

Tokit库

使用手册

洪坤安(psydollar@gmail.com)

©2014 年 10 月 8 日 TOKIT

| | |
|--|-----------|
| 1. Tokit 库简介..... | 1 |
| 1.1 简介..... | 1 |
| 1.2 Tokit 可以做到什么 | 1 |
| 1.3 Tokit 的由来 | 2 |
| 1.4 如何使用 Tokit..... | 2 |
| 1.5 示例..... | 6 |
| 2. 主流技术简介..... | 10 |
| 2.1 使用 xml 文件..... | 10 |
| 2.2. excel 转 csv 文件 | 18 |
| 2.3. 直接存在数据库中 | 19 |
| 2.4 流程小结 | 20 |
| 3. 关于 Tokit 库的一些补充..... | 21 |
| 3.1 问：一般的做法不是已经很方便了吗，为什么还要有 Tokit..... | 21 |
| 3.2 问：如果希望对 Tokit 生成的文件进行调整，该怎么做？ | 21 |
| 3.3 问：Tokit 生成的 c++代码载入 xml 的速度快吗？ | 22 |

1. Tokit 库简介

1.1 简介

Tokit 是一个小工具库，服务于网游配置数据场景，用于方便策划在 excel 中编辑配置数据、程序人员根据 excel 中的数据编写对应代码，旨在方便程序和策划的对接，改善数据制作流程以实现快速开发。目前仅支持 c++。

Tokit 的发布地址是：<https://github.com/tokitgit/tokit>

1.2 Tokit 可以做到什么

tokit 支持只通过一个 excel 就生成对应的 c++文件和 xml 数据文件，并支持生成 xsd 文件。其中，c++文件将包含自动载入 xml 的方法并生成出必要的接口，可以减小程序人员方面的工作量。

tokit 目前仅支持生成 c++文件。

1.3 Tokit 的由来

12年的时候，我还在一家端游研发公司工作做服务器研发，其时，因项目需要，由我负责对原有的配置数据导入导出流程进行整理，并被明确要求通过对象序列化的技术进行配置数据载入速度优化。

当时服务端(c++)采用的是 CodeSynthesis XSD 库，虽然成功完成了优化，但基于这个库流程的繁琐和复杂仍给我留下深刻印象，也因此，我开始留意其他项目采用的配置数据流程。

后来，公司内开新的页游项目，我又根据主管以前项目的经验，抛弃了 CodeSynthesis XSD 库，基于《游戏编程精粹 4》中《使用 XML 而不牺牲速度》文章提供的 xdstoolkit 1.03 库重新构建了一整套新的流程。

由于天生为速度而生，所以，虽然 xdstoolkit 不易理解，且与流程的契合仍然不足，在编译速度和处理速度还是给我留下了深刻的印象。

也因此，尽管后来去了新公司，并大概了解到了更多项目通行的做法，如 excel 转 csv、excel 中使用 vba 宏、直接存数据库，我仍然觉得 xdstoolkit 的优势实在巨大。于是，尽管已经拿不到我自己写的源码和齐备的文档，我还是决定重写一套。

然而，时过境迁，在重写的过程中，也慢慢意识到快速开发才是关键，对于服务端来说配置数据的载入速度如何似乎并不显得如何重要，我也便渐渐萌发了写一套单独的，仅为了快速开发的 c++小工具的想法。

这个工具将专门服务于网游的配置数据过程，旨在方便策划人员配置数据以及 c++程序方面与数据的载入、对接，并尽可能的方便使用。

工具的名称从 xdstoolkit 中截取，取 to 和 kit 的结合。

这也就是 tokit 库的由来。

1.4 如何使用 Tokit

1.4.1 使用 Tokit 命令

使用 tokit 要求有一个符合 tokit 格式的 excel 文件，使用方式如下

```
tokit.exe <excel 文件>
```

```
[-xsd <放置xsd 文件的目录>] |
```

```
[-c++ <h 模板> <cpp 模板> <放置c++文件的目录>] |
```

```
[-saveasxml <放置xml 文件的目录>]
```

选项:

```
-xsd
```

根据 excel 生成 xsd 文件

```
-c++
```

根据 excel 生成 c++文件

```
-saveasxml
```

将 excel 文件导出成为 xml 数据文件

注意:

其中的 excel 文件必须符合 tokit 要求的格式，否则将提示出错

比如

```
假设有一个符合 tokit 格式的 excel 文件《装备.xlsx》，里面有一个工作表 item，则

tokit.exe ./装备.xlsx -xsd ./xsd/ 「将在 xsd 目录下生成 item.xsd 文件」
tokit.exe ./装备.xlsx -c++ ./template/c++_template.h ./template/c++_template.cpp ./c++/ 「将在 c++ 目录下生成 itemh
和 item.cpp 文件」
tokit.exe ./装备.xlsx -saveasxml ./xml/ 「将在 xml 目录下生成 item.xml 数据文件」

这些参数可结合起来：
tokit.exe ./装备.xlsx -xsd ./xsd/ -c++ ./template/c++_template.h ./template/c++_template.cpp ./c++/
-savesaxml ./xml/ 「将生成 item.xsd、item.h、item.cpp 和 item.xml 文件」
```

1.4.2 制作 Tokit 所需要的 excel

tokit 需要的的 excel 格式如下表：

| 程序英文名 | | 最多有几条数据（0 表示不限） | | 文件定义区 | |
|-----------------------------------|--|-----------------|-----------|-------|-------|
| 必填，例如:do taequipcfg，决定生成的程序文件名和类名 | | 0 | | | |
| 字段 1 的英文名，例如:id | | 字段 2 的英文名 | 字段 3 的英文名 | ... | 字段定义区 |
| 字段类型，必填，例如:uint | | ... | ... | ... | |
| 字段属性，选填，例如:唯一 | | | | | |
| 字段 1 的描述，例如：装备 Id，用于标识装备 | | 字段 2 的描述 | 字段 3 的描述 | ... | 配置数据区 |
| 数据区域..... | | ... | ... | ... | |
| 数据区域..... | | ... | ... | ... | |
| ... | | ... | ... | ... | |

其中，excel 分为 3 个区域，文件定义区、字段定义区、配置数据区，这 3 块区域的起始行都必须固定，否则无法被 tokit 识别。

我们以一个典型的装备表为例，装备表的英文名称为 dotaequipcfg，详细各个单元格的含义可参见下图（可放大查看）。

tokit的配置定义

| A | B | C | D | E | F | G | H |
|----|--------------|-----------------|--------------------------|-------|--------|--------|----------|
| 1 | 程序总文名 | 最多有几条数据 (0表示不限) | | | | | |
| 2 | dotaequipcfg | 0 | | | | | |
| 3 | | | | | | | |
| 4 | id | name | desc | price | isdrop | attack | bornlist |
| 5 | uint | string | string | uint | bool | int | uint8 |
| 6 | 唯一 | 唯一 | | | | | 数组 |
| 7 | | | | | | | |
| 8 | 物品ID | 物品名称 | 描述 | 售出价格 | 是否死亡掉落 | 攻击力 | 合成列表 |
| 9 | 1 | 圣剑遗物 | | 1000 | 0 | 80 | |
| 10 | 2 | 恶魔刀链 | | 1050 | 0 | 60 | |
| 11 | 3 | 圣剑 | 圣剑在法师之战中由神亲自授予叛军之手，死亡后掉落 | 1100 | 0 | 300 | 1,2 |
| 12 | 4 | 斯梅斯特的掠夺 | | | 0 | 32 | 1,1 |
| 13 | 5 | 活力之球 | | | 0 | 10 | |
| 14 | 6 | 魔法之心合成书 | | | 0 | -100 | |
| 15 | 7 | 魔法之心 | 每秒恢复最大生命值的%2 | | 1 | 100 | 4,5,6 |

这是一个典型的装备表示例

配置描述区

| A | B |
|---|-----------------|
| 1 | 程序总文名 |
| 2 | dotaequipcfg |
| | 最多有几条数据 (0表示不限) |
| | 0 |

| | |
|---|--------------|
| 1 | 程序总文名 |
| 2 | dotaequipcfg |

程序英文名 (必填) : 决定导出的文件和对应结构体名称

最多有几条数据 (必填) : 分为1和0

1: 只有一行数据, 生成的程序代码载入时将只读取一行, 并且只分配一行的空间
0: 可能有很多行数据, 生成的程序代码载入配置时将把数据存入数组

字段定义区

| A | B | C | D | E | F | G | H |
|---|------|--------|--------|-------|--------|--------|----------|
| 4 | id | name | desc | price | isdrop | attack | bornlist |
| 5 | uint | string | string | uint | bool | int | uint8 |
| 6 | 唯一 | 唯一 | | | | | 数组 |

4 id 字段的英文名 (必填), 比如: id, name, desc等
5 uint 字段的类型 (必填)

string 字符串
bool bool类型
char 单字节整数
int16 双字节整数
int 四字节整数
int64 八字节整数
uint8 单字节无符号整数
uint16 双字节无符号整数
uint 四字节无符号整数
uint64 八字节无符号整数
float 浮点数
double 双精度浮点数

6 唯一 字段的属性

唯一: 表示所有该字段的值是不相同的, 可以根据该字段找到对应的配置, 例如, 在本装备表中, 由于id被注明唯一, 因此将自动生成id->装备的映射

主键: 跟数据库primary key一样, 一个表可以由多个字段共同组成一个主键, 比如, id和name可以共同组成主键

数组: 表示该字段的内容是一个数组, 由逗号分割, 例如, '1,3,4,6,11' 将被分割成数组[1,3,4,6,11]

比如
uint数组 = std::vector<uint>, 如"-10,11,12"
string数组 = std::vector<string>, 如"first,second,third"
float数组 = std::vector<float>, 如"0.1,0.2,-1.44,3.14"

集合: 表示该字段的内容是一个集合, 由逗号分割, 例如, '1,3,4,6,11' 将被分割成集合[1,3,4,6,11]

比如
uint集合 = std::set<uint>, 如"-10,11,12"
string集合 = std::set<string>, 如"first,second,third"
float集合 = std::set<float>, 如"0.1,0.2,-1.44,3.14"

配置数据区

| 8 | 物品ID | 物品名称 | 描述 | 售出价格 | 是否死亡掉落 | 攻击力 | 合成列表 | 合成数组 |
|----|------|---------|--------------------------|------|--------|------|-------|-------|
| 9 | 1 | 圣剑遗物 | | 1000 | 0 | 80 | | |
| 10 | 2 | 恶魔刀链 | | 1050 | 0 | 60 | | |
| 11 | 3 | 圣剑 | 圣剑在法师之战中由神亲自授予叛军之手，死亡后掉落 | 1100 | 0 | 300 | 1,2 | 1,1 |
| 12 | 4 | 斯梅斯特的掠夺 | | | 0 | 32 | | |
| 13 | 5 | 活力之球 | | | 0 | 10 | | |
| 14 | 6 | 魔法之心合成书 | | | 0 | -100 | | |
| 15 | 7 | 魔法之心 | 每秒恢复最大生命值的%2 | | 1 | 100 | 4,5,6 | 1,1,1 |

8 物品ID 字段的中文名 (必填) : 将被导入到程序代码中作为注释, 比如: 物品ID、物品名称、描述

1.4.3 在项目中接入 tokit

c++项目想要和 tokit 对接，必须先包含 tokit 需要的文件

1. `<type.h>`、`<tokit_util.h>`、`<tokit_util.cpp>`
2. `rapidxml` 第三方开源库（用于载入 xml 数据）

项目跟tokit对接后，载入数据的工作很简单，包含所需文件后，添加一行语句即可：

```
xxxxx::instance().load();
```

详细可参照 `example\c++\c++_example` 项目

1.5 示例

我们以最普遍的场景<装备表>、<英雄表>为例，除了一般的装备数据和英雄数据以外，现假设有需求：其中的一些装备可由其他装备合成而来。

于是策划制订了如下的 excel 表格（装备名和英雄名直接取自 dota）。

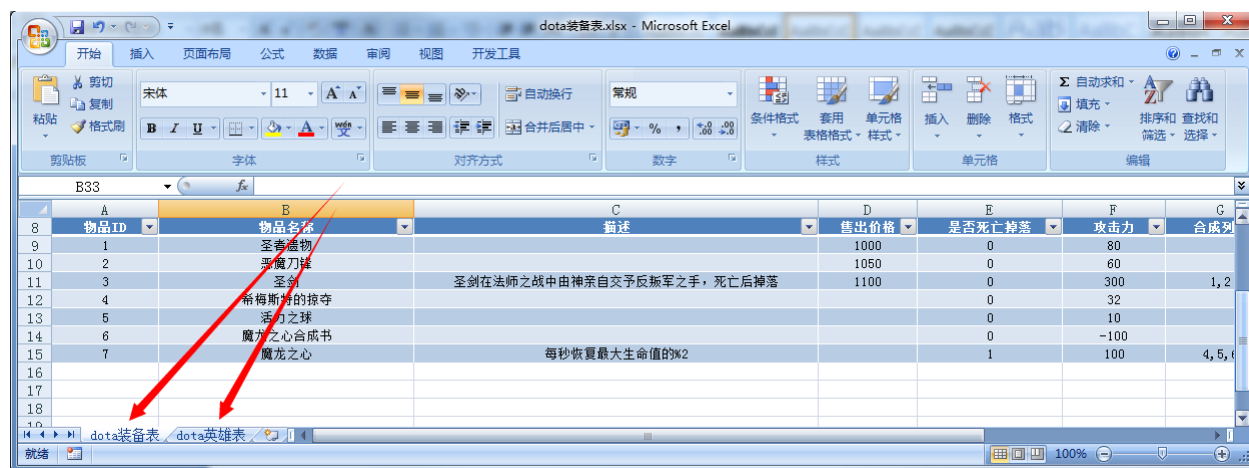
| 物品 ID | 物品名称 | 描述 | 售出价格 | 是否死亡掉落 | 攻击力 | 合成列表 | 合成数量 |
|-------|---------|---------------------------|------|--------|------|---------|---------|
| 1 | 圣者遗物 | | 1000 | 0 | 80 | | |
| 2 | 恶魔刀锋 | | 1050 | 0 | 60 | | |
| 3 | 圣剑 | 圣剑在法师之战中由神亲自授予反叛军之手，死亡后掉落 | 1100 | 0 | 300 | 1, 2 | 1, 1 |
| 4 | 希梅斯特的掠夺 | | | 0 | 32 | | |
| 5 | 活力之球 | | | 0 | 10 | | |
| 6 | 魔龙之心合成书 | | | 0 | -100 | | |
| 7 | 魔龙之心 | 每秒恢复最大生命值的%2 | | 1 | 100 | 4, 5, 6 | 1, 1, 1 |

dota 装备表（<合成列表> 字段表示合成该装备所需的装备 Id 列表，由逗号分割）

| 英雄 ID | 英雄名称 | 力量 | 敏捷 | 智力 |
|-------|------|------|----------|----------|
| 1 | 小黑 | 10.5 | 20.00001 | 301.1111 |
| 2 | 风行 | 10.5 | 20.00001 | 301.1111 |
| 3 | 凤凰 | 10.5 | 20.00001 | 301.1111 |
| 4 | 斧王 | 10.5 | 20.00001 | 301.1111 |
| 5 | 宙斯 | 10.5 | 20.00001 | 301.1111 |
| 6 | 虚空 | 10.5 | 20.00001 | 301.1111 |
| 7 | 猴子 | 10.5 | 20.00001 | 301.1111 |

dota 英雄表

两个表分别存放在一个 excel 的 2 个工作表中



这时候，策划的工作已经完成了。

一般情况下，程序人员要想使用里面的数据，得开始准备写一些载入代码以及必要的查找装备和伙伴接口。

但使用 tokit 可以减少大部分的工作量，因为很多重复的代码都可以由 tokit 自动生成出来，免去重复劳动。

不过此时的格式是无法被 tokit 识别的，因为缺少每个字段的类型信息，所以需要程序人员再定义好各个字段的类型，并配置好生成的 c++类的名称。

➡ 从这一步开始接入 tokit。

程序人员补上其余信息，使装备表和英雄表变成如下。

| 程序英文名 | 最多有几条数据（0 表示不限） |
|------------------|-----------------|
| dotaequipmentcfg | 0 |

| id | name | desc | price | isdrop | attack | bornlist | bornnum |
|------|--------|--------|-------|--------|--------|----------|---------|
| uint | string | string | uint | bool | int | uint | uint8 |
| 唯一 | 唯一 | | | | | 数组 | 数组 |

| 物品 ID | 物品名称 | 描述 | 售出价格 | 是否死亡掉落 | 攻击力 | 合成列表 | 合成数量 |
|-------|---------|---------------------------|------|--------|------|---------|---------|
| 1 | 圣者遗物 | | 1000 | 0 | 80 | | |
| 2 | 恶魔刀锋 | | 1050 | 0 | 60 | | |
| 3 | 圣剑 | 圣剑在法师之战中由神亲自交予反叛军之手，死亡后掉落 | 1100 | 0 | 300 | 1, 2 | 1, 1 |
| 4 | 希梅斯特的掠夺 | | | 0 | 32 | | |
| 5 | 活力之球 | | | 0 | 10 | | |
| 6 | 魔龙之心合成书 | | | 0 | -100 | | |
| 7 | 魔龙之心 | 每秒恢复最大生命值的%2 | | 1 | 100 | 4, 5, 6 | 1, 1, 1 |

dota 装备表（相比上面的装备表，添加了：装备表的英文名 dotaequipmentcfg、各个字段的英文名和程序类型等）

| 程序英文名 | 最多有几条数据（0 表示不限） |
|----------------|-----------------|
| dotaheroconfig | 0 |

| id | name | strength | agile | intelligence |
|------|--------|----------|--------|--------------|
| uint | string | double | double | double |
| 唯一 | 唯一 | | | |

| 英雄 ID | 英雄名称 | 力量 | 敏捷 | 智力 |
|-------|------|------|----------|----------|
| 1 | 小黑 | 10.5 | 20.00001 | 301.1111 |
| 2 | 风行 | 10.5 | 20.00001 | 301.1111 |
| 3 | 凤凰 | 10.5 | 20.00001 | 301.1111 |
| 4 | 斧王 | 10.5 | 20.00001 | 301.1111 |
| 5 | 宙斯 | 10.5 | 20.00001 | 301.1111 |
| 6 | 虚空 | 10.5 | 20.00001 | 301.1111 |
| 7 | 猴子 | 10.5 | 20.00001 | 301.1111 |

dota 英雄表

可以注意到，〈装备表〉和〈英雄表〉的 id 和 name 都填上了〈唯一〉，表示所有装备和英雄 id 和名称都是唯一的，因此 tokit 将认为，可根据 id 和 name 找到对应的装备和伙伴。

并且，〈合成列表〉填上了〈数组〉，并且里面的值都由“，”分割开来，表示一个装备可能由多个装备合成而来。tokit 将自动将〈合成列表〉识别成数组，并在载入的时候将里面的值按“，”进行分割。

修改后的 excel 内容如下所示

开始

插入

页面布局

公式

数据

审阅

视图

开发工具

剪贴板

复制

格式刷

粘贴

格式刷

宋体

11

由于装备表排在第一个，名为 `dotaequipcfg`，于是 tokit 生成了对应的 c++ 文件 `<dotaequipcfg.h>` 和 `<dotaequipcfg.cpp>`，里面包含了 `dotaequipcfgmgr` 类及其实现，并定义了 2 个结构体：`dotaequipcfg_t` 和 `dotaheroefg_t`。如下：

```
// dota装备表
struct dotaequipcfg_t{
    typedef uint32 id_t;
    typedef std::string name_t;
    typedef std::vector<uint32> bornlistvec_t;
    typedef std::vector<uint8> bornnumvec_t;

    dotaequipcfg_t();

    id_t id; // 物品ID<<<唯一>>>
    name_t name; // 物品名称<<<唯一>>>
    std::string desc; // 描述
    uint32 price; // 售出价格
    bool isdrop; // 是否死亡掉落
    int32 attack; // 攻击力
    bornlistvec_t bornlist; // 合成列表<<<数组>>>
    bornnumvec_t bornnum; // 合成数量<<<数组>>>
};

// dota英雄表
struct dotaheroefg_t{
    typedef uint32 id_t;
    typedef std::string name_t;

    dotaheroefg_t();

    id_t id; // 英雄ID<<<唯一>>>
    name_t name; // 英雄名称<<<唯一>>>
    double strength; // 力量
    double agile; // 敏捷
    double intelligence; // 智力
};
```

可以看到，`<合成列表>`和`<合成数量>`均已被自动识别为整型数组 `bornlistvec_t` 和 `bornnumvec_t`，程序载入的时候将自动分割 “，” 并存入数组。

并且，`dotaequipcfgmgr` 类中还定义了几个 `map` 映射，表示可由对应的 `id` 和 `name` 找到装备或者英雄

如下。

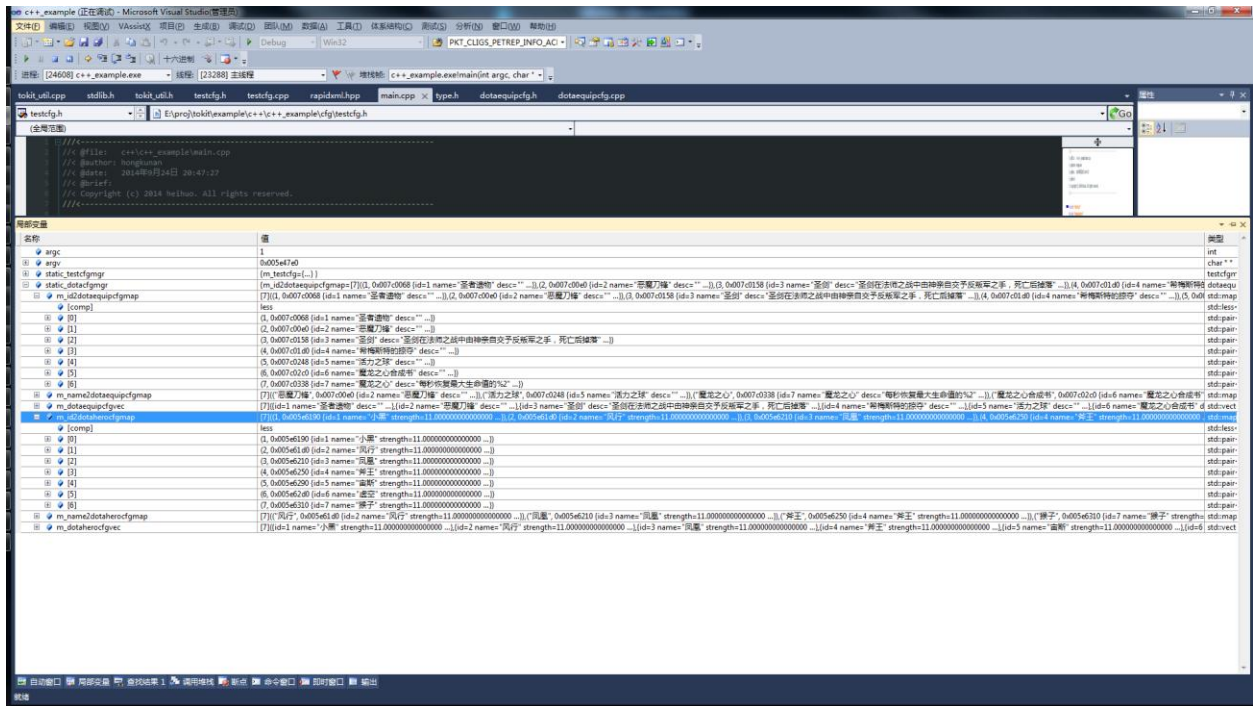
```
// dota装备表
typedef std::map<dotaquipcfg_t::id_t, dotaquipcfg_t*> id2dotaquipcfgmap; // 物品ID -> dotaquipcfg_t
typedef std::map<dotaquipcfg_t::name_t, dotaquipcfg_t*> name2dotaquipcfgmap; // 物品名称 -> dotaquipcfg_t
typedef std::vector<dotaquipcfg_t> dotaquipcfgvec;

// dota英雄表
typedef std::map<dotaheroconf_t::id_t, dotaheroconf_t*> id2dotaheroconfmap; // 英雄ID -> dotaheroconf_t
typedef std::map<dotaheroconf_t::name_t, dotaheroconf_t*> name2dotaheroconfmap; // 英雄名称 -> dotaheroconf_t
typedef std::vector<dotaheroconf_t> dotaheroconfvec;
```

项目跟tokit生成的c++文件对接后，载入数据的工作很简单，添加一行语句：

```
dotaquipcfgmgr::instance().load();
```

可以在 vs 里面调试跟踪到，调用 load() 之后，xml 数据已被载入到内存当中



同时，载入成功后可根据tokit自动生成的接口进行调用：

```
dotaquipcfg_t *dotaquipcfg = dotaquipcfgmgr::instance().get_dotaquipcfg_by_id(1);
if (NULL == dotaquipcfg){
    return;
}

dotaheroconf_t *dotaheroconf = dotaquipcfgmgr::instance().get_dotaheroconf_by_name("小黑");
if (NULL == dotaheroconf){
    return;
}

const dotaquipcfgmgr::id2dotaquipcfgmap &dotaquipcfgmap = dotaquipcfgmgr::instance().get_id2dotaquipcfgmap();
```

更加详细的用法可参照 `example\c++\c++_example` 项目或自己查看 tokit 生成的 c++ 文件接口。

2. 主流技术简介

配置数据需要的是一整套流程，为了说明为什么会专门写这个库，有必要先介绍一下我所了解的一般通行的做法。

2.1 使用 xml 文件

xml 作为项目配置数据的存储格式，具有强大的表达能力，也方便在 xml 文件内调整数据进行测试，优点主要集中在设计和开发阶段。基于 xml 的工具以及 c++ 与 xml 结合的方式比较多样，所以关于 xml 文件的部分会重点说明。

一、Excel 转 xml


一般，xml 文件都是由 excel 导出而来，以方便策划编写数据，excel 转 xml 的方式一般分以下几种

1.1 excel 中直接使用 vba 宏转成 xml

我以 <http://www.codeproject.com/Articles/6950/Export-Excel-to-XML-in-VBA> 为例，这个页面讲解了如何在 excel 中导出 xml 文件并附带了源代码。

下载示例后可以看到，excel 中的内容如下

| /student/id | /student/name | /student/age | /student/mark |
|-------------|---------------|--------------|---------------|
| 1 | Raymond | 11 | 0 |
| 2 | Moon | 11 | 100 |
| 3 | Billy | 11 | 100 |
| 4 | Pan | 12 | 80 |
| 5 | Queenie | 10 | 90 |

点击  按钮后将生成一个 xml 文件，内容如下

```
<?xml version="1.0" ?>
<data>
  <student>
    <id>1</id>
    <name>Raymond</name>
    <age>11</age>
    <mark>0</mark>
  </student>
  <student>
    <id>2</id>
    <name>Moon</name>
    <age>11</age>
    <mark>100</mark>
  </student>
  <student>
    <id>3</id>
    <name>Billy</name>
    <age>11</age>
    <mark>100</mark>
  </student>
  <student>
    <id>4</id>
    <name>Pan</name>
    <age>12</age>
    <mark>80</mark>
  </student>
  <student>
```

```
<id>5</id>
<name>Queenie</name>
<age>10</age>
<mark>90</mark>
</student>
</data>
```

1.2 excel 中直接使用 xml 映射源导出 xml

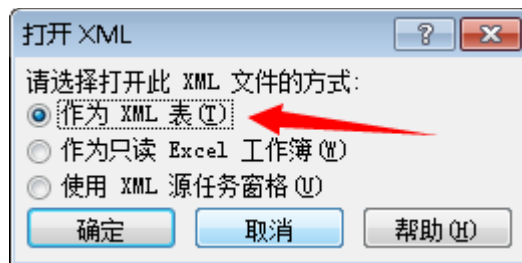
1.2.1 通过 xml 添加 xml 映射源

详见 <http://jingyan.baidu.com/article/20b68a8851721e796cec6290.html> (Excel 导出至 XML 格式) 和 <http://zfei.com/excel-to-xml/> (从 EXCEL 导出数据到 XML)。

以下面这个装备表<item.xml>为例，现在只有 3 条数据。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<equipcfgs>
  <equipcfg>
    <id>1</id>
    <name>单手剑</name>
    <lvl>1</lvl>
    <price>100</price>
  </equipcfg>
  <equipcfg>
    <id>2</id>
    <name>单手剑 2</name>
    <lvl>2</lvl>
    <price>200</price>
  </equipcfg>
  <equipcfg>
    <id>3</id>
    <name>单手剑 3</name>
    <lvl>3</lvl>
    <price>300</price>
  </equipcfg>
</equipcfgs>
```

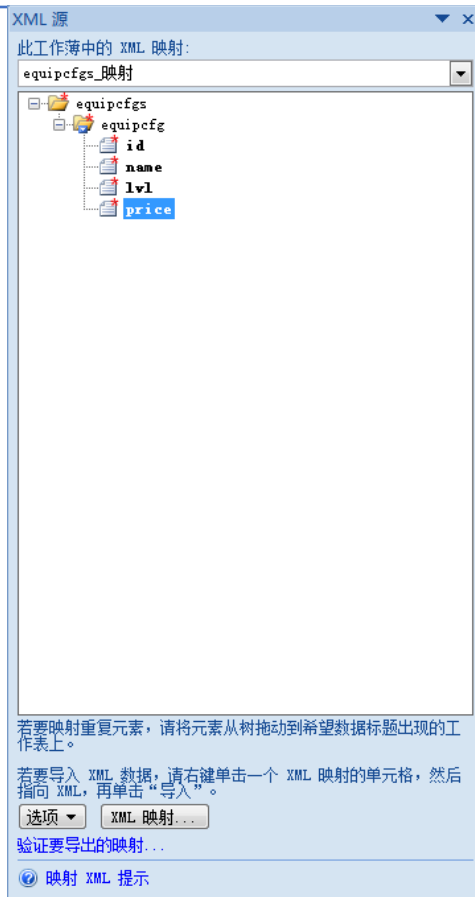
添加 xml 映射的方式有 2 种，一种是按照上面网址里的教程所说，另外一种相对来说更为方便，直接将 xml 拖动到 excel 中，在弹出的选择框中选择第一项：作为 XML 表，点击确定之后，excel 将自动识别 xml 的格式并转换为 excel 的内容。这 2 种方式的效果是一样的。



将出现如下 3 行数据

| id | name | lvl | price |
|----|-------|-----|-------|
| 1 | 单手剑 | 1 | 100 |
| 2 | 单手剑 2 | 2 | 200 |
| 3 | 单手剑 3 | 3 | 300 |

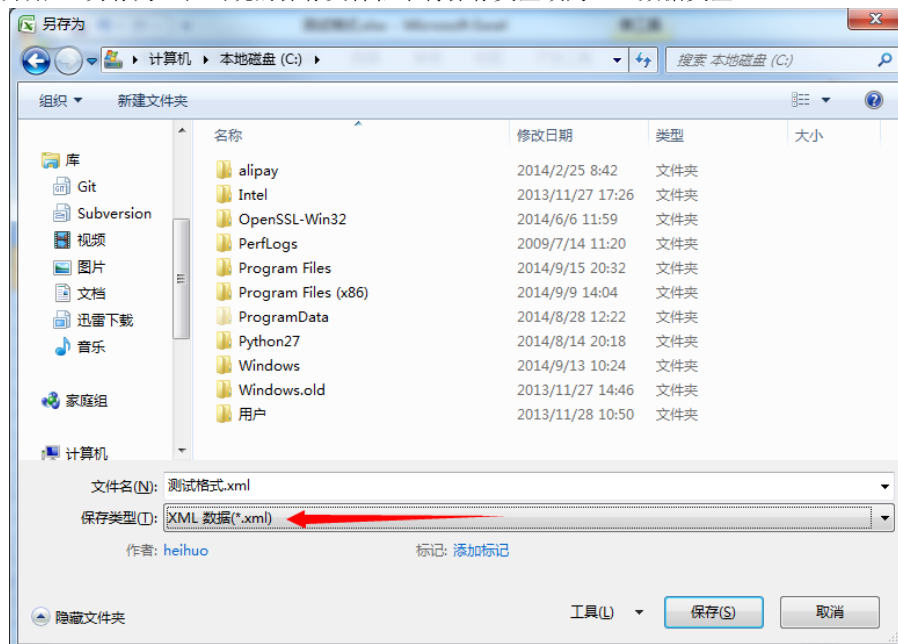
这时候可以查看 xml 映射框，可以看出 excel 已经识别出了 xml 的节点结构信息。如下图：



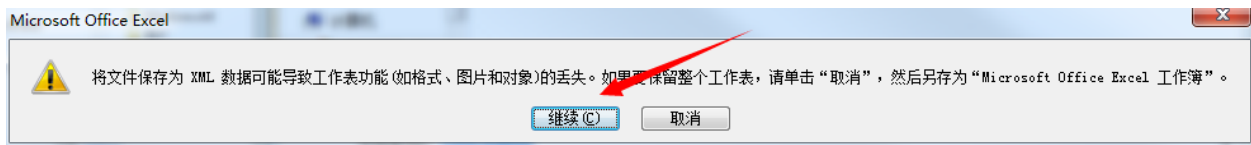
并在 excel 多添加 2 行数据

| | | | |
|---|-------|---|-----|
| 4 | 单手剑 4 | 4 | 400 |
| 5 | 单手剑 5 | 5 | 500 |

添加后点击开始 - 另存为，在出现的保存文件框中将保存类型改为 xml 数据类型



此时点击保存将弹出



最后将成功导出 xml 数据, 此时 xml 数据由原先的 3 条变成了现在的 5 条。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<equipcfgs>
  <equipcfg>
    <id>1</id>
    <name>单手剑</name>
    <lvl>1</lvl>
    <price>100</price>
  </equipcfg>
  <equipcfg>
    <id>2</id>
    <name>单手剑 2</name>
    <lvl>2</lvl>
    <price>200</price>
  </equipcfg>
  <equipcfg>
    <id>3</id>
    <name>单手剑 3</name>
    <lvl>3</lvl>
    <price>300</price>
  </equipcfg>
  <equipcfg>
    <id>4</id>
    <name>单手剑 4</name>
    <lvl>4</lvl>
    <price>400</price>
  </equipcfg>
  <equipcfg>
    <id>5</id>
    <name>单手剑 5</name>
    <lvl>5</lvl>
    <price>500</price>
  </equipcfg>
</equipcfgs>
```

1.2.2 通过 xsd 添加 xml 映射源

具体步骤与上面类似, 点击 xml 映射按钮后, 弹出添加 xml 映射框。



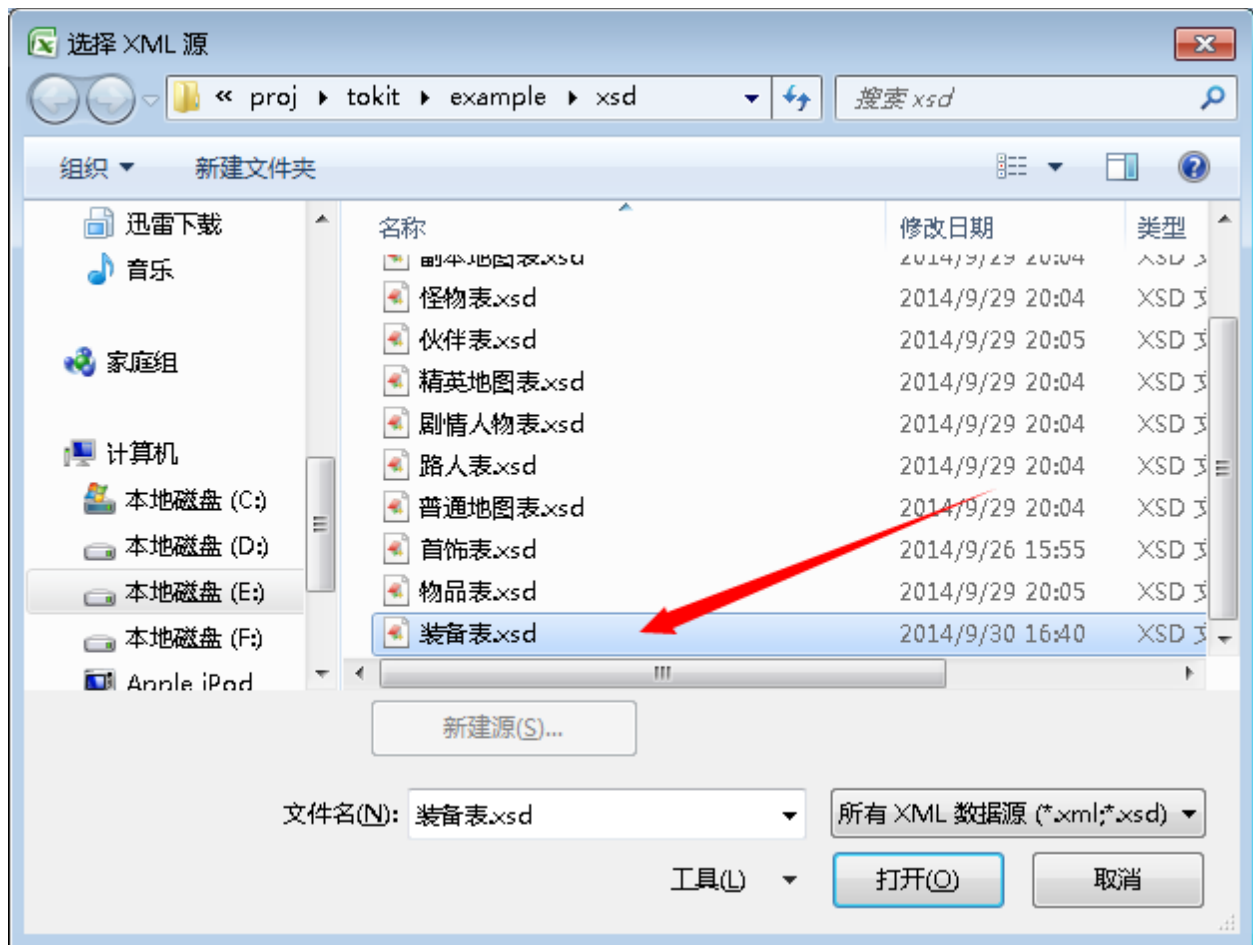
```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="equipcfg">
    <xs:complexType>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="equipcfg">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="id" type="xs:unsignedInt" />
              <xs:element name="name" type="xs:string" />
              <xs:element name="lvl" type="xs:unsignedInt" />
              <xs:element name="price" type="xs:unsignedInt" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

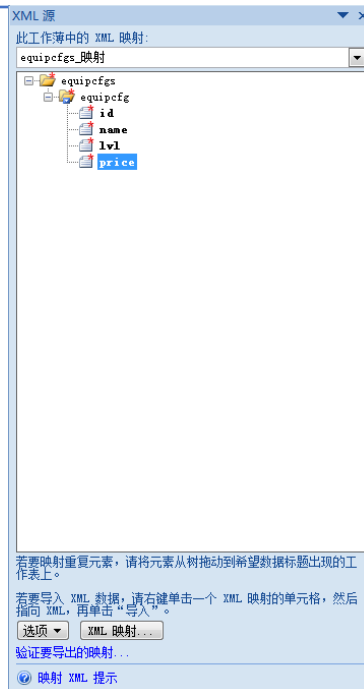
```

〈装备表.xsd〉

以上方的〈装备表.xsd〉为例，点击添加后，选中〈装备表.xsd〉然后点击打开。



成功打开后再点击确定，关闭 xml 映射框，将得到下面的图示结果。



拖动节点到 excel 中时，则有

| id | name | lvl | price |
|----|------|-----|-------|
| | | | |

编辑数据后再另存为 xml 将成功导出。

二、 c++载入 xml

2.1 硬编码载入 xml

纯粹手动编写读取和解析代码，这种方式在配置表结构不常变动以及配置表量小的情况下是最方便的。但应付比较大一点的文件量则显得工作繁琐。

2.2 xml 数据绑定

关于 xml 数据绑定与正常的手动编写 dom 解析代码，两者差别可见下面的对比

| // 手动编写 dom 解析代码 | // xml 数据绑定 |
|---|--|
| <pre>DOMElement* c = ... DOMNodeList* l; l = c->getElementsByTagName ("name"); DOMNode* name = l->item (0); l = c->getElementsByTagName ("email"); DOMNode* email = l->item (0); l = c->getElementsByTagName ("phone"); DOMNode* phone = l->item (0); cout << name->getTextContent () << ", " << email->getTextContent () << ", " << phone->getTextContent () << endl;</pre> | <pre>Contact c = ... cout << c.name () << ", " << c.email () << ", " << c.phone () << endl;</pre> |

本表格直接取自 codesynthesis 官网页面 <http://www.codesynthesis.com/products/xsd/>

xml 与对象的绑定关系可见下表:

| | |
|---|--|
| <pre><contact> <name>John Doe</name> <email>j@doe.com</email> <phone>555 12345</phone> </contact></pre> | <pre>auto_ptr<Contact> c = contact ("c.xml"); cout << c->name () << ", " << c->email () << ", " << c->phone () << endl;</pre> |
|---|--|

本表格直接取自 codesynthesis 官网页面 <http://www.codesynthesis.com/products/xsd/>

具体原理可参见 <http://www.cnblogs.com/mywoldr/archive/2012/04/24/2468908.html>, 里面的《更好的解决方案》一小栏大概讲解了 xml 与 c++对象绑定的过程。

支持 xml 数据对象绑定的 c++库有 CodeSynthesis XSD (跨平台), xdstoolkit, 以及 gsoap 等。

CodeSynthesis XSD 见 <http://www.codesynthesis.com/products/xsd/>

xdstoolkit 来源于《游戏编程精粹 4》中的《使用 XML 而不牺牲速度》, 作者 Mark T. Price, 参考资料可见 <http://blog.csdn.net/mikefeng/article/details/1327330>

gsoap 则参见 <http://www.cnblogs.com/diylab/archive/2009/02/13/1390287.html>

由于要对 c++和 xml 进行绑定, 所以需要提供结构的定义信息, 一般是要求 xsd 文件, 比如 CodeSynthesis XSD 根据 xsd 文件, 可以生成对应的 c++文件, 而有的则根据使用场景自行定义, 如 xdstoolkit 要求提供的是 c++结构体文件, 通过对文件进行语法解析, 再生成对应的 xsd 文件。

以 codesynthesis xsd 中的范例<hello.xsd>为例,

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="hello_t">
    <xsd:sequence>
      <xsd:element name="greeting" type="xsd:string" />
      <xsd:element name="name" type="xsd:string" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="hello" type="hello_t" />
</xsd:schema>
```

<hello.xsd>

通过 codesynthesis xsd 的工具将生成 c++文件<hello.hxx>和<hello.cxx>

```
class hello_t: public ::xml_schema::type
{
public:
  // greeting
  //
  typedef ::xml_schema::string greeting_type;
  typedef ::xsd::cxx::traits< greeting_type, char > greeting_traits;

  const greeting_type&
  greeting () const;

  greeting_type&
  greeting ();

  void
  greeting (const greeting_type& x);

  void
  greeting (::std::auto_ptr< greeting_type > p);

  // name
  //
  typedef ::xml_schema::string name_type;
  typedef ::xsd::cxx::tree::sequence< name_type > name_sequence;
  typedef name_sequence::iterator name_iterator;
```

```

typedef name_sequence::const_iterator name_const_iterator;
typedef ::xsd::cxx::tree::traits< name_type, char > name_traits;

const name_sequence&
name () const;

name_sequence&
name ();

void
name (const name_sequence& s);

// Constructors.
//
hello_t (const greeting_type&);

hello_t (const ::xercesc::DOMElement& e,
::xml_schema::flags f = 0,
::xml_schema::container* c = 0);

hello_t (const hello_t& x,
::xml_schema::flags f = 0,
::xml_schema::container* c = 0);

virtual hello_t*
clone (::xml_schema::flags f = 0,
::xml_schema::container* c = 0) const;

hello_t&
operator= (const hello_t& x);

virtual
~hello_t ();

// Implementation.
//
protected:
void
parse (::xsd::cxx::xml::dom::parser< char >&,
::xml_schema::flags);

protected:
::xsd::cxx::tree::one< greeting_type > greeting_;
name_sequence name_;
};

```

<hello.hxx>中的代码片段

与之对应的 xml 格式是

```

<?xml version="1.0" ?>
<!--
file : examples/cxx/tree/hello/hello.xml
copyright : not copyrighted - public domain
-->
<hello xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="hello.xsd">
  <greeting>Hello</greeting>
  <name>sun</name>
  <name>moon</name>
  <name>world</name>
</hello>

```

而在 `xdstoolkit` 中，则是根据 c++ 头文件生成对应的 xsd 文件，以下面的 `<itemprop.h>` 为例。

```

#ifndef _ItemProp_h_
#define _ItemProp_h_

#include <map>

struct ItemProp
{
  unsigned char Id;
  char Name[128];
};

```

```

    unsigned char Quality;
    unsigned char Price;
};

extern struct ItemProp *g_ItemProps;

#endif // _ItemProp_h

```

<itemprop.h>文件

与之对应的 xml 数据格式如下，每个 ItemProp 结构体将对应一个 entry 的 xml 节点

```

<?xml version="1.0" encoding="UTF-8" ?>
<ItemProps xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<ItemProp>
  <g_ItemProps>
    <entry>
      <Id>0</Id>
      <Name>物品 0</Name>
      <Quality>0</Quality>
      <Price>0</Price>
    </entry>
    <entry>
      <Id>255</Id>
      <Name>物品 255</Name>
      <Quality>255</Quality>
      <Price>255</Price>
    </entry>
    <entry>
      <Id>1</Id>
      <Name>物品 1</Name>
      <Quality>1</Quality>
      <Price>1</Price>
    </entry>
    <entry>
      <Id>254</Id>
      <Name>物品 254</Name>
      <Quality>254</Quality>
      <Price>254</Price>
    </entry>
  </g_ItemProps>
</ItemProp>
</ItemProps>

```

2.3 加速载入 xml

关于加速载入 xml 的思路，如果是采用对象序列化的技术，可以预先将 xml 读出并存入对应的对象，序列化成二进制串存入文件中，在读取的时候再进行反序列化操作即可，由于省去了 xml 的读取、解析和转换过程，所以反序列化式读取的耗时一般低于纯文本 xml 的载入。比如 CodeSynthesis XSD 就支持序列化。

另外一种思路是自行转换成相应的格式，也是 xdtoolkit 所做的，先将 xml 数据转成自定义格式 xds，直接用 xds 文件作为已发布版本的配置数据。如果有跟踪过 xdtoolkit 的源码，可以发现，经过转换成 xds，载入过程基本等价于将 xds 文件内容直接映射到内存中，速度是非常快的。以一般的游戏配置数据而言，读取时间一般在 0.001 毫秒级别，比序列化式加速要快上至少 2 个数量级。

2.2. excel 转 csv 文件

csv 格式的优势十分明显，相较于 xml，更易于从 excel 转换而来，且格式清晰简洁无冗余，载入速度更快，缺点是当 csv 文件内行数较多时，不易直接查找到某个值所对应的字段。

导出 csv 时在 excel 中的数据区域外最好不要有多余的字符，否则会导致导出非数据区的文本内容可以按如下方式使用 csv。

| 装备 id | 装备名称 | 装备等级 | 价格（铜钱） |
|-------|-------|------|--------|
| id | name | lvl | price |
| 1 | 单手剑 | 1 | 100 |
| 2 | 单手剑 2 | 2 | 200 |
| 3 | 单手剑 3 | 3 | 300 |
| 4 | 单手剑 4 | 4 | 400 |
| 5 | 单手剑 5 | 5 | 500 |

excel 另存为 csv 后，导出结果如下

```
装备 id,装备名称,装备等级,价格（铜钱）
id,name,lvl,price
1,单手剑,1,100
2,单手剑 2,2,200
3,单手剑 3,3,300
4,单手剑 4,4,400
5,单手剑 5,5,500
```

采用 csv 格式对策划来说最为友好，但对程序方面的对接就不利了，因为程序方面必须确保字段的加载顺序和 csv 的字段顺序严格一致，否则数据量大了之后易出疏漏，在这方面最好有自动机制加以保证。

2.3. 直接存在数据库中

直接将配置存在数据库中，缺点十分明显，由于策划一般是先在 excel 中编辑数据再导出，所以必须有一个从 excel 导出数据到数据库的过程，这个过程一般需要借助数据库 gui 工具的使用或者带格式数据的导入导出功能，比如，我现在（2014 年 10 月）的项目采用 mysql 数据库，使用的 navicat premium 图形化数据库工具就支持在 excel 表中直接鼠标选中区域复制再手动粘贴到数据库（经亲身体验，非常低效），不管哪种方式，都必须先在数据库中建立起和策划制作的配置数据字段严格一致的表。此外，一个弊端是，无法校验数据的正确性。就我所知的 mysql GUI 工具，似乎没有哪款产品能有效地监测数据变动的，经常容易出现误改数据又发现不了的情况。存在数据库中的配置数据在发布前如果未经过 diff 工具的检测，仅靠人工操作，那么数据的正确性是很值得怀疑的。相应的解决办法是 diff 旧配置数据的 sql 文件。

但数据存储在数据库中有一个巨大的好处是配置统一存放，当有多台服务器共用同一套配置时，更新配置会显得非常方面。

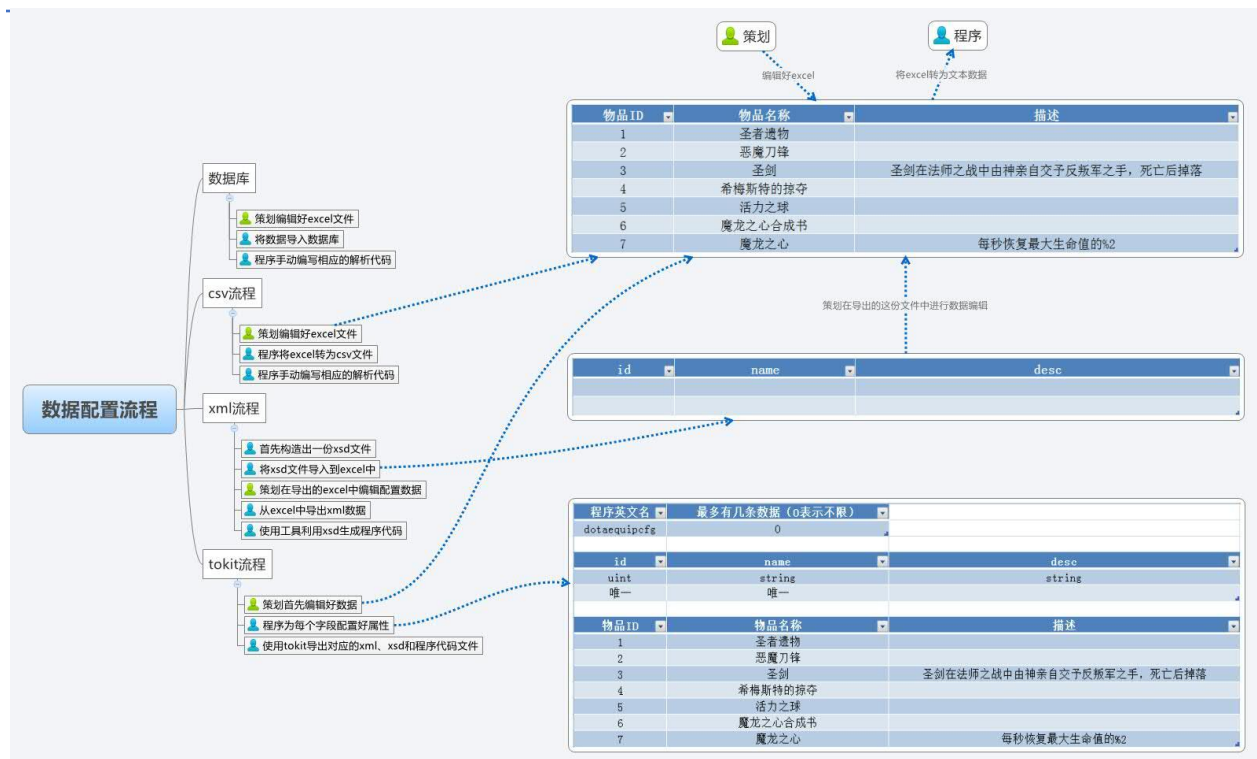
比如

| Name | Type | Length | Decimals | Not null |
|-------|----------|--------|----------|-------------------------------------|
| id | int | 4 | 0 | <input checked="" type="checkbox"/> |
| name | varchar | 256 | 0 | <input checked="" type="checkbox"/> |
| lvl | smallint | 6 | 0 | <input checked="" type="checkbox"/> |
| price | int | 3 | 0 | <input checked="" type="checkbox"/> |

| id | name | lvl | price |
|----|------|-----|-------|
| 1 | 单手剑 | 1 | 100 |
| 2 | 单手剑2 | 2 | 200 |
| 3 | 单手剑3 | 3 | 300 |
| 4 | 单手剑4 | 4 | 400 |
| 5 | 单手剑5 | 5 | 500 |

2.4 流程小结

上述的方式各有优劣，但由于大部分的策划都是用 excel 配置数据，其实都绕不过 excel。具体粗略的分别可由下图来表示。



3. 关于 Tokit 库的一些补充

3.1 问：一般的做法不是已经很方便了吗，为什么还要有 Tokit

因为还不够方便。

以我所了解的技术而言：

vba-->可以导出合适的数据格式，但用于自动生成代码则不是那么容易了。

xml数据对象绑定-->只做到了绑定，类代码自动创建，但接口还是要程序员自行编写。

数据库、csv-->不仅要自行实现类，解析代码也要手写

而Tokit希望做到的是：尽量少写代码

3.2 问：如果希望对 Tokit 生成的文件进行调整，该怎么做？

tokit在生成c++代码时，要求提供2个模板文件，这2个文件放在example\tool\template下，名为<c++_template.h>和<c++_template.cpp>。

而这2个模板文件就是为了方便调整所生成的代码的。

所以，如果希望稍微变动生成的c++文件，可以适当的改动2个文件的内容。

注意到，文件中有大量的%xxx%文本，tokit将自动把对应的文本替换成实际的内容。

其中%cfg%可能会是最经常用到的，含义是当前文件的英文名。

比如，现有一个装备表，名为item，则tokit将自动把模板文件中的%cfg%替换为item。

如果需要更多的定制，就需要改动tokit的源码了。

3.3 问： Tokit 生成的 c++代码载入 xml 的速度快吗？

是的，很快。

这要归功于tokit使用了rapidxml开源库作为xml解析器。

rapidxml设计的初衷是高速读取xml文件，并且也真的做到了，据说比tinyxml快30倍起（我知道很多人都在用tinyxml）。

具体数据可查看rapidxml的官方手册<http://rapidxml.sourceforge.net/manual.html>中的4. Performance小节。

下面表格就是取自4. Performance小节：

| Platform | Compiler | strlen() | RapidXml | pugixml 0.3 | pugxml | TinyXml |
|-----------|-----------|----------|----------|-------------|--------|---------|
| Pentium 4 | MSVC 8.0 | 2.5 | 5.4 | 7.0 | 61.7 | 298.8 |
| Pentium 4 | gcc 4.1.1 | 0.8 | 6.1 | 9.5 | 67.0 | 413.2 |
| Core 2 | MSVC 8.0 | 1.0 | 4.5 | 5.0 | 24.6 | 154.8 |
| Core 2 | gcc 4.1.1 | 0.6 | 4.6 | 5.4 | 28.3 | 229.3 |
| Athlon XP | MSVC 8.0 | 3.1 | 7.7 | 8.0 | 25.5 | 182.6 |
| Athlon XP | gcc 4.1.1 | 0.9 | 8.2 | 9.2 | 33.7 | 265.2 |
| Pentium 3 | MSVC 8.0 | 2.0 | 6.3 | 7.0 | 30.9 | 211.9 |
| Pentium 3 | gcc 4.1.1 | 1.0 | 6.7 | 8.9 | 35.3 | 316.0 |

(*) All results are in CPU cycles per character of source text (中文翻译：表格中的数值表示源文件中每个字符耗费的cpu周期)

从表格中可以看到，rapidxml : strlen() < 10 : 1。表明rapidxml读取一个xml文件的时间小于对一个xml文件进行strlen所花费时间的10倍。

这是非常快的。
