配置文件 Configuration

本章节内容由 **Time_Traveller** 自行撰写，供IC插件开发者学习。

Yaml的配置文件格式广泛运用在Bukkit各类大型插件当中。为什么要使用Yaml而不通过File类直接读取文件信息并进行解析，原因想必大家也都已经猜到了：**Bukkit直接就提供了操作Yaml格式配置文件的API！** 那我何必大费周章还要自己写一份文件的解析格式呢？况且Yaml的使用那么方便！

接下来大家就随着我的步伐，一起揭开Yaml神秘的面纱。

Yaml的基础格式

首先，Yaml文件的格式应该是这样的：

```
第一层1:
  第二层1:
    第三层1: ...
    第三层2: ...
  第二层2:
    第三层3: ...
    第三层4: ...
第一层2:
  第二层3: ...
  第二层4: ...
```

其中`...`代表值，可以是int, float, String, List<String>, 甚至是ItemStack等诸多可读取的内容。每一层与前一层区分的标志即为每一行前面的空格数量。第一层顶格，第二层前面空**2**格，第三层前面空**4**格，以此类推。

上面文件中的从属关系应是这样的：

首先第三层1和2从属于第二层1，第三层3和4从属于第二层2，而第二层1和2又从属于第一层1，第二层3和4从属于第一层2。仔细观察，你可以发现，Yaml的格式类似于一个嵌套的表(Map)，每一层都有对应的键和值，以此类推下去，直到最后一层。

接下来让我们来看看`YamlConfiguration`的源码是如何编写的。进入`YamlConfiguration`，再进入`FileConfiguration`，再进入`MemoryConfiguration`，再进入`MemorySection`。到这里，我们发现`MemorySection`实现了一个叫做`ConfigurationSection`的接口，查看其中的抽象方法，发现里面有非常多的set，get方法，这就是我们要找的东西了。

查看其实现类`MemorySection`，截取其中部分代码片段：

```
public class MemorySection implements ConfigurationSection {
    protected final Map<String, Object> map = new LinkedHashMap();
    private final Configuration root;
    private final ConfigurationSection parent;
    private final String path;
    private final String fullPath;
```

```
    // ...

    protected MemorySection(@NotNull ConfigurationSection parent, @NotNull String
path) {
        Validate.notNull(parent, "Parent cannot be null");
        Validate.notNull(path, "Path cannot be null");
        this.path = path;
        this.parent = parent;
        this.root = parent.getRoot();
        Validate.notNull(this.root, "Path cannot be orphaned");
        this.fullPath = createPath(parent, path);
    }

    // ...
}
```

我们先看其中的成员变量。观察变量名称即可发现，`map`应是一个表，`root`代表存储它的文件，`parent`即代表它的上级节点。容易发现，它的结构像是**一棵树**，与我们之前所展示的格式不谋而合。根据推测，这个类的对象会代表配置文件中的**一个层**，其中`map`用于存储这一层当中的数据，`parent`代表它的上一层。

通过上述讲解，相信你已经大概了解了Yaml配置文件的基本格式，接下来便是如何编写一个良好的Yaml配置文件供插件使用者阅读了。

编写一个配置文件

绝大部分插件都有配置文件。配置文件可以让插件使用者根据自身需求对插件的某些参数进行自定义配置，无需每一次修改都要直接改插件的源码。因此，我们建议你在开发插件的过程当中（极小规模插件除外），应当尽量多使用配置文件，增加配置文件中的可配置项，提高插件的可调节性。

接下来，我将截取`Residence`插件`config.yml`配置文件的一个片段为大家讲解如何编写一个良好的配置文件。

```
# These are Global Settings for Residence.
Global:
  # Starts UUID conversion on plugin startup
  # DONT change this if you are not sure what you doing
  UUIDConvertion: false
  # STRONGLY not recomended to be used anymore. Only enable if you are sure you
want to use this
  # If you running offline server. Suggestion would be to keep this at false and
base residence ownership from UUID and not on players name
  OfflineMode: false
  # Players with residence.versioncheck permission node will be noticed about new
residence version on login
  versionCheck: true
  # This loads the <language>.yml file in the Residence Language folder
  # All Residence text comes from this file. (NOT DONE YET)
  Language: Chinese
  # Wooden Hoe is the default selection tool for Residence.
  SelectionToolId: WOODEN_HOE
  Selection:
```

```
    # By setting this to true, all selections will be made from bedrock to sky
ignoring Y coordinates
    IgnoreY: false
    # When this set to true, selections inside existing residence will be from
bottom to top of that residence
    # When this set to false, selections inside existing residence will be exactly
as they are
    IgnoreYInSubzone: false
    # Defines height of nether when creating residences. This mostly applies when
performing commands like /res select vert or /res auto which will expand residence
to defined height
    # This cant be higher than 319 or lower than 1
    netherHeight: 128
    # By setting this to true, player will only pay for x*z blocks ignoring height
    # This will lower residence price by up to 319 times, so adjust block price
BEFORE enabling this
    NoCostForYBlocks: false
    # Enable or disable world edit integration into Residence plugin
    WorldEditIntegration: true

# ...
```

首先，配置文件的开头应该有相关说明，介绍本配置文件的用途。接着，同类的参数应该放在同一层当中，例如上面的`IgnoreY`，`IgnoreYInSubzone`等选项都放入了`Selection`中。除此之外，每一个选项的名称应该**通俗易懂**，且其头顶最好含有相关注释，以提示插件使用者如何配置。

> 如果要在配置文件中添加注释，可以使用#进行标注。
>
> 如：
>
> ```
> # 这一行代表菜单名称
> Name: "智慧IC"
> # ...
> ```
>
> #和代码注释中的//基本相同，但我们一般不将#加在一行的后面，而是选择加在该行的上面一行。

读取配置文件

写好了一个配置文件以后，接下来就是要读取你所写的配置文件当中的内容了。

首先你需要将文件从磁盘中读取到插件中，这时候你需要使用`File`类和`YamlConfiguration`类。具体操作如下：

1. **读取File文件**：初始化对应的文件，一般使用`File file = new File(plugin.getDataFolder(), "config.yml")`，其中plugin指的是你插件的实例，getDataFolder()方法可以获取到你插件所对应的插件文件夹，即`plugins\插件名称\`；`config.yml`即为配置文件名称，可以替换成其他想要读取的文件。
2. **检查文件是否存在**：使用`file.exist()`方法即可，若不存在，可使用`plugin.saveResource(String file, boolean replace)`方法将打包到插件jar中的配置文件保存到本地，然后再读取；

3. **将File读取为FileConfiguration**：使用`YamlConfiguration.loadConfiguration(File file)`即可将指定配置文件解析，生成一个`YamlConfiguration`对象。上面的方法是静态方法。获取到`YamlConfiguration`对象后，你就可以进行下面的操作了。

`ConfigurationSection`中提供了非常多获取配置文件中内容的方法，（`YamlConfiguration`的父类实现了`ConfigurationSection`这个接口）下面罗列了一部分：

方法名称	返回类型	说明
get(String path)	Object	获取指定路径所对应的值
getInt(String path)	int	获取指定路径所对应的int
getBoolean(String path)	boolean	获取指定路径所对应的boolean
getFloat(String path)	float	获取指定路径所对应的float
getDouble(String path)	double	获取指定路径所对应的double
getString(String path)	String	获取指定路径所对应的字符串
getStringList(String path)	List<String>	获取指定路径所对应的List<String>
getItemStack(String path)	ItemStack	获取指定路径所对应的物品
getConfigurationSection(String path)	ConfigurationSection	获取指定路径所对应的层

> 如果还需查阅其他方法，请点击这里查看`ConfigurationSection`的javadoc。

路径的格式：

```
Version: 1
Global:
  Timer: 100
  TimerSettings:
    allowPVP: false
    Message: "距离比赛开始还有 {time} s"
```

在上述配置文件中，如果要获取到Version的1，那么路径格式即为`Version`；如果要获取到Timer的100，那么路径格式即为`Global.Timer`；如果要获取到allowPVP的false，那么路径格式为`Global.TimerSettings.allowPVP`，以此类推。获取Version应使用`getInt("Version")`，获取allowPVP应使用`getBoolean("Global.TimerSettings.allowPVP")`，而获取Message应使用`getString("Global.TimerSettings.allowPVP")`。我们一般不直接使用`get(...)`方法。

在Yaml中，List<String>的格式一般如下：

```
# 含有3个String的List<String>
List:
- "time"
- "wpk"
- "Hydrogen"
```

```
# 空的List<String>
Empty: []
```

在插件的`onEnable`方法执行时，就应该对配置文件进行读取操作了。读取后的数据可以保存在插件中的某一处，供其他方法调用。

> 在读取数据之前，可以使用`contains(String path)`的方法检测该路径是否有对应值。使用`getString`等方法，如果对应路径不存在会返回null；直接使用`getInt`等方法，如果数据类型不匹配可能会报`java.lang.NumberFormatException`等异常。

保存数据到配置文件当中

有的时候我们不仅仅需要从配置文件当中读取数据，还需要保存某些数据到配置文件当中，那么这时候我们就需要使用`set`方法了。

```
void set(String path, Object value)
```

使用这个方法可以将指定的值设置到某个路径当中。如果该路径不存在，那么会创建该路径。**但请注意，此时的数据还未保存到本地文件中，而仅仅只是存在于该配置文件的对象中。**

进行完数据的set操作后，可以调用方法`void save(File file)`保存相应配置文件。该方法可能会抛出`IOException`的异常，记得使用try catch块捕获异常或将异常抛出。

小结

写了这么多，其实挺唠叨的，但我也是想一次把尽可能多的东西讲明白。如果还有任何问题，可以询问开源社区内人员。当然最重要的还是练习，看完以后，一定记得自己写一两个示例插件进行练习，熟练掌握这一方面的知识。

关于读取配置文件方面，还有遍历配置文件的部分没有讲，但这一部分在这个教程中已经讲解，这里就不再赘述。**但记得先看完本教程再去看上面那个教程**，否则你会看不懂。

祝各位在插件开发的旅程中一路顺利！