# bazel编译C++方法

——slwang，6/4/2018

## 1.建立工作区（workspace）

Bazel的编译是基于工作区（workspace）的概念。工作区是一个存放了所有源代码和Bazel编译输出文件的目录，也就是整个项目的根目录。同时它也包含一些Bazel认识的文件：

WORKSPACE文件，用于指定当前文件夹就是一个Bazel的工作区。所以WORKSPACE文件总是存在于项目的根目录下。

一个或多个BUILD文件，用于告诉Bazel怎么构建项目的不同部分。（如果工作区中的一个目录包含BUILD文件，那么它就是一个package。）

那么要指定一个目录为Bazel的工作区，就只要在该目录下创建一个空的WORKSPACE文件即可。

当Bazel编译项目时，所有的输入和依赖项都必须在同一个工作区。属于不同工作区的文件，除非linked否则彼此独立。

## 2.理解BUILD文件

一个BUILD文件包含了几种不同类型的指令。其中最重要的是编译指令，它告诉Bazel如何编译想要的输出，比如可执行二进制文件或库。BUILD文件中的每一条编译指令被称为一个target，它指向一系列的源文件和依赖，一个target也可以指向别的target。

下面这个hello-world的target利用了Bazel内置的cc\_binary编译指令，来从hello-world.cc源文件（没有其他依赖项）构建一个可执行二进制文件。指令里面有些属性是强制的，比如name，有些属性则是可选的，srcs表示的是源文件。

|  |
| --- |
| cc\_binary(  name = "hello-world",  srcs = ["hello-world.cc"],  ) |

## 3.使用Bazel编译项目

Bazel提供了一些编译的例子，在<https://github.com/bazelbuild/examples/>，可以clone到本地试一下。其中examples/cpp-tutorial目录下包含了这么些文件：

|  |
| --- |
| examples  └── cpp-tutorial  ├──stage1  │ └── main  │ ├── BUILD  │ ├── hello-world.cc  │ └── WORKSPACE  ├──stage2  │ ├── main  │ │ ├── BUILD  │ │ ├── hello-world.cc  │ │ ├── hello-greet.cc  │ │ ├── hello-greet.h  │ └── WORKSPACE  └──stage3  ├── main  │ ├── BUILD  │ ├── hello-world.cc  │ ├── hello-greet.cc  │ └── hello-greet.h  ├── lib  │ ├── BUILD  │ ├── hello-time.cc  │ └── hello-time.h  └── WORKSPACE |

可以看到分成了3组文件，分别对应本文中的3个例子。在第一个例子中，我们首先学习如何构建单个package中的单个target。在第二个例子中，我们将把整个项目拆分成单个package的多个target。第三个例子则将项目拆分成多个package，用多个target编译。

### 3.1构建第一个bazel项目

首先进入到cpp-tutorial/stage1目录下，然后运行以下指令：

|  |
| --- |
| bazel build //main:hello-world |

注意target中的//main:是BUILD文件相对于WORKSPACE文件的位置，hello-world则是我们在BUILD文件中命名好的target的名字。

然后Bazel就会有一些类似这样的输出：

|  |
| --- |
| INFO: Found 1 target...  Target //main:hello-world up-to-date:  bazel-bin/main/hello-world  INFO: Elapsed time: 2.267s, Critical Path: 0.25s |

这样第一个Bazel target就编译好了！Bazel将编译的输出放在项目根目录下的bazel-bin目录下，可以看一下这个目录，理解一下Bazel的输出结构。

测试刚刚生成的二进制文件：

|  |
| --- |
| bazel-bin/main/hello-world |

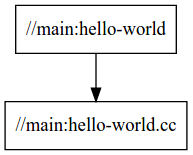
### 3.2 查看依赖图

一个成功的build将所有的依赖都显式定义在了BUILD文件中。Bazel使用这些定义来创建项目的依赖图，这能够加速编译的过程。

现在可视化项目的依赖，首先，生成依赖图的一段文字描述（即在工作区根目录下运行下述指令）：

|  |
| --- |
| bazel query --nohost\_deps --noimplicit\_deps 'deps(//main:hello-world)' --output graph |

这个指令告诉Bazel查找target //main:hello-world的所有依赖项（不包括host和隐式依赖），然后输出图的文字描述。再把文字描述贴到<https://dreampuf.github.io/GraphvizOnline/>里，就可以看到如下的依赖图。可以看出这个项目是用单个源文件编译出的单个target，并没有别的依赖。



### 3.3多个target的编译

单个target的方式对于小项目来说是高效的，但是对于大项目来说，需要拆分成多个target和多个package来实现快速增量的编译（这样就只需要重新编译改变过的部分）。

首先尝试着把项目拆分成两个target。查看cpp-tutorial/stage2/main目录下的BUILD文件，：

|  |
| --- |
| cc\_library(  name = "hello-greet",  srcs = ["hello-greet.cc"],  hdrs = ["hello-greet.h"],  )  cc\_binary(  name = "hello-world",  srcs = ["hello-world.cc"],  deps = [  ":hello-greet",  ],  ) |

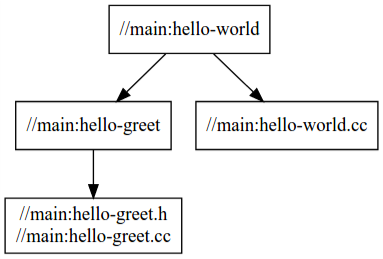
BUILD文件中，Bazel首先编译了hello-greet这个库（利用Bazel内置的cc\_library编译指令），然后编译hello-world这个二进制文件。hello-world这个target的deps属性告诉Bazel，要构建hello-world这个二进制文件需要hello-greet这个库。

进入到cpp-tutorial/stage2目录下然后运行以下指令：

|  |
| --- |
| //编译  bazel build //main:hello-world  //测试  bazel-bin/main/hello-world |

注意，现在修改一下hello-greet.cc然后重新编译整个项目，Bazel只会编译修改过的那个文件。

然后再来看一下依赖图，发现hello-world在编译时候的结构和之前有所不同，现在是有两个targets。hello-world这个target从一个源文件编译而来，同时依赖于另一个target//main:hello-greet，这个target又是从两个源文件编译而来。



### 3.4多个package的编译

现在再将项目拆分成多个package。查看cpp-tutorial/stage3目录下的内容：

|  |
| --- |
| └──stage3  ├── main  │ ├── BUILD  │ ├── hello-world.cc  │ ├── hello-greet.cc  │ └── hello-greet.h  ├── lib  │ ├── BUILD  │ ├── hello-time.cc  │ └── hello-time.h  └── WORKSPACE |

注意到有两个子目录了，每个子目录中都包含了BUILD文件。因此，对于Bazel来说，整个工作区现在就包含了两个package：lib和main。

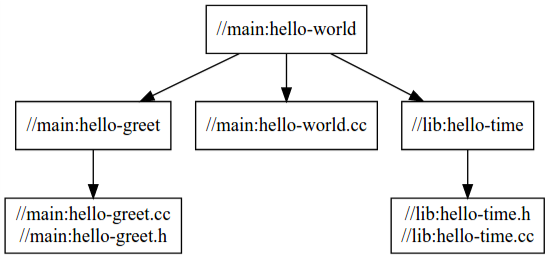
lib/BUILD文件：

|  |
| --- |
| cc\_library(  name = "hello-time",  srcs = ["hello-time.cc"],  hdrs = ["hello-time.h"],  visibility = ["//main:\_\_pkg\_\_"],  ) |

main/BUILD文件：

|  |
| --- |
| cc\_library(  name = "hello-greet",  srcs = ["hello-greet.cc"],  hdrs = ["hello-greet.h"],  )  cc\_binary(  name = "hello-world",  srcs = ["hello-world.cc"],  deps = [  ":hello-greet",  "//lib:hello-time",  ],  ) |

可以看出hello-world这个mainpackage中的target依赖于lib package中的hello-time target（即target label为：//lib:hello-time）- Bazel是通过deps这个属性知道自己的依赖项的。现在的依赖图如下所示：



编译和测试

|  |
| --- |
| //编译  bazel build //main:hello-world  //测试  bazel-bin/main/hello-world |

## 4.参考

<https://docs.bazel.build/versions/master/tutorial/cpp.html>

### supplements：

Run test file in tensorflow with bazel command.

|  |
| --- |
| # All tests (for C++ changes).  $ bazel test //tensorflow/...  # All Python tests (for Python front-end changes).  $ bazel test //tensorflow/python/...  # All tests (with GPU support).  $ bazel test -c opt --config=cuda //tensorflow/...  $ bazel test -c opt --config=cuda //tensorflow/python/... |