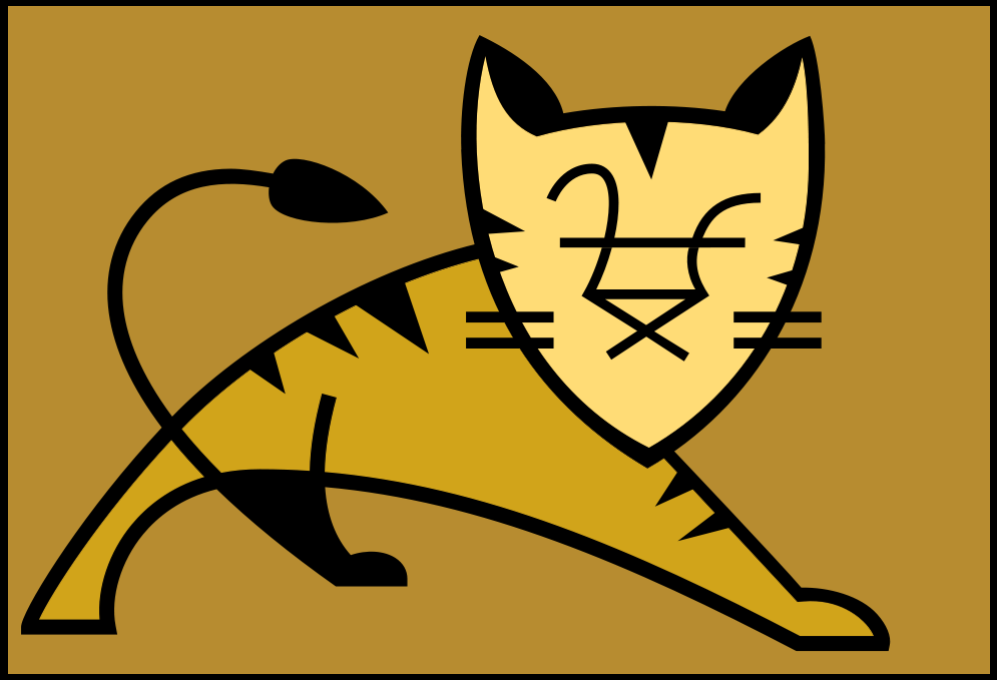
I

CONFIGURACIÓN AVANZADA DE TOMCAT



Iñaki Respaldiza

CONFIGURACIÓN AVANZADA DE TOMCAT

- Uso de las válvulas
- Habilitar el Single Sign-On (SSO)
- Gestión y configuración de sesiones en Tomcat
- El componente SessionManager
- Sesiones en base de datos
- SeguridadenTomcat
- Autenticación y autorización
- Definición de un keystore para SSL/TLS
- Tomcat y la configuración LDAP

CONFIGURACIÓN AVANZADA DE TOMCAT

Las válvulas del tomcat son una técnología introducida a partir de Tomcat 4 que permite asociar una instacia de una clase Java al un contenedor Catalina.

Esta configuración permite que la clase asociada actue como un pre-procesador de las peticiones. Estas clases se llaman válvulas, y deben implementar la intefaz org.apache.catalina.Valve interface o extender la clase org.apache.catalina.valves.ValveBase. Las válvulas son propias de Tomcat y no pueden ser usadas en otros contenedores de servlet.

CONFIGURACIÓN AVANZADA DE TOMCAT

Las válvulas disponibles son:

- Access Log
- Remote Address Filter
- Remote Host Filter
- Request Dumper
- Single Sign On
- Form Authenticator

CONFIGURACIÓN AVANZADA DE TOMCAT

Access Log Valve

La primera válvula por defecto del tomcat es Access Log Valve, que está implementada por la clase org.apache.catalina.valves.AccessLogValve. Crea ficheros de log para rastrear el acceso a la información de los clientes.
Alguna de la infomación que rastrea son clicks, actividad de la sesión del usuario, información de la autenticación del usuario entre otras. Esta válvula se puede asociar a un engine, host o context container del Tomcat.

CONFIGURACIÓN AVANZADA DE TOMCAT

Ejemplo de uso :

<Valve className=”org.apache.catalina.valves.
AccessLogValve” directory=”logs” prefix=”localhost_
access_log.” suffix=”.txt” pattern=”common”/>
Este código indica que los logs se guardaran en el directorio /logs, con el prefijo “localhost_access_log.”, y con el sufijo “.txt”.

CONFIGURACIÓN AVANZADA DE TOMCAT

Remote Address Filter

El Remote Address filter lo implementa la clase org.apache.catalina.valves.RemoteAddrValve, permite comparar direcciones IP de los clientes que realizan la petición con una o varias expresiones regulares para prohibir o permitir el acceso. Esta válvula se puede asociar al Engine, Host, o Context container.

CONFIGURACIÓN AVANZADA DE TOMCAT

Ejemplo de uso :

<Valve className=”org.apache.catalina.valves.
AccessLogValve” directory=”logs” prefix=”localhost_
access_log.” suffix=”.txt” pattern=”common”/>
Este código indica que los logs se guardaran en el directorio /logs, con el prefijo “localhost_access_log.”, y con el sufijo “.txt”.

CONFIGURACIÓN AVANZADA DE TOMCAT

Remote Host Filter

El Remote Host filter está implementado por org.apache.catalina.valves.RemoteHostValve es muy parecido al RemoteAddrValve, con la diferencia que te permite comparar al nombre del host en vez de la IP. Se puede asociar al Engine, Host, o Context container.

CONFIGURACIÓN AVANZADA DE TOMCAT

Ejemplo de uso:

<Valve className=”org.apache.catalina.valves.
RemoteHostValve” deny=”virtuas*”/>
Con esta sentencia prohibimos el acceso a todos los host con que incluyan en su nombre virtuas.

CONFIGURACIÓN AVANZADA DE TOMCAT

Request Dumper Valve

ElRequest Dumper valve lo implenta la clase org.apache.catalina.valves.RequestDumperValve es una herramienta de depuración que escribe en el log el detalle de cada petición realizada. Se puede asociar al Engine, Host, o Context container.

Habilitar el Single Sign-On (SSO)

Esta válvula se utiliza cuando queremos que los usuarios puedan identificarse en cualquier aplicacion en nuestro virtual host, y que su identidad sea reconocida por cualquier aplicación que esté en ese host.

Gestión y configuración de sesiones en Tomcat

El elemento Manager representa el gestor de sesiones que se utilizará para crear y mantener sesiones HTTP solicitadas por la aplicación web asociada.

Un elemento Manager puede ser anidado dentro del componente Context. Si incluimos uno, se creará automáticamente una configuración predeterminada de Manager, lo cual es suficiente para la mayoría de las aplicaciones.

Gestión y configuración de sesiones en Tomcat

Attribute	Description
`className`	Java class name of the implementation to use. This class must implement the `org.apache.catalina.Manager` interface. If not specified, the standard value (defined below) will be used.
`distributable`	**Deprecated**: This should be configured via the Context. Set to `true` to ask the session manager to enforce the restrictions described in the Servlet Specification on distributable applications (primarily, this would mean that all session attributes must implement `java.io.Serializable`). Set to `false` (the default) to not enforce these restrictions. **NOTE** - The value for this property is inherited automatically based on the presence or absence of the `<distributable>` element in the web application deployment descriptor (`/WEB-INF/web.xml`).
`maxActiveSessions`	The maximum number of active sessions that will be created by this Manager, or `-1` (the default) for no limit. When the limit is reached, any attempt to create a new session (e.g. with `HttpServletRequest.getSession()` call) will fail with an `IllegalStateException`.
`maxInactiveInterval`	**Deprecated**: This should be configured via the Context. The initial maximum time interval, in seconds, between client requests before a session is invalidated. A negative value will result in sessions never timing out. If the attribute is not provided, a default of 1800 seconds (30 minutes) is used. This attribute provides the initial value whenever a new session is created, but the interval may be dynamically varied by a servlet via the `setMaxInactiveInterval` method of the `HttpSession` object.
`sessionIdLength`	The length of session ids created by this Manager, measured in bytes, excluding subsequent conversion to a hexadecimal string and excluding any JVM route information used for load balancing. The default is 16. You should set the length on a nested **SessionIdGenerator** element instead.

Gestión y configuración de sesiones en Tomcat

Exitend dos implementaciones predefinidas. La estandar que almacena las sesiones activas y la persistente que almacena las sesiones activas aunque cambien o se reinicie Tomcat.

El componente SessionManager

La implementación estándar de Manager es org.apache.catalina.session.StandardManager. Soporta los siguientes atributos adicionales:

Attribute	Description
`pathname`	Absolute or relative (to the work directory for this Context) pathname of the file in which session state will be preserved across application restarts, if possible. The default is "SESSIONS.ser". See Persistence Across Restarts for more information. This persistence may be disabled by setting this attribute to an empty string.
`processExpiresFrequency`	Frequency of the session expiration, and related manager operations. Manager operations will be done once for the specified amount of backgroundProcess calls (i.e., the lower the amount, the more often the checks will occur). The minimum value is 1, and the default value is 6.
`secureRandomClass`	Name of the Java class that extends `java.security.SecureRandom` to use to generate session IDs. If not specified, the default value is `java.security.SecureRandom`.
`secureRandomProvider`	Name of the provider to use to create the `java.security.SecureRandom` instances that generate session IDs. If an invalid algorithm and/or provider is specified, the Manager will use the platform default provider and the default algorithm. If not specified, the platform default provider will be used.
`secureRandomAlgorithm`	Name of the algorithm to use to create the `java.security.SecureRandom` instances that generate session IDs. If an invalid algorithm and/or provider is specified, the Manager will use the platform default provider and the default algorithm. If not specified, the default algorithm of SHA1PRNG will be used. If the default algorithm is not supported, the platform default will be used. To specify that the platform default should be used, do not set the secureRandomProvider attribute and set this attribute to the empty string.

El componente SessionManager

`sessionAttributeNameFilter`	A regular expression used to filter which session attributes will be distributed. An attribute will only be distributed if its name matches this pattern. If the pattern is zero length or `null`, all attributes are eligible for distribution. The pattern is anchored so the session attribute name must fully match the pattern. As an example, the value `(userName|sessionHistory)` will only distribute the two session attributes named `userName` and `sessionHistory`. If not specified, the default value of `null` will be used.
`sessionAttributeValueClassNameFilter`	A regular expression used to filter which session attributes will be distributed. An attribute will only be distributed if the implementation class name of the value matches this pattern. If the pattern is zero length or `null`, all attributes are eligible for distribution. The pattern is anchored so the fully qualified class name must fully match the pattern. If not specified, the default value of `null` will be used unless a `SecurityManager` is enabled in which case the default will be `java\\.lang\\.(?:Boolean|Integer|Long|Number|String)`.
`warnOnSessionAttributeFilterFailure`	If **sessionAttributeNameFilter** or **sessionAttributeValueClassNameFilter** blocks an attribute, should this be logged at `WARN` level? If `WARN` level logging is disabled then it will be logged at `DEBUG`. The default value of this attribute is `false` unless a `SecurityManager` is enabled in which case the default will be `true`.

El componente SessionManager

La implementación persistente de Manager es org. apache.catalina.session.PersistentManager. Además de las operaciones habituales de creación y eliminación de sesiones, PersistentManager tiene la capacidad de intercambiar sesiones activas que no están en uso con un mecanismo de almacenamiento persistente, así como de guardar todas las sesiones a través de un reinicio normal de Tomcat. El mecanismo de almacenamiento persistente utilizado se selecciona mediante la elección de un elemento de almacén anidado dentro del elemento Administrador, esto es necesario para el uso de PersistentManage

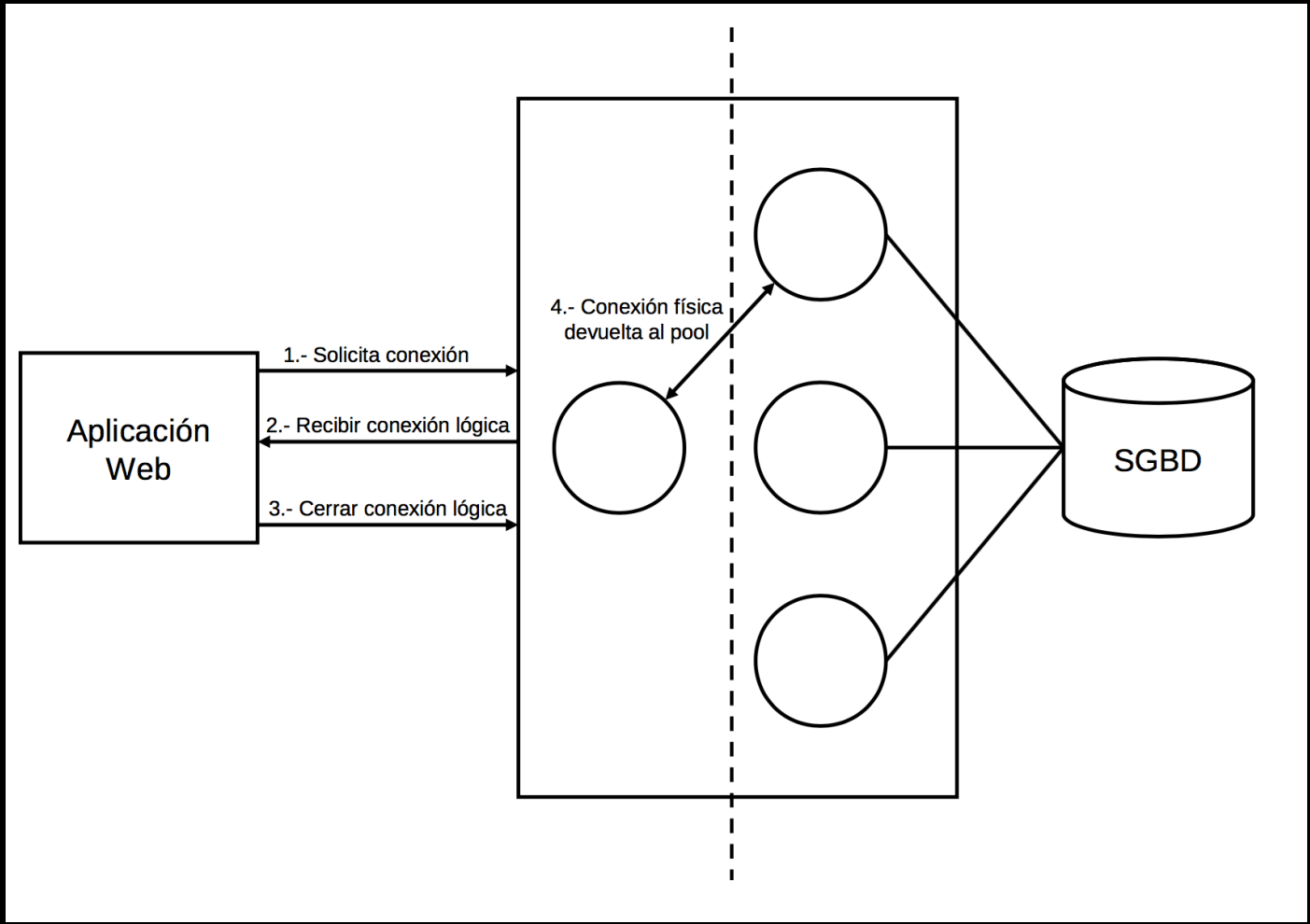
El componente SessionManager

Attribute	Description
`className`	It has the same meaning as described in the Common Attributes above. You **must** specify `org.apache.catalina.session.PersistentManager` to use this manager implementation.
`maxIdleBackup`	The time interval (in seconds) since the last access to a session before it is eligible for being persisted to the session store, or `-1` to disable this feature. By default, this feature is disabled.
`maxIdleSwap`	The maximum time a session may be idle before it is eligible to be swapped to disk due to inactivity. Setting this to `-1` means sessions should not be swapped out just because of inactivity. If this feature is enabled, the time interval specified here should be equal to or longer than the value specified for `maxIdleBackup`. By default, this feature is disabled.
`minIdleSwap`	The minimum time in seconds a session must be idle before it is eligible to be swapped to disk to keep the active session count below maxActiveSessions. Setting to `-1` means sessions will not be swapped out to keep the active session count down. If specified, this value should be less than that specified by `maxIdleSwap`. By default, this value is set to `-1`.
`processExpiresFrequency`	It is the same as described above for the `org.apache.catalina.session.StandardManager` class.
`saveOnRestart`	Should all sessions be persisted and reloaded when Tomcat is shut down and restarted (or when this application is reloaded)? By default, this attribute is set to `true`.

El componente SessionManager

`secureRandomClass`	It is the same as described above for the `org.apache.catalina.session.StandardManager` class.
`secureRandomProvider`	It is the same as described above for the `org.apache.catalina.session.StandardManager` class.
`secureRandomAlgorithm`	It is the same as described above for the `org.apache.catalina.session.StandardManager` class.
`sessionAttributeNameFilter`	A regular expression used to filter which session attributes will be distributed. An attribute will only be distributed if its name matches this pattern. If the pattern is zero length or `null`, all attributes are eligible for distribution. The pattern is anchored so the session attribute name must fully match the pattern. As an example, the value `(userName|sessionHistory)` will only distribute the two session attributes named `userName` and `sessionHistory`. If not specified, the default value of `null` will be used.
`sessionAttributeValueClassNameFilter`	A regular expression used to filter which session attributes will be distributed. An attribute will only be distributed if the implementation class name of the value matches this pattern. If the pattern is zero length or `null`, all attributes are eligible for distribution. The pattern is anchored so the fully qualified class name must fully match the pattern. If not specified, the default value of `null` will be used unless a `SecurityManager` is enabled in which case the default will be `java\\.lang\\.(?:Boolean|Integer|Long|Number|String)`.
`warnOnSessionAttributeFilterFailure`	If **sessionAttributeNameFilter** or **sessionAttributeValueClassNameFilter** blocks an attribute, should this be logged at `WARN` level? If `WARN` level logging is disabled then it will be logged at `DEBUG`. The default value of this attribute is `false` unless a `SecurityManager` is enabled in which case the default will be `true`.

Sesiones en base de datos



Sesiones en base de datos

- Añadir una etiqueta <Resource> en el elemento <Context> (META-INF/context.xml) o en el elemento <DefaultContext> (en server.xml)

- Utilizar llamadas JNDI en el código de la aplicación para buscar la fuente de datos JDBC

Sesiones en base de datos

Atributo	Descripción
name	El nombre JNDI para este elemento. El nombre lógico accesible desde la aplicación será: java:comp/env/nombre
auth	Puede valer Container o Application e indica quién realizará la autenticación
type	En este caso debe valer javax.sql.DataSource
maxActive	Número máximo de conexiones acitvas en el pool. El valor 0 significa sin límite
maxIdle	Número de conexiones inactivas en el pool antes de que se cierre alguna de ellas. El valor -1 indinca que no hay límite
maxWait	Número máximo de milisegundos que el gestor esperará para que una conexión responda antes de lanzar una excepción. El valor -1 hará que espera de forma indefinida
username	Usuario que se pasará al driver JDBC
password	Password que se pasará al driver JDBC
driverClassName	Clase que implementa el driver JDBC
url	Url de conexión que se pasará al driver JDBC

Seguridad en Tomcat

Consideraciones previas:

- Defensa en Profundidad: Este principio se basa en la suposición de que cualquier medida anterior ha podido fallar y debemos proteger el sistema desde el punto actual. La política de defensa en profundidad implica la aplicación de todas las medidas de seguridad que se puedan aplicar sin deteriorar el rendimiento del sistema, pues hay que tener en cuenta que la disponibilidad es una de las características de un sistema seguro y toda medida de seguridad lleva un coste en tiempo que hay que evaluar.

Seguridad en Tomcat

- Mínimo Privilegio Posible: Esta regla se basa en ejecutar cada uno de los procesos de un sistema con sólo los privilegios necesarios para que si esa parte queda comprometida el atacante obtenga los mínimos permisos posibles. Por ejemplo, si la página Web del sitio público accede al motor de bases de datos con más privilegios de los necesarios un atacante podría, si encuentra un fallo en la aplicación Web, acceder a información sensible e incluso modificar datos en tablas críticas de otras bases de datos.

Seguridad en Tomcat

- Mínimo punto de exposición: Los servidores deben ejecutar sólo los procesos que son necesarios para el cumplimiento de sus roles. Si un servidor no tiene ninguna impresora configurada, el servicio de spooler de impresión debe ser deshabilitado. Esto permite que un servidor tan sólo pueda ser atacado por fallos que de verdad le ataquen y no que quede comprometido por un software que no es necesario para su funcionamiento.

Seguridad en Tomcat

Lo que aplicado a Tomcat se traduce en:

Defensa en Profundidad La configuración de Tomcat no puede ser la unica barrera que apliquemos el resto de los componentes, incluida la aplicación han de ser seguros.

- Mínimo Privilegio Posible: Tomcat no puede ser ejecutado por el usuario ROOT. Debemos crear un usuario especifico, con los permisos justos, para ejecutarlo.

- Mínimo punto de exposición: los permisos de los archivos han de estar restringidos, en lectura escritura y ejecucion.

Seguridad en Tomcat

Lo que aplicado a Tomcat se traduce en:

Defensa en Profundidad La configuración de Tomcat no puede ser la unica barrera que apliquemos el resto de los componentes, incluida la aplicación han de ser seguros.

- Mínimo Privilegio Posible: Tomcat no puede ser ejecutado por el usuario ROOT. Debemos crear un usuario especifico, con los permisos justos, para ejecutarlo.

- Mínimo punto de exposición: los permisos de los archivos han de estar restringidos, en lectura escritura y ejecucion.

Seguridad en Tomcat

Al desplegar una aplicación web que proporciona funciones de administración para la instancia de Tomcat, debemos seguir las siguientes directrices:

- Cercionarnos de que los usuarios autorizados a acceder a la aplicación de gestión tengan contraseñas seguras.
- El uso de LockOutRealm que evita ataques de fuerza bruta contra contraseñas de usuario.
- Deberíamos descomentar RemoteAddrValve en /META-INF/context.xml que limita el acceso a localhost. Si requerimos acceso remoto, deberiamos limitar el acceso a direcciones IP específicas usando esta válvula.

Seguridad en Tomcat

Security manager:

El gestor de seguridad hace que las aplicaciones web se ejecuten en un entorno aislado, limitando significativamente la capacidad de una aplicación web de realizar acciones maliciosas como llamar a System.exit (), establecer conexiones de red o acceder al sistema de archivos fuera de los directorios raíz o temporales de la aplicación web . Sin embargo, debe tenerse en cuenta que hay algunas acciones maliciosas, tales como desencadenar alto consumo de CPU a través de un bucle infinito, que el administrador de seguridad no puede evitar.

Seguridad en Tomcat

Security manager:

El gestor de seguridad hace que las aplicaciones web se ejecuten en un entorno aislado, limitando significativamente la capacidad de una aplicación web de realizar acciones maliciosas como llamar a System.exit (), establecer conexiones de red o acceder al sistema de archivos fuera de los directorios raíz o temporales de la aplicación web . Sin embargo, debe tenerse en cuenta que hay algunas acciones maliciosas, tales como desencadenar alto consumo de CPU a través de un bucle infinito, que el administrador de seguridad no puede evitar.

Seguridad en Tomcat

El modulo de seguridad ha de aplicarse desde el principio del desarrollo ya que muy frecuentemente interfiere con los usuarios de Tomcat y algunas soluciones Java.

Seguridad en Tomcat

Es buena idea eliminar todas los comentarios que trae de serie para facilitar su lectura. Así como todos los componentes que no vayamos a usar para minimizar las dianas de ataque.

Seguridad en Tomcat

Server:
Situando el valor -1 en port podremos deshabilitar la función shutdown. Si nuestro proyecto requiere tenerlo habilitado deberíamos protegerlo con una contraseña segura.

Seguridad en Tomcat

Server:
Situando el valor -1 en port podremos deshabilitar la función shutdown. Si nuestro proyecto requiere tenerlo habilitado deberíamos protegerlo con una contraseña segura.

Seguridad en Tomcat

Conectores: De serie viene activo tanto el conector para HTTP como el de AJP. Deberemos desactivar, en server.xml, aquel que no vayamos a usar. El atributo address define las direcciones IPs de escucha de los conectores. Si no restringimos las direcciones escucharán en todas las direcciones.

Seguridad en Tomcat

The **address** attribute may be used to control which IP address the connector listens on for connections. By default, the connector listens on all configured IP addresses.

The **allowTrace** attribute may be used to enable TRACE requests which can be useful for debugging. Due to the way some browsers handle the response from a TRACE request (which exposes the browser to an XSS attack), support for TRACE requests is disabled by default.

The **maxPostSize** attribute controls the maximum size of a POST request that will be parsed for parameters. The parameters are cached for the duration of the request so this is limited to 2MB by default to reduce exposure to a DOS attack.

The **maxSavePostSize** attribute controls the saving of POST requests during FORM and CLIENT-CERT authentication. The parameters are cached for the duration of the authentication (which may be many minutes) so this is limited to 4KB by default to reduce exposure to a DOS attack.

The **maxParameterCount** attribute controls the maximum number of parameter and value pairs (GET plus POST) that can be parsed and stored in the request. Excessive parameters are ignored. If you want to reject such requests, configure a FailedRequestFilter.

The **xpoweredBy** attribute controls whether or not the X-Powered-By HTTP header is sent with each request. If sent, the value of the header contains the Servlet and JSP specification versions, the full Tomcat version (e.g. Apache Tomcat/7.0.0), the name of the JVM vendor and the version of the JVM. This header is disabled by default. This header can provide useful information to both legitimate clients and attackers.

The **server** attribute controls the value of the Server HTTP header. The default value of this header for Tomcat 4.1.x, 5.0.x, 5.5.x, 6.0.x and 7.0.x is Apache-Coyote/1.1. This header can provide limited information to both legitimate clients and attackers.

Seguridad en Tomcat

The **SSLEnabled**, **scheme** and **secure** attributes may all be independently set. These are normally used when Tomcat is located behind a reverse proxy and the proxy is connecting to Tomcat via HTTP or HTTPS. They allow Tomcat to see the SSL attributes of the connections between the client and the proxy rather than the proxy and Tomcat. For example, the client may connect to the proxy over HTTPS but the proxy connects to Tomcat using HTTP. If it is necessary for Tomcat to be able to distinguish between secure and non-secure connections received by a proxy, the proxy must use separate connectors to pass secure and non-secure requests to Tomcat. If the proxy uses AJP then the SSL attributes of the client connection are passed via the AJP protocol and separate connectors are not needed.

The **sslEnabledProtocols** attribute determines which versions of the SSL/TLS protocol are used. Since the POODLE attack in 2014, all SSL protocols are considered unsafe and a secure setting for this attribute in a standalone Tomcat setup might be `sslEnabledProtocols="TLSv1,TLSv1.1,TLSv1.2"`

The **ciphers** attribute controls the ciphers used for SSL connections. By default, the default ciphers for the JVM will be used. This usually means that the weak export grade ciphers will be included in the list of available ciphers. Secure environments will normally want to configure a more limited set of ciphers.

The **tomcatAuthentication** and **tomcatAuthorization** attributes are used with the AJP connectors to determine if Tomcat should handle all authentication and authorisation or if authentication should be delegated to the reverse proxy (the authenticated user name is passed to Tomcat as part of the AJP protocol) with the option for Tomcat to still perform authorization.

The **allowUnsafeLegacyRenegotiation** attribute provides a workaround for CVE-2009-3555, a TLS man in the middle attack. This workaround applies to the BIO connector. It is only necessary if the underlying SSL implementation is vulnerable to CVE-2009-3555. For more information on the current state of this vulnerability and the work-arounds available see the Tomcat 7 security page.

The **requiredSecret** attribute in AJP connectors configures shared secret between Tomcat and reverse proxy in front of Tomcat. It is used to prevent unauthorized connections over AJP protocol.

Seguridad en Tomcat

El elemento host controla la implementación. El despliegue automático permite una administración más sencilla, pero también facilita que un atacante implemente una aplicación malintencionada. El despliegue automático es controlado por los atributos autoDeploy y deployOnStartup. Si ambos son falsos, sólo se implementarán los Contextos definidos en server.xml y cualquier cambio requerirá un reinicio de Tomcat.

Seguridad en Tomcat

El elemento host controla la implementación. El despliegue automático permite una administración más sencilla, pero también facilita que un atacante implemente una aplicación malintencionada. El despliegue automático es controlado por los atributos autoDeploy y deployOnStartup. Si ambos son falsos, sólo se implementarán los Contextos definidos en server.xml y cualquier cambio requerirá un reinicio de Tomcat.

Seguridad en Tomcat

The **crossContext** attribute controls if a context is allowed to access the resources of another context. It is `false` by default and should only be changed for trusted web applications.

The **privileged** attribute controls if a context is allowed to use container provided servlets like the Manager servlet. It is `false` by default and should only be changed for trusted web applications.

The **allowLinking** attribute controls if a context is allowed to use linked files. If enabled and the context is undeployed, the links will be followed when deleting the context resources. To avoid this behaviour, use the **aliases** attribute. Changing this setting from the default of `false` on case insensitive operating systems (this includes Windows) will disable a number of security measures and allow, among other things, direct access to the WEB-INF directory.

The **sessionCookiePathUsesTrailingSlash** can be used to work around a bug in a number of browsers (Internet Explorer, Safari and Edge) to prevent session cookies being exposed across applications when applications share a common path prefix. However, enabling this option can create problems for applications with Servlets mapped to `/*`. It should also be noted the RFC6265 section 8.5 makes it clear that different paths should not be considered sufficient to isolate cookies from other applications.

Definición de un keystore para SSL/TLS

Transport Layer Security (TLS) y su predecesor, Secure Sockets Layer (SSL), son tecnologías que permiten a los navegadores web y servidores web comunicarse a través de una conexión segura. Esto significa que los datos que se envían se cifran por un lado, se transmiten, y luego se descifran por el otro lado antes del procesamiento. Se trata de un proceso bidireccional, lo que significa que tanto el servidor como el navegador cifran todo el tráfico antes de enviar los datos.

Definición de un keystore para SSL/TLS

Otro aspecto importante del protocolo SSL / TLS es la autenticación. Esto significa que durante su intento inicial de comunicarse con un servidor web a través de una conexión segura, ese servidor presentará su navegador web con un conjunto de credenciales, en forma de un "Certificado", como prueba de que el sitio es quién y lo que reclama ser. En algunos casos, el servidor también puede solicitar un Certificado desde su navegador web, solicitando pruebas de que usted es quien dice ser. Esto se conoce como "Autenticación de cliente", aunque en la práctica se utiliza más para las transacciones B2B (B2B) que con usuarios individuales. La mayoría de los servidores web habilitados para SSL no solicitan autenticación de cliente.

Definición de un keystore para SSL/TLS

Cuando usamos Tomcat detras de otro servidor, como puede ser Apache, dicho servidor se encargará de la administración del SSL. De tal forma que en dicha configuración será Apache quien lo gestione y hara un intercambio de datos con el Servlet de tomcat

Definición de un keystore para SSL/TLS

Cuando usamos Tomcat detras de otro servidor, como puede ser Apache, dicho servidor se encargará de la administración del SSL. De tal forma que en dicha configuración será Apache quien lo gestione y hara un intercambio de datos con el Servlet de tomcat.

Definición de un keystore para SSL/TLS

Para implementar SSL, un servidor web debe tener un Certificado asociado para cada interfaz externa (dirección IP) que acepte conexiones seguras. La teoría detrás de este diseño es que un servidor debe proporcionar una cierta clase de seguridad razonable que su dueño es quién quien dice ser, particularmente antes de recibir cualquier información sensible.

Definición de un keystore para SSL/TLS

Los certificados tienen cifrada la firma de tal forma que es dificil copiarlos o alterarlos. Además para que un navegador acente un certificado sin advertencias es necesario que tambien sea firmado por un tercero de confianza.
Normalmente autoridades cetificadoras (CAs).
Hay una enorme cantidad de entidades certificadoras y cada una de ellas pone sus condiciones y tarifas para obtener su firma.
Salvo en proyectos donde la entidad certificadora sea importante, por ejemplo tiendas online, podemos usar autoridades certificadoras gratuitas como Let script

Definición de un keystore para SSL/TLS

Los certificados tienen cifrada la firma de tal forma que es dificil copiarlos o alterarlos. Además para que un navegador acente un certificado sin advertencias es necesario que tambien sea firmado por un tercero de confianza.
Normalmente autoridades cetificadoras (CAs).
Hay una enorme cantidad de entidades certificadoras y cada una de ellas pone sus condiciones y tarifas para obtener su firma.
Salvo en proyectos donde la entidad certificadora sea importante, por ejemplo tiendas online, podemos usar autoridades certificadoras gratuitas como Let's Encrypt SSL

Definición de un keystore para SSL/TLS

Para entornos de desarrollo o escenarios donde es suficiente con tener un certificado autofirmado Java proporciona una herramienta llamada keytool. Existiendo alternativas gratuitas yo no la uso.