Application servers and web servers are both crucial components in web-based applications, but they serve different purposes and have distinct functionalities. Here's a detailed comparison:

**Web Server**

**Purpose**

* **Function**: Primarily handles HTTP requests from clients (browsers) and serves static content such as HTML, CSS, JavaScript files, and images.
* **Usage**: Commonly used to serve static web pages and handle simple requests.

**Capabilities**

* **Static Content Serving**: Efficient at serving static content quickly.
* **Request Handling**: Processes incoming HTTP requests and sends HTTP responses back to clients.
* **Basic Application Support**: Can handle server-side scripting languages like PHP or CGI scripts, but generally lacks the advanced features required for complex applications.

**Examples**

* **Apache HTTP Server**: Widely used for serving static content and as a reverse proxy.
* **Nginx**: Known for its high performance and efficient handling of static content, also used as a reverse proxy and load balancer.

**Application Server**

**Purpose**

* **Function**: Manages and executes business logic, application services, and dynamic content generation. It typically hosts web applications and provides additional services beyond serving static content.
* **Usage**: Used for running complex applications, handling business logic, managing transactions, and providing connectivity to databases and other services.

**Capabilities**

* **Dynamic Content Generation**: Can generate dynamic content and execute server-side business logic. It supports languages and frameworks like Java (J2EE), .NET, and Python.
* **Enterprise Features**: Provides additional features like transaction management, security, messaging, and resource pooling.
* **Middleware Services**: Offers services such as database connection pooling, caching, and advanced application management.

**Examples**

* **Apache Tomcat**: Often used for running Java Servlets and JSPs, can act as both a web server and an application server.
* **JBoss (WildFly)**: A Java EE application server with extensive support for enterprise features.
* **WebLogic**: A comprehensive Java EE application server with enterprise-level features.
* **WebSphere**: IBM's application server for deploying and managing enterprise applications.

**Key Differences**

1. **Primary Function**:
   * **Web Server**: Mainly serves static content and handles HTTP requests.
   * **Application Server**: Handles business logic, dynamic content, and provides additional services and features.
2. **Content Handling**:
   * **Web Server**: Efficient at serving static files; limited support for dynamic content.
   * **Application Server**: Designed to generate dynamic content and manage complex business processes.
3. **Complexity**:
   * **Web Server**: Generally simpler and focused on delivering content quickly.
   * **Application Server**: More complex, offering a range of services required for enterprise applications.
4. **Deployment**:
   * **Web Server**: Typically deployed in front of an application server to handle static content and act as a reverse proxy.
   * **Application Server**: Deployed behind a web server or on its own to handle application logic and dynamic content.
5. **Services**:
   * **Web Server**: Limited to serving web pages and basic server-side scripting.
   * **Application Server**: Provides advanced features such as transaction management, security, and messaging.

**Integration**

In practice, web servers and application servers are often used together in a layered architecture:

* **Web Server**: Handles incoming HTTP requests, serves static content, and forwards requests for dynamic content to the application server.
* **Application Server**: Processes requests forwarded by the web server, handles business logic, interacts with databases, and generates dynamic content.

By combining both types of servers, organizations can optimize performance and functionality, leveraging the strengths of each to build scalable and robust web applications.

AppServer=WebServer+EJB container , Application Server will take care of all these issues like  Security,Trasaction,MultiThreaidng,Resource pooling ,etc..

Contents: Application Server vs Web Server

The concept you’re referring to is one way to understand the role of an application server in a Java EE (Enterprise Edition) environment. In this context, an **application server** can be thought of as a combination of a **web server** plus additional features provided by an **EJB (Enterprise JavaBeans) container**. Here’s a breakdown of this idea:

**Web Server**

A **web server** is designed to handle HTTP requests and serve static content like HTML, CSS, JavaScript files, and images. It can also handle dynamic content using server-side scripting languages.

**EJB Container**

An **EJB (Enterprise JavaBeans) container** is a component of an application server that provides a managed environment for enterprise beans, which are Java components used to encapsulate business logic. The EJB container offers several features:

1. **Transaction Management**: Handles transactions automatically, ensuring data integrity and consistency.
2. **Security**: Provides authentication and authorization mechanisms.
3. **Concurrency**: Manages concurrent access to beans.
4. **Persistence**: Supports integration with databases and provides object-relational mapping.
5. **Remote Access**: Allows beans to be accessed from remote locations.
6. **Lifecycle Management**: Manages the lifecycle of enterprise beans, including their creation, destruction, and state management.

**Combining Web Server and EJB Container**

An **application server** combines the functionalities of a web server with the additional capabilities of an EJB container:

* **Web Server Functions**: The application server can handle HTTP requests and serve static content, similar to a traditional web server.
* **EJB Container Functions**: The application server manages business logic, transaction processing, security, and other enterprise-level features through its EJB container.

**Example: Java EE Application Server**

A Java EE application server like **JBoss (WildFly)**, **WebLogic**, or **WebSphere** typically includes:

1. **Web Server Capabilities**:
   * Handling HTTP requests.
   * Serving static content.
   * Running servlets and JavaServer Pages (JSPs) for generating dynamic content.
2. **EJB Container Capabilities**:
   * Managing enterprise beans (EJBs).
   * Providing support for transactions, security, concurrency, and persistence.
   * Facilitating remote access to enterprise beans.

**Summary**

* **Application Server**: An application server integrates the functionalities of a web server with additional enterprise-level features provided by an EJB container. It manages both the presentation layer (web content) and the business logic layer (enterprise beans).
* **Web Server**: Focuses on handling HTTP requests and serving static content, with limited support for dynamic content.
* **EJB Container**: Provides additional services such as transaction management, security, and business logic execution, which are essential for enterprise applications.

By combining these components, an application server provides a comprehensive platform for developing, deploying, and managing enterprise applications.

## Application Server is superset of web server

ChatGPT

Yes, an **application server** can be considered a superset of a **web server**. Here’s how that comparison breaks down:

**Web Server**

* **Purpose**: Primarily designed to handle HTTP requests, serve static content (like HTML, CSS, JavaScript files, and images), and support basic server-side scripting (e.g., PHP, CGI).
* **Capabilities**:
  + Serves static files and generates dynamic content using server-side scripting.
  + Basic request handling and processing.

**Application Server**

* **Purpose**: Provides a broader range of services beyond what a web server offers. It includes all the functionality of a web server but also supports complex business logic, transactions, and enterprise-level features.
* **Capabilities**:
  + **Web Server Functions**: Handles HTTP requests, serves static and dynamic content, and runs server-side components like servlets and JSPs.
  + **Business Logic Execution**: Manages enterprise beans (EJBs) or other business logic components.
  + **Transaction Management**: Manages transactions to ensure data consistency and integrity.
  + **Security**: Provides mechanisms for authentication, authorization, and secure communication.
  + **Persistence**: Integrates with databases and provides object-relational mapping.
  + **Concurrency Management**: Handles concurrent access to resources and business logic.
  + **Remote Access**: Allows remote access to business logic and services.
  + **Resource Management**: Manages resources such as connection pools and messaging services.

**Integration**

* **Web Server within Application Server**: Many application servers include a web server as part of their architecture. This means that they can handle HTTP requests and serve static content just like a standalone web server, in addition to providing enterprise-level services.

**Examples**

1. **Apache Tomcat**: While primarily a web server and servlet container, it also provides some features of an application server but lacks full EJB support.
2. **JBoss (WildFly)**: A full-fledged application server that includes web server capabilities and supports EJBs, transactions, and other enterprise services.
3. **WebLogic and WebSphere**: Comprehensive application servers that provide all the functionality of a web server plus extensive enterprise features.

**Summary**

* **Application Server**: A superset of a web server, offering all the features of a web server along with additional services for managing business logic, transactions, security, and more.
* **Web Server**: Focuses on handling HTTP requests and serving content, and may be part of an application server’s architecture.

By encompassing the functionalities of a web server and extending them to include enterprise-level features, an application server provides a more robust and versatile platform for developing, deploying, and managing complex applications.

## Understanding How the Application Server's Web Container Works

To understand how to do proper application server development, you need to understand how the two basic containers of an application server work, namely the Web container and the [EJB container](http://www.theserverside.com/feature/Working-with-the-EJB-Container). This article will take a look at how a client application interacts with the application server's Web container.

**How does an application server's Web container work?**Let’s look at a typical Web-based interaction with our application server at runtime.  
Let’s use a typical scenario where the request is coming in from an Internet browser, such as Internet Explorer, Chrome or Safari. We’ll also simplify the interaction by eliminating the complications presented by workload enhancements such as network sprayers and caching proxies. We’ll just focus on a typical Web-based interaction between a Web-based client and our application server.

**From Client to Web Server**

When dealing with Web-based requests, before tunneling through to our [application server](http://www.theserverside.com/feature/What-is-an-Application-Server), a client will always hit a Web server first. An application server does not replace the need for a Web server. A Web server remains as pivotal a part of the application server architecture as ever.  
Web servers are great at doing one thing: serving up files. A Web server takes requests from clients, maps that request to a file on the file system, and then sends that file back to the client.  
If you want an HTML file, a Web server can efficiently and reliably find that file and send it back to you. If you need an image, a Web server can serve it up to you as well. You want to download a .zip file or a .pdf file quickly and efficiently? A Web server can make that happen.

Unfortunately though, your Web server is about as intelligent as a male model. A Web server can serve up static files until the cows come home, but ask your Web server to add ‘one plus one’ and you’ll be waiting there for a very, very long time.

If our applications use any images, HTML, .pdf or .zip files, we like to keep all of those static files on the Web server. If we need some logic or dynamic content in our applications, we will delegate to our Servlets, JSPs, EJBs and JavaBeans that are running on our application server.

Now here is the dilemma. Our application server contains all of our Servlets and JSPs, but all of the requests go through the Web server, and the Web server, not being a very clever machine, tries to handle all requests, regardless of whether the request is for an image, HTML file, or to our detriment, a Servlet or a JSP.

*How do you stop a Web server from trying to handle requests for our Servlets and JSPs?*

The key to stopping a Web server from trying to serve up JSPs or Servlets, is to install something called the "application server plug-in" on the Web server.

The general idea, although not a hard and fast rule is that before you install an application server, you should install your Web server first. Even when you do a full installation of an application server, behind the scenes, the application server typically installs a Web server first, and then installs an application server plug-in into that Web server to forward non-static request that it can't handle.

**What exactly does the Web server plug-in do?**

As was stated earlier, the Web server tries to handle every single request that it receives. However, when the Application server comes onto the scene, it introduces itself to the Web server and has a conversation that goes something like this:  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Application server:* Hey, WebServer...  
**Web Server:** Yo, what’s up?  
*Application server*: Hey, not much.  
**Web Server:** What can I do for you?  
*Application server:* Well, I know that you’re really great at serving up static files and all, but you’re going to get some crazy requests for JSPs and Servlets that you won’t be able to find on your file system.  
**Web Server:** Really? What am I going to do? I won’t be able to find any of these JSPs and Servlets, and I’ll end up sending a bunch of 404 errors back to clients, and the clients will be pissed!  
*Application server:* Hey, calm down. Here’s what you do: just take those requests and send them to me. I’ll handle the request, generate some HTML, give that HTML back to you, and you can send the HTML back to the client.  
**Web Server:** Kewl. You do the work, but the client thinks it’s me handling the request? I like this arrangement already. How do I know what files to send to you though?  
*Application server:* Don’t worry. I’ll make a thorough list and write it all down in a special XML file. Just read that file every once in a while and keep up to date on which files you need to send back to me.  
**Web Server:** Great. But when I do get a request for an item on the list, how will I know where to send it.  
*Application server:* Hey, don’t worry. I’ve got it all covered. That XML file also contains a list of which IP addresses/port combinations to send the requests to. It’s all right there in that XML file. And if you have a problem understanding how to use it, here’s a .dll file that explains everything to you as well. Read it every time you start up.  
**Web Server:** Kewl. I think this is going to be a great relationship.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**From Web Server to Application Server**

When a client makes a request for a JSP or a Servlet, the request initially goes to the Web server. The Web server reads the special XML file the [application server](http://www.theserverside.com/feature/What-to-look-for-in-an-application-server) provides, and realizes that the request that came in should be sent to the application server for processing.

The special XML file also provides the IP address/port combination of listening application servers. The Web server, using the http protocol, then sends the request to the Application server JVM listening on the appropriate port.

The JVM listening on the appropriate port represents our application server, and the port the JVM listens on can be configured through that JVM’s Web container.

The Web server handles the incoming request, and matches that request to the application server set up to handle the given Servlet or JSP.

**Inside the Web Container**

If the Servlet hasn’t been called before, the JVM loads the Servlet and then generates a thread to handle the request.

Servlets are shy little creatures. They sit on the hard drive just minding their own business, and don’t bother anyone until they've been invoked. However, feed a few drinks to those Servlets – get them loaded – and they remain resident in memory until the party ends, which happens when someone pulls the plug on the application server.

So, the request gets sent from the client, to the Web server, and the Web server passes the request to the application server, who in turn invokes and threads the appropriate Servlet.

**What does our Servlet do?**

Well, the Servlet can do pretty much anything the developer wants it to do. When programming Servlets, a developer is only limited by their creativity, and more likely, their Java programming skills.

Typically, a Servlet implements some control logic. For example, a Servlet might figure out what a user typed into some text fields in a web-based form. It might then take that information and save it to a database.

Servlets are intended to be controllers. While Servlets can interact directly with a database, they’re not really supposed to. Instead, Servlets are supposed to delegate to a JavaBean or an EJB to do such things. Let’s say, for the sake of argument, our Servlet calls an EJB.