Smilart Phoenix. The Integration Guide

Table of Contents

Glossary

Person in Phoenix
: is a named set of photos that allows you to identify it.

A client
: is an application that directly sends requests to Phoenix.

A person
: is a collection of data about a person from a client database and in Phoenix.

Phoenix. Managing persons

When creating a person in Phoenix, the client can save the received photos in its database or do not save them. Let's analyze the features of each approach.

The client stores the received photos in its database

Props:

- Easy backup and restore. It is enough to make a backup copy of only the client database. In this case, the recovery of the state of the Phoenix database is performed in the following steps:
    1. Clear the Phoenix database (by sending `KeepPerson` request with an empty list)
    2. Run the required number of `AddPerson` requests to fill the Phoenix database.
- No need to get photos from Phoenix while working. It is made possible to take photos of persons from the client's database without performing `GetPerson` request.
- Possibility to realize the centralized filling of independent copies of Phoenix by persons from central client's database.

Cons:

- It is need to store photos of persons in the client database.

Storage of photos only in Phoenix

Props:

- There is no need for the client to adapt the database for storing photos.

Cons:

- Need to access Phoenix every time you want to display a photo of a person.
- Person data is divided into 2 databases (client database and Phoenix), which imposes certain requirements for backup and ensuring the consistency of data.
- When working with several independent Phoenix, you need the availability of all Phoenix to

make changes.

Consistent re-creation of a person

When creating a person in the Phoenix database, it is possible to set its identifier in the Phoenix database and the IDs for the photos to be added. This allows you to consistently recreate a person without changing its identifier and the identifiers of its photos. You do not need to save photos obtained as a result of the `AddPerson` request.

Synchronization of Phoenix database and client database

If you need to synchronize the state of Phoenix database and the client database, you should send `KeepPerson` request with the list of persons that must be present in Phoenix. When this query is executed, the specified persons will be left in Phoenix database, and all the others will be removed. In addition, information about missing persons in Phoenix will be returned. Synchronization via `ListPersons` and `RemovePersons` requests is also possible.

Non-synchronized work of several clients with Person service in Phoenix

Non-synchronized work of several clients is not recommended.

`KeepPersons`, `UpdatePerson` requests should be avoided, as it can result in deletion of persons and photos created by another client.

Receiving data from Phoenix via HTTP

Using services which can send URLs to resources which can be accessed via HTTP GET requests, it is need to make sure that the clients can resolve the DNS names where the Phoenix instances are located.

Miscellaneous

- To avoid identification problems, it is not recommended to create two different Phoenix persons containing photos of the same person.
- It is not recommended to use information related to personal data as a Phoenix person's identifier.
- By default, each instance of Phoenix has its own database and does not share any resources with other instances of Phoenix.

RPC requests and subscriptions to AMQP

- It is necessary to learn how RPC requests and subscriptions work in the programming language used by the client.
- First of all you must create a response queue and bind it to direct exchange via `routing key = <the name of the queue>` before sending a request with the name of this queue in `reply-to` field.
- It is recommended to use the sending of messages with receipt of confirmation from the broker about receipt.
- WAGNING: If the library you are using supports auto-recovery when the connection with the broker is down, you should make sure that the client continues to work correctly when the connection is lost and restored. For example, after restoring a connection, it is possible for the broker to recreate the queues with different names, if they were created by the broker, and not by the client.

Sending requests to Phoenix

- It is recommended to limit the waiting time for the client to respond to Phoenix by 5 seconds (this is a very high time limit) for all requests, **except** for those that work with the entire database as a whole: `KeepPersons`, `RemovePersons`, `ListPersons`. For these requests **timeout is highly dependent on the specific equipment** and to determine it, **it is recommended to perform stress tests on the specific equipment to be used**.
- It is recommended to implement the Circular Breaker pattern when sending requests to Phoenix.

Phoenix cluster

Phoenix cluster is implemented by connecting MongoDB in a single cluster. Phoenix then uses this database cluster as a shared storage.

In master mode Phoenix connected to a master MongoDB node should be used to manipulate persons in database. This Phoenix should be configured to the master mode. All other Phoenixes in cluster should be configured to the slave mode.

For more information see `Phoenix System User Guide`.

Usage of the Instant Photo Analytics (IPA) Service

IPA service was designed as an add-on to the main recognition workflow which works with frames from connected cameras. Due to high priority of IPA requests a high rate of IPA requests can reduce the number of processed frames from cameras, thus quality of recognition capabilities of the whole system can be reduced. High constant load of large messages can also negatively affect the work of the RabbitMQ.

Therefore, IPA service **SHOULD NOT** be used to process continues streams of images (e.g. frames from external sources) and it is recommended to wait for a response to the request before sending a new one.

Receiving video stream from cameras

It maybe very usefull feature for users to view a live stream from cameras. It may be used both for initial installation and for periodic manual control what camera sees if it was not installed in a fixed position or if the enviroment has changed.

This feature may be implemented via subscription **only for frames** in VCA service which does not affect the load on recognition services.

> Total number of processed frames (no matter will any recognition services be used for their processing or not) may be limited by your license.

Implementation of a live desktop video player usually consists of 3 steps:

1. Receiving events about frames.
2. Downloading content of frames. Downloaded images will have original resolution of frames comes from a camera and may be a quite large.
3. Rendering.

Implementing of a long-running streaming in soft real-time mode may be quite complicated task. Following recommendations may be usefull for this:

- Seperate different kind of jobs into independent workers with bounded queues between them: receiving events from transport (one worker), frames content download (multiple parallel workers), rendering (one worker). This will help to cope with the situation when slowing down or the occurrence of errors in downloading files will not block rendering. It will also help to protect your application from a memory leak or appearance of delays in result video stream observed by a user.
- Limit maximum "lifetime of frame event" for 200 ms. This time should be enough to receive frame content and it is quite enough for good UX.
- Implement some kind of a throttling algorithm in any step to decrease incoming FPS.
- Render should skip frames events in queue (regardless whether content was downloaded or not) if it is too old and should render the youngest frame event that has dowloaded content (or the last element in the queue if queue becomes empty).
- Limit a live session to a short period of time (about a couple of minutes).
- Render frames in a low resolution.