# Codeit指令格式说明

Codeit指令格式设计成形如“enable -m=1 --position”类linux命令字符串。用户可以自定义解析规则、简写规则和默认值等，系统会对该字符串解析成命令与参数包，。

### 命令字符串的规则 ###

/// 所输入的字符串应该符合以下规则：

/// - 第一个空格以前应为命令名字

/// - 第一个空格以后的为命令参数，参数可以带值，那么参数名和参数值之间以“=”分割（不可以有空格），也可以不带值，那么系统会取默认值

/// - 参数名若以“-”开始，那么该参数为简写，例如“-a”，“-m=1”

/// - 参数名若以“--”开始，那么该参数为全写，例如“--all”,“--motion=1”

/// - 参数名若以字母开始，那么这是一组复合参数，每个字符都为参数的简写，例如“ap”等同于“-a -p”

///

/// 假设对于命令“enable”，需要能让用户指定所需使能的电机以及运动模式。那么该命令有以下参数：

/// - “all” 表示所有的电机都要使能，简写为 “a”, 无需默认值

/// - “motion” 表示需要使能的电机ID，简写为“m”，默认值为“0”

/// - “velocity” 表示使能成速度模式，简写为“v”，无需默认值

/// - “position” 表示使能成位置模式，简写为“p”，无需默认值

/// - “current” 表示使能成电流模式，无简写，无需默认值

///

/// 字符串“enable --all -p” 会被解析成：

///

/// 命令：enable

///

/// 参数：

/// - all =

/// - position =

///

/// 字符串“enable -m=1 --position” 会被解析成：

///

/// 命令：enable

///

/// 参数：

/// - motion = 1

/// - position =

///

/// 字符串“enable ap” 会被解析成：

///

/// 命令：enable

///

/// 参数：

/// - all =

/// - position =

///

/// ### 命令解析的规则 ###

///

/// 开发者可以根据需要建立自己的命令解析规则，这主要包括：

/// - 建立参数的简写规则以及默认值

/// - 指定参数之间的关系

///

/// 在上例中，参数之间应该有如下关系：

/// - 参数“all”和“motion”不能共存，用户不能即指定某个电机动，又指定所有的电机都动

/// - 参数“position”、“velocity”，“current”不能共存

/// - 当用户没有指定模式时，系统自动指定模式为“position”

///

/// 那么以下字符串是合法的：

/// - “enable --all --velocity” ，意味着所有电机都使能成速度模式

/// - “enable -m=1 -p”,意味着电机1使能成位置模式

/// - “enable ap”,等同于“enable -a -p”,将所有电机使能成位置模式

/// - “enable -a”,将所有电机使能成位置模式

///

/// 以下字符串非法：

/// - “enable -a -m=1 --position”，因为不能即指定所有电机，又指定电机1

/// - “enable -p”,因为不知道是使能所有电机还是某一个电机

///

/// 可知，开发者需要根据业务场景来设定参数之间的逻辑与分层关系，在codeit中，用户通过UniqueParam和GroupParam两种节点来指定参数之间的关系，

/// 再加上具体的参数节点，共有三种节点：

/// - \ref codeit::core::Param "Param" ：参数节点，无子节点，可以设定简写，可以设定默认值

/// - \ref codeit::core::UniqueParam "UniqueParam"：是父节点，各个子节点只能出现一个，无简写，如果有默认值，默认值必须为某个子节点的名字

/// - \ref codeit::core::GroupParam "GroupParam"：是父节点，所有子节点必须全部出现，无简写，无默认值

///

/// 那么“enable”命令的节点设置模式可以用以下XML文件来表示：

///

/// ~~~{.xml}

/// <Command name="enable" default="group">

/// <GroupParam>

/// <UniqueParam>

/// <all type="Param" abbreviation="a"/>

/// <motion type="Param" abbreviation="m" default="0"/>

/// </UniqueParam>

/// <UniqueParam default="position">

/// <position type="Param" abbreviation="p"/>

/// <velocity type="Param" abbreviation="v"/>

/// <current type="Param"/>

/// </UniqueParam>

/// </GroupParam>

/// </Command>

/// ~~~

///

/// 以上xml节点确定了enable命令的解析规则，首先该命令包含一个名为“group”的“GroupParam”

/// 这说明了这个节点以下的所有参数（unique1和unique2）都必须有值，而unique1和unique2都是“UniqueParam”

/// 这说明这两个节点下的所有子节点只能出现一个，于是all和motion不能共存，position、velocity和current也不能共存。

/// unique2有名为“default”的参数，它指向position，这意味着当用户没有指定unique2中的参数时，parser会自动选择position作为

/// 默认参数。

///

/// ### 代码实现 ###

///

/// \ref demo\_command\_parser\_xml/main.cpp "基于xml和c++的代码示例"

///

/// \ref demo\_command\_parser\_cpp/main.cpp "基于纯c++的代码示例"

///

/// @{

///

**1、Enable (--all/--motion\_id=i)**

(1)enable:使能指令，enable的缩写

(2)--all/--motion\_id=i：两个互斥的可选参数（互斥参数用"/"隔开，可选参数用"()"包含，后续指令类似），可以选定1个或者全部电机，--all为默认参数，即指令en

和指令en --all等效；--motion\_id=i表示选择第i+1个电机

(3)指令和参数之间、参数与参数之间，都用一个空格隔开

例如，

en：使能全部电机

en --motion\_id=0：使能第1个电机

(4)若系统中存在多个模型，比如有两只机械臂。则en指令需同时对这两台臂起作用。子指令数目必须与模型数目相同。

例如，

en：使能第一个模型的电机

{en}：使能所有模型的电机

{en||en}:共有两个模型，同时使能这两个模型的电机；当模型数与指令数不同时，会抛出异常

{en --motion\_id=0||en --motion\_id=1}：使能模型1的第1个电机与模型2的第二个电机

Codeit系统中增加指令即是要增加一个类，该类处于命名空间codeit::function下。该类继承于基类codeit::function::BasisFunc。类的成员参考。

// plan 依次执行3个函数：

// ## prepareNrt

// ## executeRT

// ## collectNrt

//

// 所有的plan都必须执行 prepare，但在prepare中可以通过选项来决定是否执行 executeRT 或 collect

//

// 其中 prepare 可以抛异常，在此情况下后面两个函数一定不执行；executeRT 和 collect 一定不能抛异常

//

// 若要报错，请在不同阶段按如下方式来做：

// prepare :直接 throw std::exception 的继承类，在此之前可以决定是否要 collect

// executeRT :返回小于0的数，并且可以通过 setErrMsgRT 来设置当前错误信息

// collect :不要报错，这个相当于析构函数，只要 prepare 不抛异常而且未设置NOT\_RUN\_COLLECT\_FUNCTION，就一定会执行

//

类的命名遵从首字母大写。

指令的返回值：

enum RetStatus : std::int32\_t

{

EXECUTED = 1,

SUCCESS = 0,

PARSE\_EXCEPTION = -1,

PREPARE\_EXCEPTION = -2,

SERVER\_IN\_ERROR = -10,

SERVER\_NOT\_STARTED = -11,

COMMAND\_POOL\_IS\_FULL = -12,

PREPARE\_CANCELLED = -40,

EXECUTE\_CANCELLED = -41,

SLAVE\_AT\_INIT = -101,

SLAVE\_AT\_SAFEOP = -102,

SLAVE\_AT\_PREOP = -103,

SLAVE\_AT\_OP = -104,

MOTION\_NOT\_ENABLED = -501,

MOTION\_POS\_BEYOND\_MIN = -502,

MOTION\_POS\_BEYOND\_MAX = -503,

MOTION\_POS\_NOT\_CONTINUOUS = -504,

MOTION\_POS\_NOT\_CONTINUOUS\_SECOND\_ORDER = -505,

MOTION\_POS\_FOLLOWING\_ERROR = -506,

MOTION\_VEL\_BEYOND\_MIN = -507,

MOTION\_VEL\_BEYOND\_MAX = -508,

MOTION\_VEL\_NOT\_CONTINUOUS = -509,

MOTION\_VEL\_FOLLOWING\_ERROR = -510,

MOTION\_INVALID\_MODE = -511,

MOTION\_IN\_INTRNG = -512,

////\*\*\*新定义的错误码，-1000~-10000之间，需加入errMap中

PLAN\_OVER\_TIME = -1001,

FORWARD\_KINEMATIC\_POSITION\_FAILED = -1002,

INVERSE\_KINEMATIC\_POSITION\_FAILED = -1003,

WRIST\_SINGULARITY = -1004,

SHOULDER\_SINGULARITY = -1005,

ELBOW\_SINGULARITY = -1006,

SINGULARITY = -1007,

THREE\_POINTS\_COLLINEAR = -1008,

NO\_MOVEJ\_PLANNER = -1100,

NO\_MOVEL\_PLANNER = -1101,

NO\_MOVEC\_PLANNER = -1102,

NO\_MOVES\_PLANNER = -1103,

NO\_MOVELL\_PLANNER = -1104,

NO\_SERVOJ\_PLANNER = -1104,

PROGRAM\_EXCEPTION = -2000,

};