baidu

bidl2sl Manual

bidl2sl 使用手册

liuxupeng
2012/10/17

目录

bidl2sl 使用手册

1 介绍

1.1 背景

BGCC 使用 BIDL 语言来定义 BGCC 客户程序与服务器程序之间的通信接口。工具 bidl2sl(bidl to special language)负责解析 BIDL 文件，将 BIDL 文件中定义的接口和类型翻译为指定语言的源代码。此源码被应用于采用指定语言编定的客户端和服务器端的通信过程中。目前 bidl2sl 仅支持生成 C++和 Java 源代码。

1.2 内容

本手册涉及的内容包括 bidl2sl 选项使用说明、BIDL 到 C++语言的映射规则、BIDL 到 Java 语言的映射规则。

2 术语

- BIDL: Baidu Interface Definition Language。百度接口定义语言。
- BGCC: Baidu General Communication Component。 百度通用通讯组件。
- bidl2sl: BIDL to Special Language。BIDL 语言解析器，将接口定义翻译成特定语言的源代码。

3 工具选项

本节详细描述了 bidl2sl 各选项的含义。

```
./bidl2sl -h
Usage: bidl2sl [options] files...
Options:
-h, --help                  Show this messages.
-v, --version               Show version.
-I DIR, --include DIR       Append DIR in the include bidl file search path.
-O DIR, --output DIR        Change output directory to DIR.
                                (default: ./output-LANG)
```

```
-g LANG, --gen LANG      Generate source code with LANG language.
                             (default: cpp)
                             (support: cpp java)
-d, --debug              Print debug message to stderr.
```

3.1 –h, --help

-h, --help 用来显示帮助信息。用法示例如下：

```
bidl2sl -h
bidl2sl --help
```

3.2 -v, --version

-v, --version 用来显示版本信息。用法示例如下：

```
bidl2sl -v
bidl2sl --version
```

3.3 -I, --include

-I, --include 用来将指定目录添加到 BIDL 文件查找路径中。用法示例如下：

假如有如下目录结构。math.bidl 引用了 address.bidl，那么在编译 math.bidl 时，需要把 address.bidl 所在目录添加到 BIDL 文件查找路径中。address.bidl 与 math.bidl 内容见附录。

```
.
|-- address.bidl
`-- Math
    `-- math.bidl
```

```
cd Math
bidl2sl -I .. math.bidl
```

3.4 -O, --output

-O, --output 用来指定源码的输出目录。有未指定该选项时，将在当前目录下生成子目录 output-xxx，用于统一放置源码。xxx 表示生成的语言名称，如 cpp, java。用法示例如下：

```
bidl2sl -O ../cpp math.bidl
```

3.5 -g, --gen

-g, --gen 用来指定输出何种语言的源码。现在仅支持生成 C++源码和 Java 源码，默认情况下输出 C++源码。用法示例如下：

```
bidl2sl -g cpp math.bidl
bidl2sl -g java math.bidl
```

3.6 -d, --debug

-d, --debug 用于显示 bidl2sl 解析 BIDL 文件时生成的结构信息到标准输出。用法示例如下：

```
bidl2sl -d math.bidl
```

3.7 *的使用

bidl2sl 工具允许一次指定多个 BIDL 文件。示例如下：

```
bidl2sl *.bidl
```

4 Cpp 生成规则

本节介绍 BIDL 文件的各部分到 Cpp 源码的生成规则。

4.1 文件生成

每个 BIDL 文件 xxx.bidl 经 bidl2sl 编译后，均将生成一个头文件和一个源文件，分别为 xxx.h 和 xxx.cpp。如下所示：

```
bidl2sl math.bidl
```

上述命令执行完毕后，生成如下目录结构。

```
output-cpp/
|-- math.cpp
`-- math.h
```

4.2 标识符

一个 BIDL 标识符在映射为 Cpp 标识符时，分为两种情况。

a. BIDL 标识符将原样映射到 Cpp 标识符。例如 BIDL 中的常量 a 映射为 Cpp 常量后仍为 a；
b. BIDL 标识符将映射为多个 Cpp 标识符。例如 BIDL 接口在映射为 Cpp 代码时，将生成可被客户调用的代理和可被服务框架调用的存根，分别有代理标识符 xxxProxy 和处理器标识符 xxxProcessor。

4.3 基本类型

BIDL 基本类型映射为 Cpp 类型，请参见下表：

boolean	bool	布尔值

int8	int8_t	8 位有符号整数
int16	int16_t	16 位有符号整数
int32	int32_t	32 位有符号整数
int64	int64_t	64 位有符号整数
float	float	浮点数
string	std::string	字符串
binary	std::string	二进制

4.4 容器类型

BIDL 容器类型映射为 Cpp 类型，请参见下表：

sequence	std::vector	序列
set	std::set	集合
map	std::map	映射

4.5 typedef

BIDL 文件中的 typedef 语句直接映射为 C++语言中的 typedef 语句。示例如下：

BIDL typedef

```
typedef boolean b_t;
```

C++ typedef

```
typedef bool b_t;
```

4.6 enum

BIDL 文件中的 enum 被映射为一个同名的 C++类，该类对其内部定义的枚举值进行包装，对外提供枚举值的获取、枚举值描述的获取、一系列比较运算符的重载及输出运算符的重载。示例如下：

BIDL enum Color

```
enum Color {
    RED,
    GREEN,
    BLUE
}
```

在 Cpp 头文件中，bidl2sl 为 BIDL enum Color 产生如下定义:

```
class Color {
```

```
public:
    enum {
        RED = 0,
        GREEN = 1,
        BLUE = 2
    };

  Color();
  Color(int32_t value);
  int32_t get_value() const;
  std::string get_desc() const;
  bool operator==(int32_t value) const;
  bool operator!=(int32_t value) const;
  bool operator< (int32_t value) const;
  bool operator!=(const Color& rhs) const;
  bool operator==(const Color& rhs) const;
  bool operator< (const Color& rhs) const;
private:
  int32_t _value;

  static const struct desc_t {
      int32_t value;
      char* desc;
  } descs[];
};
bool operator==(int32_t value, const Color&);
bool operator!=(int32_t value, const Color&);
bool operator< (int32_t value, const Color&);
std::ostream& operator<<(std::ostream& o, const Color&);
```

4.7 struct

BIDL 文件中的 struct 被映射为一个同名的 C++类。对于 BIDL struct 中定义的每一个成员，在该类中都有一个对应的公开的 C++类成员。除了构造函数与析构函数外，生成的 C++类中还包括三个比较运算符，一个 write 函数——用于将本类成员打包并通过参数 proto 发送，两个 read 函数——其中一个 read 函数直接从参数 proto 中读取数据，另一个 read 函数从数据缓冲区 request 中读取数据。read 函数除了读取数据外，还负责对数据进行解包并还原类成员。注意：write 与 read 函数由 BGCC 运行时调用，外部程序不要对其进行直接调用。

BIDL struct Person

```
struct Person {
```

```
        string name;
        int32 age;
}
```

在 Cpp 头文件中，bidl2sl 为 BIDL struct Person 产生如下定义:

```
class Person {
public:
    Person();
    ~Person();

    std::string name;
    int32_t age;

    bool operator==(const Person& rhs) const;
    bool operator!=(const Person& rhs) const;
    bool operator< (const Person& rhs) const;
    int32_t read(bgcc::SharedPointer<bgcc::IProtocol> proto);
    int32_t read(bgcc::SharedPointer<bgcc::IProtocol> proto, char* request, int32_t request_len);
    int32_t write(bgcc::SharedPointer<bgcc::IProtocol> proto) const;
};
```

4.8 const

BIDL 文件中的 const 语句直接映射为 C++语言中的 const 常量定义语句。示例如下：

BIDL const

```
const int8 i8 = 8;
const boolean ok = true;
const sequence<string> names = ["jack", "mery"];
```

在 Cpp 头文件中，bidl2sl 为 BIDL const 产生如下定义:

```
extern const int8_t i8;
extern const bool ok;
extern const std::vector<std::string> names;
```

4.9 class

BIDL class 是 BIDL 接口定义文件中最关键的部分。一个 BIDL class 对应生成的 C++源文件中的 7 部分，包括接口类、参数序列化类、参数反序列化类、结果序列化类、结果反序列化类、代理类以及处理器类。其中，接口类定义了服务器逻辑必须实现的接口；参数序列化类定义了在客户

端调用接口时，接口参数序列化的过程；参数反序列化类定义了服务端接收到调用请求时，如何对接口参数进行反序列化的过程；结果序列化类定义了服务器端对函数调用结果进行打包的过程；结果反序列化类定义了客户端收到函数调用结果后，对其进行拆包的过程；代理类负责与客户程序进行交互，将客户程序的调用转化为对服务器功能的 RPC 调用，并负责将调用结果传递给客户程序使用；处理器类负责将服务逻辑实现加入到 BGCC 服务器框架中，以对外提供服务。

BIDL class

```
class Math {
    void add(int32 a, int32 b, [out] int32 sum);
}
```

C++ 接口类

```
class Math : public bgcc::Shareable {
    public:
        virtual ~Math() { }
        virtual void add(
            int32_t a,
            int32_t b,
            int32_t & sum,
            const std::map<std::string, std::string>& ctx) = 0;
    };
```

C++参数序列化类

```
class Math_add_pargs {
    public:
        virtual ~Math_add_pargs();
        const int32_t* a;
        const int32_t* b;
        int32_t write(bgcc::SharedPointer<bgcc::IProtocol> proto) const;
    };
```

C++参数反序列化类

```
class Math_add_args {
public:
    Math_add_args();
    virtual ~Math_add_args();
    int32_t a;
```

```
        int32_t b;
        bool operator==(const Math_add_args& rhs) const;
        bool operator!=(const Math_add_args& rhs) const;
        bool operator< (const Math_add_args& rhs) const;
        int32_t read(bgcc::SharedPointer<bgcc::IProtocol> proto, char* request, int32_t request_len);
};
```

C++结果序列化类

```
class Math_add_result {
    public:
        Math_add_result();
        virtual ~Math_add_result();
        int32_t sum;
        bool operator==(const Math_add_result& rhs) const;
        bool operator!=(const Math_add_result& rhs) const;
        bool operator< (const Math_add_result& rhs) const;
        int32_t write(bgcc::SharedPointer<bgcc::IProtocol> proto) const;
    };
```

结果反序列化类

```
class Math_add_presult {
    public:
        virtual ~Math_add_presult();
        int32_t* sum;
        int32_t read(bgcc::SharedPointer<bgcc::IProtocol> proto) const;
    };
```

C++代理类

```
class MathProxy : public bgcc::BaseProxy {
    public:
        MathProxy(
            bgcc::ServerInfo serverinfo,
            int32_t nprotocols = 1,
            bgcc::ServiceManager* mgr = NULL,
            int32_t tryCount = 5,
            int32_t tryInterval = 500);
        MathProxy(const std::string& proxy_name);
        void add(
            int32_t a,
            int32_t b,
```

```
            int32_t& sum,
            bool last = false);
    private:
        void send_add(
            const int32_t& a,
            const int32_t& b,
            int32_t seqid,
            bgcc::SharedPointer<bgcc::IProtocol> proto);
        void recv_add(
            int32_t& sum,
            bgcc::SharedPointer<bgcc::IProtocol> proto);
    private:
        std::string _proxy_name;
        bool _use_existing_socket;
    };
```

C++处理器类

```
class MathProcessor : public bgcc::BaseProcessor {
    public:
        MathProcessor(bgcc::SharedPointer<Math> intf);
        virtual ~MathProcessor(){ }
        virtual int32_t process(
            char* request,
            int32_t request_len,
            bgcc::SharedPointer<bgcc::IProtocol> proto);
        virtual std::string get_name() const;
    protected:
        virtual int32_t do_function__(
            char* request,
            int32_t request_len,
            bgcc::SharedPointer<bgcc::IProtocol> proto,
            const std::string& fname, int32_t seqid);
        bgcc::SharedPointer<Math> __intf;
    private:
        int32_t do_add(
            char* request,
            int32_t request_len,
            bgcc::SharedPointer<bgcc::IProtocol> proto,
            int32_t seqid);
        typedef int32_t (MathProcessor::* do_function_ptr)(
            char* request,
            int32_t request_len,
```

```
                bgcc::SharedPointer<bgcc::IProtocol> proto,
                int32_t seqid);
        std::map<std::string, do_function_ptr> __fun_map;
    };
```

4.10 namespace

BIDL 文件中的 namespace 语句直接映射为 C++语言中的 namespace 语句。示例如下：

BIDL namespace

```
namespace ims {
    typedef boolean b_t;
}
```

在 Cpp 头文件中，bidl2sl 为 BIDL namespace 产生如下定义:

```
namespace ims {
    typedef bool b_t;
}
```

5 Java 生成规则

5.1 文件生成

一个 BIDL 文件在映射为 Java 源代码时，可能会生成多个 Java 源文件。分为以下情况：

a. 一个 BIDL const 语句会对应生成一个 Java 源文件；
b. 一个 BIDL enum 语句会对应生成两个 Java 源文件；
c. 一个 BIDL struct 语句会对应生成两个 Java 源文件；
d. 一个 BIDL class 语句会对应生成一个 Java 源文件；
e. 一个 BIDL namespace 语句会对应生成一系列目录，用于表示 Java 语言的名称空间的概念。

5.2 标识符

一个 BIDL 标识符在映射为 Java 标识符时，分为两种情况。

c. BIDL 标识符将原样映射到 Java 标识符。例如 BIDL 中的常量 a 映射为 Java 常量后仍为 a；
d. BIDL 标识符将映射为多个 Java 标识符。例如 BIDL 接口在映射为 Java 代码时，将生成可被客户调用的代理和可被服务框架调用的存根，分别有代理标识符 XxxProxy 和处理器标识符 XxxProcessor。

5.3 基本类型

BIDL 基本类型映射为 Java 类型，请参见下表：

	in 型参数	out 型参数	all 型参数	描述
boolean	boolean	BooleanHolder	BooleanHolder	布尔值
int8	byte	ByteHolder	ByteHolder	8 位有符号整数
int16	short	ShortHolder	ShortHolder	16 位有符号整数
int32	int	IntHolder	IntHolder	32 位有符号整数
int64	long	LongHolder	LongHolder	64 位有符号整数
float	float	FloatHolder	FloatHolder	浮点数
string	java.lang.String	StringHolder	StringHolder	字符串
binary	java.lang.String	StringHolder	StringHolder	二进制

5.4 容器类型

BIDL 容器类型映射为 Java 类型，请参见下表：

	in 型参数	out 型参数	all 型参数	描述
sequence	java.util.List	Holder<List>	Holder<List>	序列
set	java.util.Set	Holder<Set>	Holder<Set>	集合
map	java.util.Map	Holder<Map>	Holder<Map>	映射

5.5 typedef

BIDL typedef 不对应生成 Java 语言源程序。

5.6 enum

BIDL 文件中的 enum xxx 被映射为两个 Java 源程序：一个是 Java Enum xxx，该 Java Enum 提供了枚举值定义，对外提供枚举值的获取、枚举值描述的获取以及将整数值转换为该 Java Enum 类型值的转换函数；一个是叫做 xxxHolder 的 Java Class，这是 Java Enum xxx 的 holder 类。示例如下：

BIDL enum Color

```
enum Color {
    RED,
    GREEN,
    BLUE
}
```

在 Java 源文件 Color.java 中，bidl2sl 为 BIDL enum Color 产生的 Java Enum 定义如下:

```
public enum Color {
        RED(0, "Color::RED"),
        GREEN(1, "Color::GREEN"),
        BLUE(2, "Color::BLUE");
        private final int value;
        private final java.lang.String desc;
        private Color(int value, java.lang.String desc) {
                this.value = value;
                this.desc = desc;
        }
        public int getValue() {
                return value;
        }
        public java.lang.String getDescription() {
                return desc;
        }
        public static Color findByValue(int value) {
                for (Color temp : Color.values()) {
                        if (value == temp.getValue()) {
                                return temp;
                        }
                }
         }
}
```

在 Java 源文件 ColorHolder.java 中，bidl2sl 为 BIDL enum Color 产生的 Holder 定义如下:

```
public final class ColorHolder {
        public ColorHolder() {
        }
        public ColorHolder(Color value) {
                this.value = value;
        }
        public Color value;
}
```

5.7 struct

BIDL 文件中的 struct xxx 被映射为两个 Java 源程序：

一个是 Java Class xxx。对于 BIDL struct 中定义的每一个成员，在该类中都有一个对应的受保护的 C++类成员及其 getter 和 setter。该类还定义有一个 write 函数——用于将本类成员打包并

通过参数 protocol 发送，两个 read 函数——其中一个 read 函数直接从参数 protocol 中读取数据，另一个 read 函数从数据缓冲区 request 中读取数据。read 函数除了读取数据外，还负责对数据进行解包并还原类成员。注意：write 与 read 函数由 BGCC 运行时调用，外部程序不要对其进行直接调用；

一个是叫做 xxxHolder 的 Java Class，这是 Java Class xxx 的 holder 类。示例如下：

BIDL struct Person

```
struct Person {
        string name;
        int32 age;
}
```

在 Java 源文件 Person.java 中，bidl2sl 为 BIDL Class Person 产生的 Java Class 定义如下:

```
import bgcc.*;
import java.util.*;
public class Person {
    protected StringHolder name;
    protected IntHolder age;
    public Person() {
        this.name = new StringHolder();
        this.age = new IntHolder();
    }
    public String getName() {
        return this.name.value;
    }
    public void setName(String name) {
        this.name.value = name;
    }
    public int getAge() {
        return this.age.value;
    }
    public void setAge(int age) {
        this.age.value = age;
    }
    public int write(Protocol protocol) {
        int ret = 0;
        ret = protocol.writeStructBegin("Person");
        if (ret < 0) { return ret; }
        ret = protocol.writeFieldBegin("name", DataType.IDSTR, 1);
```

```
            if (ret < 0) { return ret; }
            ret = protocol.writeString(this.name.value);
            if (ret < 0) { return ret; }
            ret = protocol.writeFieldEnd();
            if (ret < 0) { return ret; }
            ret = protocol.writeFieldBegin("age", DataType.IDINT32, 2);
            if (ret < 0) { return ret; }
            ret = protocol.writeInt(this.age.value);
            if (ret < 0) { return ret; }
            ret = protocol.writeFieldEnd();
            if (ret < 0) { return ret; }
            ret = protocol.writeFieldStop();
            if (ret < 0) { return ret; }
            return protocol.writeStructEnd();
        }
        public int read(Protocol protocol) {
            int ret = 0;
            StringHolder fname = new StringHolder();
            DataTypeHolder ftype = new DataTypeHolder();
            IntHolder fid = new IntHolder();
            ret = protocol.readStructBegin(fname);
            if (ret < 0) { return ret; }
            while (true) {
                ret = protocol.readFieldBegin(fname, ftype, fid);
                if (ret < 0) { return ret; }
                if (ftype.value == DataType.IDSTOP) {
                    break;
                }
                switch(fid.value) {
                case 1:
                    ret = protocol.readString(this.name);
                    if (ret < 0) { return ret; }
                    break;
                case 2:
                    ret = protocol.readInt(this.age);
                    if (ret < 0) { return ret; }
                    break;
                }
            }
            return protocol.readFieldEnd();
        }
        public int read(Protocol protocol, long request, int request_len) {
            int ret = 0;
            int nread = 0;
```

```
            StringHolder fname = new StringHolder();
            DataTypeHolder ftype = new DataTypeHolder();
            IntHolder fid = new IntHolder();
            ret = protocol.readStructBegin(request + nread, request_len - nread, fname);
            if (ret < 0) { return ret; }
            nread += ret;
            while (true) {
                ret = protocol.readFieldBegin(request + nread, request_len - nread, fname, ftype, fid);
                if (ret < 0) { return ret; }
                nread += ret;
                if (ftype.value == DataType.IDSTOP) {
                    break;
                }
                switch(fid.value) {
                case 1:
                    ret = protocol.readString(request + nread, request_len - nread, this.name);
                    if (ret < 0) { return ret; }
                    nread += ret;
                    break;
                case 2:
                    ret = protocol.readInt(request + nread, request_len - nread, this.age);
                    if (ret < 0) { return ret; }
                    nread += ret;
                    break;
                }
            }
            ret = protocol.readFieldEnd();
            if (ret < 0) { return ret; }
            nread += ret;
            return nread;
        }
    }
```

5.8 const

BIDL 文件中的 const 语句映射为 Java 语言的一个类。示例如下：

BIDL const

```
const boolean be_used = false;
```

在 Java 源文件中，bidl2sl 为 BIDL const 产生如下定义:

```
public final class be_used {
```

```
        public static final boolean value = false;
    }
```

5.9 class

BIDL class 是 BIDL 接口定义文件中最关键的部分。一个 BIDL class xxx 对应生成一个 Java 源文件 xxx.java，其中包含 7 个部分，分别是接口类、参数序列化类、参数反序列化类、结果序列化类、结果反序列化类、代理类以处理器类。其中，接口类定义了服务器逻辑必须实现的接口；参数序列化类定义了在客户端调用接口时，接口参数序列化的过程；参数反序列化类定义了服务端接收到调用请求时，如何对接口参数进行反序列化的过程；结果序列化类定义了服务器端对函数调用结果进行打包的过程；结果反序列化类定义了客户端收到函数调用结果后，对其进行拆包的过程；代理类负责与客户程序进行交互，将客户程序的调用转化为对服务器功能的 RPC 调用，并负责将调用结果传递给客户程序使用；处理器类负责将服务逻辑实现加入到 BGCC 服务器框架中，以对外提供服务。

BIDL class

```
class Math {
    void add(int32 a, int32 b, [out] int32 sum);
}
```

Java 源文件

```
import bgcc.*;
import java.util.*;
public class Math {
    public interface Intf {
        public void add(int a, int b, IntHolder sum);
    }
    public static class add_args {
        public IntHolder a;
        public IntHolder b;
        public add_args() {
            this.a = new IntHolder();
            this.b = new IntHolder();
        }
        public int write(Protocol protocol) {
            int ret = 0;
            ret = protocol.writeStructBegin("add_args");
            if (ret < 0) { return ret; }
```

```
            ret = protocol.writeFieldBegin("a", DataType.IDINT32, 1);
            if (ret < 0) { return ret; }
            ret = protocol.writeInt(this.a.value);
            if (ret < 0) { return ret; }
            ret = protocol.writeFieldEnd();
            if (ret < 0) { return ret; }
            ret = protocol.writeFieldBegin("b", DataType.IDINT32, 2);
            if (ret < 0) { return ret; }
            ret = protocol.writeInt(this.b.value);
            if (ret < 0) { return ret; }
            ret = protocol.writeFieldEnd();
            if (ret < 0) { return ret; }
            ret = protocol.writeFieldStop();
            if (ret < 0) { return ret; }
            return protocol.writeStructEnd();
        }
        public int read(Protocol protocol) {
            int ret = 0;
            StringHolder fname = new StringHolder();
            DataTypeHolder ftype = new DataTypeHolder();
            IntHolder fid = new IntHolder();
            ret = protocol.readStructBegin(fname);
            if (ret < 0) { return ret; }
            while (true) {
                ret = protocol.readFieldBegin(fname, ftype, fid);
                if (ret < 0) { return ret; }
                if (ftype.value == DataType.IDSTOP) {
                    break;
                }
                switch(fid.value) {
                case 1:
                    ret = protocol.readInt(this.a);
                    if (ret < 0) { return ret; }
                    break;
                case 2:
                    ret = protocol.readInt(this.b);
                    if (ret < 0) { return ret; }
                    break;
                }
                ret = protocol.readFieldEnd();
                if (ret < 0) { return ret; }
            }
            return protocol.readStructEnd();
        }
```

```
        public int read(Protocol protocol, long request, int request_len) {
            int ret = 0;
            int nread = 0;
            StringHolder fname = new StringHolder();
            DataTypeHolder ftype = new DataTypeHolder();
            IntHolder fid = new IntHolder();
            ret = protocol.readStructBegin(request + nread, request_len - nread, fname);
            if (ret < 0) { return ret; }
            nread += ret;
            while (true) {
                ret = protocol.readFieldBegin(request + nread, request_len - nread, fname,
ftype, fid);
                if (ret < 0) { return ret; }
                nread += ret;
                if (ftype.value == DataType.IDSTOP) {
                    break;
                }
                switch(fid.value) {
                case 1:
                    ret = protocol.readInt(request + nread, request_len - nread, this.a);
                    if (ret < 0) { return ret; }
                    nread += ret;
                    break;
                case 2:
                    ret = protocol.readInt(request + nread, request_len - nread, this.b);
                    if (ret < 0) { return ret; }
                    nread += ret;
                    break;
                }
                ret = protocol.readFieldEnd();
                if (ret < 0) { return ret; }
                nread += ret;
            }
            ret = protocol.readStructEnd();
            if (ret < 0) { return ret; }
            nread += ret;
            return nread;
        }
    }

    public static class add_result {
        public IntHolder sum;
        public add_result() {
            this.sum = new IntHolder();
```

```
    }
    public int write(Protocol protocol) {
        int ret = 0;
        ret = protocol.writeStructBegin("add_result");
        if (ret < 0) { return ret; }
        ret = protocol.writeFieldBegin("sum", DataType.IDINT32, 3);
        if (ret < 0) { return ret; }
        ret = protocol.writeInt(this.sum.value);
        if (ret < 0) { return ret; }
        ret = protocol.writeFieldEnd();
        if (ret < 0) { return ret; }
        ret = protocol.writeFieldStop();
        if (ret < 0) { return ret; }
        return protocol.writeStructEnd();
    }
    public int read(Protocol protocol) {
        int ret = 0;
        StringHolder fname = new StringHolder();
        DataTypeHolder ftype = new DataTypeHolder();
        IntHolder fid = new IntHolder();
        ret = protocol.readStructBegin(fname);
        if (ret < 0) { return ret; }
        while (true) {
            ret = protocol.readFieldBegin(fname, ftype, fid);
            if (ret < 0) { return ret; }
            if (ftype.value == DataType.IDSTOP) {
                break;
            }
            switch(fid.value) {
            case 3:
                ret = protocol.readInt(this.sum);
                if (ret < 0) { return ret; }
                break;
            }
            ret = protocol.readFieldEnd();
            if (ret < 0) { return ret; }
        }
        return protocol.readStructEnd();
    }
    public int read(Protocol protocol, long request, int request_len) {
        int ret = 0;
        int nread = 0;
        StringHolder fname = new StringHolder();
        DataTypeHolder ftype = new DataTypeHolder();
```

```
                IntHolder fid = new IntHolder();
                ret = protocol.readStructBegin(request + nread, request_len - nread, fname);
                if (ret < 0) { return ret; }
                nread += ret;
                while (true) {
                    ret = protocol.readFieldBegin(request + nread, request_len - nread, fname,
ftype, fid);
                    if (ret < 0) { return ret; }
                    nread += ret;
                    if (ftype.value == DataType.IDSTOP) {
                        break;
                    }
                    switch(fid.value) {
                    case 3:
                        ret = protocol.readInt(request + nread, request_len - nread, this.sum);
                        if (ret < 0) { return ret; }
                        nread += ret;
                        break;
                    }
                    ret = protocol.readFieldEnd();
                    if (ret < 0) { return ret; }
                }
                return protocol.readStructEnd();
            }
        }

        public static class Proxy extends BaseProxy {
            public Proxy(ServerInfo serverInfo) {
                super(serverInfo);
                setWhoAmI("global.math.Math");
            }
            public void add(int a, int b, IntHolder sum) {
                this.add(a, b, sum, false);
            }
            public void add(int a, int b, IntHolder sum, boolean last) {
                IntHolder seqHolder = new IntHolder(0);
                Protocol protocol = getProtocol();
                if (protocol == null) { return ;}
                int ret = getTicketId("add", seqHolder, last, protocol, protocol);
                if (ret < 0) { return ;}
                send_add(a, b, seqHolder.value, protocol);
                recv_add(sum, protocol);
                putProtocol(protocol);
            }
```

```
        void send_add(int a, int b, int seqid, Protocol protocol) {
            protocol.writeMessageBegin(getWhoAmI(), "add", bgcc.MessageType.CALL,
seqid);
            protocol.writeString(getName());
            add_args args = new add_args();
            args.a.value = a;
            args.b.value = b;
            args.write(protocol);
            protocol.writeMessageEnd();
        }
        void recv_add(IntHolder sum, Protocol protocol) {
            StringHolder fname = new StringHolder();
            IntHolder seqid = new IntHolder();
            MessageTypeHolder msgtype= new MessageTypeHolder();
            protocol.readMessageBegin(fname, msgtype, seqid);
            add_result _result = new add_result();
            _result.read(protocol);
            protocol.readMessageEnd();
            sum.value = _result.sum.value;
        }
    }
    public static class Processor extends BaseProcessor {
        public Processor(Intf intf) {
            _intf = intf;
            _processMap.put("add", new add());
        }
        public java.lang.String getName() {
            return "global.math.Math";
        }
        private Intf _intf;
        private class add implements ProcessFunction {
            public int process(long request, int request_len, Protocol out,
                java.lang.String fname, int seqid) {
                int ret;
                int nread = 0;
                StringHolder xxHolder = new StringHolder();
                ret = out.readString(request + nread, request_len - nread, xxHolder);
                if (ret < 0) { return ret; }
                nread += ret;
                add_args args = new add_args();
                ret = args.read(out, request + nread, request_len - nread);
                out.readMessageEnd();
                add_result result = new add_result();
                _intf.add(args.a.value, args.b.value, result.sum);
```

```
                        out.writeMessageBegin("global.math.Math", "add",
bgcc.MessageType.REPLY, 0);
                        result.write(out);
                        out.writeMessageEnd();
                        return 0;
                    }
                }
            }
        }
```

5.10 namespace

一个 BIDL namespace 语句会对应生成一系列目录，用于表示 Java 语言的名称空间。

BIDL namespace

```
namespace ims {
    const boolean be_used = false;
}
```

经 bidl2sl 编译后，生成的目录结构如下 ：

```
output-java/
`-- ims
    `-- be_used.java
```

6 附录

address.bidl

```
namespace dummy {
}
```

math.bidl

```
include "address.bidl"
```