Tensorflow的Bazel编程(二)

2017年01月10日 13:38:45 4231人阅读 (0) 收藏 举报

分类： TF（19）

安装官网：https://bazel.build/versions/master/docs/tutorial/java.html

Build Java

创建一个java项目，然后

```python
cd /home/mi/git/TF_pro/Bazel/project
```

在终端运行：

```python
$ mkdir -p src/main/java/com/example
```

```
$ cat > src/main/java/com/example/ProjectRunner.java <<EOF
package com.example;

public class ProjectRunner {
    public static void main(String args[]) {
        Greeting.sayHi();
    }
}
EOF
$ cat > src/main/java/com/example/Greeting.java <<EOF
package com.example;

public class Greeting {
    public static void sayHi() {
        System.out.println("Hi!");
    }
}
EOF
```


在项目根目录下建一个BUILD文件：

```
[python]
java_binary(
    name = "my-runner",
    srcs = glob(["**/*.java"]),
    main_class = "com.example.ProjectRunner",
)
```

然后build

```
[python]
bazel build //:my-runner
```

进入生成的文件夹

```
[python]
1. bazel-bin
```

运行可执行文件

```
[python]
1. my-runner
```


输出：Hi！

成功构建了第二个Bazel项目了！

随着项目的增加，我们讲java项目拆成两个部分独立构建，同时设置它们之间的依赖关系。

重新构建一下BUILD文件：

```
[python]
1.  java_binary(
2.      name = "my-other-runner",
3.      srcs = ["src/main/java/com/example/ProjectRunner.java"],
4.      main_class = "com.example.ProjectRunner",
5.      deps = [":greeter"],
6.  )
7.
8.  java_library(
9.      name = "greeter",
10.     srcs = ["src/main/java/com/example/Greeting.java"],
11. )
```

虽然源文件是一样的，但是现在Bazel将采用不同的方式来构建：首先是构建`greeter`库，然后是构建`my-other-runner`。可以在构建成功后立刻运行`//:my-other-runner`：

Deconvolutional N...
langb2014 : [reply]peiheng90[/reply] 请查收
Deconvolutional N...
langb2014 : [reply]To_Be_The_One[/reply] 请查收
Deconvolutional N...

```[python]
$ bazel run //:my-other-runner
INFO: Found 1 target...
Target //:my-other-runner up-to-date:
  bazel-bin/my-other-runner.jar
  bazel-bin/my-other-runner
INFO: Elapsed time: 2.454s, Critical Path: 1.58s

INFO: Running command line: bazel-bin/my-other-runner
Hi!
```


更多的包

对于更大的项目，我们通常需要将它们拆分到多个目录中。你可以用类似//path/to/dire target-name的名字引用在其他BUILD文件定义的目标。假设src/main/java/com/example/有一个cmdline/子目录，包含下面的文件：

```[python]
$ mkdir -p src/main/java/com/example/cmdline
$ cat > src/main/java/com/example/cmdline/Runner.java <<EOF
package com.example.cmdline;

import com.example.Greeting;

public class Runner {
    public static void main(String args[]) {
        Greeting.sayHi();
    }
}
EOF
```

Runner.java依赖com.example.Greeting，因此我们需要在src/main/java/com/example/cmdline/BUILD构建文件中添加相应的依赖规则：

[python]

```
# src/main/java/com/example/cmdline/BUILD
java_binary(
    name = "runner",
    srcs = ["Runner.java"],
    main_class = "com.example.cmdline.Runner",
    deps = ["//:greeter"]
)
```

然而，默认情况下构建目标都是 私有 的。也就是说，我们只能在同一个BUILD文件中被引用。这可以避免将很多实现的细节暴漏给公共的接口，但是也意味着我们需要手工允许runner所依赖的//:gre 标。就是类似下面这个在构建runner目标时遇到的错误：

[python]

```
$ bazel build //src/main/java/com/example/cmdline:runner
ERROR: /home/user/gitroot/my-project/src/main/java/com/example/cmdline/BUILD:2:1:
  Target '//:greeter' is not visible from target '//src/main/java/com/example/cmdline:r
unner'.
  Check the visibility declaration of the former target if you think the dependency is
 legitimate.
ERROR: Analysis of target '//src/main/java/com/example/cmdline:runner' failed; build ab
orted.
INFO: Elapsed time: 0.091s
```

可用通过在BUILD文件增加visibility = level属性来改变目标的可间范围。下面是通过在~/gitroot/my-project/BUILD文件增加可见规则，来改变greeter目标的可见范围：

[python]

```
java_library(
    name = "greeter",
    srcs = ["src/main/java/com/example/Greeting.java"],
```

```
    visibility = ["//src/main/java/com/example/cmdline:__pkg__"],
)
```

这个规则表示//:greeter目标对于//src/main/java/com/example/cmdline包是可见的。现在我们可以重新构建runner目标程序：

```
[python]
$ bazel run //src/main/java/com/example/cmdline:runner
INFO: Found 1 target...
Target //src/main/java/com/example/cmdline:runner up-to-date:
  bazel-bin/src/main/java/com/example/cmdline/runner.jar
  bazel-bin/src/main/java/com/example/cmdline/runner
INFO: Elapsed time: 1.576s, Critical Path: 0.81s

INFO: Running command line: bazel-bin/src/main/java/com/example ine/runner
Hi!
```

0

部署

如果你查看 bazel-bin/src/main/java/com/example/cmdline/runner.jar 的内容，可以看到里面只包含了Runner.class，并没有保护所依赖的Greeting.class：

```
[python]
$ jar tf bazel-bin/src/main/java/com/example/cmdline/runner.jar
META-INF/
META-INF/MANIFEST.MF
com/
com/example/
com/example/cmdline/
com/example/cmdline/Runner.class
```

这只能在本机正常工作（因为Bazel的runner脚本已经将greeter jar添加到了classpath），但是如果将runner.jar单独复制到另一台机器上讲不能正常运行。如果想要构建可用于部署发布的自包含所有依赖的目标，可以构建runner_deploy.jar目标（类似<target-name>_deploy.jar以_deploy为后缀的名字对应可部署目标）。

```
[python]
1. $ bazel build //src/main/java/com/example/cmdline:runner_deploy.jar
2. INFO: Found 1 target...
3. Target //src/main/java/com/example/cmdline:runner_deploy.jar up-to-date:
4.   bazel-bin/src/main/java/com/example/cmdline/runner_deploy.jar
5. INFO: Elapsed time: 1.700s, Critical Path: 0.23s
```


runner_deploy.jar中将包含全部的依赖。

Build C++

首先创建头文件和源文件：

切记一定要先建一个WORKSPACE，这个可以为空，不然找不到工程下面的一些目录，build会报错。

在project1下

```
[python]
1. $ mkdir ./main
2. $ cat > main/hello-world.cc <<'EOF'
3. #include "lib/hello-greet.h"
4. #include "main/hello-time.h"
5. #include <iostream>
6. #include <string>
7.
```

```
int main(int argc, char** argv) {
  std::string who = "world";
  if (argc > 1) {
    who = argv[1];
  }
  std::cout << get_greet(who) <<std::endl;
  print_localtime();
  return 0;
}
EOF
$ cat > main/hello-time.h <<'EOF'
#ifndef MAIN_HELLO_TIME_H_
#define MAIN_HELLO_TIME_H_

void print_localtime();

#endif
EOF
$ cat > main/hello-time.cc <<'EOF'
#include "main/hello-time.h"
#include <ctime>
#include <iostream>

void print_localtime() {
  std::time_t result = std::time(nullptr);
  std::cout << std::asctime(std::localtime(&result));
}
EOF
$ mkdir ./lib
$ cat > lib/hello-greet.h <<'EOF'
#ifndef LIB_HELLO_GREET_H_
#define LIB_HELLO_GREET_H_

#include <string>

std::string get_greet(const std::string &thing);

#endif
EOF
$ cat > lib/hello-greet.cc <<'EOF'
```

```
#include "lib/hello-greet.h"
#include <string>

std::string get_greet(const std::string& who) {
  return "Hello " + who;
}
EOF
```

在lib下添加BUILD文件：

```
[python]
cc_library(
    name = "hello-greet",
    srcs = ["hello-greet.cc"],
    hdrs = ["hello-greet.h"],
    visibility = ["//main:__pkg__"],
)
```

在main下添加BUILD文件：

```
[python]
cc_library(
    name = "hello-time",
    srcs = ["hello-time.cc"],
    hdrs = ["hello-time.h"],
)

cc_binary(
    name = "hello-world",
    srcs = ["hello-world.cc"],
    deps = [
        ":hello-time",
        "//lib:hello-greet",
    ],
)
```

```
mi@u:~/git/TF_pro/Bazel/project1$ bazel build main:hello-world
.........
INFO: Found 1 target...
Target //main:hello-world up-to-date:
  bazel-bin/main/hello-world
INFO: Elapsed time: 3.901s, Critical Path: 1.89s
mi@u:~/git/TF_pro/Bazel/project1$ bazel-bin/main/hello-world
Hello world
Tue Jan 10 14:07:20 2017
mi@u:~/git/TF_pro/Bazel/project1$ bazel-bin/main/hello-wor b
Hello lb
Tue Jan 10 14:15:15 2017
```


依赖传递

如果包含了一个头文件，那么需要将头文件对应的库添加到依赖中。不过，只需有添加直接依赖的库。假设三明治对应的`sandwich.h`文件包含了面包对应的`bread.h`文件，同时`bread.h`又包含了面粉对应的`flour.h`文件。但是，三明治`sandwich.h`文件并没有直接包含面粉`flour.h`文件。

BUILD文件是这样的：

```python
cc_library(
    name = "sandwich",
    srcs = ["sandwich.cc"],
    hdrs = ["sandwich.h"],
    deps = [":bread"],
)

cc_library(
    name = "bread",
    srcs = ["bread.cc"],
    hdrs = ["bread.h"],
    deps = [":flour"],
)
```

```
cc_library(
    name = "flour",
    srcs = ["flour.cc"],
    hdrs = ["flour.h"],
)
```

上面表示了sandwich三明治库依赖bread面包库，bread又依赖flour对应的面粉库。

添加头文件路径

0

很多时候你可能并不希望基于工作区根路径的相对路径来包含每个头文件。因为很多[illegible]在的第三方库的头文件包含方式并不是基于工作区的根路径。假设有以下目录结构：

[python]

```
[workspace]/
    WORKSPACE
    third_party/
        some_lib/
            include/
                some_lib.h
            BUILD
            some_lib.cc
```

Bazel希望用third_party/some_lib/include/some_lib.h方式包含some_lib.h，但是some_lib.cc可能跟希望用"include/some_lib.h"方式包含。为了使得包含路径有效，需要在third_party/some_lib/BUILD文件中将some_lib/目录添加到头文件包含路径的搜索列表中：

[python]

```
cc_library(
    name = "some_lib",
    srcs = ["some_lib.cc"],
```

```
    hdrs = ["some_lib.h"],
    copts = ["-Ithird_party/some_lib"],
)
```

这对于依赖的外部第三方库特别有效，因为可以避免在头文件路径中出现无关的external/[repository-name]/前缀。

添加外部库：

假设使用了Google Test。可以在WORKSPACE文件中使用new_开头的仓库相关的函数，载依赖的GTest代码到当前仓库中：

[python]

```
new_http_archive(
    name = "gtest",
    url = "https://googletest.googlecode.com/files/gtest-1.7.0.zip",
    sha256 = "247ca18dd83f53deb1328be17e4b1be31514cedfc1e3424f672bf11fd7e0d60d",
    build_file = "gtest.BUILD",
)
```

创建gtest.BUILD文件，对应Google Test的构建配置文件。配置文件中有几个需要特别注意的地方：

- gtest-1.7.0/src/gtest-all.cc文件已经采用#include语法包含了gtest-1.7.0/src/目录中其它*.cc文件，因此需要将它排除在外（也可以只包含它一个文件，但是需要正确配置包含路径）。
- 它的头文件在gtest-1.7.0/include/目录，需要将它添加到头文件包含路径列表中
- GTest依赖pthread多线程库，通过linkopt选项指定。

最终的规则大概是这样：

[python]

```
cc_library(
    name = "main",
    srcs = glob(
        ["gtest-1.7.0/src/*.cc"],
        exclude = ["gtest-1.7.0/src/gtest-all.cc"]
    ),
    hdrs = glob([
        "gtest-1.7.0/include/**/*.h",
        "gtest-1.7.0/src/*.h"
    ]),
    copts = [
        "-Iexternal/gtest/gtest-1.7.0/include"
    ],
    linkopts = ["-pthread"],
    visibility = ["//visibility:public"],
)
```


这是有点混乱：所有以gtest-1.7.0为前缀的其实都是生成的临时文件。我们可以通过new_http_archive函数中的strip_prefix属性来忽略它：

```[python]
new_http_archive(
    name = "gtest",
    url = "https://googletest.googlecode.com/files/gtest-1.7.0.zip",
    sha256 = "247ca18dd83f53deb1328be17e4b1be31514cedfc1e3424f672bf11fd7e0d60d",
    build_file = "gtest.BUILD",
    strip_prefix = "gtest-1.7.0",
)
```

现在gtest.BUILD简洁多了：

[python]

```
cc_library(
    name = "main",
    srcs = glob(
        ["src/*.cc"],
        exclude = ["src/gtest-all.cc"]
    ),
    hdrs = glob([
        "include/**/*.h",
        "src/*.h"
    ]),
    copts = ["-Iexternal/gtest/include"],
    linkopts = ["-pthread"],
    visibility = ["//visibility:public"],
)
```


现在cc_相关的规则可以通过//external:gtest/main引用GTest了。

测试：./test/hello-test.cc

[python]

```
#include "gtest/gtest.h"
#include "lib/hello-greet.h"

TEST(FactorialTest, Negative) {
  EXPECT_EQ(get_greet("Bazel"), "Hello Bazel");
}
```

./test/BUILD：

[python]

```
cc_test(
    name = "hello-test",
```

```
    srcs = ["hello-test.cc"],
    copts = ["-Iexternal/gtest/include"],
    deps = [
        "@gtest//:main",
        "//lib:hello-greet",
    ],
)
```

lib/BUILD:

```
[python]

cc_library(
    name = "hello-greet",
    srcs = ["hello-greet.cc"],
    hdrs = ["hello-greet.h"],
    visibility = ["//test:__pkg__"],
)
```

```
mi@u:~/git/TF_pro/Bazel/project2$ bazel test test:hello-test
INFO: Found 1 test target...
Target //test:hello-test up-to-date:
  bazel-bin/test/hello-test
INFO: Elapsed time: 7.673s, Critical Path: 6.15s
//test:hello-test                                                PASSED
in 0.1s
                    http://blog.csdn.net/langb2014

Executed 1 out of 1 test: 1 test passes.
There were tests whose specified size is too big. Use the --test_verbose_timeout
_warnings command line option to see which ones these are.
```

依赖预编译的库

如果要依赖一个已经编译好的库（可能只有头文件和对应的*.so库文件），可以使用cc_library规则包装一个库对象：

```
[python]
1. cc_library(
2.     name = "mylib",
3.     srcs = ["mylib.so"],
4.     hdrs = ["mylib.h"],
5. )
```

其它的目标就可以依赖这个包装的库对象了。

0

参考：

https://my.oschina.net/chai2010/blog/674110

https://my.oschina.net/chai2010/blog/701807

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

https://bazel.build/versions/master/docs/tutorial/java.html

https://bazel.build/versions/master/docs/bazel-user-manual.html

- 上一篇 Tensorflow的Bazel编程(一)
- 下一篇 Tensorflow的Bazel编程(三)

tensorflow 之 bazel安装 & 使用

u010700335 2017年04月06日 19:14 25647

写在文章前面： 当一个人从一个领域跨到另一个领域的时候会面临很大的改变，理论[illegible]，方法变换了，遇到这样挑战的时候，很多人都需要长时间去适应和习惯；这种领域的转换其实有三种，一种[illegible]的改变，一种是方...

Bazel教程

u010510350 2016年08月20日 17:20 6689

Bazel的入门上一篇博文讲解了在Ubuntu14.04下Bazel的安装，接下来讲解一下Bazel的[illegible]门。 *使用工作区* Bazel builds应该在一个工作区内运行，这个工作区应该包括...

Tensorflow的Bazel编程(二)

langb2014 2017年01月10日 13:38 4231

安装官网：https://bazel.build/versions/master/docs/tutorial/java.html Build Java 创建一个java项目，然后 cd /home/...

从零开始使用tensorflow（1）——安装

juanjuan1314 2016年08月23日 19:16 4253

1. Python和jdk1.8之前已经安装好了。 2. Pip install https://storage.googleapis.com/tensorflow/mac/...

学习Tensorflow，使用源码安装

helei001 2016年04月30日 13:52 28578

PC上装好Ubuntu系统，我们一步一步来讲解如何使用源码安装tensorflow?（我的Ubuntu系统是15.10）根据你的系统型号选择相应的cuda版本下载 https://deve...

Tensorflow的Bazel编程(四)

langb2014 2017年01月10日 20:22 1506

C/C++ Rules cc_binary(name, deps, srcs, data, args, compatible_with, copts, defines, deprecation,...

Tensorflow的Bazel编程(一)

langb2014 2017年01月09日 14:01 3107

在了解Bazel先看一下 Google Bazel原理篇： Google分布式构建软件之第一部分：访问源代码 Google分布式构建软件之第二部分：构建系统如何工作 Google分布式构建软件之第三...

源码安装TensorFlow问题总结

idwtwt 2016年08月27日 00:08 4767

1.按教程：bazel build -c opt --jobs 1 //tensorflow/cc:tutorials_example_trainer 出现 INFO: Reading option...

ubuntu15.10 源码安装 tensorflow

autumnqin 2015年11月27日 18:32 10521

刚买的新机器，安装上最新的ubuntu系统。按照官网的流程安装的，没有选择GPU支持。 下载源码。需要git，如果没有安装git需要先安装。 git clone --recurse-submodule...

Bazel教程

u010510350 2016年08月19日 10:02 22305

Bazel的安装bazel支持的平台有Ubuntu linux、Mac OS X、Windows等，本教程基于Ubuntu14.04下的bazel，其它平台的安装参考官网安装教程。Ubuntu下baz...

bazel选项的部分翻译，希望给编译tf的童鞋参考

0

部分翻译了bazel的选项的含义，仅供参考

xizero00 2017年03月03日 13:48 3026

Redhat环境下编译安装bazel

u013668392 2016年10月11日 11:30 1837

目前Google Bazel没有提供各个操作系统下的二进制安装包，只提供源代码，需要我们自己编译安装，详情可以见我翻译的中文版Google Bazel FAQ。Google Bazel官方安装文档在这...

Tensorflow的Bazel编程(五)

langb2014 2017年01月10日 20:26 1495

只了解一下常用的这几种语言的Rules，这一篇是python的rules。 py_binary py_binary(name, deps, srcs, data, args, compati...

Bazel入门：编译C++项目

 elaine_bao 2017年11月29日 18:46 824

官网：https://www.bazel.build Github: https://github.com/bazelbuild/bazel最近用到tensorflow的时候遇到了个新的编译工具Ba...

Bazel介绍——创建一个C++项目

 sdlypyzq 2017年07月20日 20:07 1245

使用Bazel构建设置工作区（workspace）在构建项目之前，需要设置其工作区。 工作区存项目源文件和Bazel构建输出的目录。 它还包含Bazel认为特殊的文件： WORKSPACE文件将目...

Bazel入门

 mydear_11000 2017年06月05日 08:49 1528

Bazel入门 上一篇博文中讲解了以Java工程bazel的使用，接下来讲解在C++工程中bazel的使用。 C++教程 *建立工作区 * 假设目录中已经有了一个项目，对应 ~/gi...

Bazel入门2：C++编译常见用例

 elaine_bao 2017年11月29日 23:05 343

1. 在一个target中包含多个文件可以利用glob在单个target中包含多个文件，例如：cc_library(name = "build-all-the-files", src...

如何编译运行tensorflow的demo

 moyimoyi123 2017年08月06日 14:02 1308

1.安装编译工具bazel，具体可以参照官方教程。 https://docs.bazel.build/versions/master/install-ubuntu.html 2. 配置tensor...

TensorFlow编译过程中遇到的问题及解决方案

最近一直尝试使用TensorFlow源码进行编译，在编译过程中遇到一个Status 4的错误码。 github问题链接其完整错误日志如下：ERROR: /home/wangbing/Git/tenso...

sydpz1987 2016年05月24日 21:44 5671

0

Tensorflow的Bazel编程(二)

haima1998 2)7月26日 16:22 188

转自：http://blog.csdn.net/langb2014/article/details/54312697 安装官网：https://bazel.build/versions/ma...