《开源软件设计与开发》课程总结

姓名：项兆坤　　学号：51194501073

1、开源理解

GitHub 这一服务，为开源世界带来了社会化编程的概念。这一概念影响了全世界众多程序员，说其是软件开发方法的一次革命都不为过[1]。

下图是 GitHub 曾经使用过的 LOGO，上面附带的 Social Coding(社会化编程)的副标题。

GitHub 这一服务创造了社会化编程的概念。随着 GitHub 的出现，软件开发者们才真正意义上拥有了源代码。世界上任何人都可以比从前更加容易地获得源代码，将其自由更改并加以公开。如今，世界众多程序员都在通过 GitHub 公开源代码，同时利用 GitHub 支持着自己日常的软件开发。

GitHub 出现之前，软件开发中只有一小部分人拥有更改源代码的权利，这个特权阶级掌握着开发的主导权。开发者在改写、发布源代码之外，往往需要花更多时间和精力去说服这个特权阶级。这导致了许多起初效率很高的流行软件越发保守化，最终被时代所抛弃。

但是，GitHub 的出现为软件开发者的世界带来了真正意义上的民主，让所有人都平等地拥有了更改源代码的权利。这在软件开发领域是一场巨大的革命。而革命领导者 GitHub 的口号便是“社会化编程”。

社会化编程让程序员接触不同开源社区的文化，接触世界上的不同文化，拓展见闻。如果只是在一个封闭的小世界中敲代码，往往会成为井底之蛙。GitHub 的出现已经让所有人平等拥有公开源代码的权利，通过查看某个程序员的 GitHub 就能了解一个程序员的实力。在不远的将来，应聘的成功与否将取决于您曾经编写过的代码。所以以编写代码为生的职业程序员们，更应该进行社会化编程。

Git 是一个开源的分布式版本控制系统，分布式相比集中式的最大区别是 Git 没有“中央服务器”，每位开发者都可以通过克隆（git clone）远程库，在本地机器上存储一个完整的 Git 仓库，还可以把代码的修改提交到本地库[2]。

从一般开发者的角度来看，使用 Git 的工作流程如下：

1.克隆远程库：从远程库上克隆完整的 Git 仓库（包括代码和版本信息）到本地；

2.在本地库上修改代码：在本地库上根据不同的开发目的，创建分支，修改代码；

3.提交到分支：在本地分支上提交代码；

4.把修改合并到本地主分支：在本地库上提交更新，也就是说，把修改合并到本地主分支；

5.把远程库合并到本地主分支：把远程库上的最新代码 fetch 下来，跟本地主分支合并，如果存在冲突，那么解决冲突。

6.把本地主分支提交到远程库：生成补丁，把补丁发送给远程库。

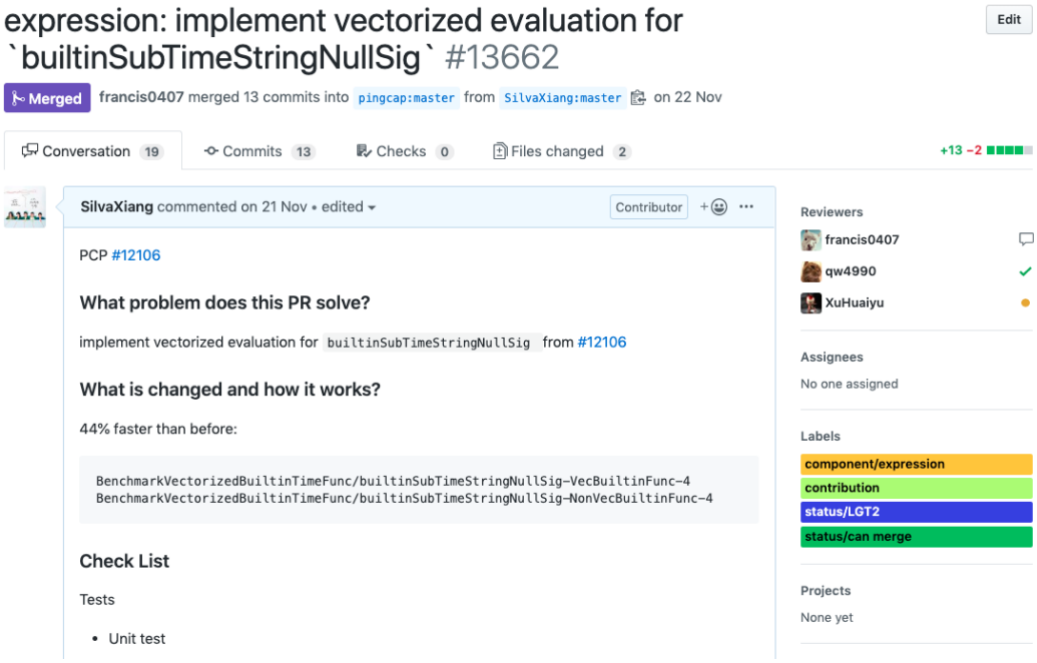
2、开源贡献

在这门课程中，我参加了 TiDB 表达式向量化这一开源项目，提交了 3 个 Pull Request，其中成功合并了两个，成功为 12 个函数实现了向量化的接口，为 TiDB 贡献 1000+行代码，

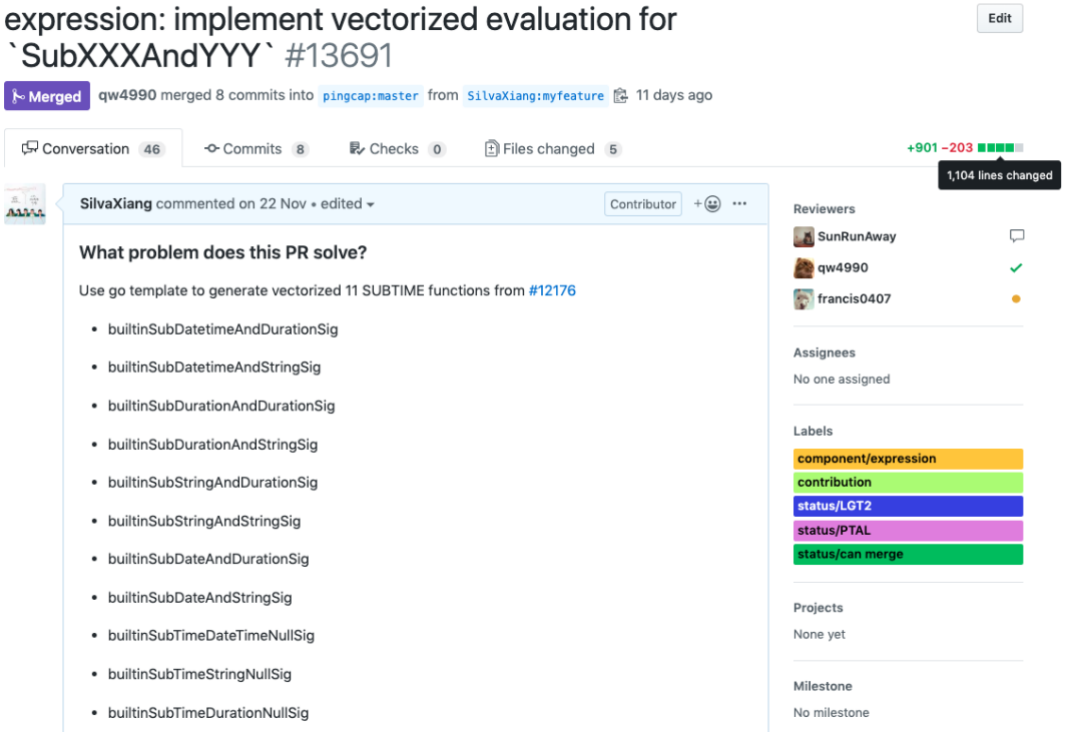
成为了 TiDB Contributor。

合并的两个 PR 的链接如下：

1. 为 builtinSubTimeStringNullSig 函数实现了向量化的接口，贡献 15 行左右的代码，链接：https://github.com/pingcap/tidb/pull/13662



2. 使用 Go Template 实现并生成 11 个 SubXXXAndXXX 函数的向量化接口，贡献 1000+ 行左右代码，链接：https://github.com/pingcap/tidb/pull/13691



具体贡献的工作描述：

1. builtinSubTimeStringNullSig 函数的向量化

该函数的功能是实现向量化的 Datetime 类型与 Datetime 类型的减法，结果为 String 类型。但是所有参与计算的值都是 NULL，结果也是 NULL，核心代码如下:

```
func (b *builtinSubTimeStringNullSig) vecEvalString(input *chunk.Chunk, result *chunk.Column) error {
    n := input.NumRows()
    result.ReserveString(n)
    for i := 0; i < n; i++ {
        result.AppendNull()
    }
    return nil
}
```

2. builtinSubDatetimeAndDurationSig 函数的向量化

该函数的功能是实现向量化的 Datetime 类型与 Duration 类型的减法，结果为 Datatime 类型。核心代码如下:

```
sc := b.ctx.GetSessionVars().StmtCtx
arg1Duration := types.Duration{Duration: arg1, Fsp: -1}
arg1time, err := arg1Duration.ConvertToTime(sc, mysql.TypeDatetime)
if err != nil {
    return err
}
tmpDuration := arg0.Sub(sc, &arg1time)
output, err := tmpDuration.ConvertToTime(sc, arg0.Type)
if err != nil {
    return err
}
```

3. builtinSubDatetimeAndStringSig 函数的向量化

该函数的功能是实现向量化的 Datetime 类型与 String 类型的减法，结果为 Datetime 类型。核心代码如下:

```
if !isDuration(arg1) {
    result.SetNull(i, true) // fixed: true
    continue
}
sc := b.ctx.GetSessionVars().StmtCtx
arg1Duration, err := types.ParseDuration(sc, arg1, types.GetFsp(arg1))
if err != nil {
    if terror.ErrorEqual(err, types.ErrTruncatedWrongVal) {
        sc.AppendWarning(err)
        result.SetNull(i, true) // fixed: true
        continue
    }
    return err
}
arg1time, err := arg1Duration.ConvertToTime(sc, mysql.TypeDatetime)
if err != nil {
    return err
```

```
}
tmpDuration := arg0.Sub(sc, &arg1time)
output, err := tmpDuration.ConvertToTime(sc, mysql.TypeDatetime)
if err != nil {
    return err
}
```

4. builtinSubDurationAndDurationSig 函数的向量化

该函数的功能是实现向量化的 Duration 类型与 Duration 类型的减法，结果为 Duration 类型。核心代码如下：

先在 overflow.go 文件实现 SubDuration 函数

```
func SubDuration(a time.Duration, b time.Duration) (time.Duration, error) {
    if (a > 0 && b < 0 && math.MaxInt64-a < -b) ||
        (a < 0 && b > 0 && math.MinInt64-a > -b) ||
        (a == 0 && b == math.MinInt64) {
        return 0, ErrOverflow.GenWithStackByArgs("BIGINT", fmt.Sprintf("(%d, %d)", a, b))
    }
    return a - b, nil
}
```

再使用:

```
// calculate
output, err := types.SubDuration(arg0, arg1)
if err != nil {
    return err
}
```

5. builtinSubDurationAndStringSig 函数的向量化

该函数的功能是实现向量化的 Duration 类型与 String 类型的减法，结果为 Duration 类型。核心代码如下：

```
if !isDuration(arg1) {
    result.SetNull(i, true) // fixed: true
    continue
}
sc := b.ctx.GetSessionVars().StmtCtx
arg1Duration, err := types.ParseDuration(sc, arg1, types.GetFsp(arg1))
if err != nil {
    if terror.ErrorEqual(err, types.ErrTruncatedWrongVal) {
        sc.AppendWarning(err)
        result.SetNull(i, true) // fixed: true
        continue
    }
    return err
}
output, err := types.SubDuration(arg0, arg1Duration.Duration)
if err != nil {
```

```
        return err
    }
```

6. builtinSubStringAndDurationSig 函数的向量化

该函数的功能是实现向量化的 String 类型与 Duration 类型的减法，结果为 String 类型。核心代码如下：

```
    sc := b.ctx.GetSessionVars().StmtCtx
    fsp1 := int8(b.args[1].GetType().Decimal)
    arg1Duration := types.Duration{Duration: arg1, Fsp: fsp1}
    var output string
    if isDuration(arg0) {
        output, err = strDurationSubDuration(sc, arg0, arg1Duration)
        if err != nil {
            if terror.ErrorEqual(err, types.ErrTruncatedWrongVal) {
                sc.AppendWarning(err)
                result.AppendNull() // fixed: false
                continue
            }
            return err
        }
    } else {
        output, err = strDatetimeSubDuration(sc, arg0, arg1Duration)
        if err != nil {
            return err
        }
    }
```

7. builtinSubStringAndStringSig 函数的向量化

该函数的功能是实现向量化的 String 类型与 String 类型的减法，结果为 String 类型。核心代码如下：

```
    sc := b.ctx.GetSessionVars().StmtCtx
    arg1Duration, err := types.ParseDuration(sc, arg1, getFsp4TimeAddSub(arg1))
    if err != nil {
        if terror.ErrorEqual(err, types.ErrTruncatedWrongVal) {
            sc.AppendWarning(err)
            result.AppendNull() // fixed: false
            continue
        }
        return err
    }
    var output string
    if isDuration(arg0) {
        output, err = strDurationSubDuration(sc, arg0, arg1Duration)
        if err != nil {
            if terror.ErrorEqual(err, types.ErrTruncatedWrongVal) {
                sc.AppendWarning(err)
```

```
                result.AppendNull() // fixed: false
                continue
            }
            return err
        }
    } else {
        output, err = strDatetimeSubDuration(sc, arg0, arg1Duration)
        if err != nil {
            return err
        }
    }
```

8. builtinSubDateAndDurationSig 函数的向量化

该函数的功能是实现向量化的 Duration 类型与 Duration 类型的减法，结果为 String 类型。核心代码如下：

```
    fsp0 := int8(b.args[0].GetType().Decimal)
    fsp1 := int8(b.args[1].GetType().Decimal)
    arg1Duration := types.Duration{Duration: arg1, Fsp: fsp1}
    sum, err := types.Duration{Duration: arg0, Fsp: fsp0}.Sub(arg1Duration)
    if err != nil {
        return err
    }
    output := sum.String()
```

9. builtinSubDateAndStringSig 函数的向量化

该函数的功能是实现向量化的 Duration 类型与 String 类型的减法，结果为 String 类型。核心代码如下：

```
    if !isDuration(arg1) {
        result.AppendNull() // fixed: false
        continue
    }
    sc := b.ctx.GetSessionVars().StmtCtx
    arg1Duration, err := types.ParseDuration(sc, arg1, getFsp4TimeAddSub(arg1))
    if err != nil {
        if terror.ErrorEqual(err, types.ErrTruncatedWrongVal) {
            sc.AppendWarning(err)
            result.AppendNull() // fixed: false
            continue
        }
        return err
    }
    fsp0 := int8(b.args[0].GetType().Decimal)
    sum, err := types.Duration{Duration: arg0, Fsp: fsp0}.Sub(arg1Duration)
    if err != nil {
        return err
```

```
}
output := sum.String()
```

10. builtinSubTimeDateTimeNullSig 函数的向量化

该函数的功能是实现向量化的 Datetime 类型与 Datetime 类型的减法，结果为 Datetime 类型。但是所有参与计算的值都是 NULL，结果也是 NULL，核心代码如下：

```
func (b *builtinSubTimeDateTimeNullSig) vecEvalTime(input *chunk.Chunk, result *chunk.Column) error {
    n := input.NumRows()
    result.ResizeTime(n, true)
    return nil
}
```

11. builtinSubTimeDurationNullSig 函数的向量化

该函数的功能是实现向量化的 Duration 类型与 Datetime 类型的减法，结果为 Duration 类型。但是所有参与计算的值都是 NULL，结果也是 NULL，核心代码如下：

```
func (b *builtinSubTimeDurationNullSig) vecEvalDuration(input *chunk.Chunk, result *chunk.Column) error {
    n := input.NumRows()
    result.ResizeGoDuration(n, true)
    return nil
}
```

12. 使用 Go Template 自动生成上述向量化函数

由于这 11 个函数很相似，都是某个类型减去某个类型，所以 TiDB 官方人员后来又添加条件，要求使用 Go Template 来生成，Go Template 就是根据 11 个函数的相同点和不同点来使用一些模版语句形成模版，然后利用模版运行自动生成上述 11 个不同的函数的代码，要学习模版语言的语法，该部分代码比较长，具体见 Pull Request 2 链接。

3、课程反馈

这学期的《开源软件设计与开发》课程让我更好的理解了开源的含义，理解了开源社区的重要意义，这是属于程序员的一种特别的文化，让我感觉特别棒。

同时，我也参与了 TiDB 表达式向量化这一开源活动，为 TiDB 贡献了 1000+行代码，成功成为了 TiDB 的贡献者，也过了一把开源的瘾。

希望该课程能成为必修课，开在白天，增加课程的课时量，因为通过该课程我真的了解了很多前沿的东西。

4、参考文献

[1] [日]大塚弘记.GitHub 入门与实践[M].支鹏浩,等,译.北京:人民邮电出版社,2015.
[2] Git 介绍. https://www.cnblogs.com/ljhdo/p/5618809.html