# Instroducción a Services Daemon (Atlas Middleware)

Es una implementación en java de un servlet, usando Jetty empotrado, usando una arquitectura de módulos auto cargados, organizado en una jerarquía de directorios, repetitiva para Managers y Services dependientes de cada Managers.

# Función del Services Daemon (Atlas Middleware)

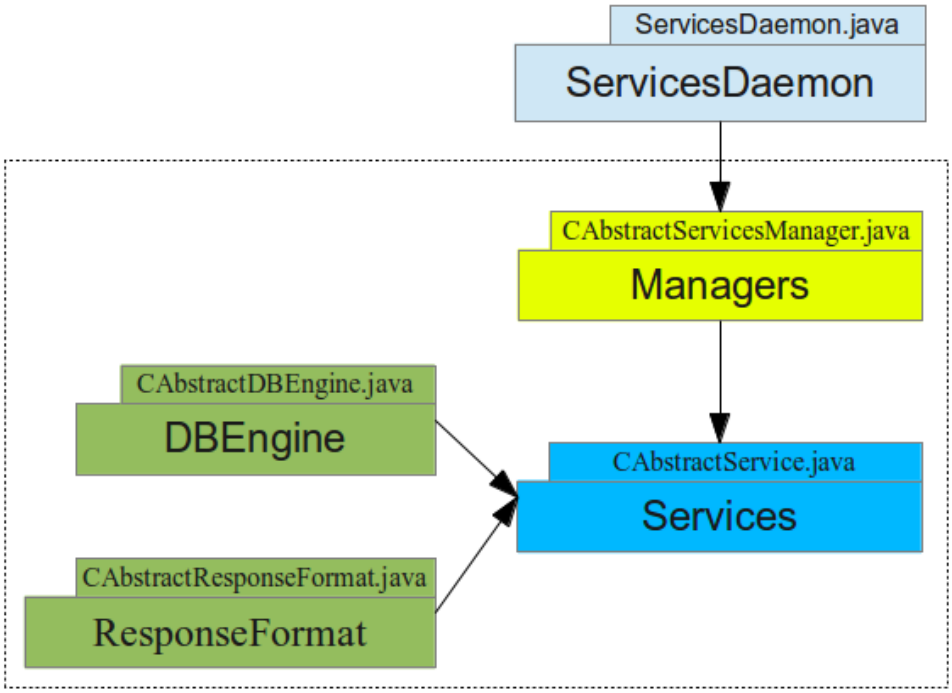
La función del services daemon es la de integración de datos, una pasarela que usa para transportar información mediante el muy conocido protocolo HTTP.

Para el caso del Manager llamado DBServicesManager, puede ser visto como una herramienta ETL o Middleware, en la que un cliente con capacidad para acceder a recursos mediante el protocolo http, opcionalmente y según la configuración se puede usar https.

Aunque personalmente lo definiría como un herramienta ET (Extract And Transforms) por que la parte del Load se le delega al cliente que solicita la información, todo depende de para que se use y como se use.

# Arquitectura del Services Daemon

La arquitectura del Services Daemon es básicamente modular




Usando la arquitectura mostrada permite desarrollar nuevos Managers (Gerentes) extendiendo la clase CAbstractServicesManager a su vez es posible implementar servicios específicos para su manager implementando la interfaz CAbstractService, eso le permite escalar prácticamente de manera

indefinida.

# Función del ServicesDaemon (ServicesDaemon)

La función del ServicesDaemon es la de coordinar mediante el uso del jetty las peticiones HTTP/HTTPS los clientes, además de la lógica del servlet container propiamente dicha, interfaz de red, puerto TCP, etc.

# Función del Manager (CAbstractServicesManager)

La función del Manager es la de coordinar la lógica del contexto que implementan, por ejemplo el DBServicesManager, tiene un contexto de /DBServices para acceder a los servicios que gerencia este manager es con la siguiente url http://127.0.0.1/DBServices

Se podría implementar N cantidad de Managers (Gerentes) cada uno con contextos distintos y acceder a los servicios de los mismos usando su contexto en la URL.

Por ejemplo imagínese que quiere implementar el manager llamado JavaObjectsManager y su contexto es /JavaObjects el url para acceder, sería el siguiente url http://127.0.0.1/JavaObjects para que el ServicesDaemon tenga conocimiento, de que es el manager llamado es el JavaObjectsManager y no el DBServicesManager, esto se hace siguiendo la misma lógica, de los servlet de java y el manejo de contextos en las URL.

# Función del Services (CAbstractService)

La función del Services es la de implementar un servicio específico que depende directamente de su manager a nivel lógico,  con el apoyo de ResponseFormat y DBEngine, en está clase se concentra toda la lógica real del sistema o de los servicios.

Cada servicio es invocado por su nombre el cual debe ser único para su manager, usando un parámetro en la llamada denominado ServiceName, por ejemplo solo debe existir un servicio de nombre System.Enum.Services en el contexto /DBServices, otro servicio con el mismo nombre será simplemente ignorado al momento de ser cargado y registrado por su Manager, proceso que sucede cuando inicia el ServicesDaemon, claro nada de esto impide que usted use el nombre System.Enum.ServicesA o el de System.Enum.ServicesB por ejemplo para tener otros dos servicios que hacen algo similar pero de distinta forma o lógica, pero para este propósito recomiendo usar los parámetros de llamada al servicio y no su nombre solamente.

Continuando con la idea anterior puede existir un System.Enum.Services en el manager /DBServices y otro de igual nombre pero con manager distinto como por ejemplo el /JavaObjects, aunque ambos realicen la misma tarea, para el caso del System.Enum.Services enumeran todos los servicios disponibles con sus parámetros de llamada.

# Función del DBEngine (CAbstractDBEngine)

La función del DBEngine es la de apoyar a los servicios con el acceso a base de datos especificas, MySQL, Firebird, PostgreSQL y M$ SQL Server entre muchos otros sistemas manejadores de base de

datos, esta clase es creada básicamente para en un futuro implementar particularidades de cada manejador en especial la manera de como tratar consultas que retornan valores con filas que son de tipo BLOB o valores binarios.

Por ejemplo PostgresSQL versión 9.1, trata a los blob como BinaryStream, el api estándar del JDBC para los BLOB no funciona adecuadamente, por eso se tomo la decisión de agregar una capa de abstracción adicional, a la ya proporcionada por el JDBC de java, el cual realiza una excelente labor estandarizando la manera de acceder a la data mediante los métodos de sus objetos, sin embargo, el uso de los mismo siempre es específico para cada manejador, y por ende a su lógica de trabajo, por eso cada implementación de esta clase abstrae al servicio de tener que tratar en el código las diversas lógicas de los SMBD.

Para poder implementar nuevos DBEngines que hagan uso del driver JDBC, primero se debe tener el driver java para acceder a la base de datos por lo general un archivo .jar que debe ser colocado en el directorio DBDrivers, luego se debe crear una clase que extienda la clase CAbstractDBEngine, crear el archivo .jar del DBEngine y colocarlo en el directorio DBEngines, tanto la carpeta de DBEngines y DBDrivers están dentro de la carpeta DBServicesManager.

# Función del ResponseFormat (CAbstractResponseFormat)

La función del ReponseFormat es la de permitir presentar la información en diversos formatos, según se indique en la configuración del manager o del propio servicio, como el DBEngine es una clase de apoyo que debe extender la clase CAbstractResponseFormat.

Los formatos de respuesta empotrados en el DBServicesManager son:

XML-DATAPACKET
JAVA-XML-WEBROWSET
JSON
CSV

## ¿Por que se les dice que son formatos de respuesta empotrados?

Por que estos formatos de respuesta son esenciales para el DBServicesManager si no tiene ninguno, el manager es incapaz de devolver a los clientes alguna respuesta ¡Ni siquiera un mensaje de error!, el manager y sus servicios estarían completamente mudos, para evitar este problema se decidió empotrar estos formatos dentro del manager, específicamente para el DBServicesManager, sin embargo es posible que para otros managers que usted implemente no tome esta decisión tan radical, si no que confié en que la carpeta ResponsesFormats nunca este vacía.

Por esa razón puede que vea la carpeta ResponsesFormats vacía, es decir, sin ningún .jar en su interior.

# Texto plano como codificación

El texto plano es el medio de codificación informático más común y simple, lo más cercano a universal que conozco en mis 16 años en el mundo de la informática, además de ellos son los más

resistentes a corrupción o perdida de datos. La familia de los sistemas operativos Unix confían en ellos para guardar la información de configuración de todos sus servicios, no se puede tener mejor referencia que esa. Sin embargo no son los más óptimos para grandes volúmenes de información, esta afirmación no es 100% correcta hoy día, debido a que los procesadores son cada vez más potentes y baratos, el procesamiento de grandes cantidades de información codificada en texto plano se realiza cada vez con mayor facilidad, aunado a que la memoria RAM de los equipos es cada día más rápida y barata, por eso se decide el uso de los mismos.

Otro de los motivos fundamentales es la facilidad para ser leídos y procesados por prácticamente por cualquier lenguaje de programación de computadoras, desde el antiguo y venerado COBOL hasta las modernas plataformas tecnológicas como Java y C#, todos tienen métodos y rutinas en sus librerías de sistemas para cargar y procesar este tipo de codificación, incluso un ser humano podría procesarlo con la ayuda de un simple editor de texto. Todo apunta que el soporte para esta clase de codificación seguirá presente en futuros lenguajes de programación por ser algo extremadamente básico y simple, tan básico y simple como las operaciones aritméticas, presentes en absolutamente todos los lenguajes.

En definitiva en mi experiencia de 14 años de programación con múltiples lenguajes me dice que el medio de codificación de texto plano es el más idóneo.

# HTTP como protocolo de transporte para la información

El super conocido protocolo de Word Wide Web, la www para resumir, diseñado para ..., si eso transportar texto, no lo confundan con el html, el html es solo un formato de presentación de información que está codificado en texto plano, no se me ocurre mejor protocolo para usar en el ServiceDaemon, aunado a la increíble robustez a prueba de bombas nucleares, de manera literal, este protocolo es prácticamente indestructible, no hay nada que no pueda transportar mientras lo que trasporte este codificado en texto plano.

Los detalles en los que puedo pensar para este protocolo son los siguientes:

1. **Para transportar archivos binarios como imágenes, musica entre otros estos archivos deben ser codificados en Base64**, para ser convertidos de una secuencia de bytes crudos a una secuencia de bytes representables en texto, de la A a la Z en mayúscula y minúsculas y otros símbolos como el =, no; no incluye la n con tilde (eñe) ni otros caracteres o símbolos, está técnica es ampliamente usada en los correos electrónicos, para transportar archivos atachados, esto es solo un inconveniente que se solventa en el ServiceDaemon, usando la técnica del Base64 (Existen otras pero yo uso solo esa).

2. **Es fácil interceptar la información, cuando viaja por las redes debido a que es texto plano**, sin embargo, al igual que el anterior punto se puede solucionar usando HTTPS, una versión con una capa de encriptación y certificados digitales, el cual es ampliamente usado por la banca electrónica y militares, no conozco a dos sectores más paranoicos que estos (aparte de los terroristas y estafadores) a si que creo es bastante sólido como para que ellos confíen en el, por lo que yo deberé hacer lo mismo.

3. **No todos los lenguajes de programación soportan acceder a recursos HTTP**, este es el problema más peliagudo que le pude encontrar, pero hoy día no es tan grave, de hecho los más viejos como COBOL, si el venerado y viejo COBOL ni siquiera saben como demonios tratar

con una página web, mucho menos acceder a ella vía una red TCP/IP estándar, ya las plataformas de los 90 o la gran mayoría tienen acceso como clientes HTTP, hoy día usted se consideraría con suerte si encontrará un lenguaje de programación que no tenga soporte para el mismo, pero de todo hay en la viña del señor, por lo que el HTTP no es tan "Universal" como podría ser el caso de la codificación de texto la cual es casi universal.

4. **La librerías del base64 tampoco son muy estándares y no están en todos los lenguajes de programación**, recordando que solo se necesitaran si se debe enviar o recibir archivos binarios, aunque el base64 es un algoritmo de hash de bastante edad, hay muchos lenguajes de programación que no lo soportan, sin embargo si el lenguaje de programación soporta en sus librerías el acceso como cliente HTTP de seguro, también soporta el base64, este algoritmo se usa mucho para otras operaciones sobre archivos.

# Los formatos de respuesta

El ServiceDaemon fue diseñado con una arquitectura modular auto cargable y para los formatos de respuesta se cuenta con la clase **CAbstractResponseFormat**, de la cual derivan todos los formatos de respuesta pero por defecto se soportan 3 familias XML (XML-DATAPACKET, JAVA-XML-WEBROWSET), JSON, CSV.