Запрос статистик в СПИК

Для запроса статистик в СПИК существует сервис **SpicStatisticsControllerService**.

Данный сервис служит для открытия сессии построения статистик, ее закрытия, получения информации о текущей порции статистики и вызова построения следующей порции. Устанавливать настройки построения статистик и забирать уже построенные порции конкретных статистик необходимо через специальные сервисы. Для каждого типа статистики существует отдельный сервис, позволяющий заказать ее построение и впоследствии забрать построенную порцию. Каждая порция содержит статистику за 1 день. Подробное описание контрактов статистик приведено здесь.

Алгоритм запроса статистик выглядит следующим образом.

1. Пройти авторизацию в системе СКАУТ (если не пройдена).
2. Сформировать запрос на открытие сессии построения статистики **SpicStatisticsSessionRequest**.
3. Вызвать метод **StartStatisticsSession** заглушки **SpicStatisticsControllerService** с параметром типа **SpicStatisticsSessionRequest**; метод возвращает объект типа **SpicStatisticsSessionResponse**. На данном этапе в сессии нет ни одного запроса на построение статистики. Есть только открытая сессия.
4. Добавление одного или более запросов на построение статистики к сессии. Для добавления запроса на построение статистики необходимо вызвать метод **AddStatisticsRequest** в сервисе по построению конкретного вида статистик (например **SpicTrackPeriodsStatisticsService**), передав туда параметры построения статистики, необходимые для ее построения. При успешном выполнении метода к сессии построения статистик добавится информация о том, что будет необходимо строить данный тип статистик. В СПИК существует ограничение, заключающееся в том, что в одной сессии не может быть несколько запросов на построение одного и того же типа конкретной статистики. После того, как все запросы на построение статистик добавлены, необходимо запустить построение этих статистик, вызвав метод **StartBuild** сервиса **SpicStatisticsControllerService**, принимающий параметр типа **SpicStatisticsSession**.
5. Ожидание построения и забор статистик. После запуска построения статистики можно начинать получение построенных результатов порциями. Построение статистики на сервере обычно занимает некоторое время, поэтому, если статистика еще не построена, при вызове метода получения статистики сервер вернет ответ без построенной статистики,

указав при этом статус операции «Processing». Если же статистика построена, то вызов данного метода вернет описание статистики со статусом «Ok» и построенную порцию статистики. Операцию забора порций статистик следует произвести для всех статистик, которые были заказаны в 4 пункте.

6. Запрос следующей порции построения статистики. Так как статистики строятся не сразу за весь период запроса, а постепенно(по дням), то после забора статистики следует дать серверу команду приступить к построению новой порции, если текущая порция не являлась последней (это указывается в **SpicStatisticsChunkInfo**). Для построения следующей порции необходимо вызвать метод **BuildNextChunk** сервиса **SpicStatisticsControllerService**, передав в параметре метода объект типа **SpicStatisticsSession**. Для получения следующей порции статистики необходимо перейти на 5 пункт.
7. Завершение построения статистики. Для завершения построения статистик, необходимо закрыть сессию, вызвав метод **CancelStatisticsSession** сервиса **SpicStatisticsControllerService**, принимающий в качестве параметра **SpicStatisticsSession**.

На **рисунке 1** приведена блок схема данного алгоритма, иллюстрирующая вышесказанное.

Пример работы со статистикой SpicTrackPeriodsMileageStatistics в C#:

```
public SpicTrackPeriodsMileageStatistics[] GetMileageStatistics()
  {
      // ??????? ???????? ????????
      var specificStatisticsClient = new
SpicSoapTrackPeriodsMileageStatisticsServiceClient();
      var statisticsClient = new SpicSoapStatisticsControllerServiceClient();

    // ??????? ?????? ?????? ?????????
      var statisticsSessionRequest = new SpicStatisticsSessionRequest
          {
              Period = new SpicDateTimeRange
                  {
                      Begin = DateTime.Now.AddDays(-10),
                      End = DateTime.Now
                  },
              TargetObject = new SpicObjectIdentity
                  {
                      ObjectId = 246
 }
          };

    // ?????????? ?????? ? ???????? ??????
      var statisticsSession =
statisticsClient.StartStatisticsSession(statisticsSessionRequest).Session;
      // ?? ????? ????, ??? ???? ? ??? ?? ????????, ?? ??? ????? ??????????
      var specificStatisticsSession = new
SpicTrackPeriodsMileageStat.SpicStatisticsSession
      {
          StatisticsSessionId = statisticsSession.StatisticsSessionId,
      };
      // ????????? ?????? ?? ?????????? ??????????
      specificStatisticsClient.AddStatisticsRequest(specificStatisticsSession);
      // ????????? ??????????
      statisticsClient.StartBuild(statisticsSession);

    var statisticsList = new List<SpicTrackPeriodsMileageStatistics>();
      SpicTrackPeriodsMileageStatisticsResult statisticsResponse;

    // ?????????, ???? ?? ??????? ????????? ?????? ?????????
      do
      {
          // ????, ???? ?????? ????????? ??????????
          do
          {
              statisticsResponse =
specificStatisticsClient.GetStatistics(specificStatisticsSession);
          } while (statisticsResponse.ChunkInfo.Status.Value == "Processing");
          statisticsList.Add(statisticsResponse.Statistics);

        // ?????????? ????????? ?????? ??????????
          statisticsClient.BuildNextChunk(statisticsSession);
      } while (!statisticsResponse.ChunkInfo.IsFinalChunk);

    // ????????? ?????? ?????????? ?????????
      statisticsClient.StopStatisticsSession(statisticsSession);
```

```
    return statisticsList.ToArray();
  }
```

Пример работы со статистикой SpicTrackPeriodsMileageStatistics в JS:

```
 var baseUrl = 'http://localhost:8081/spic/';
  var baseTrackPeriodsMileageUrl = baseUrl + 'trackPeriodsMileage/rest/';
  var baseStatisticsUrl = baseUrl + 'StatisticsController/rest/';

var startStatisticsSessionUrl = baseStatisticsUrl + 'StartStatisticsSession';
  var stopStatisticsSessionUrl = baseStatisticsUrl + 'CancelStatisticsSession';
  var getCurrentChunkInfoUrl = baseStatisticsUrl + 'GetCurrentChunkInfo';
  var buildNextChunkUrl = baseStatisticsUrl + 'BuildNextChunk';
  var startBuildUrl = baseStatisticsUrl + 'StartBuild';

var addStatisticsTrackPeriodMileageRequestUrl = baseTrackPeriodUrl +
'AddStatisticsRequest';
  var getStatisticsTrackPeriodMileageUrl = baseTrackPeriodUrl + 'GetStatistics';

var provider = new HttpJsonRequestProvider();

// ??????? ?????? ?????? ?????????
  var now = new Date();
  var statisticsSessionRequest =
  {
      Period:
      {
          Begin: new Date(now.getTime() - 24 * 3600 * 1000),
          End: now
      },
      TargetObject:
      {
          ObjectId: 246

}
  };

// ?????????? ?????? ? ???????? ??????
  var statisticsSession = startStatisticsSession(statisticsSessionRequest);
  // ????????? ?????? ?? ????????? ?????????
  addStatisticsTrackPeriodMileageRequest(statisticsSession);
  // ????????? ?????????
  startBuild(statisticsSession);

var statisticsArray = [];
  var statisticsResponse = null;

do {
      // ????, ???? ?????? ????????? ?????????
      do {
          statisticsResponse =
getStatisticsTrackPeriodMileage(statisticsSession);
      } while (statisticsResponse.ChunkInfo.Status.Value == "Processing");
      statisticsArray.Add(statisticsResponse.Statistics);

    // ????????? ????????? ?????? ?????????
      buildNextChunk(statisticsSession);
```

```
  } while (!statisticsResponse.ChunkInfo.IsFinalChunk);

// ????????? ?????? ?????????? ????????
  stopStatisticsSession(statisticsSession);

  function startStatisticsSession(sessionRequest) {
      return provider.POSTAuthorized(startStatisticsSessionUrl,
          sessionRequest,
          getAuthorizationToken()).response;
  }

function stopStatisticsSession(session) {
      return provider.POSTAuthorized(stopStatisticsSessionUrl,
          session,
          getAuthorizationToken()).response;
  }

function buildNextChunk(session) {
      return provider.POSTAuthorized(buildNextChunkUrl,
          session,
          getAuthorizationToken()).response;
  }

function startBuild(session) {
      return provider.POSTAuthorized(startBuildUrl,
          session,
          getAuthorizationToken()).response;
  }

function addStatisticsTrackPeriodMileageRequest(session) {
      return provider.POSTAuthorized(addStatisticsTrackPeriodMileageRequestUrl,
          session,
          getAuthorizationToken()).response;
  }

function getStatisticsTrackPeriodMileage(session) {
      return provider.POSTAuthorized(getStatisticsTrackPeriodMileageUrl,
          session,
          getAuthorizationToken()).response;
  }
```

<<Назад К оглавлению

Далее>>