Tce介绍

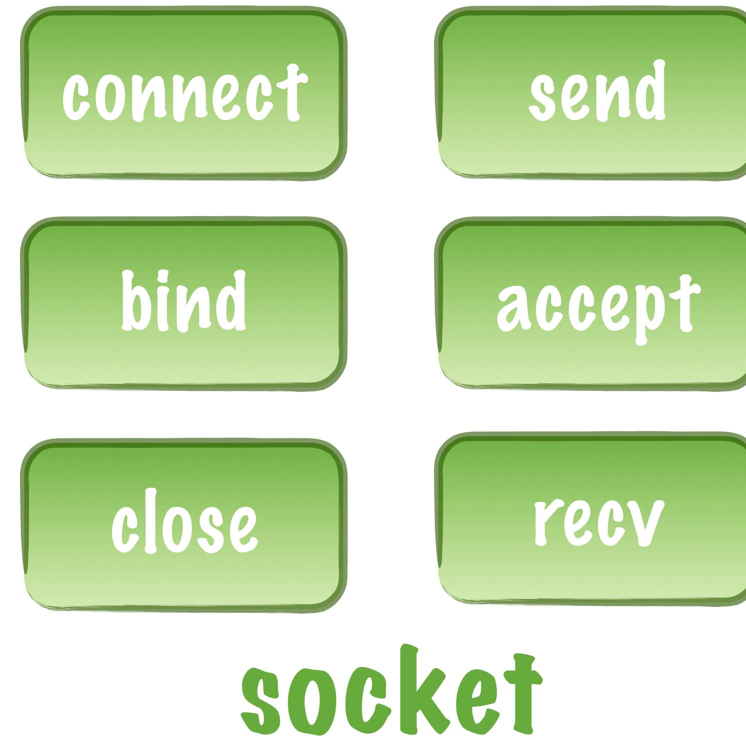
小型的Rpc通信框架库

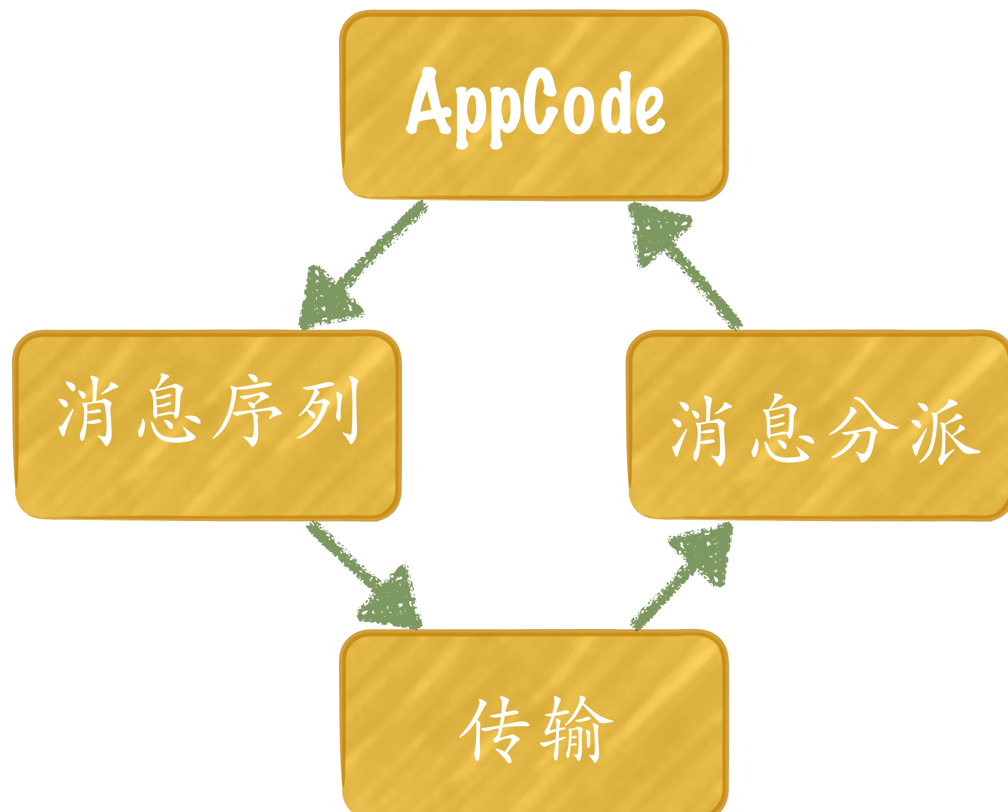
Tiny Communication Engine

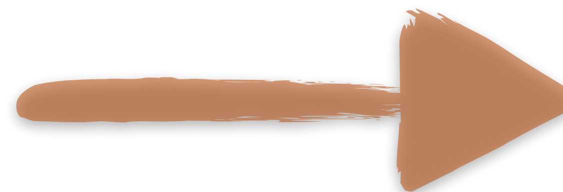
*

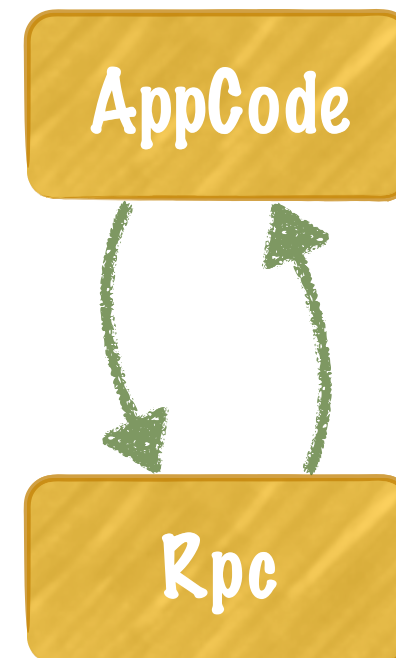
如何进行端到端交互

* socket
* web service
* websocket
* mq
* ...









java
c++
python
objc
actionscript
javascript

语言

系统

android
ios
linux
windows

通信方式

socket
websocket
http-xml
mq

idl 语言

school.idl

```
include <other.idl>
module{

    sequence<string> ids_t;
    struct  classInfo_t{
        string name;
        string teacher;
        int   elements;
    };
    sequence<classInfo_t>  classInfoList_t;
    dictionary<string,classInfo_t>  classInfoMap_t;
```

数据类型

```
    interface ISchool{
        void hello(string text);
        classInfoList_t getClassInfoList(string which);
    };

}
```

功能接口

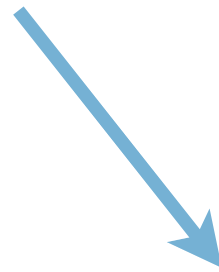
idl 语言 - 数据类型

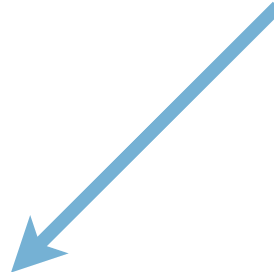
byte
short
int
long
float
double
string
bool

基础类型

复合类型

struct





容 器

sequence<T>
dictionary<T,V>

idl 语言规格

基础类型

IDL	size	java	c++	python	objc
byte	1	byte	uint8		
short	2	short	short		
int	4	int	int		
long	8	long	longlong		
float	4	float	float		
double	8	double	double		
string	n+4	String	std::string		
bool	1	boolean	bool		NSBoolean

idl 语言规格

复合类型-struct

IDL	java	c++	python	objc
struct	class	struct	class	interface

汽车对象的描述

```
struct Engine{
    ...
};

struct Door{
    ...
};

sequence<Door> DoorList;

struct Vehicle{
    string brand;
    int color;

    Engine engine;
    DoorList doors;
};
```

* 描述对象属性的集合
* 支持复合类型和容器的嵌套

idl 语言规格

容器 - sequence/dictionary

IDL	java	c++	python	objc
sequence	Vector.	std::vector	list	NSArray
dictionary	HashMap.	std::map	dict	NSDictionary

* 基础数据类型和容器可组装成复合数据类型
* 容器之间可以嵌套

idl 语言规格

对于字节流的特殊处理:

TCE	java	c++	python	objc
sequence<byte>	byte[]	std::vector<byte>	str	NSData

* sequence<byte> 应用于二进制数据

idl 语言规格

dictionary字典类型的特殊规定

* dictionary<K,V>的K必须是基础类型，复合类型和容器类型不能用作Key使用

K must be in [byte,int,float,double,long,string,bool,short]

```
struct S{};
sequence<K> A;
dictionary<k,v> K;

dictionary<S,v> obj;
dictionary<A,v> obj;
dictionary<K,v> obj;
```

idl 语言规格

接口 interface

school.idl

```
module{
    interface ISchool{
        void hello(string text);
        classInfoList_t getClassInfoList(string which);
    };
}
```

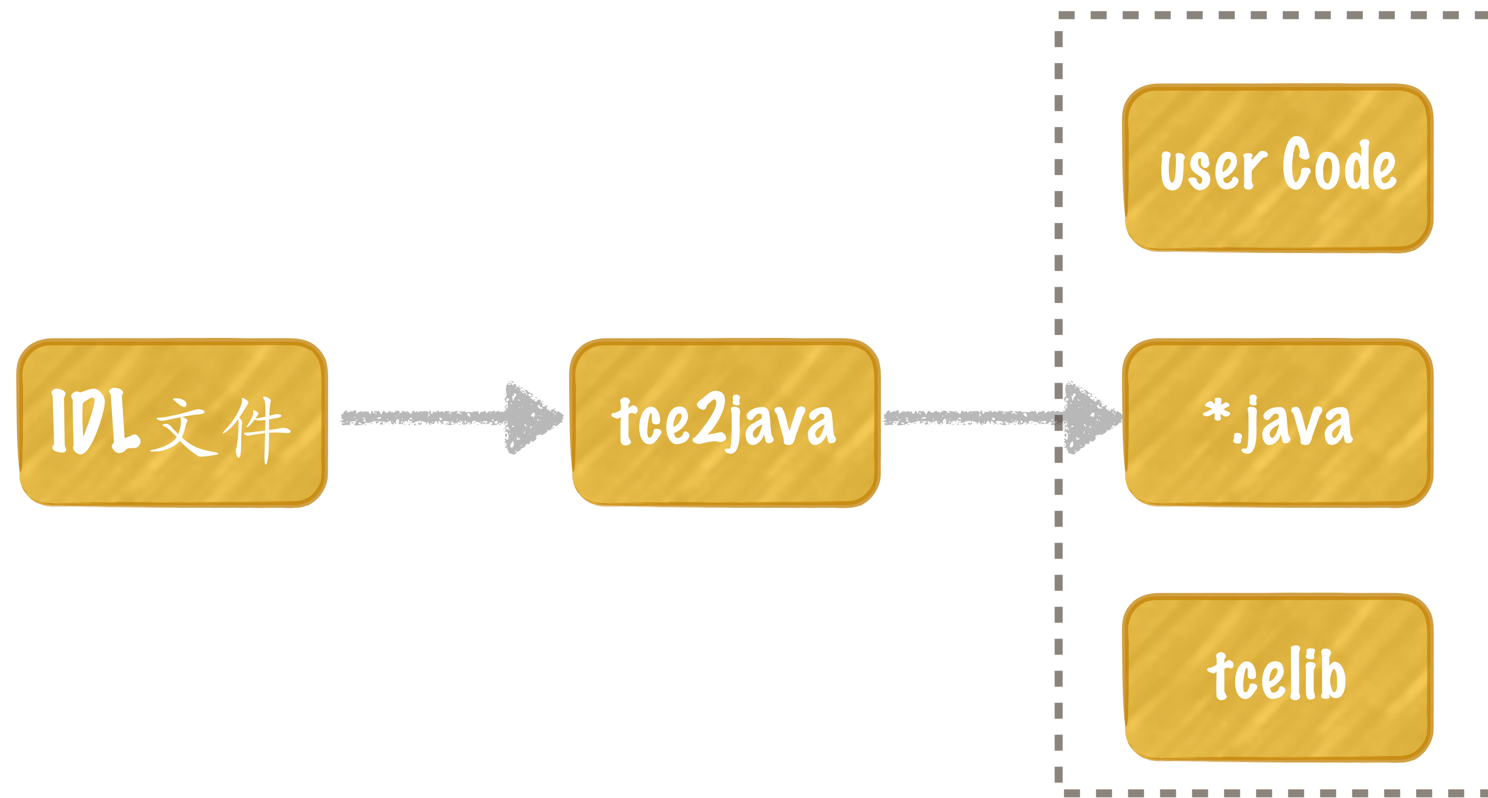
idl 语言规格

行注释 //

school.idl

```
module{
    interface ISchool{
        // void hello(string text);
        classInfoList_t getClassInfoList(string which);
    };
}
```

tce如何使用



* 接口文件将被翻译成不同语言的实现
* tce的应用都包含用户代码、接口实现代码和tce通信库

tcelib的组成

Commnicator

* 应用程序的通信管理器，负责管理本app内的与外部通信的适配器对象（Adapter）
* 管理和控制rpc的状态、请求分派
* ...

Adapter

* 通信适配器是Servant对象的容器，处理接收Rpc请求并分派到不同的Servant对象
* 通信适配器是Connection的容器，它可以包含不同的通信连接

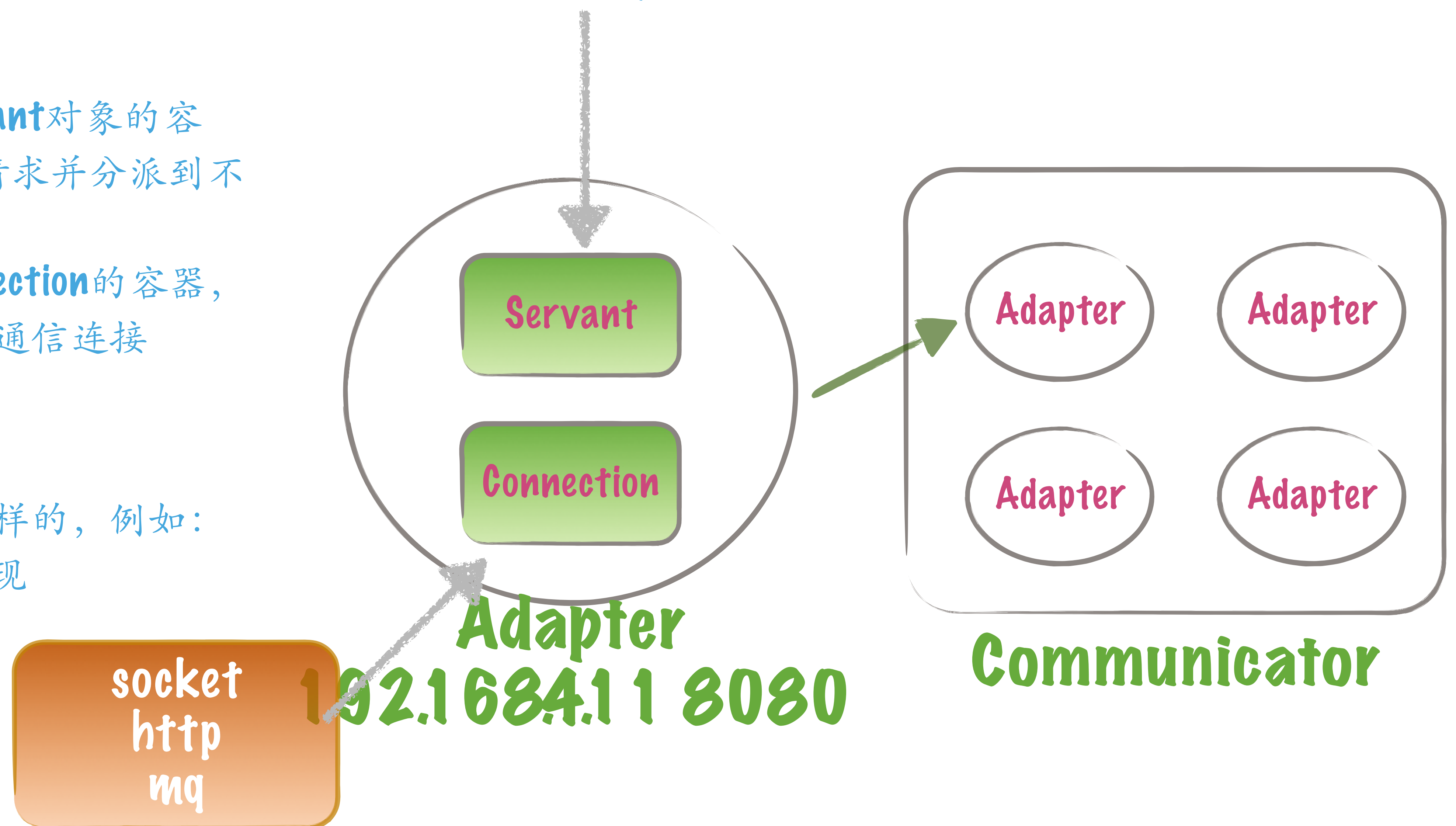
Connection

* 通信连接可以是多样的，例如：socket,http,mq的实现

Servant

* 服务接口的实现

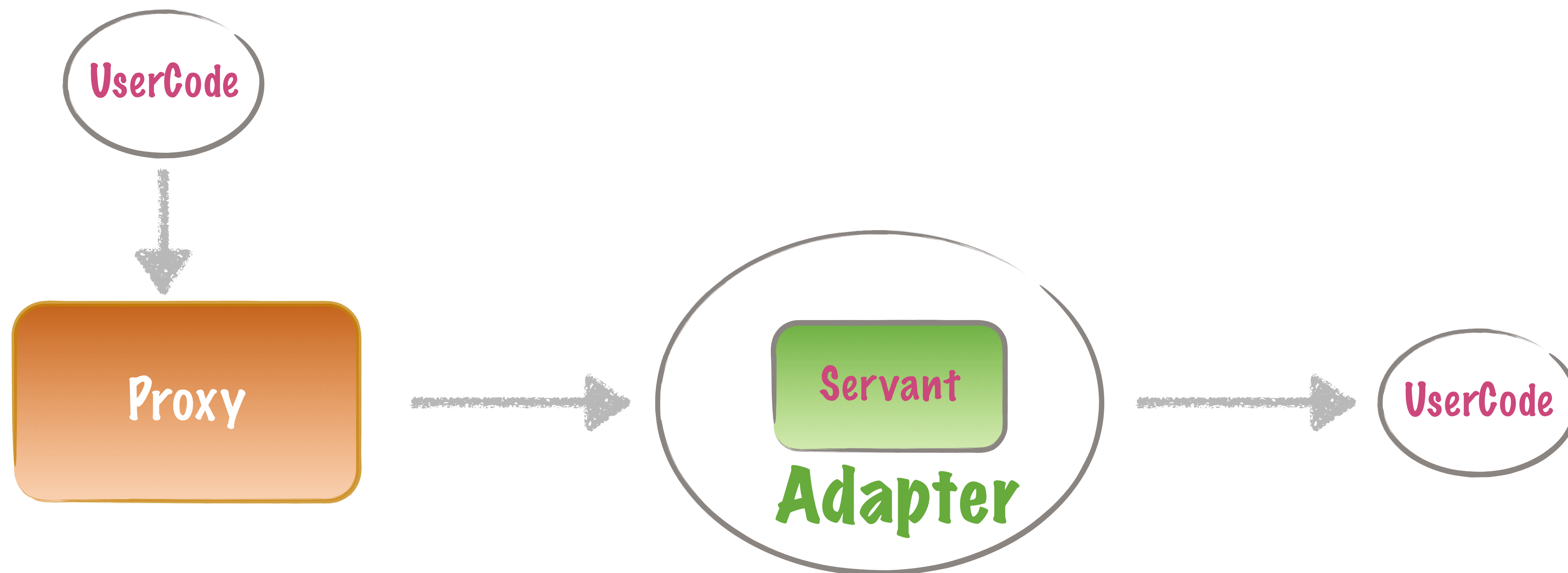
```
interface ISchool{
        void hello(string text);
        classInfoList_t getClassInfoList(string which);
    };
```



tcelib的组成

Proxy

* 列集了服务功能接口，是访问interface的客户端设施
* 使用proxy等同调用本地函数
* 由tce根据idl定义自动产生
* 完成通信和消息序列化工作
* 多种调用模式： 阻塞、异步、单向、超时



tce调用方式

* 阻塞调用

  最常见的调用方式，用户发起远程调用之后，等待结果返回，**tce**内部实现阻塞。这种编程接口使用简单，但需要消耗线程资源，故效率低

* 单向调用**(oneway)**

  单向意味着调用无返回消息，调用发起，即可返回。只有声明为**void**类型的接口函数支持单向模式

* 异步调用

  异步请求不会阻塞调用线程，通过设置回调来接收函数处理的结果。

* 超时调用

  与阻塞调用相似，同样阻塞调用线程，调用时可传入超时等待时间，如发生超时**tce**将传递异常到用户

tce for python

A. idl定义
B. 实现一个server
C. client访问server
D. 阻塞调用
E. 非阻塞调用
F. 超时

tce for python

idl定义

base.idl

```
module sns{

sequence<string> StringList_t;
sequence<int> IntList_t;

sequence<string> UserIdList_t;
sequence<string> SIDS_t;
dictionary<string,string> StrStr_t;
sequence<StrStr_t> StrStrList_t;

//interface and data modals definations

struct GeoPoint_t{
	float lon;
	float lat;
};

struct GeoSize_t{
	float cx;
	float cy;
};

struct GeoCircle_t{
	GeoPoint_t center;
	float radius;
};

struct GeoRect_t{
	float x;
	float y;
	float width;
	float height;
};

struct GpsInfo_t{
	GeoPoint_t loc;
	float speed;
	float direction;
	int   timesec;
};

struct LocationInfo_t{
	string      user_id;
	GpsInfo_t   gps;
	string      desc;
};
sequence<LocationInfo_t> LocationInfoList_t;


interface IBaseServer{
	int getServerTimestamp();
};


}
```

tce for python

idl定义

simple.idl

```
import base

module sns{


struct IoDetail_t{
        string name;
        int max;
        int min;
};

sequence<IoDetail_t> IoDetailList;
sequence<string> StrList;

struct TerminalInfo_t{
        string name;
        string address;
        IoDetailList ios;
};

struct CapacityInfo_t{
        string term;
        int flow;
        int ram;
        int max;
        string desc;
};

sequence<TerminalInfo_t> TerminalInfoList;
dictionary<string,CapacityInfo_t> CapacityInfoList;


interface ITerminal{
   void onNotifyMessage(string  notify);
};

interface ICtrlServer extends IBaseServer{
        string register(string user,string passwd );
        void start_bidirection();
        string  echo(string msg);
        void   show(StrList sids);
        CapacityInfo_t  getCapacityInfo(string termid);
        TerminalInfoList    getTerminals(string type);
        bool    save(string termid,CapacityInfoList
capacities);
        string  timeout(int waitsecs);
};


}
```

tce for python

实现server

定义servant

```
class ServerImpl(ICtrlServer):
    def __init__(self):
        ICtrlServer.__init__(self)
    def work_thread(self):
    def register(self,user,passwd,ctx):
    def start_bidirection(self,ctx):
    def show(self,sids,ctx):
    def save(self,termid,capacities,ctx):
    def timeout(self,waitsecs,ctx):
    def getCapacityInfo(self,termid,ctx):
```

初始化server

```
tce.RpcCommunicator.instance().init('server')                 #初始化通信器对象
ep = tce.RpcEndPoint(host='127.0.0.1',port=16005)             #创建通信端点对象
adapter = tce.RpcCommunicator.instance().createAdapter('first_server',ep) #创建通信适配器
servant = ServerImpl()                                        #实例化服务对象
adapter.addServant(servant)                                   #服务对象加入适配器
tce.RpcCommunicator.instance().waitForShutdown()              #进入服务循环
```

tce for python

client调用

定义回调接口

```
class TerminalImpl(ITerminal):
        def __init__(self):
                ITerminal.__init__(self)

        def onNotifyMessage(self,notify,ctx):
                print 'onNotifyMessage:',notify
```

接口调用

```
def call_twoway():
        prx.register('scott','1'*10)
        ids = range(10)
        prx.show(ids)
        cap = prx.getCapacityInfo("term_01")
        terms = prx.getTerminals('normal')
        caps ={}
        caps['speed'] =
CapacityInfo_t(term='term_01',flow=100)
        caps['times'] =
CapacityInfo_t(term='term_01',flow=200)
        r = prx.save('term_01',caps)
```

```
def getCapacityInfo_async_result(result,proxy):
   print 'async return:',result
   print 'proxy:',proxy

def call_async():
   prx.getCapacityInfo_async('test001',getCapacityInfo_async_res
   ult)
   tce.sleep(2)

def call_extras():
   print prx.show(range(20),extra={'name':'scott','age':'100'})

def call_oneway():
   prx.show_oneway(range(10))

def call_bidirection():
   adapter = tce.RpcCommAdapter('adapter')
   impl = TerminalImpl()
   adapter.addConnection(prx.conn)
   adapter.addServant(impl)
   communicator.addAdapter(adapter)
   prx.start_bidirection_oneway()

   communicator =tce.RpcCommunicator.instance().init()
   prx = ICtrlServerPrx.create(('127.0.0.1',16005))
```

tce for python

传送额外数据 extra={}

每个proxy的方法末尾参数是一个extra变量，类型是dictionary<string,string>

可以通过extra传递非应定义的数据

```
def call_extras():
        prx.show(range(20),extra={'name':'scott','age':'100'})
```

接收端利用ctx获取extra数据:

```
def show(self,sids,ctx):
        print ctx.msg.extra.props
```

ctx 的类型为tcelib.RpcContext

```
class RpcContext:
    def __init__(self):
        self.conn = None        #RpcConnection
        self.msg = None
```

tce for java

idl定义

service.idl

```
import base

module sns{


interface ITerminal{
    void onPushMessage(string  msg);
};

interface IGateway{
        bool login(string token);
        void heartbeat();
        string description();
};

interface ICtrlServer extends IBaseServer{
        bool changeUserPasswd(string old,string new);
        void userOnline(string userid) ;
        void userOffline(string userid);
        string userAuth(string user,string passwd);

        void uploadGps(GpsInfo_t gps);
        LocationInfo_t getUserLocation(string user);
        LocationInfoList_t findUsers(GeoRect_t rect);
};

}
```

```
ICtrlServerProxy(RpcConnection)
create(String, int): ICtrlServerProxy
createWithProxy(RpcProxyBase): ICtrlServerProxy
destroy(): void
changeUserPasswd(String, String): Boolean
changeUserPasswd(String, String, int, HashMap<String, String>): Boolean
changeUserPasswd_async(String, String, ICtrlServer_AsyncCallBack, HashMap<String, String>): void
userOnline(String): void
userOnline(String, int, HashMap<String, String>): void
userOnline_oneway(String, HashMap<String, String>): void
userOnline_async(String, ICtrlServer_AsyncCallBack, HashMap<String, String>): void
userOffline(String): void
userOffline(String, int, HashMap<String, String>): void
userOffline_oneway(String, HashMap<String, String>): void
userOffline_async(String, ICtrlServer_AsyncCallBack, HashMap<String, String>): void
userAuth(String, String): String
userAuth(String, String, int, HashMap<String, String>): String
userAuth_async(String, String, ICtrlServer_AsyncCallBack, HashMap<String, String>): void
uploadGps(GpsInfo_t): void
uploadGps(GpsInfo_t, int, HashMap<String, String>): void
uploadGps_oneway(GpsInfo_t, HashMap<String, String>): void
uploadGps_async(GpsInfo_t, ICtrlServer_AsyncCallBack, HashMap<String, String>): void
getUserLocation(String): LocationInfo_t
getUserLocation(String, int, HashMap<String, String>): LocationInfo_t
getUserLocation_async(String, ICtrlServer_AsyncCallBack, HashMap<String, String>): void
findUsers(GeoRect_t): Vector<LocationInfo_t>
findUsers(GeoRect_t, int, HashMap<String, String>): Vector<LocationInfo_t>
findUsers_async(GeoRect_t, ICtrlServer_AsyncCallBack, HashMap<String, String>): void
```

tce for java

Proxy对象

接口函数

bool changeUserPasswd(string old,string new);
void userOnline(string userid) ;

```
public static ICtrlServerProxy create(String host,int port)
public static ICtrlServerProxy createWithProxy(RpcProxyBase proxy)

public Boolean changeUserPasswd(String old,String new_)
public Boolean changeUserPasswd(String old,String new_,int timeout,HashMap<String,String> props)
public void changeUserPasswd_async(String old,String new_,ICtrlServer_AsyncCallBack
async,HashMap<String,String> props)
public void changeUserPasswd_async(String old,String new_,ICtrlServer_AsyncCallBack
async,HashMap<String,String> props,Boolean dispatchMainThread))

public void userOnline(String userid)
public void userOnline(String userid,int timeout,HashMap<String,String> props)
public void userOnline_oneway(String userid,HashMap<String,String> props)
public void userOnline_async(String userid,ICtrlServer_AsyncCallBack async,HashMap<String,String>
props)
public void userOnline_async(String userid,ICtrlServer_AsyncCallBack async,HashMap<String,String>
props,Boolean dispatchMainThread)
```

oneway - 单向调用,无返回值，无需等待;
仅void类型才能使用
async - 异步调用，无需等待;
返回值通过派生异步回调对象接收返回值 ICtrlServer_AsyncCallBack

android环境不能阻塞调用和超时调用

async(...,dispatchMainThread) 执行接收返回数据的代码将在主线程中执行

tce for java

Proxy的功能接口

class xxxProxy extends RpcProxyBase
tce自动生成的proxy对象都从RpcProxyBase派生

public xxxProxy(RpcConnection conn)
构造函数，可以指定连接对象来创建一个代理

static xxxProxy create(String host,int port,Boolean ssl_enable)
代理创建的辅助函数，通过指定目标主机地址和端口来创建**proxy**对象。
ssl_enable指示是否启用**ssl**加密

static xxxProxy createWithProxy(RpcProxyBase proxy)
代理创建的辅助函数，通过一个**proxy**来创建新的**proxy**，这种技巧原理是两个**proxy**共享了同一个**Connection**对象

void destroy()
显式的关闭**Proxy**对象持有的**Connection**

string foo(int p1,..)
阻塞式的函数调用

string foo(int p1,..,int timeout,HashMap<String,String> props)
阻塞式的函数调用**,**但可以指定等待超时时间**.**除了参数之外，接口调用时可通过**props**携带额外数据

void foo_async(p1,..,foo_AsyncCallBack async,HashMap<String,String> props,Object cookie)
异步函数调用。**async –** 异步消息接收对象**; props –** 额外数据**; cookie –** 用户数据

void foo_oneway(p1,..,HashMap<String,String> props)
单向函数调用。 **props –** 额外数据**;**

tce for java

使用异步函数调用

```
interface AsyncTest{
     string whatColor(int position);
}
```

```
class AsyncTest_AsyncCallBack extend RpcAsyncCallBackBase{
     void whatColor(string result,RpcProxyBase proxy,Object cookie);
     void whatColor_async(int position,AsyncTest_AsyncCallBack async,HashMap<String,String> props,Object cookie);
}
```

```
class AsyncTestProxy extend RpcProxyBase{
     static AsyncTestProxy create(host,port,ssl);
     void whatColor_async(int position,AsyncTest_AsyncCallBack async,HashMap<String,String> props,Object cookie);
}
```

void destroy()
显式的关闭**Proxy**对象持有的**Connection**

string foo(int p1,..)
阻塞式的函数调用

string foo(int p1,..,int timeout,HashMap<String,String> props)
阻塞式的函数调用,但可以指定等待超时时间.除了参数之外，接口调用时可通过**props**携带额外数据

void foo_async(p1,..,foo_AsyncCallBack async,HashMap<String,String> props,Object cookie)
异步函数调用。**async** – 异步消息接收对象; **props** – 额外数据; **cookie** – 用户数据

void foo_oneway(p1,..,HashMap<String,String> props)
单向函数调用。 **props** – 额外数据;

tce for java

ICtrlServerProxy

接口函数

bool changeUserPasswd(string old,string new);
void userOnline(string userid) ;

```
public static ICtrlServerProxy create(String host,int port)
public static ICtrlServerProxy createWithProxy(RpcProxyBase proxy)

public Boolean changeUserPasswd(String old,String new_)
public Boolean changeUserPasswd(String old,String new_,int timeout,HashMap<String,String> props)
public void changeUserPasswd_async(String old,String new_,ICtrlServer_AsyncCallBack
async,HashMap<String,String> props)
public void changeUserPasswd_async(String old,String new_,ICtrlServer_AsyncCallBack
async,HashMap<String,String> props,Boolean dispatchMainThread))

public void userOnline(String userid)
public void userOnline(String userid,int timeout,HashMap<String,String> props)
public void userOnline_oneway(String userid,HashMap<String,String> props)
public void userOnline_async(String userid,ICtrlServer_AsyncCallBack async,HashMap<String,String>
props)
public void userOnline_async(String userid,ICtrlServer_AsyncCallBack async,HashMap<String,String>
props,Boolean dispatchMainThread)
```

oneway - 单向调用,无返回值，无需等待;
仅void类型才能使用

async - 异步调用，无需等待;
返回值通过派生异步回调对象接收返回值 ICtrlServer_AsyncCallBack

android环境不能阻塞调用和超时调用

async(...,dispatchMainThread) 执行接收返回数据的代码将在主线程中执行

tce for java

简单的客户端示例

```
class Terminal extends sns.ITerminal{
     public Terminal(){
          super();
     }
     @Override
     public void onPushMessage(String msg, RpcContext ctx) {
          Main.instance().text.setText("msg:"+msg+" from server
     }
}
```

```
//定义代理访问对象
ICtrlServerProxy prxCtrlServer = null;
IGatewayProxy prxGateway = null;

//环境初始化
RpcCommunicator__Android.instance().init();
//创建通信适配器
tce.RpcCommAdapter adapter = tce.RpcCommunicator.instance().createAdapterWithProxy("local", prxCtrlServer);

//创建服务实现对象
Terminal servant = new Terminal();
adapter.addServant(servant);

//初始化代理对象
prxCtrlServer = ICtrlServerProxy.create(TARGET__HOST,TARGET__PORT);
prxGateway = IGatewayProxy.createWithProxy(prxCtrlServer);
```

```
//请求服务
GpsInfo_t gps = new GpsInfo_t();
gps.loc.lon = (float)121.03; gps.loc.lat= (float)31.;
prxCtrlServer.uploadGps_async(gps, new ICtrlServer_AsyncCallBack(){
     @Override
     public void uploadGps(RpcProxyBase proxy) {

     }
}, null);
```

tce for java

ICtrlServer_AsyncCallBack

异步处理:　ICtrlServer_AsyncCallBack
代理类:　　ICtrlServerProxy

rpc with mq

mq 两种消息模式： topic, queue