# 一种基于协议的打包和拆包方法及其实现模块

## 技术领域

本发明属于卫星通信和控制技术领域，主要涉及卫星指令数据的处理方法，包括协议定义、数据打包和拆包等关键技术。该方法可广泛应用于航天器测控、卫星遥测遥控、空间数据传输等场景，为卫星系统的指令处理和数据交互提供了标准化、自动化的解决方案。

## 背景技术

在航天领域，卫星通信系统的指令处理是确保卫星正常运行和任务执行的核心环节。随着航天技术的快速发展，卫星系统的功能日益复杂，传统的人工编码方法逐渐暴露出不足，难以满足新一代卫星系统的需求。现有技术主要存在以下问题：

1. 开发效率低：手动编写代码耗时长，难以快速适应协议变更和扩展需求。
2. 容错性差：缺乏自动化错误检测机制，易因人为失误导致指令处理失败。
3. 可维护性不足：代码复用率低，协议变更时需大幅修改代码，维护成本高。
4. 实时性不足：现有方案在处理大量数据时性能不佳，无法满足卫星系统对指令处理的时效性要求。
5. 可靠性不强：指令处理中的错误可能导致严重后果，现有方案的容错性和可靠性有待提升。

目前，业内常用的卫星指令处理方案包括：

1. 基于固定格式的解析方案  
   实现简单，但灵活性差，无法适应动态协议格式的变更和扩展。
2. 基于脚本语言的动态解析方案  
   具有一定灵活性，但执行效率低，不适用于实时性要求高的场景。
3. 基于二进制协议的专用解决方案  
   性能优越，但定制化程度高，通用性差，难以在不同系统间复用。
4. 这些方案在面对新一代卫星系统的挑战时，普遍存在指令格式多样性、数据处理复杂性、实时性要求和可靠性保障等方面的不足。因此，亟需一种新的技术方案，提供一个统一的、高效的、可靠的卫星指令处理框架，支持灵活的协议定义、自动化的数据处理和可靠的错误处理，同时具备良好的扩展性和维护性。

## 3. 发明内容

## 发明目的

本发明的目的在于提供一种基于XML格式的通用协议描述方法，用于实现卫星指令的编制与反编，解决现有技术中存在的开发效率低、灵活性差、实时性不足和可靠性不高等问题。该方法通过以下特点和优势实现：

1. 通用协议描述能力  
   采用XML格式定义指令协议，提供标准化、结构化的协议描述方式，支持复杂的协议结构定义，包括多层嵌套、数组、可选字段等，实现协议定义的规范化和标准化。
2. 多层嵌套协议支持  
   允许协议定义中包含子协议，支持任意层次的协议嵌套，子协议可以独立定义和复用，提高协议描述的模块化程度，同时支持协议间的继承和组合关系，便于构建复杂的协议体系。
3. 节点依赖关系处理  
   支持协议节点之间的复杂依赖关系定义，提供前向引用和后向引用机制，处理节点间的数据关联，自动维护和验证节点间的依赖完整性。
4. 灵活的计算转换功能  
   支持多种数据类型转换，提供丰富的计算函数库，支持算术运算、位运算、条件判断等，允许自定义计算规则和转换方法，满足特殊需求。
5. 自动化处理机制  
   基于协议描述自动生成打包和解包代码，自动处理数据类型转换、字节序调整等底层操作，提供完整的数据校验和错误处理机制。

通过这些技术特征，本发明能够显著提高卫星指令处理的效率和可靠性，为航天领域的数据通信提供强有力的技术支持。

## 技术方案

本发明的技术方案主要包括协议结构设计、节点类型和属性定义、函数库、关键技术难点及解决方案等部分。

## 协议结构设计

协议结构采用XML格式进行描述，支持多层嵌套的协议定义。以下是一个复杂的多级嵌套的指令协议示例：

|  |
| --- |
| <protocol name="遥测指令">  <header name="一级协议头" length="8" lengthUnit="BYTE">  <node name="版本号" length="1" lengthUnit="BYTE" type="HEX" value="0x01"/>  <node name="状态标志位" length="3" lengthUnit="BIT" type="BIT" value="0b001"/>  <node name="优先级标志" length="5" lengthUnit="BIT" type="BIT" value="0b10001"/>  <node name="协议体长度" length="4" lengthUnit="BYTE" type="INT" refValue="${../一级协议体}.length 1"/>  </header>  <body name="一级协议体">  <header name="二级协议头">  <node name="版本号" length="1" lengthUnit="BYTE" type="HEX" value="0x01"/>  <! 枚举类型 >  <node name="指令类型" length="1" lengthUnit="BYTE" type="ENUM" value="温度采集">  <enum value="0x01">温度采集</enum>  <enum value="0x02">姿态调整</enum>  </node>  <node name="协议体长度" length="4" lengthUnit="BYTE" type="INT" refValue="${../二级协议体}.length 1"/>  </header>  <body name="二级协议体">·  <header name="三级协议头" length="10" lengthUnit="BYTE">  <!accumulatedDay(x ,'20200101 00:00:00') 1 代表：计算时间戳距离2020年1月1日的天数1>  <node name="积日" length="8" lengthUnit="BYTE" type="TIMESTAMP" value="20200102 12:00:01"  convert="accumulatedDay(x , '20200101 00:00:00') 1"/>  <!accumulatedMillSeconds(x , '20200101 00:00:00') 1 代表：计算时间戳距离当前时间戳的毫秒数1>  <node name="积毫秒" length="8" lengthUnit="BYTE" type="TIMESTAMP" value="20200102 12:00:01"  convert="accumulatedMillSeconds(x ,'20200101 00:00:00') 1"/>  <node name="校验使能位" length="1" lengthUnit="BIT" type="BIT" value="0b1"/>  <node name="保留位" length="5" lengthUnit="BIT" type="BIT" value="0b00001"/>  <!x >> 2 代表：将整型数据右移2位>  <node name="电压值" length="2" lengthUnit="BYTE" type="INT" convert="x >> 2" optional="true"  value="16"/>  </header>  <body name="三级协议体" length="20" lengthUnit="BYTE">  <nodeGroup name="姿态参数组" repeat="1">  <!accumulatedDay(x , '20200101 00:00:00') 1 代表：计算时间戳距离2020年1月1日的天数+1>  <node name="积日" length="8" lengthUnit="BYTE" type="TIMESTAMP" value="20200102 12:00:01"  convert="accumulatedDay(x , '20200101 00:00:00') + 1"/>  <!accumulatedMillSeconds(x , '20200101 00:00:00') 1 代表：计算时间戳距离当前时间戳的秒数+1>  <node name="积毫秒" length="8" lengthUnit="BYTE" type="TIMESTAMP" value="20200102 12:00:01"  convert="accumulatedMillSeconds(x ,'20200101 00:00:00') + 1"/>  <!代表body中参数组个数>  <node name="参数组个数" length="2" lengthUnit="BYTE" type="INT" value="3"/>  <! 具体参数数据 >  <nodeGroup name="参数组" repeat="header.参数个数">  <node name="参数ID" length="2" lengthUnit="BYTE" type="INT" value="1"/>  <node name="数据类型" length="1" lengthUnit="BYTE" type="ENUM" value="整型">  <enum value="0x01">整型</enum>  <enum value="0x02">浮点型</enum>  <enum value="0x03">字符串</enum>  </node>  <node name="控制位" length="2" lengthUnit="BIT" type="ENUM" value="正常">  <enum value="0b01">正常</enum>  <enum value="0b11">紧急</enum>  <enum value="0b10">调试</enum>  </node>  <node name="工作模式" length="1" lengthUnit="BYTE" type="INT"  convert="if(x>80) then 'HIGH' else if(x>50) then 'MEDIUM' else 'LOW'" optional="true"  value="100"/>  <node name="参数值" length="4" lengthUnit="BYTE" type="DYNAMIC" endian="LITTLE"  convert="if(parent.数据类型=='整型') then x else x\*0.001" value="100"/>  <node name="参数状态" length="1" lengthUnit="BYTE" type="INT" optional="true" value="0"/>  <node name="温度阈值" length="2" lengthUnit="BYTE" type="INT" convert="(x \* 0.1) + 273.15"  optional="true" value="255"/>  </nodeGroup>  </nodeGroup>  </body>  <check name="三级协议校验" length="2" lengthUnit="BYTE">  <! 三级协议校验 >  <node name="数据校验" length="2" lengthUnit="BYTE" type="CRC16" refValue="crc16(../三级协议体)"/>  </check>  </body>  <check name="二级协议校验" length="4" lengthUnit="BYTE">  <! 二级协议校验 >  <node name="协议校验" length="4" lengthUnit="BYTE" type="CRC32" refValue="crc16(${../二级协议体})"/>  </check>  </body>  <check name="一级协议校验" length="4" lengthUnit="BYTE">  <! 一级协议校验 >  <node name="总校验" length="4" lengthUnit="BYTE" type="CRC32" refValue="crc16(${../一级协议体})"/>  </check> </protocol> |

## XML节点类型定义

协议由以下几个部分组成：

1. protocol: 协议对象，xml跟节点，包含header、body和check三个子节点
2. header: 协议头，包含一组固定的节点
3. body: 协议体，可以包含节点、节点组或子协议header、body、check，
4. nodeGroup: 节点组，可以包含节点和子节点组，支持重复
5. node: 节点，协议的叶子节点
6. check: 校验部分，包含一组校验节点

## 节点属性定义

1. Name: 节点名称,唯一标识符
2. length: 数据长度
3. lengthUnit: 长度单位,支持BYTE(字节)和BIT(比特)
4. refLength: 引用其他节点的长度进行计算，支持转换表达式：

* 数学运算: +,,\*,/,()
* 条件: if,then,else
* 位运算: &,|,>>,<<

示例：

|  |
| --- |
| <node name="数据字段" length="${header}.length \* 2" lengthUnit="BYTE" type="HEX"/> |

在此例中，数据字段的长度是header中数据长度字段值的两倍。

1. type: 数据类型,包括:

|  |  |  |
| --- | --- | --- |
| 类别 | 类型 | 含义 |
| 基本类型 | INT | 整数 |
| FLOAT | 浮点数 |
| BIT | 二进制 |
| HEX | 十六进制 |
| STRING | 字符串 |
| TIMESTAMP | 时间 |
| 复杂类型 | ENUM | 枚举 |
| DYNAMIC | 动态类型 |
| 校验类型 | CRC16 |  |
| CRC32 |  |
| CHECKSUM |  |

1. refValue: 引用其他节点值进行计算，支持转换表达式：

* 数学运算支持 +,,\*,/,()等运算符号
* 计算方法:支持自定义方法，可多层嵌套，内置方法列表如下：

|  |  |  |
| --- | --- | --- |
| 类型 | 方法 | 说明 |
| 数学方法 | math.abs(d) | 求 d 的绝对值 |
| math.round(d) | 四舍五入 |
| math.floor(d) | 向下取整 |
| math.ceil(d) | 向上取整 |
| math.sqrt(d) | 求 d 的平方根 |
| math.pow(d1,d2) | 求 d1 的 d2 次方 |
| math.log(d) | 求 d 的自然对数 |
| math.log10(d) | 求 d 以 10 为底的对数 |
| math.sin(d) | 正弦函数 |
| math.cos(d) | 余弦函数 |
| math.tan(d) | 正切函数 |
| math.atan(d) | 反正切函数 |
| math.acos(d) | 反余弦函数 |
| math.asin(d) | 反正弦函数 |
| 日期方法 | accumulatedDay(a,b) | 求日期a相对基准时间b的天数差 |
| accumulatedSeconds(a,b) | 求日期a相对基准时间b的秒数差 |
| accumulatedMillSeconds(a,b) | 求日期a相对基准时间b的毫秒数差 |

* 示例：

|  |
| --- |
| <node name="协议体长度" length="4" lengthUnit="BYTE" type="INT" refValue="${../一级协议体}.length 1"/> |

协议体长度的值引用了../一级协议体的长度并减去1

* 条件: if,then,else
* 位运算: &,|,>>,<<

1. endian: 字节序(BIG/LITTLE)
2. optional: 是否可选(true/false)
3. convert: 转换表达式，支持数学运算、函数、条件、位运算等，例如：

|  |
| --- |
| <node name="电压值" length="2" lengthUnit="BYTE" type="INT" convert="x >> 2" optional="true" value="16"/> |

* convert=“x >> 2”表示将节点值x右移2位。

|  |
| --- |
| <node name="工作模式" length="1" lengthUnit="BYTE" type="INT" convert="if(x>80) then 'HIGH' else if(x>50) then 'MEDIUM' else 'LOW'" optional="true" value="100"/> |

根据x的值返回不同的工作模式。

|  |
| --- |
| <node name="参数值" length="4" lengthUnit="BYTE" type="DYNAMIC" endian="LITTLE" convert="if(parent.数据类型=='整型') then x else x\*0.001" value="100"/> |

* 根据数据类型动态选择是否对x进行缩放。

|  |
| --- |
| <node name="积日" length="8" lengthUnit="BYTE" type="TIMESTAMP" value="20200102 12:00:01" convert="accumulatedDay(x, '20200101 00:00:00') 1"/> |

计算时间戳距离基准时间的天数并减1。

## 关键技术难点及解决方案

* 1. 多层协议嵌套
     + 难点：传统方法难以处理复杂的多层协议结构。
     + 解决方案：采用层次化XML结构，通过header/body/check三层结构清晰表达协议层级关系，支持任意深度的协议嵌套，实现协议的模块化设计。
  2. 复杂的数据引用
     + 难点：协议中节点间的依赖关系复杂，数据引用困难。
     + 解决方案：通过refLength和refValue属性实现跨层级引用，支持多级路径访问和动态计算，确保数据关联的准确性。
  3. 动态数据类型
     + 难点：数据类型在运行时才确定，传统方法无法灵活处理。
     + 解决方案：引入DYNAMIC类型，支持根据上下文动态解析数据类型，提供类型转换机制，确保解析准确。
  4. 校验机制
     + 难点：多层协议的校验逻辑复杂，传统方法难以实现精细控制。
     + 解决方案：设计独立的check节点，支持多种校验算法（如CRC16、CRC32），实现多层级校验，确保数据完整性。
  5. 比特级操作
     + 难点：比特级数据的存取和对齐处理复杂。
     + 解决方案：支持BIT单位的长度定义，自动处理跨字节比特存取和对齐，确保数据边界正确。
  6. 值转换处理
     + 难点：数据转换逻辑多样，传统方法难以统一处理。
     + 解决方案：通过convert属性支持复杂的数学表达式、条件判断和函数调用，满足各种转换需求。
  7. 可选参数
     + 难点：协议中的可选字段处理不灵活。
     + 解决方案：引入optional属性，动态控制字段存在性，支持默认值和条件判断，实现协议的向前向后兼容。

## 5. 应用场景

1. 卫星指令系统

* 遥测指令生成：自动采集多种遥测参数，支持物理量单位转换、数据压缩和校验，确保传输可靠性。
* 遥控指令解析：多级指令结构解析，支持指令合法性校验、确认和重传机制，提供实时状态反馈。
* 状态监控数据处理：实时监控卫星各系统状态，支持阈值告警、趋势分析和异常自动处理。

1. 地面站系统

* 指令封装：支持多种指令类型的标准化封装，提供指令模板管理和参数自动校验。
* 数据解析：自动识别和解析多协议格式，支持数据重组、丢包处理和可视化展示。
* 实时控制：实现毫秒级指令响应，支持多任务并行处理和紧急控制优先机制。

1. 测试验证系统

* 协议一致性测试：自动化生成测试用例，覆盖协议各字段的边界值和异常场景。
* 性能测试：支持高并发指令处理测试，评估数据吞吐量、时延和资源占用。
* 可靠性验证：进行长期稳定性测试、故障注入和恢复测试，确保系统在极限条件下的表现。

## 6. 创新点

1. 协议定义创新

* 分层设计：支持任意深度的协议嵌套，各层级独立且完整，模块化设计提高复用性。
* 动态配置：运行时协议参数动态调整，支持协议版本平滑升级。
* 灵活扩展：支持新协议类型快速接入，提供标准化扩展接口，兼容第三方协议格式。

1. 实现方法创新

* 自动化处理：协议解析代码自动生成，测试用例自动构建，配置自动校验和加载。
* 智能计算：优化数据压缩和校验算法，智能选择资源调度策略，自动诊断异常。
* 可靠性保障：多重数据校验机制，容错和恢复处理，安全加密保护。

1. 应用模式创新

* 场景适应：支持多种部署方式，适应不同硬件平台和操作系统，提供云端和容器化部署。
* 快速部署：一键式安装，配置模板快速导入，自动化环境检测。
* 维护简便：可视化运维界面，远程监控和管理，故障自动报警。

## 8. 结论

本发明通过创新的协议设计和实现方法，有效解决了卫星通信领域中的数据处理问题，显著提升了卫星指令处理的效率、可靠性和扩展性。该方案已在多个实际项目中得到验证，取得了良好的应用效果，具备广泛的市场前景和应用价值。

## 7. 附图说明

## 9. 参考文献

1. [相关标准文献]
2. [技术规范文档]
3. [研究论文引用]