JavaOne℠

Sun's 2000 Worldwide Java Developer Conference℠

JavaOne℠

Sun's 2000 Worldwide Java Developer Conference℠

Tomcat at Jakarta.Apache.ORG

Anil Vijendran, Mandar Raje, Craig McClanahan

Java™ Software,

Sun Microsystems, Inc.

Session Overview

- Introduction
- Tomcat:
  The Servlet Container
- Jasper:
  The JSP™ Technology-based Engine
- Future Directions:
  The Catalina Architecture

Introduction

- Goals of the Jakarta project
- Projects under the Jakarta umbrella
- Where are we now at Jakarta?

Jakarta Project Goals

- Ensure widespread availability of Java™ technology-based web technologies, mainly Servlets and JavaServer Pages™ technology; Tomcat as a reference implementation
- Build a development community around Tomcat and evolve Tomcat into a high quality servlet engine
- Source availability under Apache License to promote easy embedding in commercial products

Projects Under the Jakarta Umbrella

- Tomcat—Servlet container
  - Core support for Servlet 2.2+
  - Jasper 1.1+ engine, for JSP™ technology
  - Native connector(s) to Apache, IIS, NES
- Watchdog—Compliance test suite for the JSP and servlet APIs
- Jakarta Taglibs—a public repository of JSP 1.1 technology-based tags

Projects Under the Jakarta Umbrella

- Ant—Java™ technology-based build tool
- Regexp—Regular expression processing
- Slide—WebDAV support
- Struts—Model-View-Controller (MVC) based framework for web applications

Where Are We Now at Jakarta?

- Tomcat
  - Stable release (04/00): Tomcat 3.1
  - Planned release (07/00): Tomcat 3.2
  - Next major release of Tomcat based on the Catalina architecture
- Key features
  - Support for Java Servlet 2.2 API, JSP 1.1
  - Integrated to run with
    - Apache
    - Microsoft IIS
    - Netscape™ Enterprise Server

JavaOne℠

Sun's 2000 Worldwide Java Developer Conference℠

Tomcat:
The Servlet Container

Anil Vijendran
(akv@Eng.Sun.COM)
Staff Engineer
Sun Microsystems, Inc.

Design Overview of the Tomcat Servlet Engine

- Brief Intro to Servlets
- Tomcat Design Goals
- Request Processing Overview
- Core Abstractions in Tomcat
- Customizing Request Processing—Interceptors
- Tomcat Configuration

Java™ Servlet API—A Quick Introduction

- Java™ Servlet API ("Java Servlets")—Java™ technology-based components that extend web servers
- Managed by a servlet container
- Have their own lifecycle—init(), destroy(), service()
- Loaded/unloaded on demand
- Mapped to URL namespace
- Multi-threaded

Tomcat Design Goals

- Main Design Goals
  - Highly customizable—encourages embedding in other app servers, web servers
  - Keep the “core” small, entirely Java technology and portable
  - Stability and compliance over performance and features
- Non-Goals
  - Replacing other webservers
  - Turn Tomcat into a full-fledged application server

Overview of Request Processing in Tomcat

- HTTP server (Apache) receives a request; finds out it is for a servlet or JSP™ page
- A Connector creates Request/Response objects and calls the ContextManager
  - Connector can use TCP to send the request to the Java technology process or any other method
  - The most common connector is AJP12, used in also in Jserv - with implementations for Apache but also IIS, NES

Overview of Request Processing in Tomcat (Cont.)

- ContextManager starts processing by calling RequestInterceptors to process the request
- Interceptors have the same role as modules in Apache - example: lookup the context, session, do authorization and authentication and find the handler (servlet) for the request
- ContextManager calls the servlet

Core Abstractions in Tomcat

- Connector
- ContextManager
- Container
- Request Interceptors
- Context Interceptors
- Request, Response

Core Abstractions in Tomcat (Cont.)

- Connector
  - Receives requests forwarded from webserver
  - One connector implementation per transport/protocol (example: AJP)
  - Create protocol-specific Request/Response and call ContextManager
- ContextManager
  - Abstract servlet execution/lifecycle from Connector

Core Abstractions in Tomcat (Cont.)

- RequestInterceptor
  - Customize various aspects of request processing—parsing, authorization, authentication, pre/post actions
- ContextInterceptor
  - Notification of various events related to contexts—add/remove web apps, changes to mappings etc

Core Abstractions in Tomcat (Cont.)

- Request/Response
  - Just containers for HTTP request/ response state
  - State variables initialized by connectors and interceptors
- Container
  - A group of URLs that share common properties (handler, security constraints)

Customizing Request Processing—Interceptors

```
class UserHomes extends
BaseInterceptor {
  public int contextMap(req, res) {
    uri=req.getRequestURI();
    if (uri.startsWith( "~" ) ) {
    // extract string following ~
    // create context with a
    //  docbase=/home/user/webapps
      req.setContext(ctx);
   }
}
```

Tomcat Configuration

- Mostly configured by APIs (to allow embeddability)
- For standalone Tomcat
  - Rule-based XML processor
  - Rules associate components of the XML tree with actions
  - These actions call Tomcat's core APIs to configure Tomcat

JavaOne℠

Sun's 2000 Worldwide Java Developer Conference℠

Jasper: The JSP™ Technology-based Engine

Mandar Raje
(mandar@Eng.Sun.COM)
Member of Technical Staff
Sun Microsystems, Inc.

A Brief Introduction to JSP™ Technology

- Generation of dynamic content
- Easier way to write servlets
- Separation of logic from presentation
- Use of tags to minimize scripting
- Better tool support
- Web layer of the Java™ 2 Platform, Enterprise Edition (J2EE™)

JSP Technology Example Page

- A simple JSP Technology printing Request parameters:

```
<html>
   <h1> Request Information </h1>
   request URI:<%= request.getURI()%>
   request method:<%= request.getMethod()%>
</html>
```

An Equivalent Servlet

```
public void service (req, res) {

  out.print ("<html>");
  out.print ("request URI:");
  out.print (request.getURI());
  out.print ("request Method:");
  out.print (request.getMethod());
  .
  out.print ("</html>");
}
```

The Jasper JSP Engine

- Initialization
- Parser and Code Generator
  - Parsing template text
  - Parsing JSP elements
- Compiling generated Servlet
- Debugging
  - Translation-time exceptions
  - Runtime exceptions

Initialization

- Generating a unique class name
  - Avoiding name collisions
  - Using path information in package names
  - Class reloading
- Assigning implicit variables
  - Setting the content type
  - Setting the buffer size
  - Creating context, session (optional)

Parser and Code Generator

- Handling template text
  - Embedded in out.print()
- Handling JSP expressions
  - Embedded in out.print()
- Handling scriptlets
  - Copied verbatim in the service() method
- Handling declarations
  - Define class variables and methods

Parser and Code Generator

- Handling jsp:useBean

```
<jsp:useBean id="foo" scope="request"
     class="myClass" />

myClass foo = pageContext.getAttribute
                    ("foo", REQUEST_SCOPE);
if (foo == null) {
  foo = (myClass) Beans.Instantiate
                    (loader, "myClass");
  pageContext.setAttribute
                    ("foo", foo, REQUEST_SCOPE);
}
```

Parser and Code Generator

- Handling setProperty and getProperty
  - Invoking methods on the beans
- Handling runtime includes and forwards
  - Use underlying Servlet RequestDispatcher mechanism
- Handling custom-tags
  - Parse Tag Library Descriptors (TLDs)
  - Generate code accordingly

Debugging

- Translation-time exceptions
  - Parse-time exceptions are errors in pages

    <jsp:getProperty nme="foo" .... />

    error message correctly points the cause and the location where the error has occurred
  - Compile-time exceptions are errors in the generated Java source code (Servlet)

    <% out.println ("Welcome") %>

    error message is generated by the Java compiler (javac or jikes)

Debugging

- Runtime exceptions
  - NullPointer, ArrayOutOfBound

```
<% String foo = null;
    if (foo.equals(....)) ...  %>
```

  - Error-page directive works only with RuntimeExceptions

```
<% page errorPage="/error.jsp" %>
```

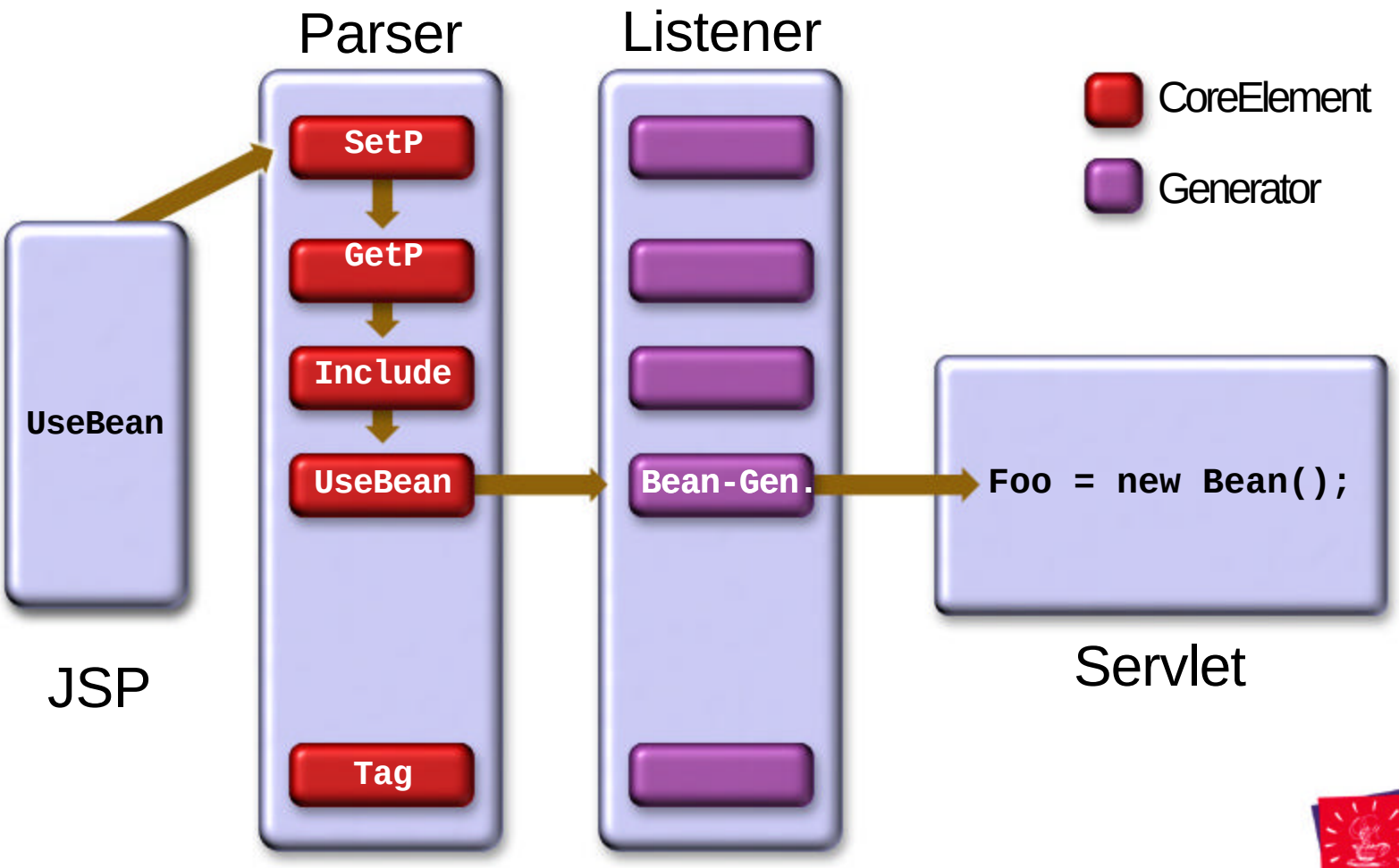
  - Error-page element in deployment descriptor

The Jasper Engine

- The Jasper engine is a servlet by design
- Registered using deployment descriptor

```
<servlet>
    <servlet-name> jsp </servlet-name>
    <servlet-class> JspServlet </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name> jsp </servlet-name>
    <url-pattern> *.jsp </url-pattern>
</servlet-mapping>
```

Jasper Architecture



Compiling Generated Servlet

- JavaCompiler Interface
  - Set encoding, classpath, outputdir
  - JikesJavaCompiler, SunJavaCompiler
  - Can be set through init-parameter
- Handling JSP components with large static data
  - Limitation on the method size
  - Read static data from a separate file
  - Can be set through init-parameter

Future Directions

- Moving to Catalina
- Better debugging support
- Better tool support
- Generating portable servlets from JSP technology pages
- Support for JSP.next

JavaOne℠
Sun's 2000 Worldwide Java Developer Conference℠

Future Directions: The Catalina Architecture

Craig McClanahan
(craigmcc@Eng.Sun.COM)
Staff Engineer
Sun Microsystems, Inc.

Diverse Deployment Environments

- Stand alone “mini-application server”
- Connected to an existing web server
- Integrated in full function application server
- Integrated in a development tool
- Embedded in a hardware device
- Multiple communications protocols

Plug-In Functionality Support

- Internal architecture based on components, described by Java interfaces
- Implementation classes configurable at run time (with appropriate defaults)
- Component configuration based on JavaBeans™ property APIs
- Component lifecycle support and events

Extensible Request Processing

- At various levels
  (server, host, context, servlet)
- Examples:
  - Security (authentication/access control)
  - Customized logging
  - Resource management
  - Transaction support
- Configurable at runtime

Catalina Component Families

- Connector Family
  - *Request* and *Response*
- Container Family
  - *Engine*, *Host*, *Context*, and *Wrapper*
- Session Management Family
  - *Manager* and *Session*
- Utility Components Family
  - *Loader*, *Logger*, *Realm*, and *Resources*

Connector Components

- *Connector*—Represents a communications connection from a remote client
- *Request*—Represents the incoming HTTP request
- *Response*—Represents the outgoing HTTP response

The *Connector* Interface

```
public interface Connector {
    ...
    public Container getContainer();
    public void setContainer(Container container);
    public Request createRequest();
    public Response createResponse();
    ...
}
```

Container Components

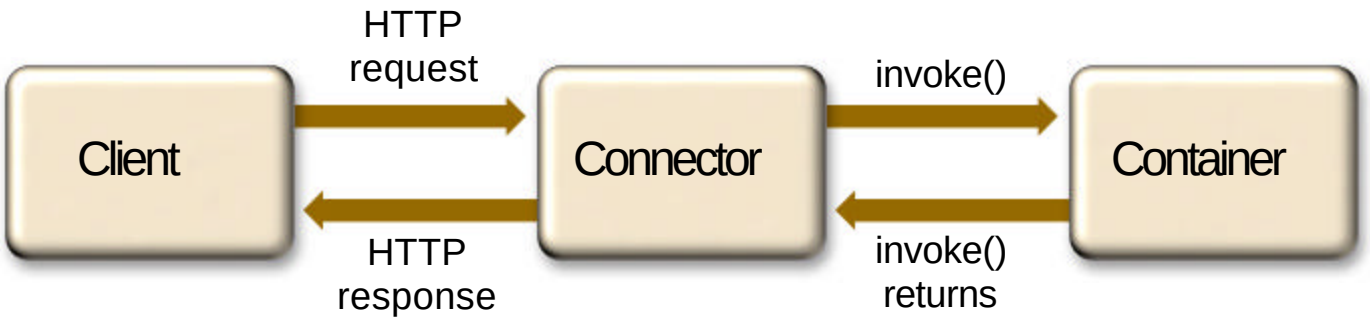
- *Engine*—the entire servlet container
- *Host*—a virtual host
- *Context*—a servlet context
  - Also known as a web application
- *Wrapper*—an individual servlet

All of these components implement the *Container* interface

The *Container* Interface

```
public interface Container {
    ...
    public void invoke(Request request,
      Response response) throws IOException,
      ServletException;
    public Container map(Request request,
      boolean update);
    ...
}
```

The Basic Request/ Response Processing Model



Adding Custom Request Processing Functionality

- *Valves* can be inserted in the request/response processing paths
- What can a *Valve* do with the request?
  - Examine the request and pass it on
  - Modify the request and pass it on
  - Complete the response and return
- What can a *Valve* do with the response?
  - Examine the response and pass it on
  - Modify the response and pass it on

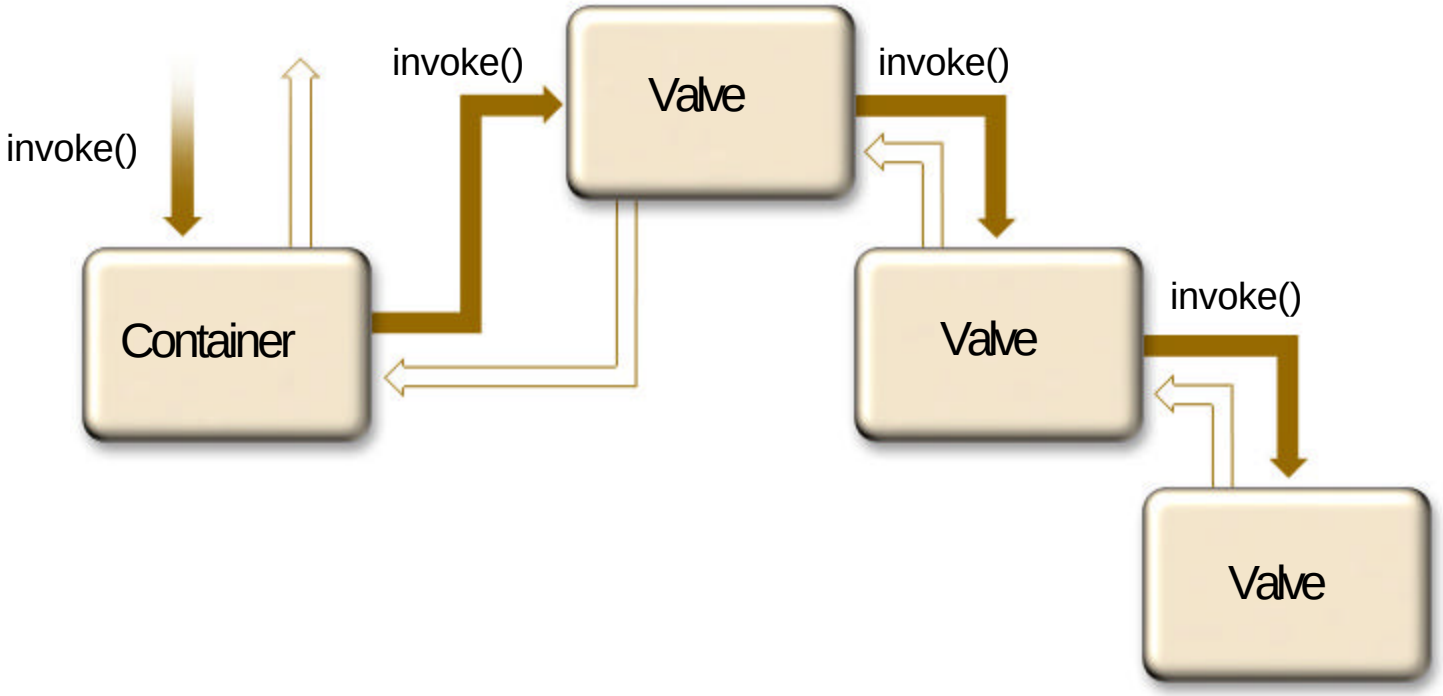
The *Valve* Interface

```
public interface Container {
    ...
    public Valve getNext();
    public void invoke(Request request,
      Response response) throws IOException,
      ServletException;
    ...
}
```

Containers Can Implement Pipelines of *Valves*

- *invoke()* method receives a request
- Passes it to the first attached *Valve*
- The first *Valve* can:
  - Preprocess the request and pass it on
  - Complete the request and return
- Each subsequent *Valve* behaves in the same manner
- After processing is complete, the call stack unwinds
  - Local variables to save state
  - You can postprocess the response

Pipelined Request Processing



Session Management Components

- *Manager*—manages all of the sessions for a *Context*
- *Session*—represents an individual **`HttpSession`**

Session managers can implement persistence, swapping, load balancing, and other features

Utility Components (per Container)

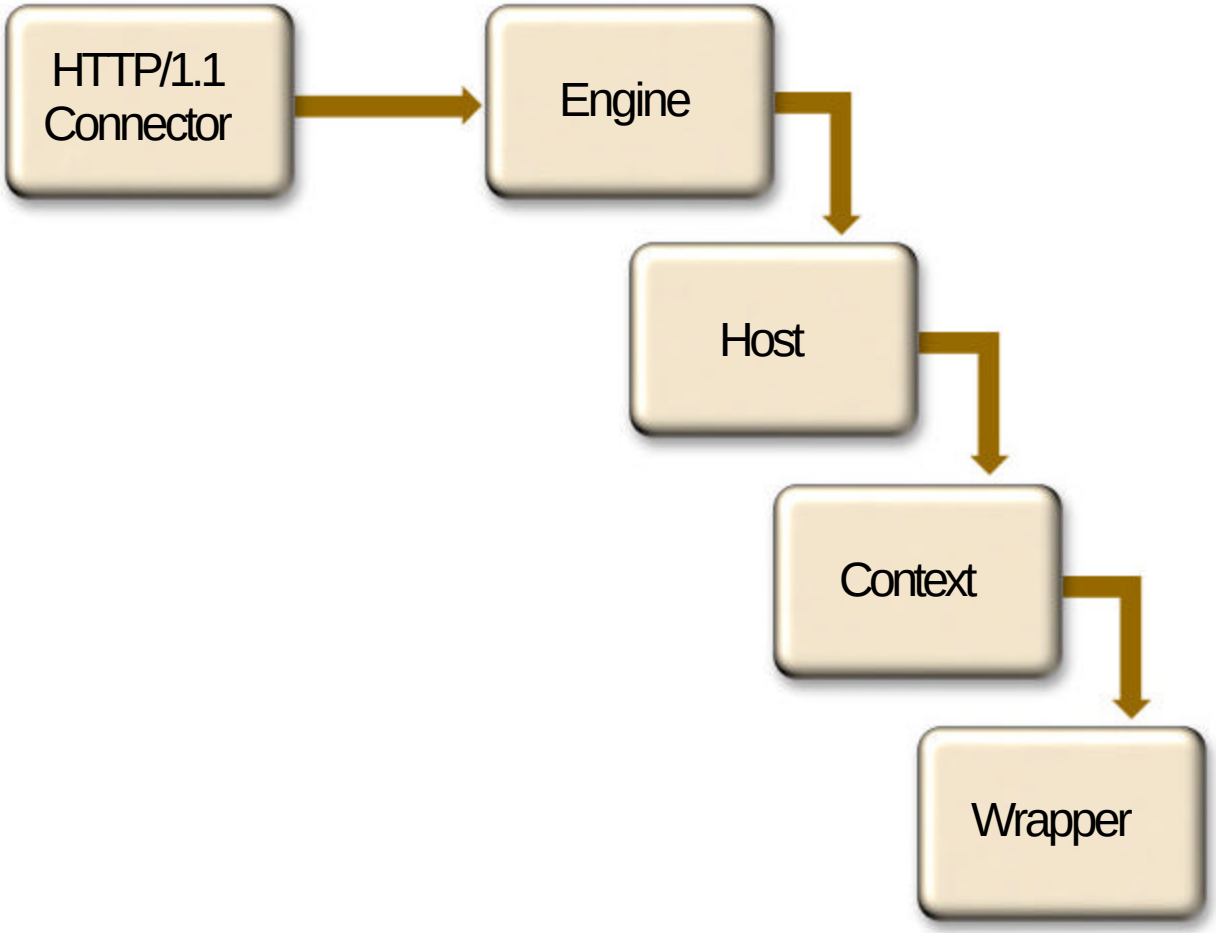
- *Loader*—class loader to use
- *Logger*—implement the **`ServletContext.log()`** methods
- *Realm*—integrate with external source of users and their roles
- *Resources*—integrate with local filesystem, web server, or application server to retrieve static resources

Containers may inherit utility components from their parent

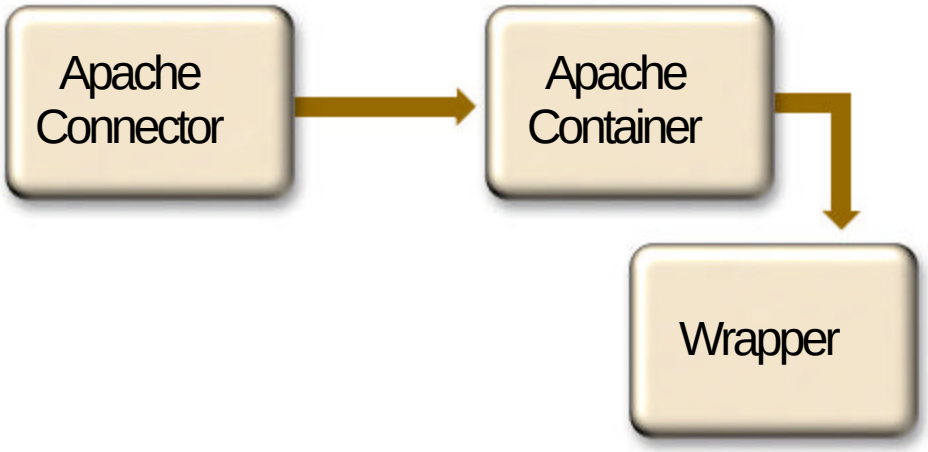
Deployment Scenarios

- Stand-alone server
- Connected to a web server
- Integrated with an application server
- Integrated with a development tool

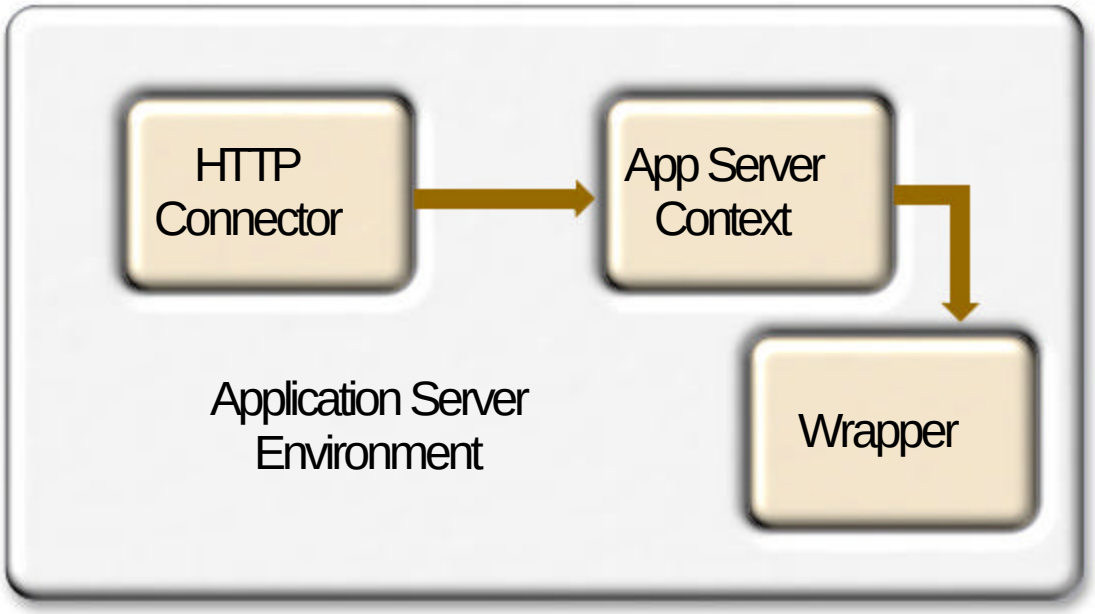
Stand-alone Server



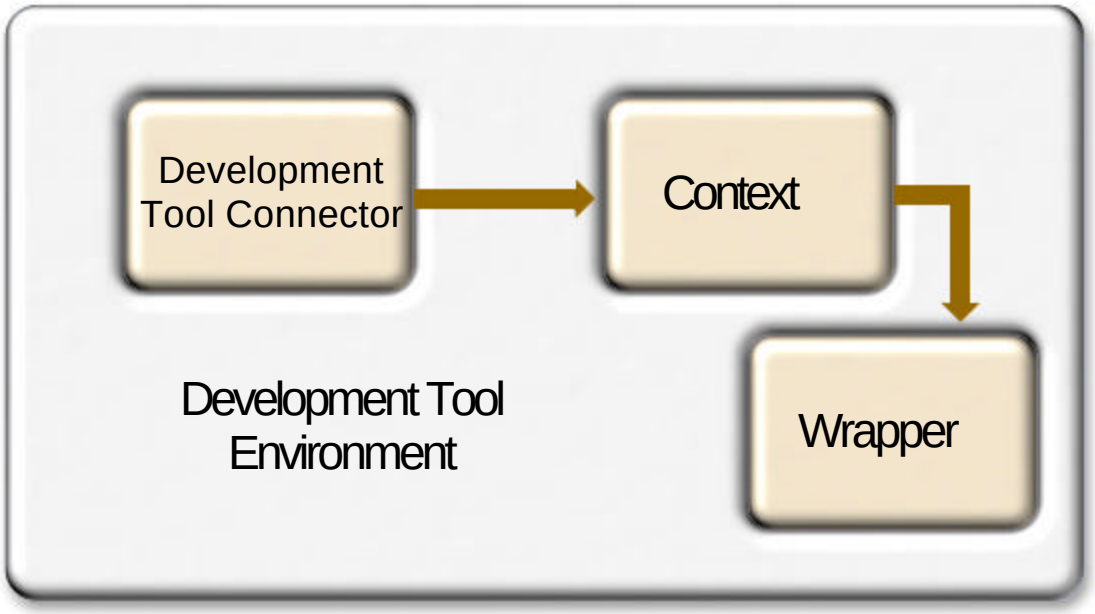
Connected To A Web Server



Integrated with an Application Server



Integrated with a Development Tool



Current Development Status

- Core containers are complete
- HTTP/1.1 connector is complete
- Major remaining functionality needed:
  - Web connectors (Apache, IIS, NES)
  - Distributed server support
  - Administration application
  - SSL/TLS support (standalone mode)
  - Support Servlet 2.3 / JSP 1.2 features
- Testing and tuning is needed

Resources

- Servlet and JSP technology information:
  - http://java.sun.com/products/servlet
  - http://java.sun.com/products/jsp
  - Mailing lists SERVLET-INTEREST and JSP-INTEREST
- Jakarta Project information
  - http://jakarta.apache.org
  - Mailing lists TOMCAT-DEV and TOMCAT-USER

JavaOne℠

Sun's 2000 Worldwide Java Developer Conference℠