Proyecto Aemet

Desarrollo de Aplicaciones para Ciencia de Datos (DACD)

Segundo curso

Grado en Ciencia e Ingeniería de Datos

Escuela de Ingeniería Informática (EII)

Universidad de Las Palmas de Gran Canaria (ULPGC)

Índice

1. Resumen:

El proyecto está diseñado a través de tres módulos cada uno con una función específica. El objetivo global es obtener registros de temperatura de Gran canaria y guardarlos en un fichero datalake que contendrá un fichero .event con los registros de cada fecha. Estos ficheros deben ser leídos y se deben extraer la temperatura mínima y máxima de cada archivo, para ser insertados en un datamart con una tabla de máximos y una de mínimos. Por último, estos datos deberán ser expuestos en una api donde se puedan consultar temperaturas mínimas y máximas filtrando por fecha.

Para ello tendremos tres módulos que se dedicarán a cada tarea en particular del proceso. El módulo feeder debe generar los archivos .events que contienen los registros, estos se crearán en la carpeta datalake dentro del módulo. Para procesar estos archivos y crear el datamart se usarán los componentes del módulo datamart-builder. El datamart se creará en una carperta datamart en el feeder. Por último, tenemos un módulo api-rest que se encargará de lanzar el servicio web y habilitar las direcciones para la consulta de los datos en el datamart. Estas se lanzarán en las siguientes:

- /v1/places/with-max-temperature Devuelve las temperaturas máximas de los días registrados y los datos relacionados.
- /v1/places/with-min-temperature Devuelve las temperaturas mínimas de los días registrados y los datos relacionados.

Para las dos direcciones podremos usar from y to como parámetros query para solo obtener las medidas entre las fechas especificadas. Estas consultas nos devolverán una respuesta en formato json que consiste en una lista de medidas donde cada medida contiene la fecha(date), hora(time), lugar(place), estación (station) y el valor de la temperatura mínima o máxima(value).

2. Diagrama de clases:

Para no llenar la memoria con tres páginas de diagrama, insertaré los enlaces de github que contienen las imágenes de los diagramas:

- Feeder: https://github.com/Alejeglez/aemet/blob/master/Diagrama/ClassDiagramFeeder.jpg?raw=true
- Datamart-builder: https://github.com/Alejeglez/aemet/blob/master/Diagrama/DatamartBuilderDiagramClass.jpg?raw=true
- Api-rest: https://github.com/Alejeglez/aemet/blob/master/Diagrama/DiagramaApRest.jpg?raw=true

3. Recursos usados:

- **IDE:** IntelliJ IDEA Community Edition 2022.2.2.
- **Control de versiones:** GIT (integrado en IntelliJ)**.**
- **Herramienta diagrama de clases:** StarUml.
- **Herramienta de documentación:** Word.

- **Versión java:** 11
- **Librerías externas:** jsoup(1.15.3), gson(2.10) y spark(2.9.4).

4. Diseño:

Como vimos en el resumen tenemos un total de tres módulos, a continuación, desglosaré cada uno de ellos.

- Feeder:

  El elemento básico de este módulo es la interfaz sensor. Esta interfaz permitirá implementar los métodos necesarios para cualquier sensor que suministré datos meteorológicos. En este caso solo tendrá un método getData que devuelve una String.

  Su implementación en nuestro programa es AemetSensor que necesita de una apiKey para poder acceder a los datos. Gracias a la librería jsoup y a la función connect nos conectamos a la url proporcionada en el enunciado haciendo uso de la función getFirstUrl() que recibe como parámetro una String. La respuesta de este primer enlace la pasamos a un JSONObject del cual obtenemos el valor del campo datos que contiene la url con todos los registros de España de las últimas 24 horas. Se repite el proceso sobre este link empleando la función getMeasures() que nos devuelve el contenido. El resultado se pasa a formato JsonArray y gracias a la función fromJsonArrayToList lo recorremos y convertimos a una lista de JsonObject. Esta lista se pasará como parámetro a la función filterList que mediante streams filtra la lista para obtener las medidas de Gran Canaria según longitud y latitud. El método getData devolverá esta lista como json.

  FeederTask recibe como parámetro este json, lo convierte a un JsonArray que recorremos como el anterior, y almacenaremos los campos necesarios del JsonObject para fabricar un objeto Weather. Estos objetos necesitan como parámetro una fecha, una estación, un lugar, una temperatura máxima, una temperatura mínima y una temperatura. El método filterToList devolverá una lista de estos objetos.

  Para el siguiente paso necesitamos una interfaz Datalake que contendrá los métodos para trabajar con datalakes. Estos son save que recibe como parámetro una lista de Weathers y las guarda en un archivo, y read que se encarga de leer un archivo devolviendo una lista de Weather.

  La clase DatalakeFile implementa esta interfaz y recibe como parámetro un path que ya viene determinado. El método save lo primero que comprueba es si el path existe y si no crea la carpeta. Luego, para cada Weather que se encuentra en la lista comprueba la fecha y crea o conecta con el archivo correspondiente. Tras eso se pasa el objeto Weather a json y si el archivo no lo contiene se incluye en él. Por su parte el método read lee el contenido de un archivo y los pasa con gson de json a una lista de Weathers.

La clase Controller se encarga de llamar a las funciones correspondientemente. Para su construcción requiere de un Sensor, un Datalake y un FeederTask. Para ejecutarlo usamos el método run.

La clase Main crea y ejecuta un Controller cada hora.

- Datamart-builder:

Este módulo contiene las clases ya explicadas DatalakeFile y Weather así como la interfaz Datalake.

Este módulo contiene las clases necesarias para conectarse a una base de datos sqlite. La primera clase básica es CreateConnection que recibe como parámetro un nombre con el cuál se creará la base de datos y se encarga de establecer una conexión con esta mediante el método connect que devuelve un Connection. Este método elimina el datamart existente y crea uno nuevo cada vez que se ejecuta.

La siguiente clase necesaria es un DdlTranslator para modificar la base de datos y crear tablas. Contiene dos métodos createTableMinTemperatures() y createTableMaxTemperatures( ). Ambos métodos crean una tabla si no existe de temperaturas mínimas y máximas correspondientemente. Cada tabla contendrá una fecha (primary key), una hora, un lugar, una estación y un valor (mínimo o máximo). Necesita una conexión a la base datos para trabajar.

La clase DmlTranslator recibe una conexión con la base de datos y contiene los métodos minToDml y maxToDml que reciben los parámetros necesarios para un registro de las tablas del datamart y crea los statements necesarios para insertarlos en las tablas.

DatamartInsert se encarga de llamar a de crear objetos de DmlTranslator para insertar los registros a través de los métodos insertStatementOfMax() e insertStatementOfMin() que reciben como parámetros un objeto Weather. Para crear objetos de esta clase necesitamos un connection.

El Weather usado en los métodos de DatamartInsert es devuelto por los métodos de FilterWeather que recibe una lista de Weather. Los métodos minWeather() y maxWeather() crean una stream a partir de la lista y usando Comparator.comparing encuentra el mínimo(tmin) y máximo(tmax) respectivamente.

El Controller al igual que en el feeder llama a las funciones en orden y en su construcción crea un CreateConnection, una connection a partir de él y que

usa para crear una DdlTranslator y un DatamartInsert. También recorrerá todos los archivos en una carpeta para conseguir el mínimo y máximo de todos los archivos. Al final cerrará la conexión con la base de datos. Para ejecutar la rutina del Controller usamos el método run.

La clase Main crea un controller cada día y ejecuta su rutina run().

- Api-rest:

  Este módulo contendrá la clase CreateConnection explicada anteriormente.

  Lo más importante de este módulo es el web service en sí por lo que creamos una interfaz para ello. La interfaz WebService incluye los métodos launch, que lanza las direcciones para consulta, y los métodos getMinTemperatures y getMaxTemperatures que reciben como parámetros un Request y un Response.

  Los objetos que son expuestos en la api son distintos a los Weather usados anteriormente porque o tienen la temperatura mínima o la máxima. La clase WeatherDB es una simplificación de Weather que contiene los mismos campos salvo los relacionados a las temperaturas que resume en un solo campo value(double).

  La clase DatamartReader requiere como parámetro de una conexión con la base de datos. Contiene los métodos readMaxTable() y readMinTable(). En ellos se crea un statement para seleccionar todos los valores de la tabla correspondiente que devolverá un result set. Iterando en este result set creamos objetos WeatherDB que añadimos a una lista. Ambos métodos devolverán un json de esta lista.

  La implementación de WebService para el caso particular de Aemet. Necesita un Controller al que llamará para interactuar con DatamartReader. El método launch lanza las direcciones mencionadas en el resumen. Los métodos getMinTemperatures y getMaxTemperatures devuelven una String. La respuesta será en formato json y primero buscará si hay algún queryParam que sea from o to. Si alguno no ha sido introducido no se filtrará por fechas y se devolverá directamente el contenido completo de una tabla de la base de datos desde el Controller. En caso contrario, llamamos a la función filterByDate que recibe como parámetros una String de la fecha origen, otra de la fecha límite y otro del json con todos los registros. El método formatea las fechas, convierte el json a un array de WeatherDB y lo recorre. Compara la fecha de cada objeto con las establecidas y si entra dentro del límite lo incluye en una lista que contiene los WeatherDB filtrados y que será devuelta en formato Json.

La clase controller lanza los métodos en el orden oportuno y necesita de un CreateConnection. El método run creará una nuevo WebService y lo lanzará. También incluye dos métodos de getMinTemperatures() y getMaxTemperatures() que llaman a los métodos con esos nombres del DatamartReader.

En cuanto al diseño, podemos ver como sea aplica el principio de inversión de la dependencia al usar interfaces en lugar de sus implementaciones.

El principal principio de diseño arquitectónico es el MVC(Model-View-Controller). En los diagramas de clases podemos ver esta característica reflejada ya que ninguna de las clases se relaciona entre ellas sino el Controller es el que las aglutina y relaciona.

5. Conclusiones:

En el desarrollo del programa he aprendido sobre entidades como el JsonObject o JsonArray que no conocía previamente y simplifican mucho las tareas en este proyecto, ya que mi idea principal era hacer las conversiones mediante el apoyo de mapas.

En el comienzo del trabajo tuve un problema con los null pointer en algunas medidas que no tenían algún valor. Para ello implemente un try, catch asi puede crear un objeto para la medida, pero en caso de que no sea posible lo ignorará y continuará su ejecución. No sé cómo de recomendable será esta práctica.

Otra de las estructuras que he podido ver en la práctica han sido las streams. Estas estructuras son de gran utilidad ya que para filtrar solo las medidas de Gran Canaria muy probablemente hubiera usado bucles for iterados y procedimientos similares para encontrar las temperaturas mínimas y máximas.

Por último, tuve problemas con la conexión entre la base de datos y la api. Para ello hice que los métodos getMinTemperatures() y getMaxTemperatures() sean los que crean una conexión para el DatamartReader y se encarguen de cerrarla. De esta manera, la probabilidad de que haya un error se basa en realizar una petición en la api y que justo coincida con el momento en el que se borra el

datamart. En este caso, daría un error por repetición de primary key, ya que no habrá podido borrar el archivo porque la api lo está usando.

6. Líneas Futuras:

El proyecto necesitaría de mejoras para poder comercializarse. Como comenté antes hay una probabilidad de excepción en caso de que la renovación del datamart coincida con una petición de la api. Por lo tanto, sería recomendable buscar una forma de que el datamart se renovará sin necesidad de borrarlo.

En el concepto del producto el dominio está limitado ya que solo tiene en cuenta a Gran Canaria. Podríamos pedir como argumento la localidad de la que deseamos obtener las medidas. Otra forma podría ser crear tablas para todas las localidades y que mediante tablas relacionales tengamos una tabla de min y max que indexe localidades.

Podría ser interesante incorporar los datos de alguna plataforma para poder realizar una comparativa, para ello podemos crear otro sensor. De esta manera los clientes recibirían una información más precisa.

7. Bibliografía:

- Ejecutar tareas cada hora: https://stackoverflow.com/questions/10748212/how-to-call-function-every-hour-also-how-can-i-loop-this

8. Notas:

- El proyecto en el repositorio de Github no se encuentra el valor de apiKey por lo que será preciso que se introduzca en la clase Controller del módulo feeder (línea 18).
- Las imágenes ocupaban mucho espacio del documento, por lo que las he subido al repositorio de Github en la carpeta diagramas.
- He estado un buen rato, intentando solucionar un problema al subir los archivos a github, y es que cada vez que los subo las clases de dos de los módulos aparecen como archivos.java. He probado bastantes cosas y no consigo que esto cambie. La solución que encontré fue subir el repositorio de github con otro nombre al parecer era eso. Lamento el retraso de una hora en la entrega.

Alejandro Jesús González Santana