

## 1. Where to access my deployed services?

### Server's Information

I find two servers which have the same configuration. Operation system of the server is Centos 7 and the Tomcat version is 7.

VM001 ip address: 159.226.111.19

VM002 ip address: 159.226.111.20

There are several RESTFul web services on VM001:

- 1) Initialize service:  
<http://159.226.111.19:8080/calgarysensorserver/initializeSensor.do>
- 2) Stimulate service: start or stop stimulation, the parameter 'start' can be 'start' or 'stop'.  
<http://159.226.111.19:8080/calgarysensorserver/stimulate.do?start=start>
- 3) Query certain sensor information service: the parameter 'id' represent sensor's id  
<http://159.226.111.19:8080/calgarysensorserver/querySensorInfoById.do?id=sid0001>
- 4) Get all sensor information service: [getSensorInfo.do](#)  
<http://159.226.111.19:8080/calgarysensorserver/getSensorInfo.do>

The call order of these services is:1)->2)->3)

## 2. Strategies and Algorithms

### 2.1 Boundary Confine

I take boundary of Calgary into account when calculate sensor's position. In this case, I confine sensor's position to a polygon boundary (The red polygon in Figure 1), which is simplifier than the reality. When the sensor will move out of Calgary, it will automatically turn around and move again.



Figure 1 A Simplified Boundary of Calgary

## 2.2 Create Sensor

All sensors, which are random spatial distribution, are created in the polygon I depict in Figure

1. I create sensors from 'sid1' to 'sidn', the n represents the total number of sensors.

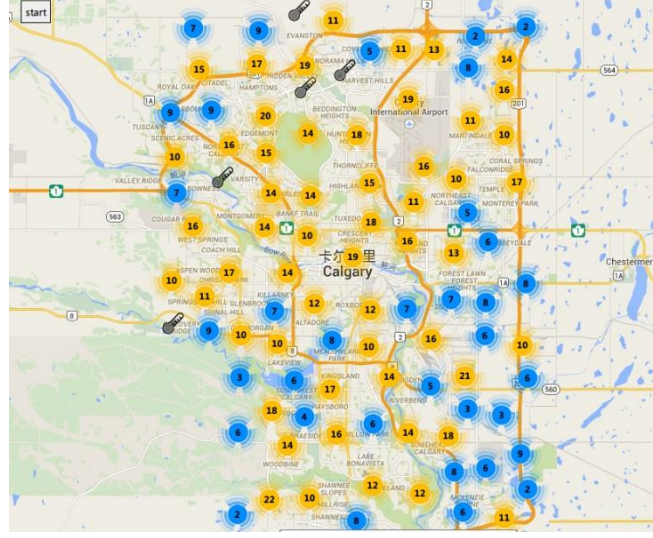


Figure 2 Create Sensors in Calgary

## 2.3 Stimulate Data

In the program, I stimulate sensor's direction, observed temperature and position.

- 1) Direction: I use 1 to 360 to represent sensor's moving direction.
- 2) Temperature: The formula I use to calculate temperature is as follows:

$$\text{Temp} = \text{min}_{\text{temp}} + (\text{max}_{\text{temp}} - \text{min}_{\text{temp}}) * \text{random};$$

- 3) Position:

Suppose that the Earth is a perfect sphere and its radius is 6371.393 kilometers.

We know the distance that separates each degree of latitude is approximate same, This distance(Latitude\_length) is  $2 * \pi * \text{EARTH\_RADIUS} / 360$ . However, the distance that separates each degree of longitude varies with the degree of latitude, this distance(Longitude\_length) is  $2 * \pi * \text{EARTH\_RADIUS} * \cos(\text{latitude}) / 360$ .

And we can calculate all sensors' moving distance( $\Delta s$ ) over a period of time. What's more, we know every sensor's moving direction( $\alpha$ ), so we can calculate every sensor's moving distance on meridian( $\Delta s_{\text{lng}}$ ) and parallel( $\Delta s_{\text{lat}}$ ). Then, we can calculate one sensor's the variation of degrees of latitude and longitude. The formula is

$$\Delta \text{lng} = \Delta s * \sin \alpha / \text{Longitude\_length}$$

$$\Delta \text{lat} = \Delta s * \cos \alpha / \text{Latitude\_length}$$

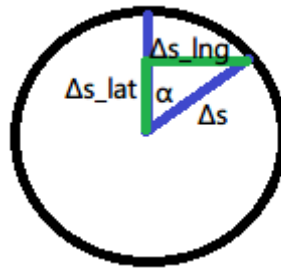


Figure 3 Variation Position Calculation

$$\begin{cases} newPosition.lat = position.lat + \Delta lat \\ newPosition.lng = position.lng + \Delta lng \end{cases}$$

When program calculates one sensor's new position, it will firstly determine whether the new position is inside the Calgary. If the new position is in the Calgary, the sensor will move to this new position. Otherwise, it will turn around and repeat the above steps.

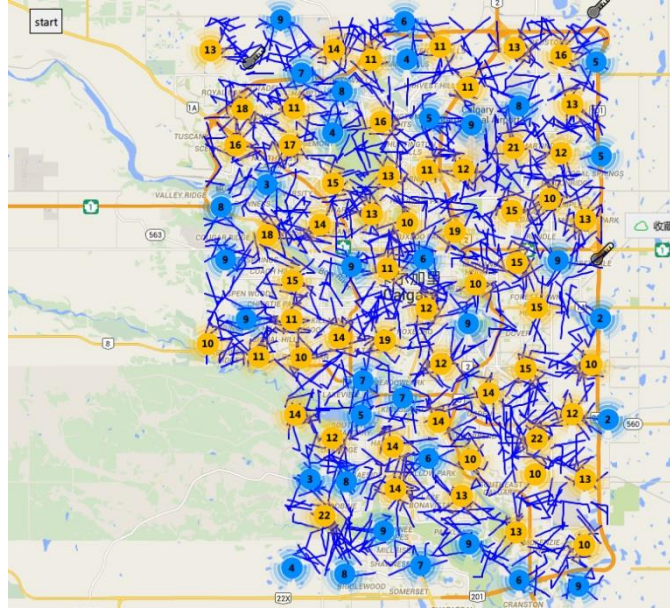


Figure 4 Stimulate sensors moving

## 2.4 Multi-threading

I write a very simple concurrency framework, which can set multiple threads to finish a task. In this project, the task is calculating all sensors' new position. Now the number of sensors is very few and the amount of calculation is very small. Therefore, it will not take full advantage of this concurrency framework. I think concurrency ability is an essential and important part of a program to deal with large data.

## 2.5 Data Push Technology

The client can request data actively or passively receive data from server. I list three ways of common methods of the client to get data in Table 1.

Table 1 Comparison of three methods of the client to get information

Method	Description	Advantages	Disadvantages
Ajax polling	The client sends requests to server and waits to receive data.	<ul style="list-style-type: none"> <li>● Easy</li> <li>● Don't need too many server-side configuration</li> </ul>	<ul style="list-style-type: none"> <li>● Pressure on the server does not have decreased significantly</li> <li>● Weak real-time performance</li> </ul>
Comet	Comet is a web application model in which a long-held HTTP request allows a web server to push data to a browser, without the	Short delay and High performance	<ul style="list-style-type: none"> <li>● Still a long polling</li> <li>● Long time connection, loss the characteristic of stateless and</li> </ul>

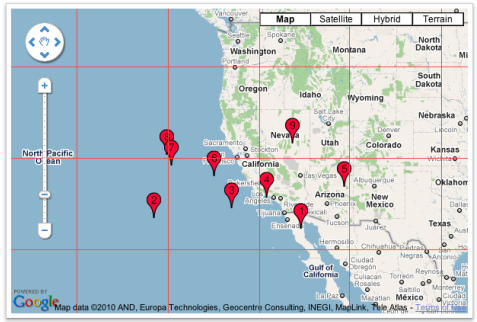
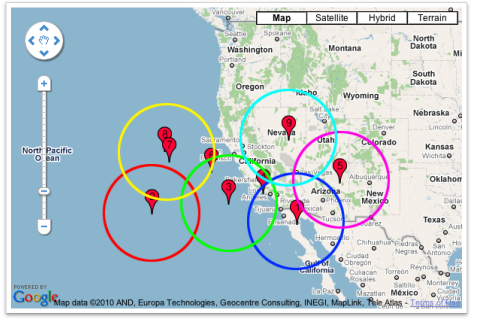
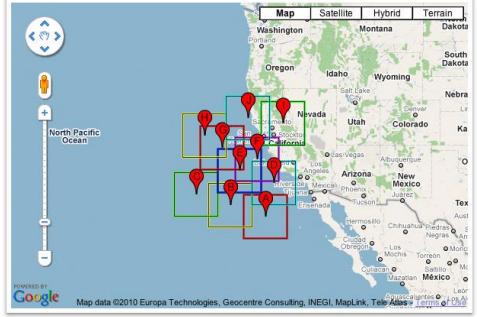
	browser explicitly requesting it.		high concurrency
Websocket	WebSocket is a protocol providing full-duplex communications channels over a single TCP connection	<ul style="list-style-type: none"> <li>● High performance</li> <li>● Future development tendency</li> </ul>	Weak compatibility

In this program, I use AJAX polling to request data from server.

## 2.6 Dynamically Grouping

I use markercluster in google map api to implement dynamically grouping. It's an elegant and efficient way to cluster sensors.

Table 2 Three algorithms of clustering points

Method	Advantages	Disadvantages	
Grid-based Clustering	<ul style="list-style-type: none"> <li>● Iterating through the markers once</li> <li>● no complicated distance calculation</li> </ul>	The ending result might not accurately describe the data	
Distance-based Clustering	Aggregation points precisely reflect the original points' location information.	Need to calculate the distance between point and point	
Grid and Distance – Based Clustering	High accuracy and efficiency	Slow than Grid-based Clustering	

Google map uses Grid and Distance Based Clustering Algorithm to implement point dynamically group. Many other company provides similar function, such as

**ESRI** <https://developers.arcgis.com/en/flex/sample-code/clustering.htm>

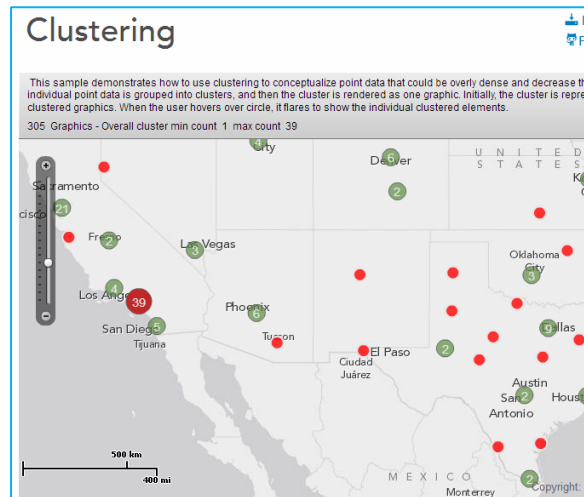


Figure 5 ESRI clustering demo

Openlayer <http://openlayers.org/dev/examples/strategy-cluster-threshold.html>

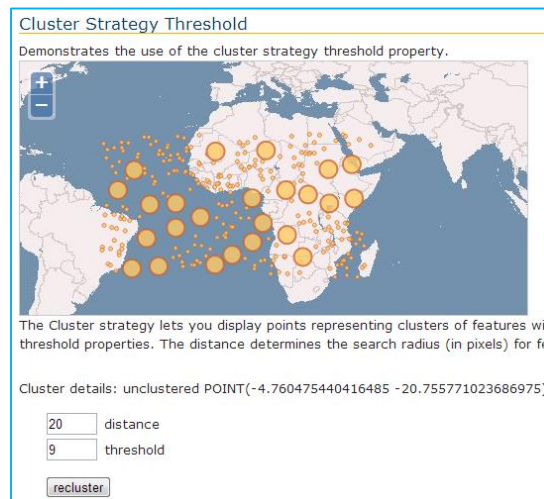


Figure 6 Openlayer clustering demo

## 2.7 Moving track Fade out

When the client adds or updates a sensor's position, it will create a new line between the new position and the old position of a sensor. Then the client will call a Javascript function named fadeout to reduce the line's opacity as the time goes on and finally this line will disappear. Other parts of the sensor's line will do the same thing. So this process will fade out all sensors' track gradually. This algorithm flow is as Figure 7 shows.

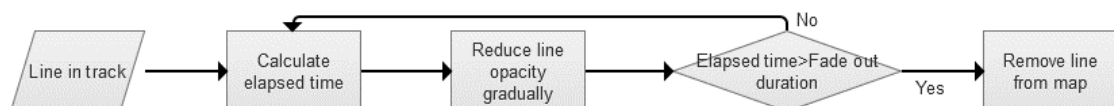


Figure 7 Track fade out algorithm flow



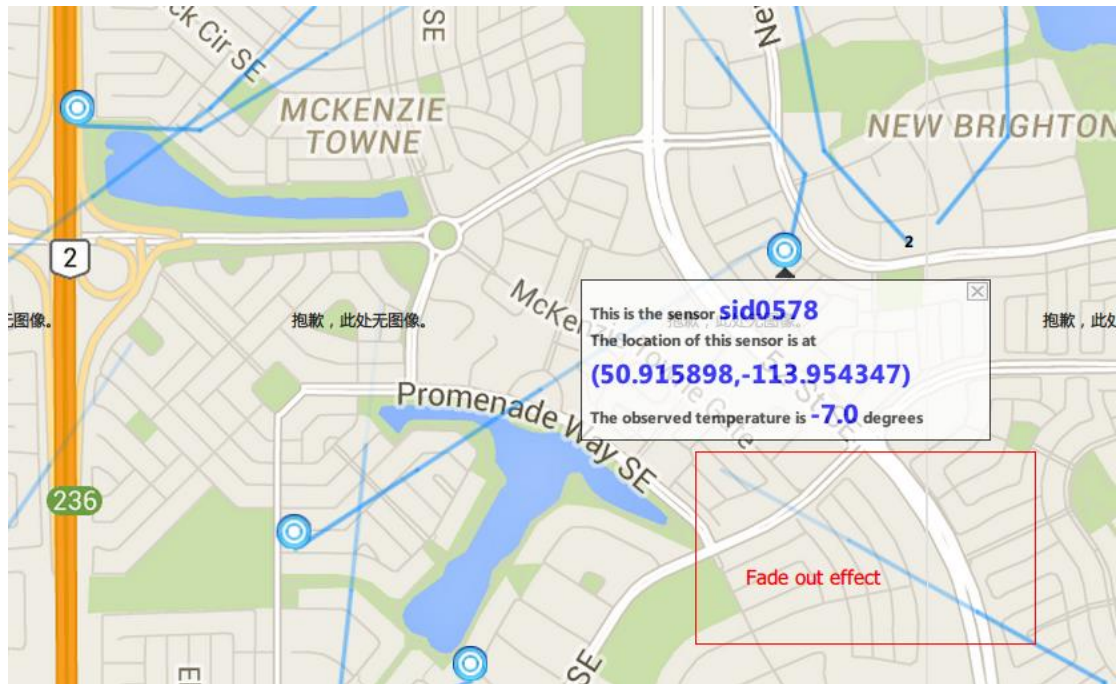


Figure 8 Fade out effect

### 3. Other information

#### 3.1 Configurable

In order to make program more scalable, I put many settings such as total number of sensors in spring properties file. So we can just change some arguments without recompiling code.

#### 3.2 Cross portable

There are many framework based on Foundation, which is an advanced responsive front-end framework. Foundation is professional and easy to use.

<http://foundation.zurb.com/>

#### 3.3 Cross-domain

When server VM002 access to RESTful service on server VM001, there exist cross-domain access problem. In order to solve the problem, I use Cross-Origin Resource Sharing (CORS) technic. I write a filter in spring which add Access-Control-Allow-Origin header in each response.

### 4. Comprehensive description regarding how to deploy your program to a VM

#### 4.1 Bindary deployment

- Copy calgarysensorserver.war to webapps folder on server VM001
- Copy calgarysensormap folder to webapps folder on server VM002

#### 4.2 Source code deployment

My development IDE is MyEclipse 2013 and Sublime Text.

The source code hierarchical structure of calgarysensorserver.war as the Figure 9 shows:

- CalgarySensorServer [CalgarySensor]
  - src
    - com.nunknown
      - controller
        - BaseController.java 2 14-12-25 下午5:47 lkcozy
        - SensorController.java 13 15-1-3 上午3:19 lkcozy
      - filter
        - CorsFilter.java 13 15-1-3 上午3:19 lkcozy
      - model
        - LatLng.java 12 15-1-2 下午6:55 lkcozy
        - Sensor.java 13 15-1-3 上午3:19 lkcozy
      - service
        - SensorService.java 13 15-1-3 上午3:19 lkcozy
      - thread
        - Executer.java 13 15-1-3 上午3:19 lkcozy
        - Job.java 13 15-1-3 上午3:19 lkcozy
        - MyLock.java 13 15-1-3 上午3:19 lkcozy
        - StimulateJob.java 13 15-1-3 上午3:19 lkcozy
      - util
        - ClassLoaderUtil.java 13 15-1-3 上午3:19 lkcozy
        - ReadPropertiesUtil.java 13 15-1-3 上午3:19 lkcozy
        - SensorConst.java 13 15-1-3 上午3:19 lkcozy
        - Util.java 13 15-1-3 上午3:19 lkcozy
    - commons-logging.properties 11 14-12-31 上午2:06 lkcozy
    - log4j.properties 13 15-1-3 上午3:19 lkcozy
    - module.properties 13 15-1-3 上午3:19 lkcozy

Figure 9 Source code hierarchical of CalgarySensorServer