命令模式巧解 HTTP 包

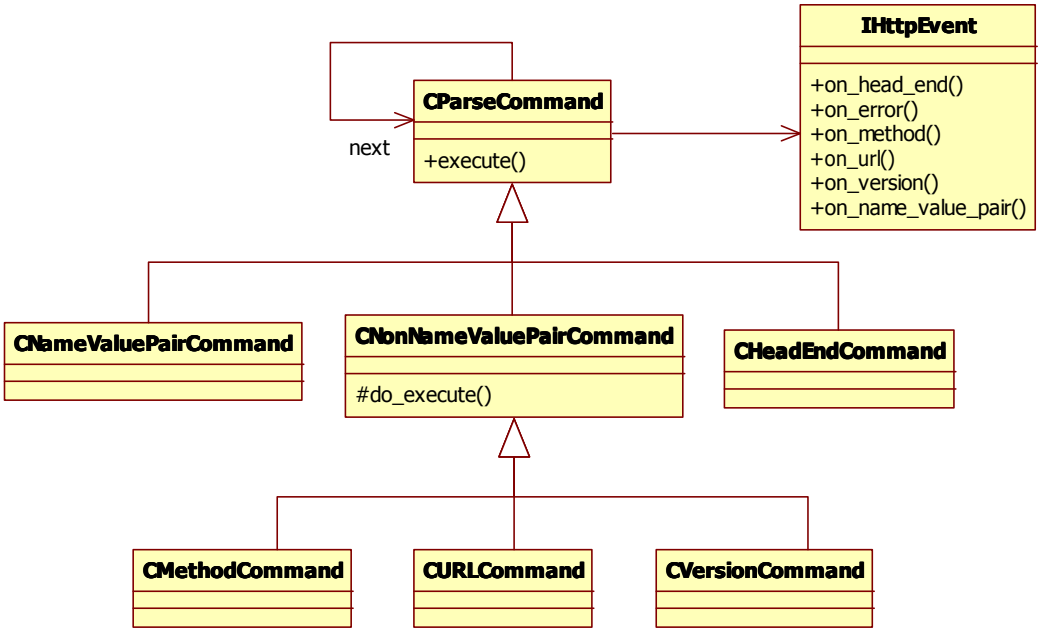
1. 前言

HTTP 是一个比较简单的协议，有着不同的解析方法，本文介绍如何通过命令模式来巧解 HTTP 包，而且格式的解析和内容是分离的，保证了实现的通用性。

2. 类视图

为了保证类图清晰，将类图分成以继承为主的继承视图和以聚合为主的聚合视图，两者实际是一起的，共同构成了 HTTP 包解析类视图。

2.1. 继承视图



2.1.1.IHttpEvent

HTTP 事件回调接口，用来提取所关心 HTTP 包内容，Command 只关心格式的解析。

2.1.2.CParseCommand

Command 基类。

2.1.3.CNameValuePairCommand

名值对命令，实现对名值对格式的解析。每条名值对均以“\n”打头，并以“\r”结尾。

2.1.4.CNonNameValuePairCommand

非名值对命令，即 HTTP 状态行解析命令，分请求和响应两类，但两者不会同时出现在同一个包的解析过程中。

2.1.5.CHeadEndCommand

HTTP 包解析完成命令，当 HTTP 解析完成时处罚该命令。HTTP 总是以连续的两个“\r\n”结尾。

2.1.6.CMethodCommand

HTTP 请求行中的 HTTP 请求方法解析命令。

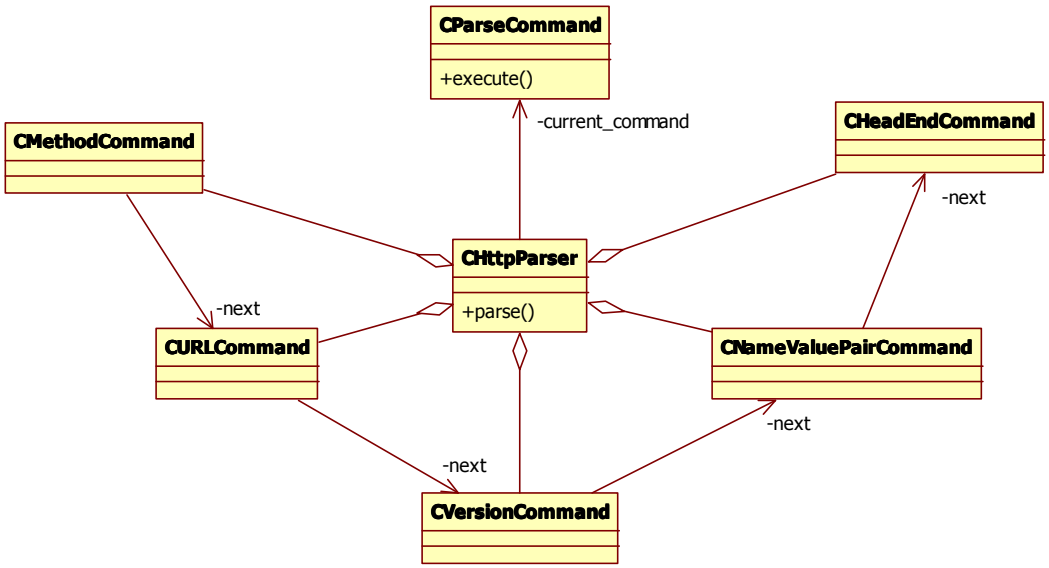
2.1.7.CURLCommand

HTTP 请求行中的 URL 解析命令。

2.1.8.CVersionCommand

HTTP 请求行中的 HTTP 版本解析命令。

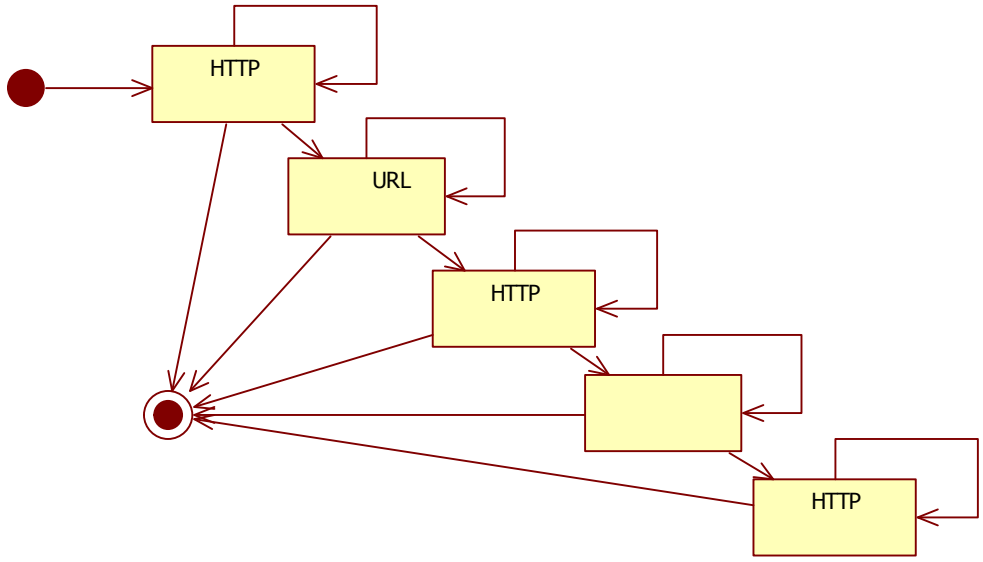
2.2. 聚合视图



2.2.1.CHttpParser

HTTP 协议解析入口和驱动类，通过驱动 Command 完成 HTTP 包的解析。一个命名解析完成之后，自动跳到下一个解析命令，所以命令组成一个命令链，组成一个 HTTP 解析自动机。

3. 解析状态图



HTTP 的状态非常简单，如上图所示，是一个线性的状态转换过程。每一步存在三个状态：要么成功完成本命令，要么未完成需要继续本命令和本命令解析出错。对于非线性的状态变化，过程稍复杂点，因为命令链将是动态变化的，每次都可能不一样，但是这个思想仍可以运用以化简代码的复杂度。

4. 总结

本方法具有以下优点：

1) 结构清晰，各代码相对独立；
2) 格式和内容分离，使得结构更为简单明了；
3) 格式和内容分离，通用性和可扩展性高；
4) 贪婪流式解析，不回溯，解析效率较高，即使按字节接收数据，也能保持解析速度。

附 1：代码位置

http://code.google.com/p/eejian/source/browse/#svn/trunk/src/http_parser

请注意，代码暂只做过基本测试，未严格测试。整个实现只有两段相对复杂的代码：非名值对解析和名值对解析代码，摘要如下，实际上还是相当简单的。

附 2：非名值对解析代码摘要

```
jian_util::TReturnResult CNonNameValuePairCommand::do_execute(const char* buffer, int& offset, char endchar, bool (IHttpEvent::*on_xxx)(const char*, const char*))
{
	const char* iter = buffer;

	for (;;)
	{
		if ('\0' == *iter) break;
		if (NULL == _begin)
		{
			iter = jian_util::CStringUtil::skip_spaces(iter);
			if (NULL == iter) //  空格过多
			{
				get_http_event()->on_error("too much spaces in the first line");
				return return_result(jian_util::rr_error);
			}

			if ('\0' == *iter) break;
			_begin = iter;
			++iter;
		}

		if ('\0' == *iter) break;

		if (endchar == *iter)
		{
			_end = iter;
			if (!(get_http_event()->*on_xxx)(_begin, _end)) return return_result(jian_util::rr_error);

			++iter; //  空格
			offset = iter-buffer;
			return return_result(jian_util::rr_finish);
		}

		++iter;
	}

	offset = iter-buffer;
	return jian_util::rr_continue;
}
```

附 3：名值对解析代码摘要

```
jian_util::TReturnResult CNameValuePairCommand::execute(const char* buffer, int& offset)
{
    const char* iter = buffer;

    for (;;)
    {
        if ('\0' == *iter) break;
        if (NULL == _name_begin)
        {
            // GET / HTTP/1.1\r\n
            // F1\r\n
            // F2\r\n
            // \r\n
            if (*iter != '\n')
            {
                get_http_event()->on_error("NV pair not started with '\\n'");
                return return_result(jian_util::rr_error);
            }

            ++iter;
            if ('\0' == *iter) break;
            if ('\r' == *iter) //  以\n\r 开头，表示包头结束
            {
                ++iter;
                offset = iter-buffer;
                return return_result(jian_util::rr_finish);
            }

            _name_begin = iter;
            ++iter;
        }
        else if ((NULL == _value_begin) && (_name_end != NULL))
        {
            iter = jian_util::CStringUtil::skip_spaces(iter);
            if (NULL == iter)
            {
                get_http_event()->on_error("too much spaces in NV pair");
                return return_result(jian_util::rr_error);
            }

            if ('\0' == *iter) break;
```

```
                _value_begin = iter;
                ++iter;
            }

            if ('\0' == *iter) break;

            if ('\r' == *iter)
            {
                _value_end = iter;
                if (!get_http_event()->on_name_value_pair(_name_begin, _name_end,
_value_begin, _value_end))
                    return return_result(jian_util::rr_error);

                // 准备下一个名字对的解析
                _name_begin  = NULL;
                _name_end    = NULL;
                _value_begin = NULL;
                _value_end   = NULL;
                _ignore_colon = false;
            }
            else if ((':' == *iter) && !_ignore_colon)
            {
                _name_end = iter;
                _ignore_colon = true;
            }

            ++iter;
        }

        offset = iter-buffer;
        return return_result(jian_util::rr_continue);
}
```

附 4：IHttpEvent 实现示例

```
class CHttpEvent: public jian::IHttpEvent
{
private:
    virtual bool on_head_end();
    virtual void on_error(const char* errmsg);
    virtual bool on_method(const char* begin, const char* end);
    virtual bool on_url(const char* begin, const char* end);
    virtual bool on_version(const char* begin, const char* end);
```

```
    virtual bool on_name_value_pair(const char* name_begin, const char* name_end
                                    ,const char* value_begin, const char* value_end);

private:
    typedef bool (CHttpEvent::*on_name_value_pair_xxx)(const char* name_begin, int name_len, const char* value_begin, int value_len);
    bool on_name_value_pair_3(const char* name_begin, int name_len, const char* value_begin, int value_len);
    bool on_name_value_pair_4(const char* name_begin, int name_len, const char* value_begin, int value_len);
    bool on_name_value_pair_5(const char* name_begin, int name_len, const char* value_begin, int value_len);
    bool on_name_value_pair_6(const char* name_begin, int name_len, const char* value_begin, int value_len);
    bool on_name_value_pair_7(const char* name_begin, int name_len, const char* value_begin, int value_len);
    bool on_name_value_pair_8(const char* name_begin, int name_len, const char* value_begin, int value_len);
    bool on_name_value_pair_9(const char* name_begin, int name_len, const char* value_begin, int value_len);
    bool on_name_value_pair_10(const char* name_begin, int name_len, const char* value_begin, int value_len);
    bool on_name_value_pair_11(const char* name_begin, int name_len, const char* value_begin, int value_len);
    bool on_name_value_pair_12(const char* name_begin, int name_len, const char* value_begin, int value_len);

private:
    CHttpHeader _header;
    on_name_value_pair_xxx _on_name_value_pair_xxx[20];   // 根据 name 的长度来决定调用哪个函数
};

bool CHttpEvent::on_method(const char* begin, const char* end)
{
    MYLOG_DEBUG("Http method: %.*s.\n", end-begin, begin);

    int len = end-begin;
    switch (len)
    {
    case 3:
        if (0 == strncasecmp(begin, "GET", len))
        {
            _header.set_method(CHttpHeader::hm_get);
```

```
                return true;
            }
        case 4:
            if (0 == strncasecmp(begin, "POST", len))
            {
                _header.set_method(CHttpHeader::hm_post);
                return true;
            }
            break;
        }

        return false;
}

bool CHttpEvent::on_url(const char* begin, const char* end)
{
        MYLOG_DEBUG("URL: %.*s.\n", end-begin, begin);
        _header.set_url(begin, end-begin);
        return true;
}

bool CHttpEvent::on_version(const char* begin, const char* end)
{
        MYLOG_DEBUG("Version: %.*s.\n", end-begin, begin);
        return true;
}

bool CHttpEvent::on_name_value_pair_10(const char* name_begin, int name_len, const char*
value_begin, int value_len)
{
        if (0 == strncasecmp(name_begin, "Connection", name_len))
        {
            if (0 == strncasecmp(value_begin, "Keep-Alive", value_len))
            {
                _header.set_keep_alive(true);
            }
            else
            {
                _header.set_keep_alive(false);
            }
        }

        return true;
}
```