebSphere Application Server SIP proxy must be configured with the appropriate outbound interfaces
    - Multihomed hosting is configured at the proxy level, not the SIP container level, and is only supported in a network deployment environment

Using multihomed hosting requires both application changes and configuration changes. The SIP servlet specification 1.1 includes new APIs for multihomed support, and any application wanting to take advantage of multihomed hosting needs to use these new APIs. The APIs make available a list of outbound interfaces that is maintained by the SIP container and available to applications through a context attribute. The application must set the interface on the Proxy, the ProxyBranch, or the SipSession object before sending any outbound requests. The container sees the interface attribute and notifies the proxy which outbound interface needs to be used to send the outbound request. In order to take advantage of multihomed hosting, the SIP proxy must be configured with the appropriate outbound interfaces. Multihomed hosting is configured at the proxy level, not the SIP container level, so a multihomed topology is only supported in a network deployment environment. The next two sections of the presentation describe the multihomed hosting APIs and SIP proxy configuration in more detail.

Section

Summary and reference

This section contains a summary and references.

Summary

- JSR 289 introduces several enhancements to SIP servlets:
  - Application routing
  - Annotation-based development
  - API changes to simplify development

The JSR 289 specification introduces several new features for SIP applications. Application routing provides a mechanism for removing application composition logic from applications and simplifying integration of application components. Annotations speed up SIP servlet application development and reduce the need for deployment descriptors. Other API changes, like improved support for converged applications and the B2buaHelper class, also simplify application development.

Reference

- JSR 289 specification

  http://jcp.org/aboutJava/communityprocess/final/jsr289/index.html

- SIP overview in IBM Education Assistant

  http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.was_v6/was/6.1/Architecture/WASv61_SIP_overview/player.html

This page contains a link to the official JSR 289 specification document, and a general SIP education module for WebSphere Application Server V6.1 that is available on IBM Education Assistant.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_CEAFP_JSR289Overview.ppt

This module is also available in PDF format at: ../CEAFP_JSR289Overview.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Enterprise JavaBeans, Java, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.