链滴

Guava EventBus 处理 JVM 异步事件（1）

作者：Eddie

原文链接：https://ld246.com/article/1585224756308

来源网站：链滴

前言

情景：当账号发生异地登陆时，需要通过客户端，邮箱，短信等方式发送警告信息。

建立事件模型

```
public class UnnormalLoginEvent {

    private String ip;
    private String userId;

    public UnnormalLoginEvent(String ip, String userId) {
        this.ip = ip;
        this.userId = userId;
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }

    public String getUserId() {
        return userId;
    }

    public void setUserId(String userId) {
        this.userId = userId;
    }
}
```

自定义监听器注解

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE})
public @interface Listener {
}
```

异常登陆监听器（可以将具体执行的方法置于此）

```
@Listener
public class UnnormalLoginListener {


    @Subscribe
    @AllowConcurrentEvents
    public void sendEmail(UnnormalLoginEvent event) throws InterruptedException {
        System.out.printf("email-send(%1$s,%2$s)...\n",event.getIp(),event.getUserId());
```

```
    }

    @Subscribe
    @AllowConcurrentEvents
    public void sendSMS(UnnormalLoginEvent event) {
        System.out.printf("sms-send(%1$s,%2$s)...\n",event.getIp(),event.getUserId());

    }

}
```

其中 @Subscribe 代表注册到 UnnormalLoginEvent 的方法，@AllowConcurrentEvents 标示其为程安全

自定义事件分发类 EventDispatcher

```
public class EventDispatcher {
    private EventBus eventBus;

    @Inject
    public EventDispatcher(EventBus eventBus) {
        this.eventBus = eventBus;
    }
    public void  postAsyncEvent(Object event){

        eventBus.post(event);
    }

}
```

EventBus 注入封装

```
@Singleton
public class EventBusProvider implements Provider<EventBus> {

    private EventBus eventBus;


    public EventBusProvider() throws IllegalAccessException, InstantiationException {
        eventBus =new AsyncEventBus(Executors.newCachedThreadPool());
        //扫描所有包含@Listener注解的类，注册到EventBus
        List<Class<?>> list = getClassOfPackage("net.i_ng.listener", Listener.class);
        for ( Class<?> c:list) {

            eventBus.register(c.newInstance());
        }

    }


    public List<Class<?>> getClassOfPackage(String packagenom, Class<? extends Annotatio
> clazz) {
```

```
        final ClassLoader loader = Thread.currentThread()
                .getContextClassLoader();
        List list = new ArrayList();

        try {

            ClassPath classpath = ClassPath.from(loader);
            for (ClassPath.ClassInfo classInfo : classpath.getTopLevelClasses(packagenom)) {
                Class<?> c =classInfo.load();
                if (c.isAnnotationPresent(clazz)) {
                    list.add(c);
                }


            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            return list;
        }

    }

    public EventBus get() {

        return eventBus;
    }
}
```

Guice 工具类

```
public class GuiceSupport {
    private static ThreadLocal<Injector> map = new ThreadLocal<Injector>();


    public static      <T> T getInstance(Class<T> var1){
        Injector injector=map.get();

        if (injector==null){
           //EventBus 实例化由EventBusProvider提供
            injector = Guice.createInjector((Module) binder -> binder.bind(EventBus.class).toProvi
er(EventBusProvider.class));
            map.set(injector);

        }

        return  injector.getInstance(var1);

    }

}
```

测试

```
public class App {
    public static void main(String args[]) {

        EventDispatcher eventDispatcher = GuiceSupport.getInstance(EventDispatcher.class);

        UnnormalLoginEvent unnormalLoginEvent = new UnnormalLoginEvent("127.0.0.1", "888
9");

        eventDispatcher.postAsyncEvent(unnormalLoginEvent);

}}
```