Hystrix在Spring MVC中的使用

为使熔断控制和现有代码解耦，hystrix官方采用了Aspect方式。现在介绍hystrix在spring mvc的使用。

1、添加依赖

使用maven引入hystrix依赖：

```
<dependency>
    <groupId>com.netflix.hystrix</groupId>
    <artifactId>hystrix-javanica</artifactId>
    <version>1.5.12</version>
</dependency>
```

2、添加配置

新建hystrix.properties文件（名字随意定，里面将定义项目所有hystrix配置信息）

新建一个类HystrixConfig

```
public class HystrixConfig
{
    public void init()
    {
        Properties prop = new Properties();
        InputStream in = null;
        try
        {
            in = HystrixConfig.class.getClassLoader().getResourceAsStream("hystrix.properties");
            prop.load(in);
            in.close();
            System.setProperties(prop);
        }
```

```
        catch (Exception e)
                                15          {
            e.printStackTrace();
        }
    }
}
```

在spring的配置文件添加内容：

```
<!-- 添加了就不用加了 -->
<aop:aspectj-autoproxy proxy-target-class="true" />

<bean name="hystrixCommandAspect" class="com.netflix.hystrix.contrib.javanica.aop.aspectj.HystrixCommandAspect"/>
<bean id="hystrixConfig"  class="com.gary.test.HystrixConfig" init-method="init"/>
```

新建hystrixConfig bean主要是因为使用spring自带的context:property-placeholder配置加载器，hystrix无法读取。目前我只想到了通过System.setProperties的方式，若有其他方式欢迎指导。

3、hystrixCommand使用

举个简单的例子(写成接口方式是方便测试，普通的方法效果是一样的)：

```
@ResponseBody
@RequestMapping("/test.html")
@HystrixCommand
public String test(int s)
{
    logger.info("test.html start,s:{}", s);
    try
    {
        Thread.sleep(s * 1000);
    }
    catch (Exception e)
```

```
    {13         logger.error("test.html error.", e);
    }
    return "OK";
}
```

根据例子，我们可以看到和其他方法相比就添加了个@HystrixCommand注解，方法执行后会被HystrixCommandAspect拦截，拦截后会根据方法的基本属性（所在类、方法名、返回类型等）和HystrixCommand属性生成HystrixInvokable，最后执行。例子中，因为HystrixCommand属性为空，所以其groupKey默认为类名，commandKey为方法名。

通过HystrixCommand源码来看下可以设置的属性：

```
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@Documented
public @interface HystrixCommand {

    String groupKey() default "";

    String commandKey() default "";

    String threadPoolKey() default "";

    String fallbackMethod() default "";

    HystrixProperty[] commandProperties() default {};

    HystrixProperty[] threadPoolProperties() default {};

    Class<? extends Throwable>[] ignoreExceptions() default {};

    ObservableExecutionMode observableExecutionMode() default ObservableExecutionMode.EAGER;

    HystrixException[] raiseHystrixExceptions() default {};

```

```
    String defaultFallback() default "";
}
```

其中比较重要的是groupKey、commandKey、fallbackMethod（Fallback时调用的方法，一定要在同一个类中，且传参和返参要一致）。threadPoolKey一般可以不定义，线程池名会默认定义为groupKey。

再来看下HystrixCommandAspect是如何实现拦截的：

```
@Pointcut("@annotation(com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand)")
public void hystrixCommandAnnotationPointcut() {
}

@Pointcut("@annotation(com.netflix.hystrix.contrib.javanica.annotation.HystrixCollapser)")
public void hystrixCollapserAnnotationPointcut() {
}

@Around("hystrixCommandAnnotationPointcut() || hystrixCollapserAnnotationPointcut()")
public Object methodsAnnotatedWithHystrixCommand(final ProceedingJoinPoint joinPoint) throws Throwable {
    Method method = getMethodFromTarget(joinPoint);//见步骤1
    Validate.notNull(method, "failed to get method from joinPoint: %s", joinPoint);
    if (method.isAnnotationPresent(HystrixCommand.class) && method.isAnnotationPresent(HystrixCollapser.class)) {
        throw new IllegalStateException("method cannot be annotated with HystrixCommand and HystrixCollapser " +
                "annotations at the same time");
    }
    MetaHolderFactory metaHolderFactory = META_HOLDER_FACTORY_MAP.get(HystrixPointcutType.of(method));//见步骤2
    MetaHolder metaHolder = metaHolderFactory.create(joinPoint);//见步骤3
    HystrixInvokable invokable = HystrixCommandFactory.getInstance().create(metaHolder);//见步骤4
    ExecutionType executionType = metaHolder.isCollapserAnnotationPresent() ?
            metaHolder.getCollapserExecutionType() : metaHolder.getExecutionType();

    Object result;
    try {
        if (!metaHolder.isObservable()) {
            result = CommandExecutor.execute(invokable, executionType, metaHolder);
```

```
        } else {
                result = executeObservable(invokable, executionType, metaHolder);//见步骤5
        }
    } catch (HystrixBadRequestException e) {
        throw e.getCause() != null ? e.getCause() : e;
    } catch (HystrixRuntimeException e) {
        throw hystrixRuntimeExceptionToThrowable(metaHolder, e);
    }
    return result;
}
```

- 步骤1：获取切入点方法；
- 步骤2：根据方法的注解HystrixCommand或者HystrixCollapser生成相应的CommandMetaHolderFactory或者CollapserMetaHolderFactory类。
- 步骤3：将原方法的属性set进metaHolder中；
- 步骤4：根据metaHolder生成相应的HystrixCommand，包含加载hystrix配置信息。

commandProperties加载的优先级为前缀hystrix.command.commandKey > hystrix.command.default > defaultValue(原代码默认)；

threadPool配置加载的优先级为 前缀hystrix.threadpool.groupKey.> hystrix.threadpool.default.> defaultValue(原代码默认).

- 步骤5：执行命令。

倘若需要给该方法指定groupKey和commandKey定义其fallback方法，则可通过添加注解属性来实现。如：

```
@ResponseBody
@RequestMapping("/test.html")
@HystrixCommand(groupKey = "groupTest", commandKey = "commandTest", fallbackMethod = "back")
public String test(int s)
{
    try
    {
```

```
        Thread.sleep(s * 1000);
    }
    catch (Exception e)
    {
    }
    logger.info("test.html start");
    return "OK";
}

private String back(int s)
{
    return "back";
}
```

- groupKey=”groupTest”是将该hystrix操作的组名定义为groupTest，该属性在读取threadPoolProperties时需要用到。读取的策略是先读取已groupTest为键值的配置缓存；若没有则读取已hystrix.threadpool.groupTest.为前缀的配置；若没有则读取hystrix.threadpool.为前缀的配置，最后才读取代码默认的值。
- commandKey=”commandTest”是将hystrix操作的命令名定义为commandTest，该属性在读取commandProperties时需要用到。读取的策略与上面的一致，只是前缀由hystrix.threadpool变为hystrix.command。
- fallbackMethod=”back”是给该hystrix操作定义一个降级fallback方法，值为降级方法的方法名，并且要与降级方法在同一个类下、相同的输入参数和返回参数。fallbackMethod可级联。

如果要给该方法指定一些hystrix属性，可通过在hystrix.properties中添加一些配置来实现。如给上述方法添加一些hystrix属性，示例如下：

```
#定义commandKey为commandTest的过期时间为3s
hystrix.command.commandTest.execution.isolation.thread.timeoutInMilliseconds=3000
#定义所有的默认过期时间为5s，不再是默认是1s。优先级小于上面配置
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=5000
#定义threadPoolKey为groupTest的线程池大小为15
hystrix.threadpool.groupTest.coreSize=15
#定义所有的线程池大小为为5，不再是默认是10。优先级小于上面配置
hystrix.threadpool.default.coreSize=5
```