实体对象高速拷贝

在实际的开发当中会出现很多实体成员的赋值操作，而赋值的对象成员 跟 被赋值的对象成员名称是一致的（编码规范，可读性）

例如：项目开发项目中，BLL层，RequirementBLL类，Pub\_ModifyRequirement方法，里面有大量的一个实体进行赋值操作，其赋值成员，与被赋值成员名称是一致的，成员类型也是一致的，十分有规律，有没有办法能让我的操作更加简单一些，我写一行代码就行了，不必写十几或几十行代码做这样 机械化的赋值操作，我们做软件开发不就是为了让电脑帮我们处理一切机械化的东西吗，答案是有的，利用反射就可以，但问题来了，少量的反射能带来编码上的便捷，但大量反射，性能就会直线下降，而现在的开发方式，大部分都基于面向实体的，如果不缓存这个反射操作将会很多,这里我们就必须要做缓存操作，缓存反射就是我们这里需要说东西，当然了这也不是新东西。

我们要缓存什么？缓存这个对象的值吗，下次访问直接返回或赋值？

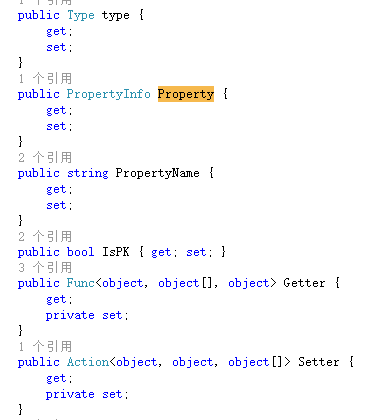
不是的，如果是这样，这就根本是个内存数据库了雏形。

我们缓存的是 成员的get set 访问器（当然了如果你有需要，也可以缓存成员的特性, 第二批次项目开发中，就有使用到 反射缓存枚举值的注释特性，该代码已经集成到内部开发框架中了）

在做日志系统的时候，这个概念，就已经日志系统里使用了。

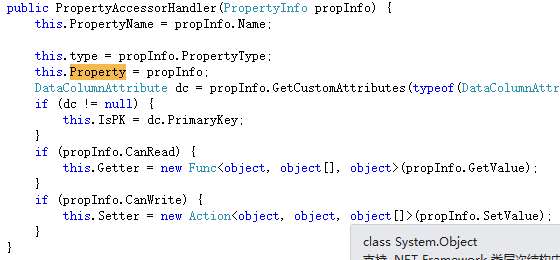
让我们从头开始说起吧，当然简单的东西，我们就忽略了。

第一步创建我们的 缓存对象集合该集合中放置了，一个我们自己定义的实体，实体成员如下：



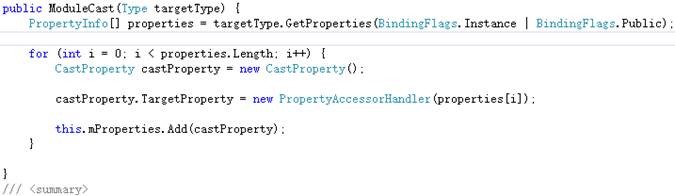
本别是 成员类型，成员的原数据，成员名称，是否主键,成员get 访问器，成员set访问器，

注意看实体构造函数



最后2个判断，是得到这个成员的元数据的set,get 访问器，并创建一个匿名的函数返回，便于后面我们对成员进行取值赋值。这个很重要，

第二步，开始反射并追加缓存集合中，这个比较简单。



注意一下我们的缓存集合的结构 ,一个类就N个成员，那么它就有N个 实体缓存对象，而我们又有N个类，所以结构是2层，N\*N

第三步，取值或赋值，

因为前面在实体缓存对象中，我们已经对gei set 访问做了匿名函数，所以使用方式就比较简单了

做一个循环：遍历一个实体 对应 实体缓存对象列表

foreach(var实体缓存对象in对应实体缓存对象列表){

    实体缓存对象. Getter(你要取数据的实体对象,null);

实体缓存对象. Setter (你要赋值数据的实体对象,你的Value, null);

}

第四步，何时反射？

Web 开发是一个并行运行的环境，大多数时候，你必须考虑多线程情况下的代码运行状况

嗯，这属于一个优化操作。

通常，我们的做法是 在缓存集合里没有检索到，就去反射

If(cache. Contains(key)){

    开始反射

}

开始赋值or取值

这种写法是错误的（如果你的业务要求一个对象必须是单例的，这种写法是致命的）

因为很可能在你正在反射的过程中，还没有加入到缓存集合中，就已经有第二个线程进入到Contains，判断，并没有找到，又开始反射了，所以我们必须加lock(作用我就不说了)

那么我们的代码就会变成这样：（不一定是this ，也可能是一个静态只读的obj）

lcok(this){

    If(cache. Contains(key)){

        开始反射

}

}

开始赋值or取值

这样对了吗？代码执行的角度上讲，对了，但优化的角度上讲错了，

因为lock的作用是，一旦有线程进入了作用域,其它线程在作用域外等待，即时你的实体缓存对象，已经反射好了，追加进了集合，但其它的线程，所有的线程，所有目前正在执行这代码的人，从并行，变成了串行，必须等待作用域中的线程离开作用域，其它线程才可以进入，注意，这是一个循环的过程，我们应该改成这样

If(cache. Contains(key)){

    lcok(this){

        If(cache. Contains(key)){

            开始反射

}

}

}

开始赋值or取值

为什么这样写大家可以想想。并不难理解，完整的代码，在附件中，稍后会加入到内部开发框架中。