



پروژه اول: لحظه‌نگاری سراسری

موعد ارسال: ۱۳۹۶/۰۱/۱۷

مقدمه

سامانه لحظه‌نگاری سراسری، یک سامانه متن‌باز با انعطاف‌پذیری بالا مبتنی بر زبان جاوا است. هسته این نرم‌افزار یک سامانه بانک‌داری توزیع شده با قابلیت انتقال پول از یک شعبه به شعب دیگر است. برای حفظ صحت، برنامه از ثبت حالت سراسری سامانه پشتیبانی می‌کند. از جمله ویژگی‌های فنی این محصول می‌توان به موارد ذکر شده در جدول ۱ اشاره کرد.

جدول ۱: ویژگی‌های فنی سامانه لحظه‌نگاری

| ویژگی | مقدار |
|------------|---------------------|
| زبان | جاوا نسخه ۸ |
| سیستم‌عامل | مستقل از سیستم‌عامل |
| پروانه | MIT |

معماری

توسعه و بهبود سامانه لحظه‌نگاری سراسری نیازمند درک مناسبی از معماری آن است. هدف سامانه توزیع شده این است که برنامه‌های کاربردی را از بستر سخت‌افزاری و سیستم‌های عامل جدا کند. این هدف با فراهم‌سازی یک لایه میان‌افزار در یک سامانه توزیع شده محقق می‌شود. سبک‌های معماری بر اساس مؤلفه‌ها، روش ارتباط مؤلفه‌ها با یکدیگر و داده‌های مبادله شده بین آن‌ها بوجود می‌آید و اینکه در کنار یکدیگر چگونه برای یک سامانه پیکربندی می‌شوند. عوامل سبک‌های معماری عبارتند از: مؤلفه‌ها و اتصال‌دهنده‌ها.

مؤلفه یک واحد در یک سامانه توزیع شده است که باید قابلیت جایگزینی داشته باشد و با توجه به واسطه‌هایش سرویس فراهم کند. اتصال‌دهنده سازوکاری است که ارتباط و همکاری بین مؤلفه‌ها را فراهم می‌سازد. مانند ارتباط از طریق فراخوانی راه دور.

سامانه لحظه‌نگاری سراسری از معماری مبتنی بر شی استفاده کرده است. سبک معماری این مدل بر خلاف مدل لایه‌ای انعطاف زیادی دارد. چندین شی (مؤلفه) وجود دارد که هر یک می‌تواند از هر شی دیگر سرویس بگیرد. یعنی هر مؤلفه یا شی سرویس‌های خاص خودش را دارد که شاید با دیگری فرق کند. اگر یک شی از سرویس‌های شی دیگر استفاده کند با استفاده از یک متد فراخوانی و مشخص کردن نوع سرویس، درخواست خود را به طور مستقیم می‌فرستد. در این معماری مؤلفه‌ها در سامانه پراکنده هستند و از طریق فراخوانی رویه با هم ارتباط برقرار می‌کنند که ممکن است از راه دور نیز باشد.

کلاس‌ها و توابع مهم

در این بخش به شرح و معرفی کلاس‌ها و توابع مهم می‌پردازیم.

Snapshot ۱.۰

از کلاس مذکور برای مدیریت رویدادهای لحظه‌نگاری استفاده می‌شود. به عنوان نمونه شعبه بانک در یک لحظه زمانی خاص عملیات لحظه‌نگاری را آغاز می‌کند.

startSnapshot ۱.۱.۰

با استفاده از این تابع عملیات لحظه‌نگاری آغاز می‌شود. شعبه بانک، شناسه‌های دیگر شعب را ذخیره می‌کند.

```
public void startSnapshot(Bank bank, int balance, List<Bank> banks){  
    ...  
    incomingChannels  
        .addAll(banks  
            .parallelStream()  
            .filter(b->b.getId()!=bank.getId())  
            .map(Bank::getId)  
            .collect(Collectors.toSet()));  
}
```

incrementMoneyInTransit ۲.۱.۰

با استفاده از این تابع مقدار اعتبار در حال انتقال به بانک، ثبت می‌شود.

```
public void incrementMoneyInTransit(int recipientBankId, Bank bank){  
    ...  
    if(incomingChannels.contains(recipientBankId))  
        this.moneyInTransit += bank.getBalance();  
}
```

stopRecording ۳.۱.۰

با استفاده از این تابع عملیات لحظه‌نگاری از بانک مقصد را متوقف می‌شود.

```
public void stopRecording(Bank bank){  
    incomingChannels.remove(bank.getId());  
}
```

BankServerRemote ۲.۰

از کلاس مذکور برای انجام عملیات انتقال، دریافت و لحظه‌نگاری سراسری به شکل از راه دور استفاده می‌شود.

sendMoney ۱.۲.۰

با استفاده از این تابع عملیات انتقال اعتبار از بانک مبدأ به مقصد انجام می‌شود.

```
public void sendMoney(...) throws RemoteException {
    withdrawLock.lock();
    try {
        boolean isWithdraw = bankDao.withdraw(bank);
        if(isWithdraw){
            ...
            boolean isTransferred = bankDao.getRemoteBank(...).recieveMoney(...);
            ...
        }
    } finally {
        withdrawLock.unlock();
    }
}
```

recieveMoney ۲.۲.۰

با استفاده از این تابع عملیات دریافت اعتبار توسط بانک مقصد انجام می‌شود.

```
public boolean recieveMoney(...) throws RemoteException {
    depositLock.lock();
    try {
        bankDao.getBank(...).getSnapshot().incrementMoneyInTransit(...);
        bankDao.deposit(bank);
        return true;
    } finally {
        depositLock.unlock();
    }
}
```

recieveToken ۳.۲.۰

با استفاده از این تابع عملیات لحظه‌نگاری سراسری توسط بانک مبدأ انجام می‌شود.

```

public void receiveToken(int receiverBankId, int senderBankId) throws RemoteException {
    tokenLock.lock();
    depositLock.lock();
    withdrawLock.lock();
    try {
I    ...
        Snapshot snapshot = bankDao.getBank(senderBankId).getSnapshot();
        if (!snapshot.isRecording()) {
            snapshot.startSnapshot(...);
            ExecutorService executorService = Executors.newFixedThreadPool(...);
            bankDao()
                .allBanks()
                .parallelStream()
                .filter(b -> b.getId() != senderBankId)
                .forEach(bank -> executorService.execute(() -> {
                    ...
                    bankDao.getRemoteBank(bank).recieveToken(...);
                    ...
                }));
        }
        snapshot.stopRecording(bankDao.getBank(...));
        if (!snapshot.isRecording()){
            ...
            bankDao.getBank(...).getSnapshot().stopSnapshot();
        }
    } finally {
        tokenLock.unlock();
        depositLock.unlock();
        withdrawLock.unlock();
    }
}

```
