TAA DISTRI

DESCRIBIR LOS SIGUIENTES CONCEPTOS

NTP significa "Protocolo de Tiempo en Red" (Network Time Protocol, en inglés). Es un protocolo de comunicación utilizado para sincronizar con precisión los relojes de dispositivos en una red informática. NTP se utiliza para asegurarse de que todos los dispositivos en una red tengan la misma hora y fecha, lo que es crucial para muchas aplicaciones informáticas, como la sincronización de servidores, transacciones financieras, registros de eventos y más. NTP se basa en una jerarquía de servidores de tiempo que proporcionan señales de tiempo precisas, permitiendo una sincronización confiable en redes locales e internet.

GIT

Git es un sistema de control de versiones distribuido ampliamente utilizado en desarrollo de software. Permite a los equipos de programadores llevar un registro de los cambios en su código fuente y colaborar de manera eficiente en proyectos.

El comando **"git push"** se utiliza para enviar o cargar los cambios locales realizados en un repositorio Git a un repositorio remoto.

En resumen, "git push" es una operación esencial en Git que permite compartir tus cambios locales con otros colaboradores en un repositorio remoto, lo que facilita la colaboración en proyectos de desarrollo de software.

SESSION BEANS

En resumen, las Session Beans son componentes esenciales en la plataforma Java EE para desarrollar aplicaciones empresariales. Los Stateless Session Beans se utilizan para lógica de negocio sin estado, mientras que los Stateful Session Beans se emplean cuando es necesario mantener el estado de un cliente a lo largo de múltiples interacciones. Estos componentes facilitan la gestión de la lógica empresarial y ofrecen servicios fundamentales para aplicaciones empresariales, como la gestión de transacciones y la seguridad.

WEB SERVICE

Un servicio web (web service en inglés) es un conjunto de protocolos y estándares que permiten que diferentes aplicaciones o sistemas puedan comunicarse entre sí a través de la World Wide Web. Estos servicios permiten que aplicaciones desarrolladas en diferentes lenguajes de programación y que se ejecutan en diferentes plataformas puedan intercambiar datos y realizar acciones en conjunto.

Los servicios web se basan en estándares como XML (Extensible Markup Language) y HTTP (Hypertext Transfer Protocol) para la transmisión de datos y la descripción de servicios. Uno de los formatos más comunes para describir servicios web es el lenguaje de descripción de servicios web (WSDL, por sus siglas en inglés).

Los servicios web se utilizan ampliamente en aplicaciones distribuidas y en la integración de sistemas heterogéneos, lo que permite a las aplicaciones compartir información y funcionalidades de manera eficiente y escalable a través de Internet o redes corporativas. Esto los hace esenciales en la construcción de aplicaciones modernas y en la interconexión de sistemas en entornos empresariales.

CITAR

códigos de estado HTTP junto con breves descripciones de lo que representan:

1. **200 OK**: La solicitud se completó con éxito y la respuesta contiene la información solicitada.
2. **201 Created**: Indica que la solicitud ha tenido éxito y ha resultado en la creación de un nuevo recurso.
3. **204 No Content**: La solicitud se completó con éxito, pero no hay contenido para devolver en la respuesta.
4. **400 Bad Request**: Indica que la solicitud del cliente es incorrecta o defectuosa de alguna manera.
5. **401 Unauthorized**: El cliente no está autenticado y se requiere autenticación para acceder al recurso.
6. **403 Forbidden**: El cliente está autenticado pero no tiene permiso para acceder al recurso solicitado.
7. **404 Not Found**: El recurso solicitado no se encuentra en el servidor.
8. **500 Internal Server Error**: Indica un error interno en el servidor que impide que la solicitud se complete con éxito.
9. **502 Bad Gateway**: Se utiliza cuando un servidor actuando como puerta de enlace o proxy recibe una respuesta no válida del servidor al que está reenviando la solicitud.
10. **503 Service Unavailable**: El servidor no puede atender la solicitud en este momento debido a una sobrecarga temporal o mantenimiento.
11. **504 Gateway Timeout**: Indica que el servidor proxy o puerta de enlace ha agotado el tiempo de espera al esperar una respuesta del servidor final.

CARACTERISTICAS DE LOS SISTEMAS DISTRIBUIDOS

Los sistemas distribuidos son sistemas informáticos que consisten en múltiples componentes o nodos que se ejecutan en hardware separado y están conectados entre sí a través de una red. Aquí tienes cuatro características clave de los sistemas distribuidos:

1. **Transparencia de la ubicación**: Los usuarios y las aplicaciones pueden acceder a los recursos distribuidos sin necesidad de conocer la ubicación física de esos recursos. Esto significa que los recursos pueden estar en diferentes lugares geográficos, pero los usuarios los perciben como si estuvieran localmente disponibles.
2. **Concurrencia y compartición de recursos**: En los sistemas distribuidos, múltiples usuarios o aplicaciones pueden acceder y compartir recursos al mismo tiempo. La gestión adecuada de la concurrencia es esencial para garantizar que los recursos se utilicen eficientemente y que no se produzcan conflictos.
3. **Escalabilidad**: Los sistemas distribuidos suelen ser escalables, lo que significa que pueden crecer o reducirse fácilmente según las necesidades. Se pueden agregar nodos o recursos adicionales para aumentar la capacidad del sistema, lo que lo hace adecuado para aplicaciones con requisitos cambiantes de rendimiento y carga.

4. **Tolerancia a fallos**: Los sistemas distribuidos a menudo incorporan mecanismos para manejar fallos en los componentes individuales o en la red. Esto implica la capacidad de detectar fallos, recuperarse de ellos y continuar funcionando de manera confiable incluso cuando algunos componentes fallan.

Estas características son fundamentales para comprender cómo funcionan y se gestionan los sistemas distribuidos, ya que permiten que múltiples recursos y nodos colaboren de manera efectiva en un entorno de red distribuido.

COMPLETAR EL ESPACIO ENTRE COMILLAS

El lenguaje de definición de servicios web está representado mediante un URL, cuyo contenido tiene el formato de un documento **"XML" (Extensible Markup Language).**

En java, al utilizar el protocolo TCP, el método **read()** es utilizado para recibir mensajes y el método **write()** es utilizado para enviar mensajes.

El modo **client/server** del protocolo NTP es similar al funcionamiento del algoritmo de Cristian. Un servidor acepta solicitudes de otros computadores, que el procesa respondiendo con su marca de tiempo (lectura actual del reloj).

Un metodo para la compracion de fallos en la transmision es: **checksum**

La **deriva** del reloj alude a la proporcion en que el reloj de un computador difiere del reloj de referencia perfecto

En una comunicación asíncrona, la primitiva "envia" es **no bloqueante** y la primitiva "recibe" es **bloqueante.**

CITAR EN JSON Y XML CODIGOS QUE INDIQUEN CAMBIO DE TIEMPO

JSON:

```
{
  "datosClimaticos": {
    "ciudad": "Nueva York",
    "hora": "12:00 PM",
    "temperatura": "25°C",
    "humedad": "60%",
    "condiciones": "Parcialmente nublado"
  }
}
```

XML:

```
<datosClimaticos>
  <ciudad>Nueva York</ciudad>
  <hora>12:00 PM</hora>
  <temperatura>25°C</temperatura>
  <humedad>60%</humedad>
  <condiciones>Parcialmente nublado</condiciones>
</datosClimaticos>
```

Ambos formatos representan la misma información: la ciudad (Nueva York), la hora de la medición (12:00 PM), la temperatura (25°C), la humedad (60%) y las condiciones climáticas (Parcialmente nublado) en un formato estructurado de datos para su fácil lectura y procesamiento.

DESAFIOS DE LOS SITEMAS DISTRIBUIDOS

Los sistemas distribuidos presentan varios desafíos debido a su naturaleza de operar en entornos en los que los recursos y procesos están distribuidos en múltiples nodos interconectados. A continuación, se citan y se explican algunos de los desafíos más importantes:

1. **Comunicación y latencia**: En los sistemas distribuidos, la comunicación entre nodos a través de una red introduce latencia y posibles retrasos. Los desafíos incluyen la gestión de la latencia, la pérdida de paquetes y la garantía de que los mensajes lleguen en el orden correcto.
2. **Consistencia de datos**: Mantener la consistencia de los datos distribuidos es un desafío fundamental. La replicación de datos en múltiples nodos puede llevar a problemas de coherencia y conflictos, que deben ser manejados mediante algoritmos de conciliación y estrategias de consistencia.
3. **Tolerancia a fallos**: Los sistemas distribuidos deben ser capaces de manejar fallos de hardware y software de manera efectiva. Esto implica detectar fallos, recuperarse de ellos y continuar operando de manera confiable. La tolerancia a fallos es esencial para garantizar la disponibilidad y la confiabilidad del sistema.
4. **Coordinación y concurrencia**: La coordinación de múltiples nodos concurrentes puede llevar a problemas como condiciones de carrera y bloqueos. La gestión de la concurrencia y la sincronización de procesos distribuidos son desafíos importantes en estos sistemas.
5. **Escalabilidad**: Los sistemas distribuidos deben ser escalables para adaptarse al crecimiento de la carga de trabajo y la adición de nuevos nodos. La escalabilidad debe diseñarse teniendo en cuenta la distribución de la carga y la gestión eficiente de recursos.
6. **Seguridad**: La seguridad en sistemas distribuidos es crítica debido a la exposición a amenazas de red y ataques. Se deben implementar mecanismos de autenticación, autorización y cifrado para proteger los datos y la integridad del sistema.

7. **Gestión de recursos**: La gestión eficiente de los recursos, como el almacenamiento y el ancho de banda de red, es un desafío importante. La asignación de recursos adecuados y su monitoreo constante son esenciales para el rendimiento y la disponibilidad.
8. **Mantenimiento y actualizaciones**: Coordinar el mantenimiento y las actualizaciones en sistemas distribuidos puede ser complicado. Es importante garantizar que las actualizaciones no causen interrupciones y que se puedan implementar de manera escalonada.
9. **Localización de servicios y descubrimiento**: En sistemas distribuidos, encontrar y acceder a servicios y recursos distribuidos de manera eficiente es un desafío. Se utilizan sistemas de descubrimiento y registros para abordar este problema.
10. **Cumplimiento de estándares y protocolos**: La interoperabilidad entre diferentes sistemas y tecnologías es crucial. Cumplir con estándares y protocolos establecidos puede ser un desafío para garantizar la compatibilidad y la comunicación efectiva entre sistemas heterogéneos.

En resumen, los sistemas distribuidos son complejos y presentan una serie de desafíos que van desde la gestión de la comunicación hasta la seguridad y la escalabilidad. Abordar estos desafíos de manera efectiva es esencial para construir sistemas distribuidos confiables y eficientes.

****Socket**:**

Un **socket** es una interfaz de programación de aplicaciones (API) que proporciona un mecanismo para que los programas de software se comuniquen a través de una red, ya sea en la misma máquina o a través de una red local o global, como Internet. Los sockets permiten que las aplicaciones envíen y reciban datos de manera eficiente y flexible, facilitando la comunicación entre procesos en diferentes dispositivos o incluso en diferentes ubicaciones geográficas.

****Funcionamiento**:**

Los sockets funcionan mediante un modelo cliente-servidor. Aquí se explica brevemente cómo funciona:

- **Servidor**: El servidor crea un socket y escucha en un puerto específico para recibir conexiones entrantes. Cuando un cliente se conecta, el servidor acepta la conexión, crea un nuevo socket para esa conexión y comienza a comunicarse con el cliente a través de ese socket.

- **Cliente**: El cliente crea un socket y se conecta al servidor especificando la dirección IP y el puerto del servidor al que desea conectarse. Una vez establecida la conexión, el cliente y el servidor pueden intercambiar datos a través de los sockets.

Tipos de Sockets:

Existen dos tipos principales de sockets:

1. **Sockets de flujo (TCP - Transmission Control Protocol)**:

- Proporcionan una comunicación orientada a la conexión y confiable.

- Aseguran que los datos se entreguen en el orden correcto y sin errores.

- Son adecuados para aplicaciones que requieren transferencia de datos confiable, como transferencia de archivos y navegación web.

2. **Sockets de datagrama (UDP - User Datagram Protocol)**:

- Proporcionan una comunicación no orientada a la conexión y menos confiable.

- No garantizan la entrega de datos ni el orden de llegada.

- Son adecuados para aplicaciones que requieren una comunicación rápida y ligera, como transmisiones en tiempo real y videojuegos.

Gráfico Explicativo:

```
+--------+            +--------+
| Cliente|              | Servidor|
+--------+            +--------+
    |                   |
    | ---Solicitud de conexión-->|
    |                   |
    | <----Aceptación de conexión--|
    |                   |
    | ---Envío de datos-------->|
    |                   |
    |                   |
    | <---Recepción de datos----|
    |                   |
    | ---Cierre de conexión--->|
    |                   |
```

Este gráfico muestra cómo el cliente y el servidor establecen una conexión, intercambian datos y finalmente cierran la conexión. Los datos se envían y reciben a través de los sockets de flujo TCP en este ejemplo.