Tornado 增强包（TORE）

为了适应模板和 Ajax 类型的 Web 应用以及 RESTful Web 服务开发，对 Tornado Web 框架进行了功能上的增强，适用于 Python 3.2，主要增加了以下功能：

1. 内置 Json Messaging 消息服务器；
2. 增强了模板引擎，使之支持 Code Behind 风格；
3. 增加认证、鉴权、Json 处理等其它功能；

下面是一些简要说明，具体说明参见软件注释文档。

1 tore.start_server(**settings)

功能：一步到位启动 tornado 服务器，tore 新增并强化了 tornado 默认的设置，如下表所示。

键	值	备注
root_dir	网站根目录，其它设置中的目录均已该目录为基准，默认为当前目录（os.getcwd()）	新增
handlers	URL 映射，功能和格式与 tornado.web.Application 类的构造函数中的 handlers 参数一致，默认有以下映射： 1、/web：映射到网站的 web 目录，其中的文件的访问规则为：*.t 文件作为模板处理；*.py 文件是模板代码文件，禁止直接访问；其它文件作为静态文件转交 tornado.web.StaticFileHandler 处理 2、/messaging：Json Messaging 消息服务的 WebSocket 地址，WebSocket 的地址和端口是与网站共享的。 3、/：根目录直接重定向到/web/index.t，所以/web/index.t 作为网站的首页必须存在	强化
port	网站端口，默认为 80	强化
gzip	是否启用 gzip 压缩，默认不启用	不变
encryption	是否启用 SSL 加密，如果启用了 SSL 加密，那么必须设置 certfile 和 keyfile 参数，默认不启用	新增
certfile	SSL 公钥文件名，功能和 tornado.httpserver.HTTPServer 的参数是一致的，可以使用绝对路径或相对路径，如果使用相对路径，那么以 root_dir 为根	新增
keyfile	SSL 私钥文件名，功能和 tornado.httpserver.HTTPServer 的参数是一致的，可以使用绝对路径或相对路径，如果使用相对路径，那么以 root_dir 为根	新增
debug	是否启用调试，默认不启用。该参数除了具备 tornado.web.Application 同名参数的功能外，tore 还增加了以下功能： 1、禁用调试将关闭 tornado 所有 HTTP 相关的调试输出，提高性能； 2、启用调试的时候 tore.web.JsonHandler 的 Json 输出是可读的，也就是不进行 ascii 转义，并且具有换行和缩进； 3、启用调试的时候每一次请求都将重新加载模板文件和对应的代码文件，禁用调试的时候只加载一次，和 tornado 默认的行为是一致的	强化
authentication	foo(username, password)格式的函数，tore.web.authenticated 修饰符将会用从 HTTP 基本认证中获取的用户名和口令调用该函数，如果该函数存在并且调用返回值为 False，那么将引发 HTTP 401 错误；否则设置这个用户名为当前	新增

	用户名并且调用被修饰的方法。默认不存在	
authorization	foo(username, path)格式的函数，tore.web.authorized 修饰符将会使用当前用户名和当前请求的 URL 调用该函数，如果该函数存在并且调用返回值为 False，那么将引发 HTTP 403 错误；否则调用被修饰的方法。默认不存在	新增
messaging_tcp_port	Json Messaging 消息服务的 TCP 端口，设置为 None 则禁用 TCP，默认为 None	新增
messaging_udp_port	Json Messaging 消息服务的 UDP 端口，设置为 None 则禁用 UDP，默认为 None	新增

2 模板引擎

tore.web.RequestHandler、tore.web.Loader、tore.web.Template 分别改写了 tornado.web.RequestHandler、tornado.template.Loader、tornado.template.Template，增强了 Tornado 自带的模板功能，一般不建议单独使用，具体改进如下：

现在模板引擎能自动寻找模板同目录下存在“模板文件名.py”文件，如果存在，则自动加载合并到模板生成的代码中，如果不存在则不加载。对于模板继承的情况，模板引擎能够自动的遍历祖先模板，然后按照祖先优先的顺序加载这些模板对应的代码文件（如果存在的话）。

现在写模板和代码就像 ASP.NET 的 Code Behind 风格一样，简单多了，不需要再像传统 Tornado 那样需要编写 Handler 然后在 URL 映射中进行繁琐的设置了。下面举个简单的例子，具体好处自己体会。

父模板 base.t

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ title }}</title>
</head>
<body>
    {% block body %}{% end %}
</body>
</html>
```

父模板代码 base.t.py

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
"""

"""
__author__ = ''
__version__ = ''

title = request.path
```

子模板 index.t

```
{% extends "base.t" %}
{% block body %}
<h1>
    {{ hello }}
    {{ world() }}
    {{ os.getcwd() }}
</h1>
<img src="test.jpg" alt="test">
{% end %}
```

子模板代码 index.t.py

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
"""

"""
__author__ = ''
__version__ = ''

import os

hello = 'hello'

def world():
    return 'world'
```

最终子模板会自动地转换成以下 Python 代码，可以看到父子模板及其代码的合并是非常“和谐”的：

```
def _execute():  # base.t:0
    _buffer = []  # base.t:0
    _append = _buffer.append  # base.t:0
    _append(b'<!DOCTYPE html>\n<html>\n<head>\n    <title>')  # base.t:4
    _tmp = title  # base.t:4
    if isinstance(_tmp, _string_types): _tmp = _utf8(_tmp)  # base.t:4
    else: _tmp = _utf8(str(_tmp))  # base.t:4
    _tmp = _utf8(xhtml_escape(_tmp))  # base.t:4
    _append(_tmp)  # base.t:4
    _append(b'</title>\n</head>\n<body>\n    ')  # base.t:7
    _append(b'\n<h1>\n    ')  # index.t:4 (via base.t:7)
```

```
    _tmp = hello  # index.t:4 (via base.t:7)
    if isinstance(_tmp, _string_types): _tmp = _utf8(_tmp)  # index.t:4 (via base.t:7)
    else: _tmp = _utf8(str(_tmp))  # index.t:4 (via base.t:7)
    _tmp = _utf8(xhtml_escape(_tmp))  # index.t:4 (via base.t:7)
    _append(_tmp)  # index.t:4 (via base.t:7)
    _append(b'\n    ')  # index.t:5 (via base.t:7)
    _tmp = world()  # index.t:5 (via base.t:7)
    if isinstance(_tmp, _string_types): _tmp = _utf8(_tmp)  # index.t:5 (via base.t:7)
    else: _tmp = _utf8(str(_tmp))  # index.t:5 (via base.t:7)
    _tmp = _utf8(xhtml_escape(_tmp))  # index.t:5 (via base.t:7)
    _append(_tmp)  # index.t:5 (via base.t:7)
    _append(b'\n    ')  # index.t:6 (via base.t:7)
    _tmp = os.getcwd()  # index.t:6 (via base.t:7)
    if isinstance(_tmp, _string_types): _tmp = _utf8(_tmp)  # index.t:6 (via base.t:7)
    else: _tmp = _utf8(str(_tmp))  # index.t:6 (via base.t:7)
    _tmp = _utf8(xhtml_escape(_tmp))  # index.t:6 (via base.t:7)
    _append(_tmp)  # index.t:6 (via base.t:7)
    _append(b'\n</h1>\n<img src="test.jpg" alt="test">\n')  # index.t:9 (via base.t:7)
    _append(b'\n</body>\n</html>')  # base.t:9
    return _utf8('').join(_buffer)  # base.t:0



#!/usr/bin/env python
# -*- coding: UTF-8 -*-
"""

"""
__author__ = ''
__version__ = ''

title = request.path

#!/usr/bin/env python
# -*- coding: UTF-8 -*-
"""

"""
```

```
__author__ = ''
__version__ = ''


import os


hello = 'hello'


def world():
    return 'world'
```

3 tore.web.JsonHandler

为 Ajax 和 RESTful Web 服务优化的 Json 处理器，继承自 tornado.web.RequestHandler，增加以下方法：

1. write_object(self, obj)：把 Python 对象输出为 Json 格式，Content Type 设置为“application/json; charset=UTF-8”
2. write_text(self, txt)：直接输出 Json 格式的文本，Content Type 设置为“application/json; charset=UTF-8”。注意此方法不对文本做编码和语法检查
3. get_params_as_dict(self)：获取字典包装的请求参数，包含 URL 附加的查询字符串（例如 GET 请求）和 HTTP 请求报文体（例如 POST 请求）中的内容
4. get_body_as_text(self)：获取纯文本格式的 HTTP 请求报文体，默认解码自 UTF-8
5. get_body_as_object(self)：将 Json 格式的 HTTP 请求报文体转化为 Python 对象，注意使用 jQuery.ajax()提交时，Json 字符串应该放在 data 参数中，并且 processData 参数必须设置为 false，否则 jQuery 会“自作聪明”地把 data 中的内容进行 URL 格式的转码

4 tore.web.authenticated 修饰符

用于修饰 RequestHandler 中的 get、post 等方法，对这些请求进行 HTTP 基本身份认证（HTTP Basic Authentication），认证调用的的函数详见 tore.start_server()的设置参数。

tore 增强的模板处理器已经加入了该修饰。

5 tore.web.authorized 修饰符

用于修饰 RequestHandler 中的 get、post 等方法，对这些请求进行鉴权，鉴权调用的的函数详见 tore.start_server()的设置参数。

tore 增强的模板处理器已经加入了该修饰。

这两个修饰符可以结合使用，同时对请求进行认证和鉴权，例如：

```
    @tore.web.authenticated
    @tore.web.authorized
    def get(self, *args, **kwargs):
        ...
```

6 Json Messaging 消息引擎

原先使用 Node.JS 编写的 Json Messaging 消息服务现已经用 Python 完全重写并嵌入到 tornado 中，参见 tore.start_server()的设置参数表。

在服务程序内部可以直接调用 tore.messaging.exchange.push(message, destination)发送消息，就不需要通过 TCP 或者 UDP 了。

帧格式的定义参见 Json Messaging 文档。

下面的 jquery.messaging.js 程序提供了通过 WebSocket 发布订阅消息服务的功能，目前支持新版本的 Chrome 和 Firefox 浏览器：

```
/*!
 * Json Messaging 消息服务客户端
 * 具有断连自动恢复功能，参见 initWs()函数
 * @author shajunxing
 * @version 0.0.0.0
 */



(function ($) {
    /**
     * 获取消息客户端
     * 消息客户段具有下面一些事件：
     * onOpen() 客户端已连接
     * onError() WebSocket 的 onerror 和消息服务器的错误帧都将触发该事件
     * onClose() 客户端已关闭事件
     * @return {*}
     */
    $.messageClient = function () {
        window.WebSocket = window.WebSocket || window.MozWebSocket;

        if (!window.WebSocket) {
            // 浏览器不支持 WebSocket
            return null;
        }

        var client = {};

        // 局部消息回调函数，可针对每一个订阅单独定义回调函数
        client.messageListeners = {};
        // 打开、关闭和错误回调函数
        client.openListener = null;
        client.closeListener = null;
```

```
        client.errorListener = null;


        /**
         * 初始化client对象中的ws成员并和client绑定
         * 此函数将在连接错误或关闭后定时自动调用以自动重新连接
         */
        client.initWs = function () {
            if (location.protocol == 'https:') {
                client.ws = new WebSocket('wss://' + location.host +
'/messaging');
            } else {
                client.ws = new WebSocket('ws://' + location.host +
'/messaging');
            }


            client.ws.onmessage = function (message) {
                try {
                    var parsed = JSON.parse(message.data);
                    switch (parsed.type) {
                        case 'message':
                            // 消息帧
                            // 局部回调函数
                            for (var destination in client.messageListeners) {
                                if
(client.messageListeners.hasOwnProperty(destination)) {
                                    if (client.messageListeners[destination]
['pattern'].exec(parsed.match[0])) {
                                        if (client.messageListeners[destination]
['listener']) {
                                            client.messageListeners[destination]
['listener'](parsed.content, parsed.match);
                                        }
                                    }
                                }
                            }
                            break;
                        case 'error':
                            // 错误帧
                            console.warn('Server error: %s', parsed);
                            if (client.errorListener) {
                                client.errorListener();
                            }
```

```
                        break;
                    default:
                        // 未知帧
                        console.warn('Unknown message type %s',
parsed.type);
                        break;
                }
            } catch (e) {
                console.warn(e);
            }
        };

        client.ws.onopen = function () {
            if (client.openListener) {
                client.openListener();
            }
        };

        client.ws.onclose = function () {
            console.debug('websocket closed');
            if (client.closeListener) {
                client.closeListener();
            }
            // 等待尝试重新连接
            // TODO: 是否会引起内存泄露?
            setTimeout(client.initWs, 3000);
        };

        client.ws.onerror = function () {
            console.debug('websocket error');
            if (client.errorListener) {
                client.errorListener();
            }
            // 关闭连接
            client.ws.close();
        };
    };

    client.onOpen = function (listener) {
        client.openListener = listener;
```

```
        };

        client.onClose = function (listener) {
            client.closeListener = listener;
        };

        client.onError = function (listener) {
            client.errorListener = listener;
        };

        /**
         * 发布消息
         * @param content 内容
         * @param destination 目的地
         */
        client.publish = function (content, destination) {
            client.ws.send(JSON.stringify({
                type:'publish',
                destination:destination,
                content:content
            }));
        };

        /**
         * 订阅消息
         * @param destination 目的地（可以是正则表达式）
         * @param listener(content, match) 该订阅的回调函数
         */
        client.subscribe = function (destination, listener) {
            client.messageListeners[destination] = {};
            // 预编译
            client.messageListeners[destination]['pattern'] = new
RegExp(destination);
            client.messageListeners[destination]['listener'] = listener;
            client.ws.send(JSON.stringify({
                type:'subscribe',
                destination:destination
            }));
        };
```

```
        /**
         * 取消订阅消息
         * @param destination 目的地
         */
        client.unsubscribe = function (destination) {
            delete client.messageListeners[destination];
            client.ws.send(JSON.stringify({
                type:'unsubscribe',
                destination:destination
            }));
        };

        /**
         * 关闭连接
         */
        client.close = function () {
            client.ws.close();
        };

        // 第一次初始化
        client.initWs();

        return client;
    };
}(jQuery));
```