在前面几篇文章中，我们分析了consumer端的代理生成过程。创建完成后，应用就可以进行调用了，调用的代码如下：

[java] view plain copy

```java
// 代理类中的sayHello方法
public String sayHello(String paramString) {
  // 将调用时的参数传入arrayOfObject中
  Object[] arrayOfObject = new Object[1];
  arrayOfObject[0] = paramString;
  // methods[0]表示的就是sayHello, handler的实现为com.alibaba.dubbo.rpc.proxy.InvokerInvocationHandler
  Object localObject = this.handler.invoke(this, methods[0], arrayOfObject);
  return (String)localObject;
}

// InvokerInvocationHandler的invoke方法
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
            // Object中的方法不用远程调用
        String methodName = method.getName();
        Class<?>[] parameterTypes = method.getParameterTypes();
        if (method.getDeclaringClass() == Object.class) {
            return method.invoke(invoker, args);
        }
        if ("toString".equals(methodName) && parameterTypes.length == 0) {
            return invoker.toString();
        }
        if ("hashCode".equals(methodName) && parameterTypes.length == 0) {
            return invoker.hashCode();
        }
        if ("equals".equals(methodName) && parameterTypes.length == 1) {
            return invoker.equals(args[0]);
        }
            // 将方法和参数封装成RpcInvocation后调用，recreate方法主要是在调用时如果发生异常则抛出异常，否则正常返回
        return invoker.invoke(new RpcInvocation(method,
```

```
args)).recreate();
30.     }
```

Invocation的主要作用是将（远程）方法调用时需要的信息+Invoker封装起来， 这些信息包括：方法名、参数类型、参数值、附加信息。结合前一篇Invoker创建流程，我们知道此处的invoker为com.alibaba.dubbo.rpc.cluster.support.wrapper.MockClusterInvoker，对应的invoke方法：

[java] view plain copy

```
1. public Result invoke(Invocation invocation) throws
RpcException {
2.     Result result = null;
3.         // 查询mock值，查询顺序methodName[sayHello].mock -> mock
-> default.mock，默认为false，即不进行mock
4.         String value =
directory.getUrl().getMethodParameter(invocation.getMethodName(),
Constants.MOCK_KEY, Boolean.FALSE.toString()).trim();
5.         if (value.length() == 0 ||
value.equalsIgnoreCase("false")){
6.             // 不进行mock的话直接调用后面的invoker
7.             result = this.invoker.invoke(invocation);
8.         } else if (value.startsWith("force")) {
9.             if (logger.isWarnEnabled()) {
10.                 logger.info("force-mock: " +
invocation.getMethodName() + " force-mock enabled , url : " +
directory.getUrl());
11.             }
12.             //如果值为force，表示强制mock，即不访问远端方法，直接调用
mock数据
13.             result = doMockInvoke(invocation, null);
14.         } else {
15.             // 其他的值，则是先调用后面的invoker，如果失败且不是业务
错误时使用mock数据，
16.                 // 非业务错误包含：网络错误、超时错误、禁止访问错误、
序列化错误及其他未知的错误，
17.                 // 业务错误则是接口实现类中的方法抛出的错误，如
sayHello调用时产生异常
18.                 try {
19.                 result = this.invoker.invoke(invocation);
20.             }catch (RpcException e) {
21.                 if (e.isBiz()) {
```

```
22.                     throw e;
23.                 } else {
24.                     if (logger.isWarnEnabled()) {
25.                         logger.info("fail-mock: " +
invocation.getMethodName() + " fail-mock enabled , url : " +
directory.getUrl(), e);
26.                     }
27.                     result = doMockInvoke(invocation, e);
28.                 }
29.             }
30.         }
31.         return result;
32. }
```

mock支持的配置方式包含多种：

1、return xxx（注意这里的xxx区分大小写）

empty: 返回空对象，根据Type返回空对象（单纯的new xxx()），如果是基础类型则返回对应的默认值

null: 返回null

true/false: 返回boolean

开头+结尾都是双引号(")或单引号(')的字符串：返回去掉开头、结尾字符的字符串

数字：返回数字

以{开头：按json格式解析，并将结果放入Map, 最终返回一个对象

以[开头：按json格式解析，并将结果放入List，最终返回一个List

其他：直接返回xxx

2、true/default

根据{interfaceName}Mock构造类名（如com.alibaba.dubbo.demo.DemoServiceMock），Mock类必须带有无参构造方法，如果失败则初始化失败，如果成功则在需要mock时会调用该类的对应方法

3、`force:return+null`

直接往注册中心写入该mock规则，在配置文件中不能使用这种方式。主要是为了临时降级。

[java] view plain copy

```
1.   private Result doMockInvoke(Invocation
invocation,RpcException e){
2. Result result = null;
3.     Invoker<T> minvoker ;
4.     // directory根据invocation中是否有
Constants.INVOCATION_NEED_MOCK，来判断获取的是一个normal invoker 还是
一个 mock invoker
5.       // 如果directorylist 返回多个mock invoker，只使用第一个
invoker.
6.     List<Invoker<T>> mockInvokers =
selectMockInvoker(invocation);
7. if (mockInvokers == null || mockInvokers.size() == 0){
8.                         // 如果没有则根据url创建一个MockInvoker
9.     minvoker = (Invoker<T>) new
MockInvoker(directory.getUrl());
10. } else {
11.     minvoker = mockInvokers.get(0);
12. }
13. try {
14.     result = minvoker.invoke(invocation);
15. } catch (RpcException me) {
16.     if (me.isBiz()) {
17.         result = new RpcResult(me.getCause());
18.     } else {
19.         throw new RpcException(me.getCode(),
getMockExceptionMessage(e, me), me.getCause());
20.     }
21. } catch (Throwable me) {
22.     throw new RpcException(getMockExceptionMessage(e, me),
me.getCause());
23. }
24. return result;
25.   }
```

MockInvoker的invoke方法：

[java] view plain copy

```
1.     public Result invoke(Invocation invocation) throws
RpcException {
2.         String mock =
getUrl().getParameter(invocation.getMethodName()+"."+Constants.MO
CK_KEY);
3.         if (invocation instanceof RpcInvocation) {
4.             ((RpcInvocation) invocation).setInvoker(this);
```

```
5.          }
6.          if (StringUtils.isBlank(mock)){
7.              mock = getUrl().getParameter(Constants.MOCK_KEY);
8.          }
9.
10.         if (StringUtils.isBlank(mock)){
11.             throw new RpcException(new
IllegalAccessException("mock can not be null. url :" + url));
12.         }
13.         mock = normallizeMock(URL.decode(mock));
14.         if
(Constants.RETURN_PREFIX.trim().equalsIgnoreCase(mock.trim())){
15.                 // 如果只含return则返回null
16.             RpcResult result = new RpcResult();
17.             result.setValue(null);
18.             return result;
19.         } else if (mock.startsWith(Constants.RETURN_PREFIX))
{
20.             // 如果是return 开头，则根据上面介绍的规则返回值
21.             mock =
mock.substring(Constants.RETURN_PREFIX.length()).trim();
22.             mock = mock.replace('`', '"');
23.             try {
24.                 Type[] returnTypes =
RpcUtils.getReturnTypes(invocation);
25.                 Object value = parseMockValue(mock,
returnTypes);
26.                 return new RpcResult(value);
27.             } catch (Exception ew) {
28.                 throw new RpcException("mock return invoke
error. method :" + invocation.getMethodName() + ", mock:" + mock
+ ", url: "+ url , ew);
29.             }
30.         } else if (mock.startsWith(Constants.THROW_PREFIX)) {
31.             // throw开头则按照throw后的字符串构造异常
32.             mock =
mock.substring(Constants.THROW_PREFIX.length()).trim();
33.             mock = mock.replace('`', '"');
34.             if (StringUtils.isBlank(mock)){
35.                 throw new RpcException(" mocked exception for
Service degradation. ");
36.             } else { //用户自定义类
37.                 Throwable t = getThrowable(mock);
38.                 throw new
RpcException(RpcException.BIZ_EXCEPTION, t);
```

```
39.             }
40.         } else { //调用实现类的对应方法，如实现类为
com.alibaba.dubbo.demo.DemoServiceMock，调用sayHello时其mock方法也为
sayHello
41.             try {
42.                 Invoker<T> invoker = getInvoker(mock);
43.                 return invoker.invoke(invocation);
44.             } catch (Throwable t) {
45.                 throw new RpcException("Failed to create
mock implemention class " + mock , t);
46.             }
47.         }
48.     }
49.
50.     private Invoker<T> getInvoker(String mockService){
51.             // Invoker由反射生成，需要缓存生成的Invoker（否则效率
低）
52.         Invoker<T> invoker =(Invoker<T>)
mocks.get(mockService);
53.         if (invoker != null ){
54.             return invoker;
55.         } else {
56.                         // 如果配置为默认，则mock类为原类名+Mock,如
com.alibaba.dubbo.demo.DemoService的mock类为
com.alibaba.dubbo.demo.DemoServiceMock
57.                         // 如果非默认配置，则按照配置的字符串创建类
58.                 Class<T> serviceType =
(Class<T>)ReflectUtils.forName(url.getServiceInterface());
59.             if (ConfigUtils.isDefault(mockService)) {
60.                 mockService = serviceType.getName() + "Mock";
61.             }
62.             // 创建mock类并判断mock类是否是原类的子类
63.             Class<?> mockClass =
ReflectUtils.forName(mockService);
64.             if (! serviceType.isAssignableFrom(mockClass)) {
65.                 throw new IllegalArgumentException("The mock
implemention class " + mockClass.getName() + " not implement
interface " + serviceType.getName());
66.             }
67.             // 这里copy的，应该是作者写重复了
68.             if (! serviceType.isAssignableFrom(mockClass)) {
69.                 throw new IllegalArgumentException("The mock
implemention class " + mockClass.getName() + " not implement
interface " + serviceType.getName());
70.             }
```

```
71.             try {
72.                                 // 创建实例，并创建对应的代理
73.                 T mockObject = (T) mockClass.newInstance();
74.                 invoker = proxyFactory.getInvoker(mockObject,
(Class<T>)serviceType, url);
75.                                 // 只缓存10000个Invoker
76.                 if (mocks.size() < 10000) {
77.                     mocks.put(mockService, invoker);
78.                 }
79.                 return invoker;
80.             } catch (InstantiationException e) {
81.                 throw new IllegalStateException("No such
empty constructor \"public " + mockClass.getSimpleName() + "()\"
in mock implemention class " + mockClass.getName(), e);
82.             } catch (IllegalAccessException e) {
83.                 throw new IllegalStateException(e);
84.             }
85.         }
86.     }
```

此时的url protocol,proxyFactory最终调用包装了JavassistProxyFactory的StubProxyFactoryWrapper，这里的getInvoker方法直接调用了JavassistProxyFactory.getInvoker(proxy, type, url)，JavassistProxyFactory根据mock类生成的代理类，在invoke方法中调用mock类的对应方法。

mock 逻辑并不复杂，但是在系统降级时非常有用。至于如何在线上修改配置进行降级，我们后面再聊。