二

06 ZooKeeper 的网络通信协议详解

本节课我们将学习 ZooKeeper 的网络通信协议。同时，本节课也是基础篇中的最后一节课。在 ZooKeeper 中无论是客户端和服务器之间的通信，还是集群之间服务器的内部协同工作都是基于网络进行通信的。而网络通信协议则是影响 ZooKeeper 性能和稳定性的核心点。

ZooKeeper 协议简述

说到网络通信协议我们最为熟悉的应该就是 TCP/IP 协议。而 ZooKeeper 则是在 TCP/IP 协议的基础上实现了自己特有的通信协议格式。在 ZooKeeper 中一次客户端的请求协议由请求头、请求体组成。而在一次服务端的响应协议中由响应头和响应体组成。

ZooKeeper 协议的底层实现

我们大概了解了 ZooKeeper 中的网络通信协议的结构后。接下来我们看一下在 ZooKeeper 中的内部对于网络通信协议的底层是怎么样实现的。

请求协议

请求协议就是客户端向服务端发送的协议。比如我们经常用到的会话创建、数据节点查询等操作。都是客户端通过网络向 ZooKeeper 服务端发送请求协议完成的。

客户端请求头底层解析

首先，我们先看一下请求头的内部的实现原理。在 ZooKeeper 中请求头是通过 RequestHeader 类实现的。首先 RequestHeader 类实现了 Record 接口，用于之后在网络传输中进行序列化操作。

我们可以看到 RequestHeader 类中只有两个属性字段分别是 xid 和 type。这两个字段在我们第一节课 ZooKeeper 的数据模型中介绍过，分别代表客户端序号用于记录客户端请求的发起顺序以及请求操作的类型。

```
class RequestHeader implements Record{

  private int xid;

  private int type;

}
```

客户端请求体底层解析

我们接下来再看一下客户端请求协议的请求体，协议的请求体包括了协议处理逻辑的全部内容，一次会话请求的所有操作内容都涵盖在请求体中。在 ZooKeeper 的内部实现中，根据不同的请求操作类型，会采用不同的结构封装请求体。接下来我们就以最常用的创建一次会话和数据节点的查询和更新这三种操作来介绍，深入底层看看 ZooKeeper 在内部是如何实现的。

会话创建

前面的课程我们已经介绍了 ZooKeeper 中的会话创建以及会话管理等相关知识。通过之前的学习我们知道了在 ZooKeeper 客户端发起会话时，会向服务端发送一个会话创建请求，该请求的作用就是通知 ZooKeeper 服务端需要处理一个来自客户端的访问链接。

而服务端处理会话创建请求时所需要的所有信息都包括在请求体内。在 ZooKeeper 中该请求体是通过 ConnectRequest 类实现的，其内部一共包括了五种属性字段。分别是 protocolVersion 表示该请求协议的版本信息、lastZxidSeen 最后一次接收到的服务器的 zxid 序号、timeOut 会话的超时时间、会话标识符 sessionId 以及会话的密码 password。有了这些信息 ZooKeeper 服务端在接收一个请求时，就可以根据请求体的信息进行相关的操作了。

```
ConnectRequest implements Record {

  private int protocolVersion;

  private long lastZxidSeen;

  private int timeOut;

  private long sessionId;

  private byte[] passwd;

 }
```

节点查询

在我们通过客户端 API 查询 ZooKeeper 服务器上的数据节点时，客户端会向服务端发送 GetDataRequest 会话请求。与上面介绍的会话请求不同。ZooKeeper 在处理获取数据节点会话请求时，选择了另一种结构作为该协议的请求体。而具体的实现类则是 GetDataRequest 。在 GetDataRequest 类中首先实现了 Record 接口用于序列化操作。其具有两个属性分别是字符类型 path 表示要请求的数据节点路径以及布尔类型 watch 表示该节点是否注册了 Watch 监控。

节点路径如下：

```
public class GetDataRequest implements Record {

  private String path;

  private boolean watch;
```

节点更新

最后，我们来看一下最后一种会话操作类型即节点的更新操作，同样的在客户端向服务端发送一个数据节点更新操作时，其在网络上实际发送的是更新操作的请求协议。而在 ZooKeeper 中对于协议内部的请求体，ZooKeeper 通过 SetDataRequest 类进行了封装。在 SetDataRequest 内部也包含了三种属性，分别是 path 表示节点的路径、data 表示节点数据信息以及 version 表示节点期望的版本号用于锁的验证。

```
public class SetDataRequest implements Record {

  private String path;

  private byte[] data;

  private int version;

 }
```

到目前为止我们就对 ZooKeeper 客户端在一次网络会话请求中所发送的请求协议的内部结构和底层实现都做了介绍，然而这些都是客户端向服务器端的请求协议，接下来我们就继续分析 ZooKeeper 服务端向客户端发送的响应协议是如何实现的。

响应协议

响应协议可以理解为服务端在处理客户端的请求后，返回相关信息给客户端。而服务端所采用的响应协议类型则要根据客户端的请求协议类型来选择。

服务端请求头解析

在服务端接收到客户端的请求后，执行相关操作将结果通知给客户端。而在 ZooKeeper 服务端向客户单发送的响应协议中，也是包括了请求头和请求体。而与客户端的请求头不同的是在 ZooKeeper 服务端的请求头多了一个错误状态字段。具体的实现类是 ReplyHeader。

```
public class ReplyHeader implements Record {

  private int xid;

  private long zxid;

  private int err;

  }
```

服务端请求体解析

下面我们再看一下响应协议的请求体部分，服务端的请求体可以理解为对客户端所请求内容的封装，一个服务端的请求体包含了客户端所要查询的数据而对于不同的请求类型，在 ZooKeeper 的服务端也是采用了不同的结构进行处理的。与上面我们讲解客户端请求体的方法一样，我们还是通过会话的创建、数据节点的查询和修改这三种请求操作来介绍，看看 ZooKeeper 服务端是如何响应客户端请求的。

响应会话创建

对于客户端发起的一次会话连接操作，ZooKeeper 服务端在处理后，会返回给客户端一个 Response 响应。而在底层代码中 ZooKeeper 是通过 ConnectRespose 类来实现的。在该类中有四个属性，分别是 protocolVersion 请求协议的版本信息、timeOut 会话超时时间、sessionId 会话标识符以及 passwd 会话密码。

```
public class ConnectResponse implements Record {

  private int protocolVersion;

  private int timeOut;

  private long sessionId;

  private byte[] passwd;

  }
```

响应节点查询

在客户端发起查询节点数据的请求时，服务端根据客户端发送的节点路径，并验证客户端具有相应的权限后，会将节点数据返回给客户端。而 ZooKeeper 服务端通过

GetDataResponse 类来封装查询到的节点相关信息到响应协议的请求体中。在 GetDataResponse 内部有两种属性字段分别是 data 属性表示节点数据的内容和 stat 属性表示节点的状态信息。

```
public class GetDataResponse implements Record {

  private byte[] data;

  private org.apache.zookeeper.data.Stat stat;

  }
```

响应节点更新

在客户端发送一个节点变更操作后， ZooKeeper 服务端在处理完相关逻辑后，会发送一个响应给客户端。而在 ZooKeeper 中更新完节点后会将操作结果返回给客户端，节点更新操作的响应协议请求体通过 SetDataResponse 类来实现。而在该类的内部只有一个属性就是 stat 字段，表示该节点数据更新后的最新状态信息。

```
public class SetDataResponse implements Record {

  private org.apache.zookeeper.data.Stat stat;

  }
```

结束

会话相关的知识点在之前的课程中我们已经讲到过了，从客户端到服务端的处理过程我们都已经很熟悉了。本节课我们从网络传输的角度来分析 ZooKeeper 底层的实现，通过本节课的学习后，我们就将 ZooKeeper 会话请求的全部过程都介绍完了。

通过本节课的学习，我们知道了 ZooKeeper 中的网络通信协议是通过不同的类具体实现的。那么我们思考这样一个问题。如果一个会话的超时时间是在服务器端实现的。会话超时服务器端会主动关闭会话。所以在会话请求的请求体中包括了设置会话超时的属性字段，而我们经常会遇到一个问题就是明明通过客户端将超时时间设置了一个值，而在实际执行的时候会话超时的时间可能远远小于我们设置的时间。这是因为在 ZooKeeper 处理会话超时时间的时候不只是简单地使用客户端传来的超时时间还会根据 minSessionTimeout 和 maxSessionTimeout 这两个参数来调整时间的设置。这里请大家在日常设置中多注意。

上一页 下一页