Healing

Agenda

As an SDC user, I would like to keep information updated with latest changes (the update should be seamless). This process is performed on a graph vertex, by calling healing implementations and applying the new (healed) data into the vertex without saving it in DB.

Before haling, we retrieved the data from DB and returned it to the user. With this process, we update the vertex data (if needed) and overriding the old data before returning it.

Healing version

Defined in dao.properties file.

Using this version, the healing mechanism runs the healers, by comparing it to vertex version and healer versions.

Heal interface

Healing interface contains two methods –

fromVersion()

returns the healer version that it is working on

heal(V parentV)

this method heals the data that is inside vertex parentV. **Notice – the healed data is not getting saved in DB. It returns to the user everytime that there is a read action from DB.** Each healer is associated to an edge label, and works on the data that's inside this node.

After the healed data is generated, it should be applied to the vertex (for example – setting the new vertex JSON data that was generated during the process).

HealingPipelineDao

Manages healing process.

This class contains a map member named **healingPipeline**, that holds the corresponding healer implementations for each EdgeLabelEnum name that needs to go through healing.

```
{
        "edge_name_1" -> healer1,
        "edge_name_2" -> healer2,
        "edge_label_3" -> [healer3, healer4]
}
```

shouldHeal (HealVersion<Integer> healerVersion, HealVersion<Integer> vertexVersion)

this method tells us if the current vertex needs to get healed by a specific healer, using their versions. healerVersion is the version of the current healer, that is getting set once the healer is created, and vertexVersion is the vertex of the current child node that we are working on. Once we have a healer for the current node (using its edge label), and shouldHeal returns true – the corresponding healer is getting called.

HealGraphDao interface

This interface contains one method – performGraphReadHealing and has three implementations, for each vertex type –

1. JanusGraphVertex
2. GraphNode
3. GraphVertex

This method retrieves the corresponding healers for the vertex that we are working on, and executes the healing for each healer.

Flow

1. **Get element**
   Once the user tries to retrieve data from DB, we call janusGraphDao in order to get this information.

2. **getChildrenVertices (HealingJanusGraphDao)**
   when we read the requested node from DB, we read its child vertices as well. Each child vertex holds an edge label for representation.

3. **transformVertex (HealingJanusGraphDao)**
   when we are in healing flow, we call HealingJanusGraphDao (that extends from the original JanusGraphDao), which calls healingPipelineDao to start the healing process

4. **preformGraphReadHealing & supplyHealer (HealingPipelineDao)**
   these methods return the appropriate HealGraphDao implementation, according to the vertex type

5. **performGraphReadHealing (HealGraphDao)**
   start of the healing process, according to the implementation.

6. **getHealersForVertex & shouldHeal (HealingPipelineDao)**
   using the edgeLabel of the child vertex, we read the healers from healingPipelineDao map, and for each healer we check if it needs to run, by checking its version and vertex version

7. **healGraphVertex (HealGraphDao)**
   after retrieving the healers list for the specific edge label, we call all of the healers' "heal" method on the current child vertex

8. **healData (Heal)**
   healing the vertex data, and appending the new data into it.