容错实例的创建

开始

```
ReferenceConfig # createProxy()
invoker = REF_PROTOCOL.refer(interfaceClass, urls.get(0));
```

```
Protocol$Adaptive # refer()
return extension.refer(arg0, arg1);
```

```
ProtocolFilterWrapper # refer()
return protocol.refer(type, url);
```

```
ProtocolListenerWrapper # refer()
return protocol.refer(type, url);
```

```
RegistryProtocol # refer()
return doRefer(cluster, registry, type, url, qs);
```

```
RegistryProtocol # refer()
```

FailoverCluster的创建

```
RegistryProtocol # refer()
Cluster cluster = Cluster.getCluster(qs.get(CLUSTER_KEY));
```

```
Cluster # getCluster(一个参数)
return getCluster(name, true);
```

```
Cluster # getCluster(两个参数)
```

FailoverClusterInvoker的创建

```
RegistryProtocol # doRefer()
return interceptInvoker(migrationInvoker, url, consumerUrl, url);
```

```
RegistryProtocol # interceptInvoker()
listener.onRefer(this, invoker, consumerUrl, registryURL);
```

```
MigrationRuleListener # onRefer()
migrationRuleHandler.doMigrate(rule);
```

```
MigrationRuleHandler # doMigrate()
if (refreshInvoker(step, threshold, rule)) {
```

```
MigrationRuleHandler # refreshInvoker()
migrationInvoker.migrateToApplicationFirstInvoker(newRule);
```

```
MigrationInvoker # migrateToApplicationFirstInvoker()
refreshInterfaceInvoker(latch);
```

```
MigrationInvoker # refreshInterfaceInvoker()
invoker = registryProtocol.getInvoker(cluster, registry, type, url);
```

```
InterfaceCompatibleRegistryProtocol # getInvoker()
return doCreateInvoker(directory, cluster, registry, type);
```

```
RegistryProtocol # doCreateInvoker()
return (ClusterInvoker<T>) cluster.join(directory);
```

```
MockClusterWrapper # join()
return new MockClusterInvoker<T>(directory, this.cluster.join(directory));
```

```
AbstractCluster # join()
return buildClusterInterceptors(doJoin(directory), ...);
```

```
FailoverCluster # doJoin()
```

容错方案的调用

开始

入手点分析

```
ConsumerApplication # main()
String greetings = greetingService.hello();
```

```
InvokerInvocationHandler # invoke()
return invoker.invoke(rpcInvocation).recreate();
```

```
MigrationInvoker # invoke()
return currentAvailableInvoker.invoke(invocation);
```

服务降级判断

```
MockClusterInvoker # invoke()
result = this.invoker.invoke(invocation);
```

FilterChain一路过滤

```
AbstractCluster # invoke()
return filterInvoker.invoke(invocation);
```

```
FilterChainBuilder # invoke()
asyncResult = filter.invoke(nextNode, invocation);
```

```
ConsumerContextFilter # invoke()
return invoker.invoke(invocation);
```

```
FilterChainBuilder # invoke()
asyncResult = filter.invoke(nextNode, invocation);
```

```
FutureFilter # invoke()
return invoker.invoke(invocation);
```

```
FutureFilter # invoke()
return invoker.invoke(invocation);
```

```
FilterChainBuilder # invoke()
asyncResult = filter.invoke(nextNode, invocation);
```

```
MonitorFilter # invoke()
return invoker.invoke(invocation);
```

路由选择

```
AbstractClusterInvoker # invoke()
return doInvoke(invocation, invokers, loadbalance);
```