راه حل های پیشنهادی مدیریت تراکنشها در فراخوانی بین مایکروسرویس ها

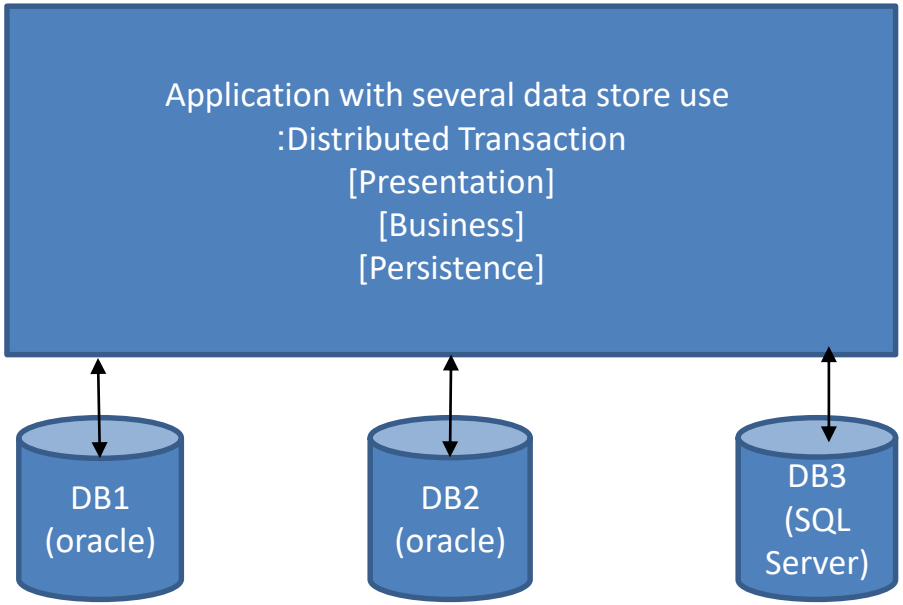
تیم CTO / مریم آزیش
۱۳۹۹/۰۴/۱۵ – ۱۳۹۹/۰۴/۲۹

تراکنشهای توزیع شده (Distributed Transaction) :

قبل از بررسی مدیریت تراکنشها در بین مایکروسرویسها موضوع تراکنشهای توزیع شده را بررسی میکنیم. این راه حل زمانی استفاده میشود که یک اپلیکیشن (یک پروژه نرم افزاری) دارای چند Data Store (دیتابیس یا ...) باشد و بخواهد با یک تراکنش توزیع شده یک بیزینس را انجام دهد. به عنوان مثال یک متد ثبت کاربر داریم که در این متد میخواهیم در یک تراکنش اطلاعات هویتی کاربر در یک دیتابیس و اطلاعات مالی او در دیتابیس دیگری با رعایت ACID ثبت شود:

- Atomicity : یا تمام اطلاعات به درستی ثبت گردد و یا تمام آن rollback شود
- Consistency : داده‌ها مطابق با قوانین تعریف شده پایگاه‌داده باشند.
- Isolation : هر تراکنش به صورتی اجرا شود که گویی مستقل از دیگری است.
- Durability : براساس این خاصیت تراکنش‌هایی که به مرحله انجام (Commit) برسند اثرشان ماندنی است و هرگز به طور تصادفی از بین نمی‌رود.

روشهای معروف پیاده سازی این تکنیک SAGA , 2PC , XA میباشد.



تراکنشها در یک پروژه نرم افزاری به دنیا می آیند و به سمت دیتابیس commit یا rollback میشوند. این تکنیک تراکنشهای توزیع شده (Distributed Transaction) به مایکروسرویسهایی که چند پروژه هستند کمکی نمیتواند بکند. زیرا موضوع آن مدیریت تراکنش در یک پروژه و چند دیتابیس است.

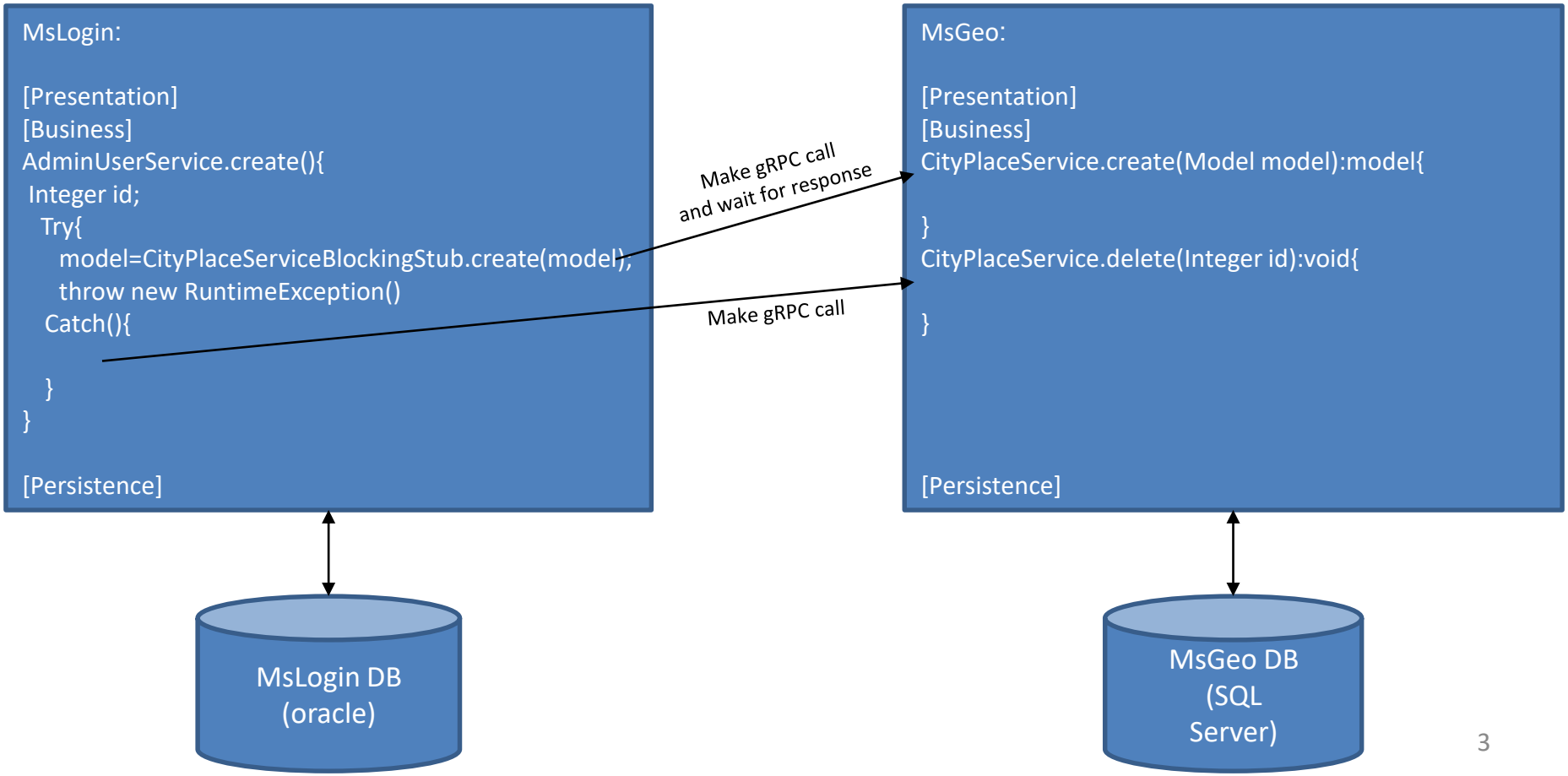
https://www.infoworld.com/article/2077963/distributed-transactions-in-spring--with-and-without-xa.html

مایکروسرویسها:

وضعیتی که ما میخواهیم بررسی کنیم چند مایکروسرویس (پروژه نرم افزاری) است که هرکدام دیتابیس خود را دارند. بنابراین تراکنشها در خود آنها ایجاد و commit و rollback میشوند. به لطف استفاده از تکنیک gRPC به همراه proto3 میتوانیم ارتباط sync و متقارن داشته باشیم.

- ارتباط sync : مایکروسرویس ها میتوانند همدیگر را فراخوانی کنند و منتظر جواب فراخونی خود بمانند و حتی به کمک proto3 میتوانند Exception را نیز از مایکروسرویس مقصد دریافت نمایند
- ارتباط متقارن : مایکروسرویس ها فراخوانی دوطرفه میتوانند داشته باشند به این معنی که یک مایکروسرویس فقط نقش فراخوانی کننده ندارد و میتواند هم مایکروسرویس رافراخوانی کند و هم مایکروسرویس دیگر آن را فراخوانی کند

ما میخواهیم در این پرزنت مستقل بودن مایکروسرویسها را حفظ کنیم و تراکنش یا پروژه واسطی بین آنها در نظر نگیریم و راه حل های پیشنهادی را ارائه کنیم



مشکلات و راه حل های پیشنهادی

- **۱- تغییرات در طول انجام یک متد در مایکروسرویس های دیگر**
  - ۱-۱ هنگام ثبت یا ویرایش روی آیدی با فراخوانی مجدد stub چک انجام شود
  - ۱-۲ اضافه کردن کش با کلید نام و آی دی انتیتی و مقداری از جنس کانتر(int) جهت لاک شدن حذف و غیرفعال سطری که از مایکروسرویس دیگر select میشود. در صورت select شدن یک عدد به کانتر کش آن انتیتی اضافه میشود و با اتمام کار یک عدد از کانتر کش آن انتیتی کم میشود. و حذف و غیرفعال در صورت ۰ بودن مقدار کانتر در کش انجام میشود.
- **۲- بررسی وابستگی یک انتیتی در مایکروسرویسهای دیگر:**
  - ۲-۱ بررسی وابستگی از طریق لیست های تشکیلی در هنگام فراخوانی متد
  - ۲-۲ فراخوانی متد های بررسی وابستگی در مایکروسرویس های دیگر
  - ۲-۳ یک انتیتی ارتباطات بیرونی به همه مایکروسرویس ها اضافه شود که تمام ارتباطات بیرونی انیتیهای بیرونی در آن ثبت شود.
  - ۲-۴ نگه داشتن شناسه انتیتی های وابسته مایکروسرویسهای دیگر در انتیتی فعلی

- **۳- حذف زباله انتیتی هایی که در مایکروسرویس های دیگر ثبت شده اند(CityPlace):**

۳.۱ تمامی عملیات select و بررسی بیزینس و خطاهای بیزینسی در ابتدای متد و قبل از فراخونی stubهای ثبت و ویرایش و حذف مایکروسرویسهای دیگر انجام شوند.

۳.۲ در صورتی که ثبت یا حذف یا ویرایش برای stubای به صورت لیست باشد (چندتایی) باید لیست را در قالب لیستی از مدل به stub موردنظر ارسال کنیم. این کار دو مزیت دارد :

- یکبار هر مایکروسرویس را فراخوانی میکنیم.
- اگر خطایی در مایکروسرویس رخ دهد کل عملیات آن مایکروسرویس رول بک میشود.

۳.۳ استفاده از روش stateMachine برای حذف زباله ها (ضمیمه ۱)

- **۴- حذف انتیتی های مایکروسرویس های دیگر (CityPlace)و دریافت خطا در انتیتی مایکروسرویس اصلی (AdminUser):**

۴.۱ فرآیندها طوری باشند که عملیات حذف در سرویس StateMachine با مشکلات وابستگی های بیزینسی روبرو نشود.(و تنها به دلیل مسایل فنی مثل بن بست ما از اسکجل داخل StateMachine استفاده کنیم).

۴.۲ استفاده از روش StateMachine برای حذف زباله ها، با استفاده از قابلیت اسکجل(در بعضی مواقع به دلیل مشکلات فنی مثل پایین آمدن دیتابیس یا بن بست امکان حذف بعد از مدت زمانی فراهم میشود) (ضمیمه ۲)

- **۵-ویرایش انتیتی های مایکروسرویسهای دیگر (CityPlace) و دریافت خطا در انتیتی مایکروسرویس اصلی (AdminUser):**

۵.۱ استفاده از روش StateMachine برای ویرایش اطلاعات ونگه داری اطلاعات در کش ردیس (ضمیمه ۳)

۵.۲ استفاده از روش StateMachine برای ویرایش اطلاعات و نگه داری اطلاعات در فیلد جیسون در خود انتیتی

ضمیمه ۱ :
استفاده از روش StateMachine برای رفع مشکل ثبت

۱- کنترلر فراخوانی کننده سرویس اصلی شامل try و catch باشد.

۲ – تولید referenceCode در کنترلر و ارسال به سرویس.

۳- ارسال مدل یا لیستی از مدل به stub موردنظر جهت ثبت

ضمیمه ۲ :
استفاده از روش StateMachine برای رفع مشکل حذف

۱- کنترلر فراخوانی کننده سرویس اصلی شامل try و catch باشد.

۲ – تولید referenceCode در کنترلر و ارسال به سرویس.

۳- در ابتدای متد حذف در سرویس موردنظر(adminUserDelete) ابتدا نام stubهایی که باید جهت حذف فراخوانی شوند و همچنین نام متد حذف خود سرویس را به ترتیب درانتیتی وضعیت ثبت می نماییم. هرکدام از مایکروسرویس ها که با موفقیت حذف شوند باید فیلد done را true کنیم.

مزیت فیلد done این است که بدانیم حذف کدام stub ها موفقیت آمیز بوده و اگر مقدار done خالی باشد یعنی حذف آنها انجام نشده است و باید در catch کنترلر موردنظر حذف را تکمیل نماییم.

4- ارسال مدل یا لیستی از مدل به stub موردنظر جهت حذف

* برای حذف هم میتوانیم مانند ویرایش از کش استفاده کنیم و در صورت بروز خطا سطرهایی را که حذف شده اند را اطلاعاتش را از کش بخوانیم و مجدد ثبت کنیم.

ضمیمه ۳ :
استفاده از روش StateMachine برای رفع مشکل ویرایش

۱- کنترلر فراخوانی کننده سرویس اصلی شامل try و catch باشد.

۲ – تولید referenceCode در کنترلر و ارسال به سرویس.

۳- در ابتدای متد ویرایش در سرویس موردنظر(adminUserUpdate) ابتدا نام stubهایی که باید جهت ویرایش فراخوانی شوند و همچنین نام متد ویرایش خود سرویس را به ترتیب در انتیتی وضعیت ثبت می نماییم. هرکدام از مایکروسرویس ها که با موفقیت ویرایش شوند باید فیلد done را true کنیم.

مزیت فیلد done این است که بدانیم حذف کدام stub ها موفقیت آمیز بوده و اگر مقدار done خالی باشد یعنی حذف آنها انجام نشده است و باید در catch کنترلر موردنظر حذف را تکمیل نماییم.

۴- یک کش منیجر مشترک برای تمامی مایکروسرویس ها فقط جهت عملیات rollback ایجاد میکنیم.

۵- ارسال مدل یا لیستی از مدل به stub موردنظر جهت ویرایش

۶- قبل از ویرایش هر آیدی در هر STUB باید اطلاعاتش را درکش ذخیره کنیم. کلید کش میتواند ترکیبی از

referenceCodeو serviceName و methodTypeEnum و grpcNameEnum و entityNameEnum و entityId

باشد تا بتوانیم در صورت بروز خطا اطلاعات را از کش بخوانیم و سطرهایی که فیلد done آنها true شده است را به حالت قبل برگردانیم.

۷- در صورتی هیچ یک از stub ها خطایی نداد، میتوانیم اطلاعات را از کش پاک نماییم.

نکات ضروری برای مایکروسرویس

۱ –تمامی عملیات select و بررسی بیزینس وخطاها ابتدای متد و قبل از فراخونی stubهای ثبت و ویرایش و حذف انجام شوند.

۲- فراخوانی stubهای ثبت و ویرایش و حذف درانتهای متد انجام شود.

۳- در صورتی که خطایی در stubهای درون سرویس اصلی رخ دهد باعث میشود که خود متد فراخوانی کننده stub رول بک شود.

۴- عملیات ثبت و ویرایش و حذف انتیتی وضعیت (State) و جزییات وضعیت (StateDetail) باید در ترنزکشن جدید انجام شود. مزیت این کار این است که اگر خطایی در سرویس اصلی یا stubهای درون سرویس اصلی رخ دهد بدانیم تا کجای کار را انجام داده ایم.

۵-کنترلر فراخوانی کننده سرویس اصلی شامل try و catch باشد.در catch کنترلر باید عملیات مربوط به Create , Delete Update, موردنظر را تکمیل کنیم.

۶- در catch کنترلر خودش هم شامل try و catch باشد که در catch دوم در صورتی که مجدد عملیات به خطا خورد یک پیام برای کارمند ارسال کنیم.

```
try {
   //referenceCode = stateSevice.generateUniqReferenceCode();
   //model.setReferenceCode(referenceCode);
   //call Service(model);
}catch (Exception ex){
   try {
      //stateSevice.rollback();
   }catch (Exception ex1){
      //notification referenceCode to employee
   }
   throw ex;
}
```

۷ – درمتد کنترلر، قبل ازفراخوانی متد سرویس (Create و Update و Delete) باید یک referenceCode یونیک در کنترلر تولید نماییم و در مدل ارسالی به سرویس ارسال کنیم.

کد referenceCode جهت این تولید میشود که بتوانیم با referenceCode سرچ کنیم برای اینکه بدانیم چه آیدی هایی از چه stub هایی را ایجاد کرده ایم تا در صورتی که ثبت stub با خطا مواجه میشود در catch کنترلر بتوانیم سطرهایی که ثبت کرده ایم را حذف نماییم.
* referenceCode باید در کل مایکروسرویس کاملا یونیک باشد.

۸- در صورتی که ثبت یا حذف یا ویرایش برای stubای به صورت لیست باشد (چندتایی) باید لیست را در قالب لیستی از مدل به stub موردنظر ارسال کنیم. این کار دو مزیت دارد :
- یکبار هر مایکروسرویس را فراخوانی میکنیم.
- اگر خطایی در مایکروسرویس رخ دهد کل عملیات آن مایکروسرویس رول بک میشود.

۹- مدل برگشتی stub شامل آیدی هایی باشد که عملیات روی آن صورت گرفته است

مثال متد ثبت AdminUserService.create

```
public AdminUserModel create(@NotNull AdminUserModel adminUserModel) throws Exception {

   //بررسی شناسه شهر
    ReadByIdRequestModel readByIdRequestModel =  ReadByIdRequestModel.newBuilder().setId(adminUserModel.getDefaultAdminUserContact_city_id()).build();
   final ReadByIdResponseModel cityResponse= this.cityStub.grpcReadById(readByIdRequestModel);

   //ثبت اطلاعات تماس ادمین
    AdminUserContact adminUserContact = new AdminUserContact();
   adminUserContact.setCityId(cityResponse.getId());
   adminUserContact.setAddress(adminUserModel.getDefaultAdminUserContact_address());
   adminUserContact = adminUserContactRepository.save(adminUserContact);

   //ثبت ادمین
    AdminUser adminUser = new AdminUser();
   adminUser.setFirstName(adminUserModel.getFirstName());
   adminUser.setLastName(adminUserModel.getLastName());
   adminUser.setPassword(PasswordEncoderGenerator.generate(adminUserModel.getPassword()));
   adminUser.setUsername(adminUserModel.getUsername());

   adminUser = adminUserRepository.save(adminUser);
   adminUser.setDefaultAdminUserContact(adminUserContact);

   //اضافه کردن لوکیشن شهری برای ادمین
    CreateRequestModel model = CreateRequestModel.newBuilder()
    .setAdminUserId(adminUser.getId()).setCityId(cityResponse.getId()).setLatitude("35.791354").setLongitude("51.356406").setTitle("OfficePlace").build();
   final CreateResponseModel createResponseModel= this.cityPlaceStub.grpcCreate(createRequestModel);
   //ثبت یک استیت با کد رفرنس در اولین بار
   stateService.create(adminUserModel.referenceCode, “adminUserServiceImpl”,”create”);
   //در صورت ثبت موفقیت آمیز هر مرحله ، یک شرح استیت جدید ایجاد میکنیم که در صورت بروز خطا بعدا آن را حذف نماییم
   stateDetailService.create(adminUserModel.getReferenceCode(),”msgeo”,”cityPlace”, createResponseModel.getId());

   //اضافه کردن .... برای ادمین
   final CreateResponseModel createResponseModel1= this.xxxxeStub.grpcCreate(createRequestModel);
   //در صورت ثبت موفقیت آمیز هر مرحله ، یک شرح استیت جدید ایجاد میکنیم که در صورت بروز خطا بعدا آن را حذف نماییم
    stateDetailService.create(adminUserModel.getReferenceCode(),” msgeo”,”xxxx”, createResponseModel1.getId());

   adminUserModel.setId(adminUser.getId());
   return adminUserModel;
}
```

مثال متد حذف AdminUserService.delete

```
public AdminUserModel delete(@NotNull Integer id) throws Exception {

   //ثبت تمام فعالیتهایی که باید انجام شود شامل یک استیت و چند شرح استیت که همگی با کد رفرنس قابل دسترسی و آپدیت هستند. شرحهای استیت باید به ترتیب درست ثبت شوند
   //بعد از انجام هرکار وضعیت شرح استیت آن کار را ترو می کنیم و بعد از تکمیل کل متد بدون خطا کل سطرهای استیت و شرحهای آن حذف میشوند
   stateService.create(adminUserModel.referenceCode, “adminUserServiceImpl”,” delete”);
   stateDetailService.create(adminUserModel.referenceCode,” msgeo”,” cityPlace”);
   stateDetailService.create(adminUserModel.referenceCode,” msgeo”,” xxxx”);
   stateDetailService.create(adminUserModel.referenceCode,” msgeo”,” adminUser”);

   AdminUserModel adminUserModel = this.readById(id);
   AdminUser adminUser = adminUserRepository.findById(adminUserModel.getId()).get();

   //حذف لوکیشن شهری ادمین
   final DeleteResponseModel deleteResponseModel= this.cityPlaceStub.grpcDelete(deleteRequestModel);
   //در صورت حذف موفقیت آمیز هر مرحله، وضعیت شرح استیت آن را ترو میکنیم
   stateDetailService.update(adminUserModel.getReferenceCode(), “cityPlace” , deleteResponseModel.getId() , true);

   //حذف ..... ادمین
   final DeleteResponseModel deleteResponseModel1= this.xxxxStub.grpcDelete(deleteRequestModel);
   //در صورت حذف موفقیت آمیز هر مرحله، وضعیت شرح استیت آن را ترو میکنیم
   stateDetailService.update(adminUserModel.getReferenceCode(), “xxxx” , deleteResponseModel1.getId() , true);

   //حذف ادمین
   adminUserContactRepository.deleteById(adminUser.getDefaultAdminUserContact().getId());
   adminUserRepository.delete(adminUser);
   //در صورت حذف موفقیت آمیز هر مرحله ، وضعیت شرح استیت آن را ترو میکنیم
   stateDetailService.update(adminUserModel.getReferenceCode(), “adminUser” , adminUser.getId() , true);

   return adminUserModel;
}
```

مثال متد ویرایش AdminUserService.update

```
public AdminUserModel delete(@NotNull Integer id) throws Exception {

   //ثبت تمام فعالیتهایی که باید انجام شود شامل یک استیت و چند شرح استیت که همگی با کد رفرنس قابل دسترسی و آپدیت هستند. شرحهای استیت باید به ترتیب درست ثبت شوند
   //بعد از انجام هرکار وضعیت شرح استیت آن کار را  ترو می کنیم و بعد از تکمیل کل متد بدون خطا کل سطرهای استیت و شرحهای آن حذف میشوند
   stateService.create(adminUserModel.referenceCode, “adminUserServiceImpl”,” update”);
   stateDetailService.create(adminUserModel.referenceCode,” msgeo”,” cityPlace”);
   stateDetailService.create(adminUserModel.referenceCode,” msgeo”,” xxxx”);
   stateDetailService.create(adminUserModel.referenceCode,” msgeo”,” adminUser”);



   AdminUserModel adminUserModel = this.readById(id);
   AdminUser adminUser = adminUserRepository.findById(adminUserModel.getId()).get();

   //قبل از شروع به ویرایش باید تمام اطلاعات را کش میکنیم و در صورت بروز خطا با آن مدل کش شده شرح استیت هایی را که ترو بوده است را با فراخوانی متد ویرایش آن به حالت اول برمیگردانیم
   this.cache(adminUser)

   //ویرایش لوکیشن شهری ادمین
   final UpdateResponseModel updateResponseModel= this.cityPlaceStub.grpcUpdate(updateRequestModel);
   //در صورت ویرایش موفقیت آمیز هر مرحله ، وضعیت شرح استیت ان را ترو میکنیم
   stateDetailService.update(adminUserModel.getReferenceCode(), “cityPlace” , updateResponseModel.getId() , true);

   //ویرایش ..... ادمین
   final UpdateResponseModel updateResponseModel1= this.xxxxStub.grpcUpdate(updateRequestModel);
   //در صورت ویرایش موفقیت آمیز هر مرحله ، وضعیت شرح استیت آن را ترو میکنیم
   stateDetailService.update(adminUserModel.getReferenceCode(), “xxxx” , updateResponseModel1.getId() , true);

   //ویرایش ادمین
   adminUserContactRepository.updateById(adminUser.getDefaultAdminUserContact().getId());
   adminUserRepository.save(adminUser);
   //در صورت ویرایش موفقیت آمیز هر مرحله ، وضعیت شرح استیت آن را ترو میکنیم
   stateDetailService.update(adminUserModel.getReferenceCode(), “adminUser” , adminUser.getId() , true);

   return adminUserModel;
}
```

مثالی از انتیتی State و StateDetail

A16 | منظور از referenceId آیدی انتیتی است که شروع کننده کار است. علت ایجاد این فیلد این است که اگر به هرعلتی عملیات ناقص انجام شد و برای کارمند ناتیفیکیشن ارسال کردیم از طریق referenceId بتوانیم انتیتی

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	state								stateDetail				
2	id	referenceCode	serviceName	methodTypeEnum	dateStart	referenceId	isFailed	isRolledBack	id	grpcNameEnum	entityNameEnum	entityId	done
3	101	125322	adminUserServiceImpl	create	2020-07-11 13:01:00	30	true	false	200	msgeo	cityPlace	11	
4	102	159922	adminUserServiceImpl	create	2020-07-12 13:02:00	31	false	false	202	msgeo	cityPlace	13	
5									203	msgeo	city	14	
6	103	235622	adminUserServiceImpl	delete	2020-07-11 13:01:00	35	true	false	205	msgeo	cityPlace		TRUE
7									206	msgeo	city		TRUE
8									207	mslogin	adminUser		
9	104	235624	adminUserServiceImpl	update	2020-07-11 14:01:00	31	true	false	208	msgeo	cityPlace		TRUE
10									209	msgeo	city		TRUE
11									210	mslogin	adminUser		

منظور از referenceIdآیدی انتیتی است که شروع کننده کار است. علت ایجاد این فیلد این است که اگر به هرعلتی عملیات ناقص انجام شد و برای کارمند ناتیفیکیشن ارسال کردیم از طریق referenceIdبتوانیم انتیتی هایی که به referenceIdموردنظر وصل هستند را پیدا کنیم.