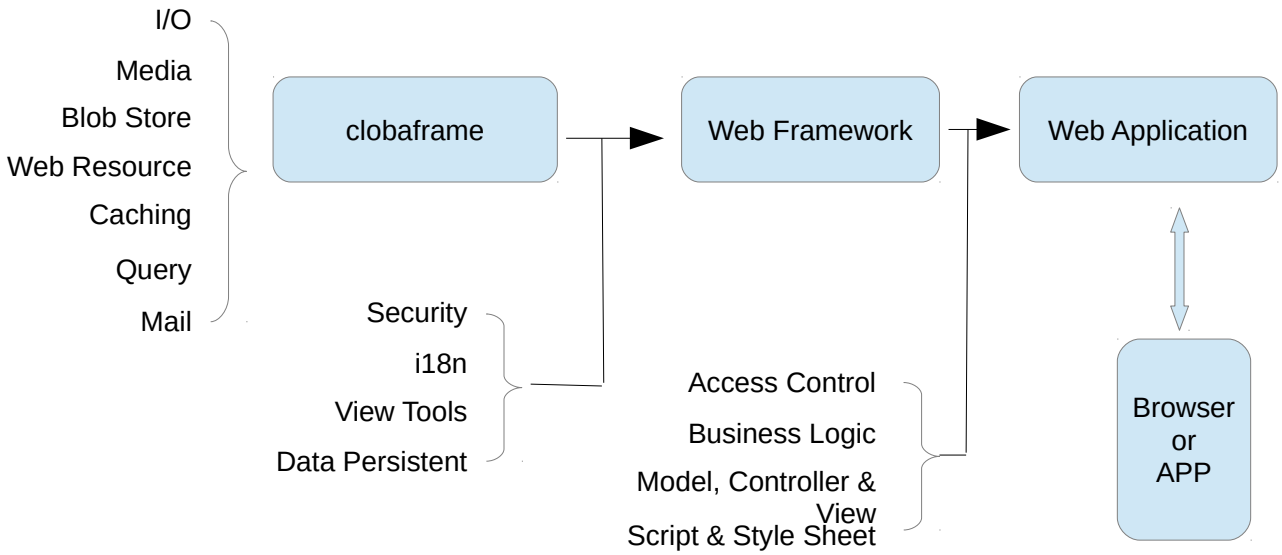
The Clobaframe Manual

For version 2.4 Last updated: 2015-04-06

内容目录

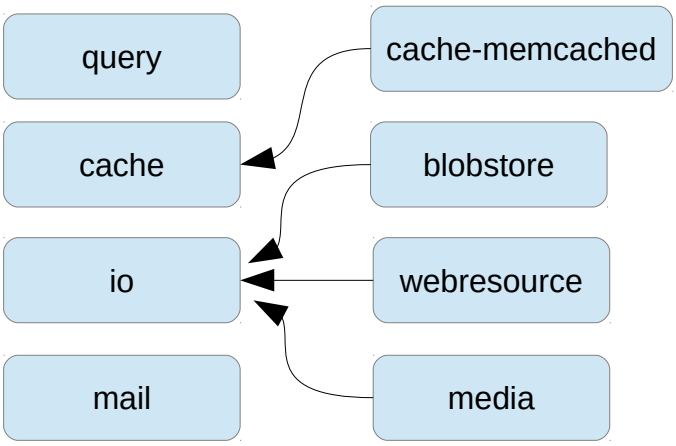
Clobaframe 是一个 web 应用程序的基础框架，提供统一的接口，封装及抽象各种云服务（如 Amazon WebService, Google App Engine 等），简化 web 程序调用云服务的过程。另外 Coapframe 提供了 web 应用程序常用的功能封装，比如图片的处理、对象集合的查询、web 资源（指脚本和样式表等）的自动化管理、音频和视频的元数据提取等，为 web 应用程序提供必要的支持。目前该框架已经通过 Apache 协议开源（源码库地址 https://github.com/ivarptr/clobaframe ）。

1 Clobaframe 在 web app 当中的角色



Clobaframe 并不是一个 web 框架，但为可以跟现有的 web 框架（比如 Spring Web MVC）结合使用。典型的组合由 Clobaframe、Spring Framework、Spring Security、Spring Web MVC 和 Apache Velocity 等组成，只需为项目添加必要的数据持久化模块和业务逻辑模块，即可以快速搭建 web 应用程序。

2 各个子模块的作用及依赖关系



Clobaframe 主要有如下几个子模块：

- Query：实现对象集合的条件查询、排序、重组等功能。
- Cache：对缓存服务的统一封装，目前有两个实现：Ehcache 和 Memcached，前者为基于 jvm 内存缓存，虽然可以配置为分布式，不过一般用于开发阶段或者单服务器的生产环境。后者使用 Memcached 服务，有良好的分布式特性，支持 Amazon ElastiCache 缓存阵列。
- Media：主要由 Image,Audio,Video 三个小模块组成，Image 包含一些常用的图片处理功能，比如缩放、裁剪、加水印等。Audio 和 Video 主要用于提取元数据（Meta Data）
- IO：对二进制数据资源进行封装，同时提供将数据送数据至客户端/浏览器，或者从客户端/浏览器接收数据的功能。
- Blob Store：储存或读取二进制资源，一般用于将图片/声音/视频等用户资源储存于分布式的储存阵列当中（或者云服务）。
- Web Resource：自动管理 web application 的资源，包括图片/css 样式表/js 脚本等。
- Mail：邮件的发送。

3 使用示例

Clobaframe 各个模块的详细使用方法可以阅读相关的 JavaDoc，或者参考各模块的单元测试的代码。下面简单罗列一下功能及使用示例：

3.1 Query

考虑有个集合 ‘members’, 它有 4 个对象：

```
{id:001, name:hello, gender:m, birth:1990-4-1}
{id:002, name:world, gender:f, birth:1992-5-1}
```

```
{id:003, name:foo, gender:f, birth:1994-8-1}
{id:004, name:bar, gender:m, birth:1996-1-1}
```

3.1.1 查询所有 ‘gender’='m’ 的对象

```
result = SimpleQuery.from(members).whereEquals("gender", "m").list();
```

3.1.2 按照 ‘birth’ 属性排序并找出第一个对象

```
result = SimpleQuery.from(members).orderBy("birth").first();
```

3.1.3 先按照 ‘gender’ 排序然后再按 ‘birth’ 排序

```
result = SimpleQuery.from(members)
  .orderBy("gender")
  .orderBy("birth")
  .list();
```

3.1.4 查询所有 ‘birth’ 大于 ’1992-1-1’ 并小于 ’1996-1-1’ 的对象

```
result = SimpleQuery.from(members)
  .whereGreaterThan("birth", date1)
  .whereLessThan("birth", date2)
  .list();
```

3.1.5 查询所有 ‘gender’='f’ 的对象然后按照 ‘name’ 排序

```
result = SimpleQuery.from(members)
.whereEquals("gender","f")
.orderBy("name")
.list();
```

3.1.6 查询所有 ‘id’ 大于 ’003’ 或者 ‘name’ 等于 ‘hello’ 的对象

```
result = SimpleQuery
```

```
.from(members)
.where(PredicateFactory.or(
  PredicateFactory.greaterThan("id", "003"),
  PredicateFactory.equals("name", "hello")))
.list();
```

3.1.7 查询所有 ‘gender’='m’ 的对象，然后返回只包含 'name'属性的新对象

```
result = SimpleQuery.from(members).whereEquals("gender",
"m").select(“name”);
```

3.1.8 返回所有对象由 'id' 和 'name' 属性组成的新对象

```
result = SimpleQuery.from(members).whereEquals("gender",
"m").select(“id”, “name”);
```

3.2 Cache

3.2.1 将字符串压入缓存

```
cache.put("key001", "foo");
cache.put("key002", "bar");
```

3.2.2 从缓存中获取

```
String s1 = (String)cache.get("key001"); // s1 will equals "foo"
String s2 = (String)cache.get("key002"); // s2 will equals "bar"
```

3.2.3 删除缓存项

```
cache.delete("key001");
```

3.3 Media

3.3.1 缩放图片，将图片缩放至 200x200px

```
File file = new File("...");
Image image = (Image)mediaFactory.make(file);
Transform transform = imaging.reize(200, 200);
Image newimage = imaging.apply(transform);
```

3.3.2 为图片添加字体为“Arial” 32 像素内容为“Watermark ”的蓝色透明度为 75%的水印

```
File file = getFileByName("test.png");
Image image = (Image)mediaFactory.make(file);
Font font = new Font("Arial", Font.BOLD, 32);
Composite composite = imaging.text("Watermark", font, Color.blue, 100, 100, 0.75F);
Image newimage = imaging.apply(image, composites);
```

3.4 IO

3.4.1 从客户端浏览器接收文件

```
List<UploadedResourceInfo> resourceInfos = resourceReceiver.receive(request);
for(UploadedResourceInfo resourceInfo : resourceInfos){
if (!resourceInfo.isFormField()){
    ResourceContent resourceContent = resourceInfo.getContentSnapshot();
    ...
}else{
    System.out.println(resourceInfo.getContentAsString());
}
```

3.4.2 发送数据至客户端

```
Resource resource = ...;
resourceSender.send(resource, request, response);
```

3.5 Blob Store

对于 BlobStore 来说，资源使用一个惟一码来标识和访问，资源之间没有目录层次结构。另外资源可以分别储存于不同的集合当中，每个集合称之为“Repository”。分集合储存资源的一个好处是可以对资源的性质进行分类管理，比如把用户的临时文件放在某一个 repository 当中，在系统维护阶段可以将这个 repository 进行清空操作。

[TODO::更新本章节的内容]

3.5.1 储存资源

```
InputStream in = ...
BlobKey blobKey = new BlobKey("bucket001", "key-001"); // make sure the "bucket001" bucket has already exists.
BlobInfo blobInfo = blobInfoFactory.createBlobInfo(blobKey, data.length, "image/jpeg", in);
blobstore.put(blobInfo);
```

3.5.2 获取指定资源

```
BlobKey blobKey = new BlobKey("bucketName", "key-002");
BlobInfo blobInfo = blobstore.get(blobKey);
BlobContent blobContent = blobInfo.getContentSnapshot();
InputStream in = blobContent.getInputStream();
...
blobContent.close();
```

3.5.3 删除指定资源

```
BlobKey blobKey = new BlobKey("bucketName", "key-003");
blobstore.delete(blobKey);
```

3.5.4 列举资源

```
BlobKey blobKeyPrefix = new BlobKey("bucketName", "key-");
PartialCollection blobs1 = blobstore.list(blobKeyPrefix);
...
while (blobs1.hasMore()){
    PartialCollection blobs2 = blobstore.listNext(blobs)
    ...
}
```

3.6 Web Resource

该模块能自动化管理所有 web 资源的版本问题，同时还提供了自动替换 css 样式表当中的资源路径。比如现有样式表 "common.css" 和图片"logo.png"：

```
#button {
    background: url("../image/logo.png") no-repeat 0px 0px;
}
```

其中的 “../image/logo.png” 将会自动替换为实际 url 地址。

3.6.1 根据资源名称获取资源对象

```
WebResourceInfo resource = webResourceService.getResource("main.css");
// get the resource location
String url = webResourceService.getLocation(resource);
// or
String url = webResourceService.getLocation("main.css");
```

3.6.2 发送指定资源给客户端

```
webResourceSender.send("main.css", request, response);
```

3.7 Mail

用于发送纯文本或者 HTML 格式邮件，支持邮件模板。

4 Clobaframe 的配置文件

Clobaframe 的默认配置文件为 clobaframe.properties，并由 Spring Framework 以 PropertyPlaceholderConfigurer 的形式引入项目，配置文件的内容可以根据所用到的子模块进行增减。配置内容也可以存在于其他属性文件，只要由 PropertyPlaceholderConfigurer 引入项目即可。

对于配置内容中的资源路径表示方法，由于采用了 Spring Framework，所以有两种形式：

- classpath:some.package.name.subname
- file:path/relate/to/src/folder

如果省略 classpath: 或者 file: 前缀，则由当前项目运行时的 Application Context 的类型决定，比如在单元测试环境下默认的是 file，而在 Servlet 容器当中，默认的是 classpath。因此建议不要省略路径前缀。

下面是主要的配置项及其作用：

[TODO::更新本章节的内容]

```
cache.agent=memcached

#指定 cahce 使用的实现的名称，目前有 3 个实现：null，ehcache，memcached。第一个用于禁用缓存，一般用于单元测试阶段，第二个用于开发环境或者单服务器，第三个用于生产环境或者多服务器。


cache.memcached.servers=127.0.0.1:11211

#指定 memcached 的服务器地址及其服务端口，多个 cache 服务器可以使用逗号分隔，比如：'host1:11211,host2:11211,host3:11211'
```

cache.memcached.protocol=BINARY

#指定 memcached 的通信协议，使用 BINARY 可以提高更好的性能，详细文档参考 memcached。

cache.ehcache.region=common

#指定 encache 服务当中用于 cache 的域的名称。因为 encache 服务可能会配置有多个域，有些域可能用于其他服务（比如 hibernate 二级缓存），因此需要在此指定用于 coapframe cache 服务的域的名称。

cache.ehcache.configuration=classpath:ehcache.xml

#指定 encache 的配置文件名称

media.maxHandleSize=2048

#指定 media 最大能处理的媒体的大小，单位为 KB，用于防止待处理的比如图片、音频、视频等过大以致消耗过多的内存资源。

blobstore.agent=local

#指定 blobstore 使用的实现的名称，Clobaframe 自带一个本地实现，使用本地硬盘作为 blobstore 的数据储存方式，由于本地硬盘不易于动态扩展容量而且不支持指定的 meta data 和 content type（mime type 名称），所以只能用于开发和调试阶段。如果需要使用云存储服务（比如 Amazon S3），可以查看另一个项目 clobaframe-amazon。

blobstore.local.path=file:/home/arch/blobstore

#本地储存的目录，如果采用相对路径，在 web application 当中这个路径是相对于'src/main/webapp'目录的路径。建议写成绝对路径。

io.maxUploadSize=1024

#指定 webio 最大支持上传数据的大小，单位为 KB。通过此项配置可以防止用户上传过大的数据/文件导

致服务器内存消耗。

webresource.strategy=local

#指定 webresource 使用的实现的名称。目前有 2 个实现：local 和 blobstore。第一个使用本地硬盘储存 web 资源（图片、样式表和脚本等），这个实现要求 web application 增加一个 URL 路由负责发送资源数据，因此适用于开发或者单服务器环境。第二个使用 Blobstore 储存 web 资源，将 web 资源交由 Blobstore 托管可以有效减轻服务器的负担，并且可以通过设置 CDN（比如 Amazon CloudFront）将 web 资源较快速地发送给用户浏览器。

webresource.cacheSeconds=0

#指定资源的缓存时间间隔，单位为秒。设置为 0 表示不使用缓存（仅适合用于开发阶段）；设置为负数（如-1）表示总是缓存；在生产环境中一般取 30～60 之间的数值即可。

webresource.local.path=file:src/test/resources/sample/web

#指定 web 资源的路径。

webresource.local.location=/web/

#指定访问 web 资源的 URL 路由。

webresource.blobstore.bucketName=test-clobaframe-bucket

#指定当使用 Blobstore 托管 web 资源时所采用的 bucket 的名称

webresource.blobstore.keyNamePrefix=r-

#储存于 Blobstore 的资源的名称的前缀。

webresource.blobstore.location=https://s3.amazonaws.com/test-clobaframe-bucket/

#web 资源的访问 URL。

webresource.blobstore.sync=true

#是否每次启动应用程序都同步储存于 Blobstore 的 web 资源。

webresource.blobstore.autoCreateBucket=true

#在同步的时候，如果指定 bucket 不存在，是否自动创建 bucket。

webresource.blobstore.deleteNoneExists=true

#同步时是否删除已经不再存在的 web 资源，即位于本地的已经被删除的资源。

mail.agent=null

#发送邮件的实现的名称，目前的实现有：null，smtp

mail.smtp.host=smtp.gmail.com

mail.smtp.port=587

mail.smtp.tls=true

mail.smtp.loginName=test@gmail.com

mail.smtp.loginPassword=no

mail.smtp.fromAddress=test@gmail.com

5 编译、测试和安装

编译需要如下软件和工具：

- Java SDK 6+
- Apache Maven 3

运行单元测试还需要：

- Memcached

5.1 检出源代码

建议把源码检出到如下目录：

~/projects/archboy/clobaframe

5.2 配置 Apache Maven 3

Maven 是项目管理和构建工具，Clobaframe 项目使用 Maven 管理和编译。在 Linux 环境下一般能从各个发行版自身的包管理工具直接安装。比如在 ArchLinux 下运行如下命令：

```
$ sudo pacman -S maven
```

除此之外也可以从官方网站下载安装，官方网站是 http://maven.apache.org/。下载已编译的版本然后解压到任意目录，比如 ~/programs/maven。然后在 ~/.bashrc 配置文件里加入如下环境变量：

```
export M2_HOME=~/programs/maven
export PATH=$PATH:$M2_HOME/bin
```

使用 source 命令加载新环境配置

```
$ source ~/.bashrc
```

然后转到任意一个目录，运行如下命令，如果出现正确的版本信息则说明 Maven 已经配置成功。

```
$ mvn -v
```

5.3 配置 memcached

可以使用各个发行版的包管理器安装 memcached ，如在 Arch Linux 下执行：

```
$ sudo pacman -S memcached
```

建议设置 memcached 以系统守护程序（daemon）形式启动，免去每次运行开发或者调试都要手动启动 memcached 的麻烦，比如，假设操作系统使用 systemd 管理系统守护程序，则执行：

```
$ sudo systemctl enable memcached.service
```

```
$ sudo systemctl start memcached.service
```

memcached 默认监听 tcp 端口 11211，可以通过检查 11211 端口是否打开以判断服务是否正常运行：

```
$ netstat -nat|grep 11211
```

5.4 创建本地 blobstore 目录

blobstore 模块的单元测试默认配置使用本地的 blobstore 实现。而本地 blobstore 默认使用目录 /var/lib/clobaframe 储存数据，所以需要手动创建这个目录，另外还需要把该目录的拥有者和组更改为当前用户以及所属的组，否则测试会因为没有文件访问权限而失败。

5.5 编译

在源码的首层目录，使用如下命令编译：

```
$ mvn clean compile
```

5.6 运行单元测试

在源码的首层目录，使用如下命令进行单元测试：

```
$ mvn test
```

如果看到成功信息，则说明测试已通过，然后继续进行下一步。

5.7 打包

在源码的首层目录，使用如下命令编译并打包：

```
$ mvn clean package -DskipTests=true
```

然后将会得到如下库文件：

```
./clobaframe-xxx/target/clobaframe-xxx-2.3.jar
```

你可以按需要将库文件拷贝到你的项目的 CLASS_PATH 之下，如果你的项目也是使用 Maven 管理和构建的，则最佳实践应该是将库安装到 Maven 本地库（见 5.9）。

5.8 生成 JavaDoc

如果你需要库的文档 JavaDoc，则运行如下命令：

```
$ mvn clean javadoc:jar
```

然后你将得到如下的文档包：

```
./clobaframe-xxx/target/clobaframe-xxx-2.3-javadoc.jar
```

5.9 安装 jar 库到 Maven 本地仓库

在源码的首层目录，使用如下命令编译、打包并安装到 Maven 本地库：

```
$ mvn clean install -DskipTests=true
```

使用如下命令可以同时安装项目的 JavaDoc 和 源代码 到 Maven 本地仓库：

```
$ mvn clean javadoc:jar source:jar install -DskipTests=true
```

6 在项目中使用 Clobaframe

首先你需要按照上一节的方法编译得到库文件，然后按需要将库文件拷贝到你的项目的 CLASS_PATH 之下或者安装到 Maven 本地库。

1、然后打开你的项目的 pom.xml 文件，添加 Clobaframe 的依赖项，如：

```
<dependencies>
    <!-- Clobaframe -->
    <dependency>
        <groupId>org.archboy.clobaframe</groupId>
        <artifactId>clobaframe-query</artifactId>
        <version>2.3</version>
    </dependency>

    <dependency>
        <groupId>org.archboy.clobaframe</groupId>
```

```
        <artifactId>clobaframe-media</artifactId>
        <version>2.3</version>
    </dependency>
     ......
</dependencies>
```

2、接下来在你的项目（必须是 Spring Framework IoC 的项目）的 applicationContext.xml 内增加对 org.archboy.clobaframe 的自动扫描，如：

```
<context:component-scan base-package="
        org.archboy.clobaframe">
        <context:include-filter type="annotation"
expression="org.aspectj.lang.annotation.Aspect"/>
    </context:component-scan>
```

3、按照文档配置好 clobaframe.properties。

4、使用 @Autowired 或者 @Inject 标注引用各个模块的主要接口，比如使用 Cache 模块：

```
    @Service public class MyService {
        @Autowired
        private Cache cache;
        public void test() {
            String key = "key001";
            cache.put(key, "FOO");
            String result = cache.get(key);
            if (result.equals("FOO")){
                System.out.println("It works");
            }
            cache.delete(key);
        }
```

```
}
```