1. 发明名称

一种基于协议的打包和拆包方法及其实现模块

1. 技术领域

本发明涉及卫星通信和控制领域，具体是一种用于卫星指令编制与反演的协议数据打包和拆包方法。该方法适用于卫星指令的生成、传输和解析，能够有效支持卫星系统的指令编制、数据传输和实时控制。

1. 背景技术

在卫星通信系统中，指令的编制与反演是确保卫星正常运行和执行任务的关键环节。卫星指令通常由地面控制中心生成，并通过通信链路发送到卫星。指令的格式和内容必须符合特定的协议，以确保卫星能够正确解析和执行。

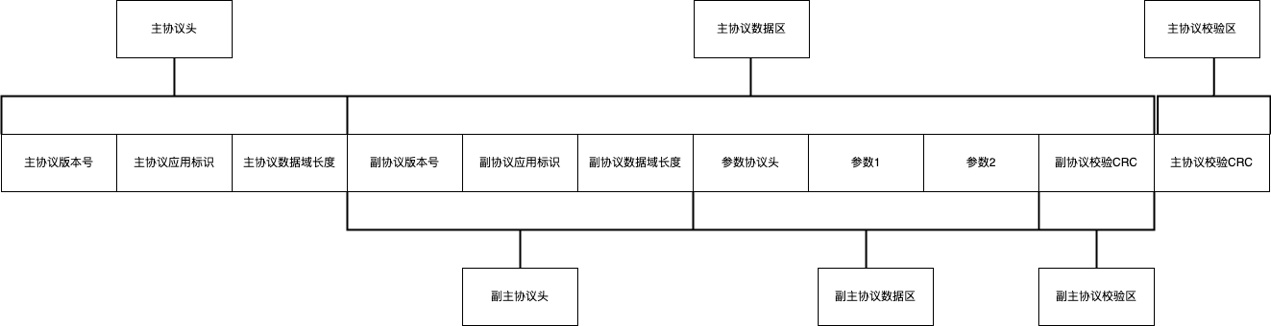
现有的卫星指令编制技术通常依赖于手动编写指令格式和解析逻辑，导致以下问题：

1. 复杂性：卫星指令的格式多样且复杂，手动编写容易出错，且难以维护。
2. 效率低下：在指令编制过程中，频繁的格式转换和数据处理会导致效率低下，影响指令的实时性。
3. 灵活性不足：现有技术难以适应不同卫星任务的需求，缺乏灵活的协议定义和扩展能力。

为了解决上述问题，本发明提出了一种基于协议的打包和拆包方法，通过定义标准化的协议结构，实现卫星指令的自动化编制与反演。该方法能够提高指令编制的效率和准确性，支持多种卫星任务的灵活配置。

1. 发明内容

本发明提供了一种基于协议的打包和拆包方法，通过定义协议结构和节点属性，实现了协议数据的自动化处理。该方法的核心在于使用枚举类型和接口设计，提供灵活的协议解析和数据处理能力。



1. 协议结构设计

协议由多个节点组成，每个节点包含以下属性：

1. 名称：节点的标识
2. 长度：节点的字节长度
3. 值类型：节点值的数据类型（如无符号整数、有符号整数、浮点数等）
4. 计算方法：节点值的计算和反向计算方法
5. 子节点：支持嵌套的子节点结构
6. 示例协议配置

以下是一个示例协议配置文件 proto.xml，展示了如何定义协议的结构，并对每个节点进行详细解释：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <protocol name="主协议">  <header name="主协议头">  <node>  <name>版本号</name>  <length>1</length>  <lengthUnit>BYTE</lengthUnit>  <value>80</value>  <valueType>HEX</valueType>  <endianType>BIG\_BYTE</endianType>  </node>  <node>  <name>主协议体长度</name>  <length>4</length>  <lengthUnit>BYTE</lengthUnit>  <valueType>SIGNED\_INT</valueType>  <endianType>LITTLE\_BYTE</endianType>  <depOnBody>true</depOnBody>  <calFuncName>数据区长度计算</calFuncName>  <calFuncExp>x-1</calFuncExp>  <reverseCalFuncName>表达式计算</reverseCalFuncName>  <reverseCalFuncExp>x+1</reverseCalFuncExp>  </node>  </header>  <body name="主协议体">  <header name="副协议头">  <node>  <name>版本</name>  <length>2</length>  <value>A1B1</value>  <lengthUnit>BYTE</lengthUnit>  <valueType>HEX</valueType>  <endianType>BIG\_BYTE</endianType>  </node>  <node>  <name>副协议体长度</name>  <length>4</length>  <lengthUnit>BYTE</lengthUnit>  <valueType>SIGNED\_INT</valueType>  <endianType>LITTLE\_BYTE</endianType>  <depOnBody>true</depOnBody>  <calFuncName>数据区长度计算</calFuncName>  <calFuncExp>x-2</calFuncExp>  <reverseCalFuncName>表达式计算</reverseCalFuncName>  <reverseCalFuncExp>x+2</reverseCalFuncExp>  </node>  </header>  <body name="副协议体">  <node>  <name>固定值</name>  <length>4</length>  <value>1024</value>  <lengthUnit>BYTE</lengthUnit>  <valueType>SIGNED\_INT</valueType>  <endianType>LITTLE\_BYTE</endianType>  </node>  <node>  <name>参数1</name>  <length>4</length>  <lengthUnit>BYTE</lengthUnit>  <valueType>FLOAT</valueType>  <endianType>BIG\_BYTE</endianType>  <calFuncName>表达式计算</calFuncName>  <calFuncExp>x\*1.25</calFuncExp>  <reverseCalFuncName>表达式计算</reverseCalFuncName>  <reverseCalFuncExp>x/1.25</reverseCalFuncExp>  </node>  </body>  <check name="副协议校验区">  <node>  <name>副协议数据区CRC码</name>  <length>2</length>  <lengthUnit>BYTE</lengthUnit>  <valueType>UNSIGNED\_INT</valueType>  <endianType>BIG\_BYTE</endianType>  <calFuncName>CRC计算</calFuncName>  </node>  </check>  </body>  <check name="主协议校验区">  <node>  <name>主协议数据区CRC码</name>  <length>2</length>  <lengthUnit>BYTE</lengthUnit>  <valueType>UNSIGNED\_INT</valueType>  <endianType>BIG\_BYTE</endianType>  <calFuncName>CRC计算</calFuncName>  </node>  </check> </protocol> |

1. 节点解释

* 协议根节点 <protocol>：表示整个协议，包含协议的名称属性。
* 头部 <header>：包含协议的头部信息。
* 节点 <node>：
  + <name>：节点的名称，例如“版本号”。
  + <length>：节点的字节长度，例如“1”表示该节点占用1个字节。
  + <lengthUnit>：长度单位，支持以下两种：
    - BYTE：表示长度以字节为单位。
    - BIT：表示长度以比特为单位（如果需要支持比特长度，可以在此处添加）。
  + <value>：节点的默认值，例如“80”。
  + <valueType>：节点值的数据类型，例如“HEX”表示十六进制，可能的值包括：
    - UNSIGNED\_INT：无符号整数
    - SIGNED\_INT：有符号整数
    - FLOAT：浮点数
    - HEX：十六进制
    - BINARY：二进制
  + <endianType>：字节序类型，支持以下两种：
    - BIG\_BYTE：表示大端字节序。
    - LITTLE\_BYTE：表示小端字节序。
  + 主协议体 <body>：包含主协议的主体信息。
  + 副协议头 <header>：包含副协议的头部信息。
  + 副协议体 <body>：包含副协议的主体信息。
    - 节点 <node>：与头部节点相同的结构，定义副协议体的具体内容。
  + 校验区 <check>：用于存放校验信息，确保数据的完整性。
    - 节点 <node>：定义校验码的相关信息。

1. 算法设计

本发明的算法设计主要包括以下几个方面：

1. 数据打包算法：
   * + - 输入：用户提供的数据和协议配置。
       - 处理：
         * 读取协议配置，构建协议节点树。
         * 遍历节点树，根据节点的属性（如长度、值类型等）将数据序列化为字节流。
         * 计算各个节点的值，并根据计算方法进行处理。

依赖计算：某些节点的值可以依赖于其他节点的数据。例如，主协议头中的“主协议体长度”节点可以依赖于主协议体的内容，通过表达式计算得出。

* + - * 输出：生成符合协议格式的字节流。
      * 打包流程图



1. 数据拆包算法：
   * + - 输入：接收到的字节流。
       - 处理：
         * 根据协议配置，解析接收到的字节流，识别协议结构。
         * 遍历节点树，依次读取每个节点的数据，反序列化为相应的对象。
         * 校验数据的完整性，确保数据未被篡改。
       - 输出：生成协议对象，供后续处理使用。
       - 拆包流程图



1. 计算方法：
   * + - 提供计算方法的接口，包含正向计算和反向计算的抽象方法。
       - 用户可以实现不同的计算器类，定义具体的计算逻辑。
       - 通过工厂模式管理所有计算器，方便获取和使用。
       - 表达式执行：支持通过表达式执行计算，用户可以在协议配置中定义计算表达式，例如：
         * calFuncExp：用于定义计算时的表达式，例如 x-1 表示从数据区获取的长度减去1。
         * reverseCalFuncExp：用于定义反向计算的表达式，例如 x+1 表示在反序列化时将长度加1。
2. 实施例

以下是本发明的一个实施例：

1. 协议配置：用户通过 XML 文件定义协议结构，包括协议头、数据区和校验区。
2. 打包过程：
   1. 读取协议配置，构建协议节点树。
   2. 根据节点属性，将数据序列化为字节流。
3. 拆包过程：
   1. 接收字节流，解析协议结构。
   2. 将字节流反序列化为协议对象，供后续处理。
4. 优点
5. 灵活性：支持多种协议格式的定义和解析，适应不同场景的需求。
6. 可扩展性：通过接口和工厂模式，用户可以方便地扩展新的计算方法和协议类型。
7. 易维护性：清晰的结构和模块化设计，降低了代码的复杂性，便于后续维护和更新。
8. 数据完整性：通过自动计算校验码，确保数据在传输过程中的完整性和可靠性。
9. 动态计算能力：支持节点间的依赖关系和动态计算，提升了协议的灵活性和适应性。
10. 结论

本发明提供了一种高效、灵活的协议打包和拆包方法，能够显著提高数据处理的效率和准确性，具有广泛的应用前景。希望通过本专利的申请，能够保护本发明的创新性和实用性。