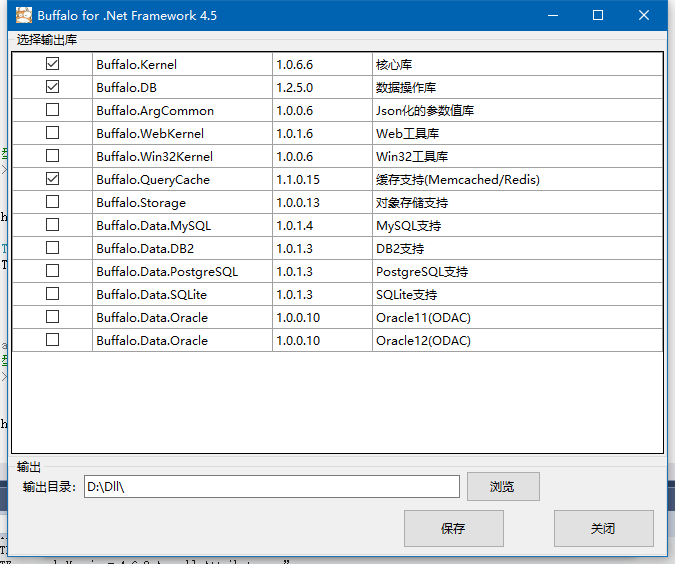
# Buffalo缓存/分布式缓存类使用

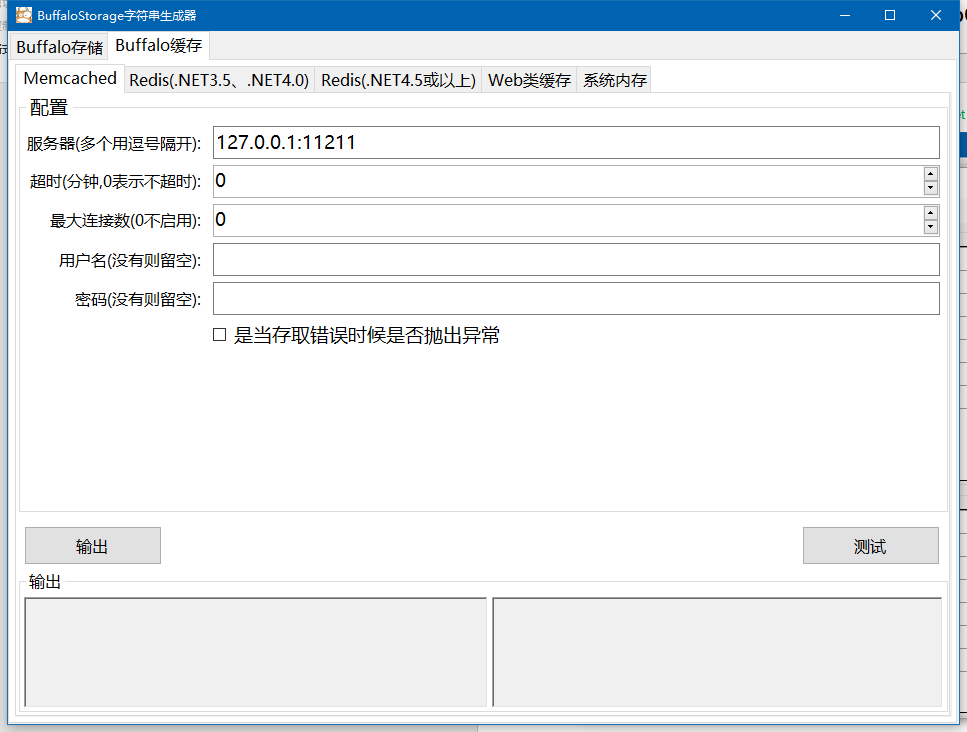
## 1.介绍

Buffalo.QueryCache是一个支持多种缓存调用的缓存封装包，通过此包可以方便使用和切换多种缓存

项目依赖文件输出可以在 Buffalo安装工具🡪选择对应的.net版本🡪输出 获取



连接字符串生成器可以在Buffalo安装工具🡪帮助🡪连接字符串🡪选择Buffalo缓存选项卡 然后选择对应的缓存选项卡，填入相关信息就可以输出对应的连接字符串



## 2.缓存类型支持

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 参数 | 连接字符串 | 说明 |
| **Memcached** | memcached | server=127.0.0.1:11211;expir=30;poolsize=30;throw=0;uid=abc;pwd=123456 | *expir=30(可选) 是表示30分钟过期，没这个项则表示不过期(可选)*  *poolsize是最大连接数，(可选)*  *throw=0 是当存取错误时候是否抛出异常(可选)*  *uid=abc 登录账户(可选)*  *pwd=123456 登录密码(可选)*  *--分布式缓存* |
| **Redis** | redis | server=127.0.0.1:6379;readserver=127.0.0.1:6381;expir=30;poolsize=30;throw=0;uid=abc;pwd=123456 | *server 是连接服务器*  *readserver 是只读服务器*  *expir=30(可选) 是表示30分钟过期，没这个项则表示不过期(可选)*  *poolsize是最大连接数(可选)*  *throw=0 是当存取错误时候是否抛出异常(可选)*  *uid=abc 登录账户(可选)*  *pwd=123456 登录密码(可选)*  *--分布式缓存* |
| **ASP.Net Cache** | web | expir=30 | expir=30 是表示30分钟过期，没这个项则默认过期方式*(可选)*  使用HttpContext.Current.Cache实现，Winform可用  *--单机缓存* |
| **系统内存** | system |  | 使用静态变量实现  *--单机缓存* |

## 3.使用

示例

QueryCache cache = CacheUnit.CreateCache(“memcached”, "server=127.0.0.1:11211,127.0.0.1:11212;expir=30;poolsize=30;throw=0");

实例化缓存后可以使用其功能

QueryCache方法

|  |  |  |
| --- | --- | --- |
| 方法 | 返回类型 | 说明 |
| SetValue(string key, int expirSeconds, object value) | void | 写入值  string key //键  int expirSeconds //过期时间(可选)  object value //值 |
| SetValue<E>(string key,  int expirSeconds, SetValueType type,E value) | void | 写入值  E //类型  string key //键  int expirSeconds //过期时间(可选)  E value //值  Type 设置值的方式(看标题4) |
| GetValue(string key) | object | 获取值  string key //键 |
| GetValue<E>(string key,E defaultValue) | E  如果不存在此键则返回defaultValue | 获取值  string key //键  E defaultValue //默认值 |
| GetValues(string[] keys) | IDictionary<string, object>  返回键值对应列表 | 批量获取值  String[] keys //要获取的键 |
| DoIncrement(string key,uint inc) | long  返回自增后的值 | 自增  string key //键  uint inc //值(可选，不填则自增1) |
| DoDecrement(string key, uint dec) | long  返回自减后的值 | 自减  string key //键  uint dec //值(可选，不填则自减1) |
| DeleteValue(string key) | void | 删除键值  string key //键 |
| Clear() | void | 清除所有数据 |
| List系列方法 | | |
| List是根据Key保存的一个集合，除了支持按索引增删改查外，还支持队列/堆栈模式的Pop和Push | | |
| ListAddValue(string key, E value, long index=-1, SetValueType setType=SetValueType.Set) | long | 增加数值到列表  Key:键  Value:要插入的值  Index:保存到的索引(0则为加到头部[push]，-1为加进尾部[push])  setType:设置值方式(看标题4) |
| ListGetValue<E>(string key, long index, E defaultValue=default(E)) | E | 获取列表指定位置的值  Key:键  Index:要获取的索引  defaultValue：如果没有值时候则返回此默认值 |
| ListGetLength(string key) | long | 获取集合长度  Key:键 |
| ListPopValue<E>(string key, bool isPopEnd=true, E defaultValue=default(E)) | E | Pop模式：移除并返回值  Key:键  isPopEnd true就从尾部弹出值，false则从头部弹出值  defaultValue 如果此集合为空，则返回此默认值 |
| ListRemoveValue(string key, object value, long count=0) | long | 从移除值  Key:键  Value 值  Count 移除几个 |
| Hash系列方法 | | |
| Hash是一个根据Key保存的一个哈希表，其内部还是根据Key/Value结构保存的数据集 | | |
| HashSetRangeValue(string key, IDictionary dicSet) | void | 批量给HashSet设置值  Key:键  dicSet:要设置的值字典 |
| HashSetValue(string key, object hashkey, object value, SetValueType type= SetValueType.Set) | bool | HashSet设置值  Key:键  Hashkey:哈希表的键  value:哈希表的值  type：设置值方式(看标题4) |
| HashGetValue<E>(string key, object hashkey, E defaultValue) | E | 获取哈希表的值  Key:键  Hashkey:哈希表的键  defaultValue：如果找不到键，则返回此默认值 |
| HashGetAllValues<K, V>(string key, V defaultValue= default(V)) | List<KeyValuePair<K, V>> | 获取所有哈希表的值  Key:键  defaultValue：如果集合的值是空，则返回此默认值 |
| HashDeleteValue(string key, object hashkey) | bool | 删除哈希表中的值  Key:键  hashkey: 要删除的哈希表的键 |
| HashDeleteValues(string key, IEnumerable hashkeys) | long | 批量删除哈希表的值  Key:键  Hashkeys:要删除的哈希表的键集合 |
| HashExists(string key, object hashkey) | bool | 哈希表的键是否存在  Key:键  hashkey: 要查找的哈希表的键 |
| HashIncrement(string key, object hashkey, long value=1) | long | 哈希表自增  Key:键  hashkey: 要自增的哈希表的键  value 自增多少 |
| HashDecrement(string key, object hashkey, long value) | long | 哈希表自减  Key:键  hashkey: 要自减的哈希表的键  value 自减多少 |

## 4.SetType类型解说

// <summary>

/// 设置值的方式

/// </summary>

[Description("设置值的方式")]

public enum SetValueType : int

{

/// <summary>

/// 新增或覆盖，任何时候都设置成功

/// </summary>

[Description("新增或覆盖")]

Set = 0,

/// <summary>

/// 覆盖模式，当存在此Key时候才能覆盖成功

/// </summary>

[Description("覆盖模式")]

Replace = 1,

/// <summary>

/// 新增模式，当不存在此Key时候才能新增成功

/// </summary>

[Description("新增模式")]

AddNew = 2,

}