**ConfigurationBuilder**

IConfigurationBulder作为配置生成器，ConfigurationBulder是默认实现

IConfigurationBulder提供注册 配置源，生成 根配置

public interface IConfigurationBuilder

{

    IEnumerable<IConfigurationSource>  Sources { get; }

    Dictionary<string, object>         Properties { get; }

    // 添加配置源

    IConfigurationBuilder     Add(IConfigurationSource source);

    // 生成根配置

    IConfigurationRoot        Build();

}

**Configuration**

IConfiguration可以看作一棵配置树

IConfiguration对象表示配置树的某个配置节点

public interface IConfiguration {

    // 获取当前配置下的所有子配置

    IEnumerable<IConfigurationSection> GetChildren ();

    // 获取某一配置

    IConfigurationSection GetSection (string key);

    IChangeToken GetReloadToken ();

    // 根据键取得配置值

    string this [string key] { get; set; }

}

IConfiguration有2个继承

IConfigurationRoot表示根配置节点

public interface IConfigurationRoot : IConfiguration

{

    // 重新加载配置

    void Reload();

}

IConfigurationSection表示非根子节点

public interface IConfigurationSection : IConfiguration {

    // 当前配置的路径

    string Path { get; }

    // 当前配置路径下的Key

    string Key { get; }

    // 如果当前节点为叶子节点，那么 Value 保存其值，否则为 NULL

    string Value { get; set; }

}

**ConfigurationSource**

IConfigurationSource表示配置源，其提供了一个方法Build用于生成配置提供者

public interface IConfigurationSource {

    // 生成配置提供者

    IConfigurationProvider Build (IConfigurationBuilder builder);

}

从这里我们可以意识到，IConfigurationSource并不提供数据访问，而提供数据访问的是IConfigurationProvider

**ConfigurationProvider**

IConfigurationProvider代表配置数据提供者，提供数据访问

public interface IConfigurationProvider {

    // 加载配置数据

    void Load ();

    // 试图获取数据

    bool TryGet (string key, out string value);

    // 设置配置数据（这个配置一般不持久化到物理数据中）

    void Set (string key, string value);

    // 获取某个配置节点下的子节点Key

    IEnumerable<string> GetChildKeys (IEnumerable<string> earlierKeys, string parentPath)

}

IConfigurationProvider有一个默认的抽象实现ConfigurationProvider

一般具体实现都继承ConfigurationProvider

public abstract class ConfigurationProvider : IConfigurationProvider

{

    // ConfigurationProvider实现了其他方法，但没有实现Load方法

    // 因为Load方法涉及到不同的数据源数据的载入

    public virtual void Load()

    {}

    ...

}

**UML关系图**

