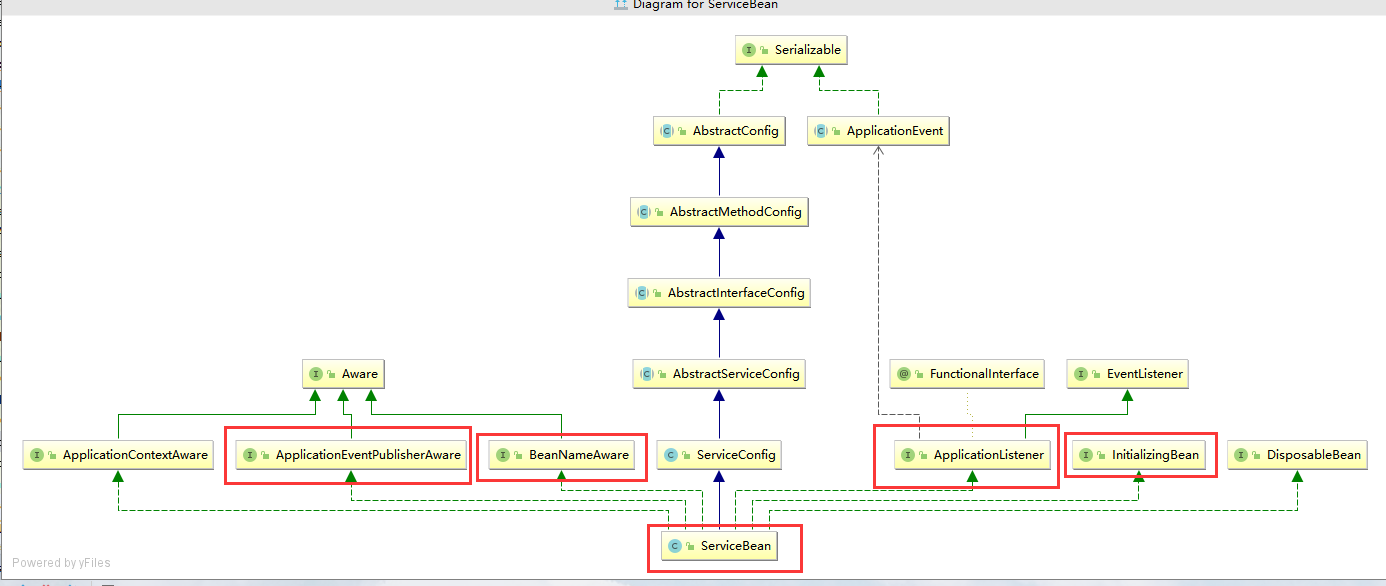
**Dubbo**

# ServiceBean

## 用途

作为<dubbo:service>标签的bean

## 类图



## 事件发布(用来启动服务)

### 代码

|  |
| --- |
| @Override **public void** onApplicationEvent(ContextRefreshedEvent event) {  **if** (!isExported() && !isUnexported()) {  **if** (***logger***.isInfoEnabled()) {  ***logger***.info(**"The service ready on spring started. service: "** + getInterface());  }  export();  } } |

# Zookeeper

## ZookeeperDynamicConfiguration

|  |
| --- |
| ZookeeperDynamicConfiguration(URL url, ZookeeperTransporter zookeeperTransporter) {  **this**.**url** = url;  **rootPath** = ***PATH\_SEPARATOR*** + url.getParameter(***CONFIG\_NAMESPACE\_KEY***, ***DEFAULT\_GROUP***) + **"/config"**;   **initializedLatch** = **new** CountDownLatch(1);  **this**.**cacheListener** = **new** CacheListener(**rootPath**, **initializedLatch**);  **this**.**executor** = Executors.*newFixedThreadPool*(1, **new** NamedThreadFactory(**this**.getClass().getSimpleName(), **true**));   **// 连接zookeeper**  **zkClient** = zookeeperTransporter.connect(url);  **zkClient**.addDataListener(**rootPath**, **cacheListener**, **executor**);  **try** {  *// Wait for connection* **this**.**initializedLatch**.await();  } **catch** (InterruptedException e) {  ***logger***.warn(**"Failed to build local cache for config center (zookeeper)."** + url);  } } |

## Zookeeper连接

双重检查

|  |
| --- |
| **@Override public ZookeeperClient connect(URL url) {  ZookeeperClient zookeeperClient;  List<String> addressList = getURLBackupAddress(url);  *// The field define the zookeeper server , including protocol, host, port, username, password* if ((zookeeperClient = fetchAndUpdateZookeeperClientCache(addressList)) != null && zookeeperClient.isConnected()) {  *logger*.info("find valid zookeeper client from the cache for address: " + url);  return zookeeperClient;  }  *// avoid creating too many connections， so add lock* synchronized (zookeeperClientMap) {  if ((zookeeperClient = fetchAndUpdateZookeeperClientCache(addressList)) != null && zookeeperClient.isConnected()) {  *logger*.info("find valid zookeeper client from the cache for address: " + url);  return zookeeperClient;  }   zookeeperClient = createZookeeperClient(toClientURL(url));  *logger*.info("No valid zookeeper client found from cache, therefore create a new client for url. " + url);  writeToClientMap(addressList, zookeeperClient);  }  return zookeeperClient; }** |

### ClientCnxn

#### SendThread

用来连接zookeeper的守护线程

|  |
| --- |
| @Override **public void run() {  clientCnxnSocket.introduce(this,sessionId);  clientCnxnSocket.updateNow();  clientCnxnSocket.updateLastSendAndHeard();  int to;  long lastPingRwServer = System.*currentTimeMillis*();  final int MAX\_SEND\_PING\_INTERVAL = 10000; *//10 seconds* while (state.isAlive()) {  try {  if (!clientCnxnSocket.isConnected()) {  if(!isFirstConnect){  try {  Thread.*sleep*(r.nextInt(1000));  } catch (InterruptedException e) {  *LOG*.warn("Unexpected exception", e);  }  }  *// don't re-establish connection if we are closing* if (closing || !state.isAlive()) {  break;  }  startConnect();  clientCnxnSocket.updateLastSendAndHeard();  }   if (state.isConnected()) {  *// determine whether we need to send an AuthFailed event.* if (zooKeeperSaslClient != null) {  boolean sendAuthEvent = false;  if (zooKeeperSaslClient.getSaslState() == ZooKeeperSaslClient.SaslState.*INITIAL*) {  try {  zooKeeperSaslClient.initialize(ClientCnxn.this);  } catch (SaslException e) {  *LOG*.error("SASL authentication with Zookeeper Quorum member failed: " + e);  state = States.*AUTH\_FAILED*;  sendAuthEvent = true;  }  }  KeeperState authState = zooKeeperSaslClient.getKeeperState();  if (authState != null) {  if (authState == KeeperState.*AuthFailed*) {  *// An authentication error occurred during authentication with the Zookeeper Server.* state = States.*AUTH\_FAILED*;  sendAuthEvent = true;  } else {  if (authState == KeeperState.*SaslAuthenticated*) {  sendAuthEvent = true;  }  }  }   if (sendAuthEvent == true) {  eventThread.queueEvent(new WatchedEvent(  Watcher.Event.EventType.*None*,  authState,null));  }  }  to = readTimeout - clientCnxnSocket.getIdleRecv();  } else {  to = connectTimeout - clientCnxnSocket.getIdleRecv();  }    if (to <= 0) {  String warnInfo;  warnInfo = "Client session timed out, have not heard from server in "  + clientCnxnSocket.getIdleRecv()  + "ms"  + " for sessionid 0x"  + Long.*toHexString*(sessionId);  *LOG*.warn(warnInfo);  throw new SessionTimeoutException(warnInfo);  }  if (state.isConnected()) {  *//1000(1 second) is to prevent race condition missing to send the second ping  //also make sure not to send too many pings when readTimeout is small* int timeToNextPing = readTimeout / 2 - clientCnxnSocket.getIdleSend() -   ((clientCnxnSocket.getIdleSend() > 1000) ? 1000 : 0);  *//send a ping request either time is due or no packet sent out within MAX\_SEND\_PING\_INTERVAL* if (timeToNextPing <= 0 || clientCnxnSocket.getIdleSend() > MAX\_SEND\_PING\_INTERVAL) {  sendPing();  clientCnxnSocket.updateLastSend();  } else {  if (timeToNextPing < to) {  to = timeToNextPing;  }  }  }   *// If we are in read-only mode, seek for read/write server* if (state == States.*CONNECTEDREADONLY*) {  long now = System.*currentTimeMillis*();  int idlePingRwServer = (int) (now - lastPingRwServer);  if (idlePingRwServer >= pingRwTimeout) {  lastPingRwServer = now;  idlePingRwServer = 0;  pingRwTimeout =  Math.*min*(2\*pingRwTimeout, *maxPingRwTimeout*);  pingRwServer();  }  to = Math.*min*(to, pingRwTimeout - idlePingRwServer);  }   clientCnxnSocket.doTransport(to, pendingQueue, outgoingQueue, ClientCnxn.this);  } catch (Throwable e) {  if (closing) {  if (*LOG*.isDebugEnabled()) {  *// closing so this is expected  LOG*.debug("An exception was thrown while closing send thread for session 0x"  + Long.*toHexString*(getSessionId())  + " : " + e.getMessage());  }  break;  } else {  *// this is ugly, you have a better way speak up* if (e instanceof SessionExpiredException) {  *LOG*.info(e.getMessage() + ", closing socket connection");  } else if (e instanceof SessionTimeoutException) {  *LOG*.info(e.getMessage() + *RETRY\_CONN\_MSG*);  } else if (e instanceof EndOfStreamException) {  *LOG*.info(e.getMessage() + *RETRY\_CONN\_MSG*);  } else if (e instanceof RWServerFoundException) {  *LOG*.info(e.getMessage());  } else {  *LOG*.warn(  "Session 0x"  + Long.*toHexString*(getSessionId())  + " for server "  + clientCnxnSocket.getRemoteSocketAddress()  + ", unexpected error"  + *RETRY\_CONN\_MSG*, e);  }  cleanup();  if (state.isAlive()) {  eventThread.queueEvent(new WatchedEvent(  Event.EventType.*None*,  Event.KeeperState.*Disconnected*,  null));  }  clientCnxnSocket.updateNow();  clientCnxnSocket.updateLastSendAndHeard();  }  }  }  cleanup();  clientCnxnSocket.close();  if (state.isAlive()) {  eventThread.queueEvent(new WatchedEvent(Event.EventType.*None*,  Event.KeeperState.*Disconnected*, null));  }  ZooTrace.*logTraceMessage*(*LOG*, ZooTrace.*getTextTraceLevel*(),  "SendThread exited loop for session: 0x"  + Long.*toHexString*(getSessionId())); }** |

dubbo://192.168.4.17:20880/com.order.service.OrderService?anyhost=true&application=order-service&bean.name=com.order.service.OrderService&bind.ip=192.168.4.17&bind.port=20880&deprecated=false&dubbo=2.0.2&dynamic=true&generic=false&interface=com.order.service.OrderService&methods=createOrder&pid=149960&register=true&release=2.7.3&revision=1.0-SNAPSHOT&side=provider&timestamp=1579319384152

# dubbo服务的暴露

## 服务的注册

### ZookeeperRegistry

|  |
| --- |
| **@Override public void doRegister(URL url) {  try {**  **// 创建节点  zkClient.create(toUrlPath(url), url.getParameter(*DYNAMIC\_KEY*, true));  } catch (Throwable e) {  throw new RpcException("Failed to register " + url + " to zookeeper " + getUrl() + ", cause: " + e.getMessage(), e);  } }** |

## Dubbo服务端口的暴露

### DubboProtocol

|  |
| --- |
| **private ExchangeServer createServer(URL url) {  url = URLBuilder.*from*(url)  *// send readonly event when server closes, it's enabled by default* .addParameterIfAbsent(*CHANNEL\_READONLYEVENT\_SENT\_KEY*, Boolean.*TRUE*.toString())  *// enable heartbeat by default* .addParameterIfAbsent(*HEARTBEAT\_KEY*, String.*valueOf*(*DEFAULT\_HEARTBEAT*))  .addParameter(*CODEC\_KEY*, DubboCodec.*NAME*)  .build();  String str = url.getParameter(*SERVER\_KEY*, *DEFAULT\_REMOTING\_SERVER*);   if (str != null && str.length() > 0 && !ExtensionLoader.*getExtensionLoader*(Transporter.class).hasExtension(str)) {  throw new RpcException("Unsupported server type: " + str + ", url: " + url);  }   ExchangeServer server;  try {  server = Exchangers.*bind*(url, requestHandler);  } catch (RemotingException e) {  throw new RpcException("Fail to start server(url: " + url + ") " + e.getMessage(), e);  }   str = url.getParameter(*CLIENT\_KEY*);  if (str != null && str.length() > 0) {  Set<String> supportedTypes = ExtensionLoader.*getExtensionLoader*(Transporter.class).getSupportedExtensions();  if (!supportedTypes.contains(str)) {  throw new RpcException("Unsupported client type: " + str);  }  }   return server; }** |

### org.apache.dubbo.remoting.transport.netty4.NettyServer

|  |
| --- |
| **protected void doOpen() throws Throwable {  bootstrap = new ServerBootstrap();   bossGroup = new NioEventLoopGroup(1, new DefaultThreadFactory("NettyServerBoss", true));  workerGroup = new NioEventLoopGroup(getUrl().getPositiveParameter(*IO\_THREADS\_KEY*, Constants.*DEFAULT\_IO\_THREADS*),  new DefaultThreadFactory("NettyServerWorker", true));   final NettyServerHandler nettyServerHandler = new NettyServerHandler(getUrl(), this);  channels = nettyServerHandler.getChannels();   bootstrap.group(bossGroup, workerGroup)  .channel(NioServerSocketChannel.class)  .childOption(ChannelOption.*TCP\_NODELAY*, Boolean.*TRUE*)  .childOption(ChannelOption.*SO\_REUSEADDR*, Boolean.*TRUE*)  .childOption(ChannelOption.*ALLOCATOR*, PooledByteBufAllocator.*DEFAULT*)  .childHandler(new ChannelInitializer<NioSocketChannel>() {  @Override  protected void initChannel(NioSocketChannel ch) throws Exception {  *// FIXME: should we use getTimeout()?* int idleTimeout = UrlUtils.*getIdleTimeout*(getUrl());  NettyCodecAdapter adapter = new NettyCodecAdapter(getCodec(), getUrl(), NettyServer.this);  ch.pipeline()*//.addLast("logging",new LoggingHandler(LogLevel.INFO))//for debug* .addLast("decoder", adapter.getDecoder())  .addLast("encoder", adapter.getEncoder())  .addLast("server-idle-handler", new IdleStateHandler(0, 0, idleTimeout, *MILLISECONDS*))  .addLast("handler", nettyServerHandler);  }  });  *// bind* ChannelFuture channelFuture = bootstrap.bind(getBindAddress());  channelFuture.syncUninterruptibly();  channel = channelFuture.channel();  }** |

# 服务调用

## FailoverClusterInvoker

### 介绍

集群默认的容错模式:失败自动切换

### 远程调用

|  |
| --- |
| **@Override @SuppressWarnings({"unchecked", "rawtypes"}) public Result doInvoke(Invocation invocation, final List<Invoker<T>> invokers, LoadBalance loadbalance) throws RpcException {  List<Invoker<T>> copyInvokers = invokers;  checkInvokers(copyInvokers, invocation);  String methodName = RpcUtils.*getMethodName*(invocation);**  **// 重试的次数  int len = getUrl().getMethodParameter(methodName, *RETRIES\_KEY*, *DEFAULT\_RETRIES*) + 1;  if (len <= 0) {  len = 1;  }  *// retry loop.* RpcException le = null; *// last exception.* List<Invoker<T>> invoked = new ArrayList<Invoker<T>>(copyInvokers.size()); *// invoked invokers.* Set<String> providers = new HashSet<String>(len);  for (int i = 0; i < len; i++) {  *//Reselect before retry to avoid a change of candidate `invokers`.  //NOTE: if `invokers` changed, then `invoked` also lose accuracy.* if (i > 0) {  checkWhetherDestroyed();  copyInvokers = list(invocation);  *// check again* checkInvokers(copyInvokers, invocation);  }  Invoker<T> invoker = select(loadbalance, invocation, copyInvokers, invoked);  invoked.add(invoker);  RpcContext.*getContext*().setInvokers((List) invoked);  try {  Result result = invoker.invoke(invocation);  if (le != null && *logger*.isWarnEnabled()) {  *logger*.warn("Although retry the method " + methodName  + " in the service " + getInterface().getName()  + " was successful by the provider " + invoker.getUrl().getAddress()  + ", but there have been failed providers " + providers  + " (" + providers.size() + "/" + copyInvokers.size()  + ") from the registry " + directory.getUrl().getAddress()  + " on the consumer " + NetUtils.*getLocalHost*()  + " using the dubbo version " + Version.*getVersion*() + ". Last error is: "  + le.getMessage(), le);  }  return result;  } catch (RpcException e) {  if (e.isBiz()) { *// biz exception.* throw e;  }  le = e;  } catch (Throwable e) {  le = new RpcException(e.getMessage(), e);  } finally {  providers.add(invoker.getUrl().getAddress());  }  }  throw new RpcException(le.getCode(), "Failed to invoke the method "  + methodName + " in the service " + getInterface().getName()  + ". Tried " + len + " times of the providers " + providers  + " (" + providers.size() + "/" + copyInvokers.size()  + ") from the registry " + directory.getUrl().getAddress()  + " on the consumer " + NetUtils.*getLocalHost*() + " using the dubbo version "  + Version.*getVersion*() + ". Last error is: "  + le.getMessage(), le.getCause() != null ? le.getCause() : le); }** |