**ServerFramework**

**DB模块**

**设计方案**

**更新记录**

|  |  |  |
| --- | --- | --- |
| 更新时间 | 更新人 | 内容 |
| 2014-01-20 | 史耀力 | 创建文档 |
|  |  |  |
|  |  |  |

1. 设计目标

作为框架的一个组成部分,希望DB模块可以尽可能友好的面对使用者,尽可能的屏蔽数据库的实现细节,避免使用者在逻辑开发过程中过多的思考数据层的实现.希望能够不需要使用者额外做配置工作,仅靠代码就可以直接完成对DB的操作.

1. 方案描述

* DBModule使用场景

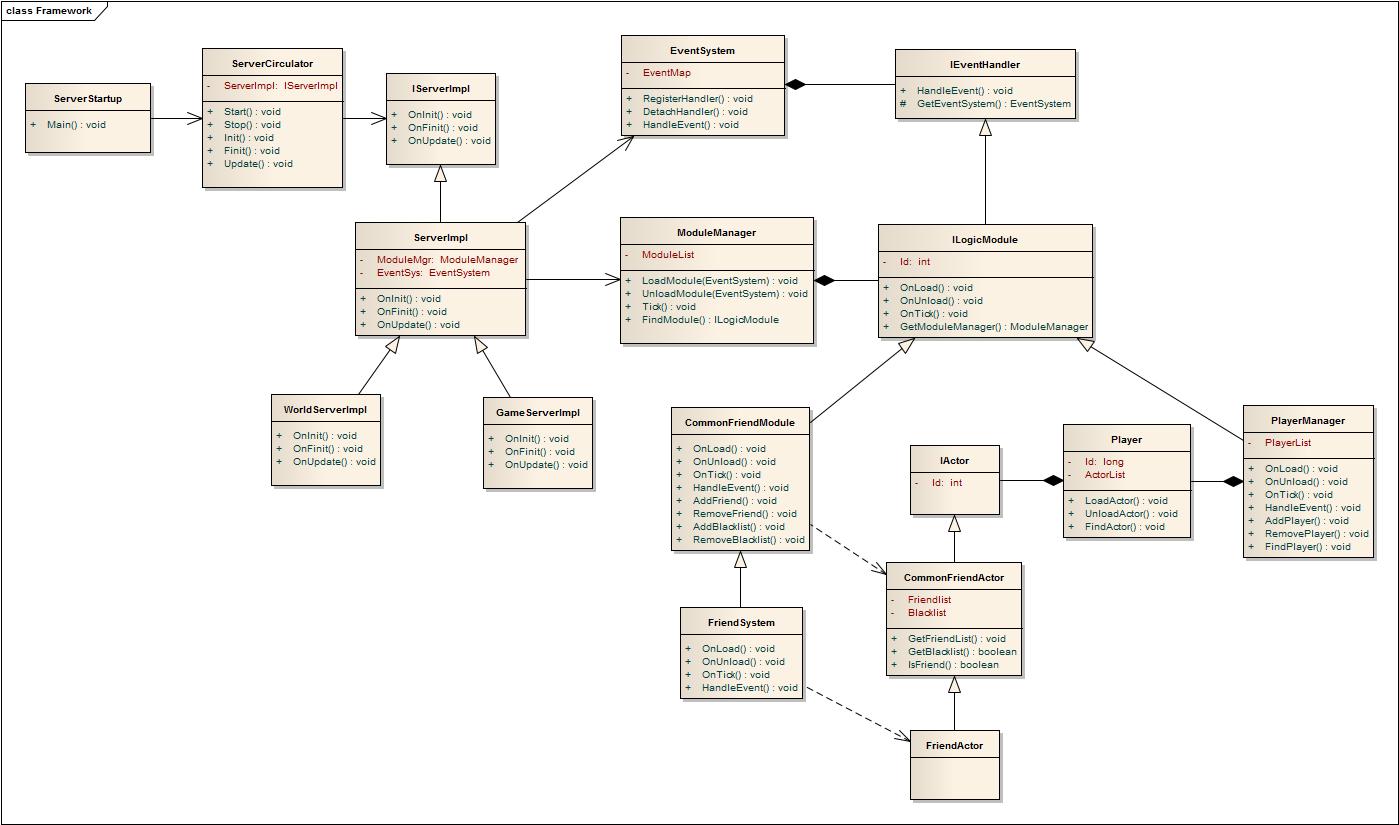
作为框架的一部分,DBModule主要有两个使用场景

1. 服务器直接连接DB
2. 服务器通过InfoSvr或其他服务器接入DB

因此DBModule需要提供模式配置,保证用户可以在两个模式间切换.实现如下图所示:



* DBModule接口



按模块化设计的程序结构,我们的数据应该由IActor所派生的各种模块Actor负责.因此定义DB模块接口如下(不完全):



模块使用自己对应的模块Actor与DB进行交互,从而实现模块化的数据读写.

* 数据存储

对于玩家数据,以如下方式在数据库中组织存储.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| RoleID | 模块1 | 模块2 | 模块3 | 模块4 |  |  |

其中RoleID是唯一索引,其余模块使用二进制K-V对方式进行存储,使用逻辑对应的模块名,如Task, Item, Attribute.

* 无配置的实现

使用C#的特性与反射机制,先由DBModule定义一系列特性,如下图所示:

namespace DBTest

{

/// <summary>

/// 特性声明

/// </summary>

[System.AttributeUsage(System.AttributeTargets.Class |

System.AttributeTargets.Struct)

]

public class ModuleName : System.Attribute

{

private string name;

public double version;

public ModuleName(string name)

{

this.name = name;

version = 1.0;

}

}

[System.AttributeUsage(System.AttributeTargets.Class |

System.AttributeTargets.Struct | System.AttributeTargets.Property)

]

public class DB : System.Attribute

{

private string key;

public double version;

public DB(string key)

{

this.key = key;

version = 1.0;

}

}

}

其中ModuleName特性用于标明模块名称,DB特性用于标明属性需要存储数据库并指明Key值.

以任务系统为范例,使用者代码如下,

[ModuleName("Task")]

public class TaskActor

{

[DB("Attr1")]

private int AttrOfInt;

[DB("Attr2")]

private char AttrOfChar;

private char AttrNotSaveToDB;

[DB("Attr3")]

private List<InnerItem> ArrayOfInnerClass;

};

public class InnerItem

{

[DB("Attr1")]

public int InnerAttrOfInt;

};

对于所有需要进行DB存储的属性,需要使用DB标示,并传入自己的属性Key,在同一个层级上,Key值是不可以重复的.一旦设定,Key值尽量不要改变,否则在数据读取时会无法正确匹配.

将TaskActor传入DBModule.Write接口, DBModule使用C#反射机制,遍历TaskActor的所有属性,查看属性是否有DB特性,如有DB特性,则以KV方式进行记录(Key值为特性所声明的字符串).

对于List,Map等常见容器,DBModule应该通过嵌套的方式予以支持.

DBModule将数据分析整理完成后,综合ModuleName特性生成对应的数据库操作语句,根据当前模式写入数据库.

同样, DBModule在读取时,也根据存储的KV数据,通过反射查找TaskActor实例属性,进行赋值.

KV数据域Actor属性的互相访问,由DBModule内部完成,不暴露给用户,用户仅需要声明对应特性即可.

数据库对应各表格,由用户自己建立.

1. 风险点分析

C#反射机制的性能尚不明确,需要进行profiling.

以KV方式存储,使用字符串做Key的话,会占用更多的存储空间.

以KV方式存储,Value应该是泛类型存储的,是否也会多占用存储空间,是否已经有较好的解决方案(protobuffer?).