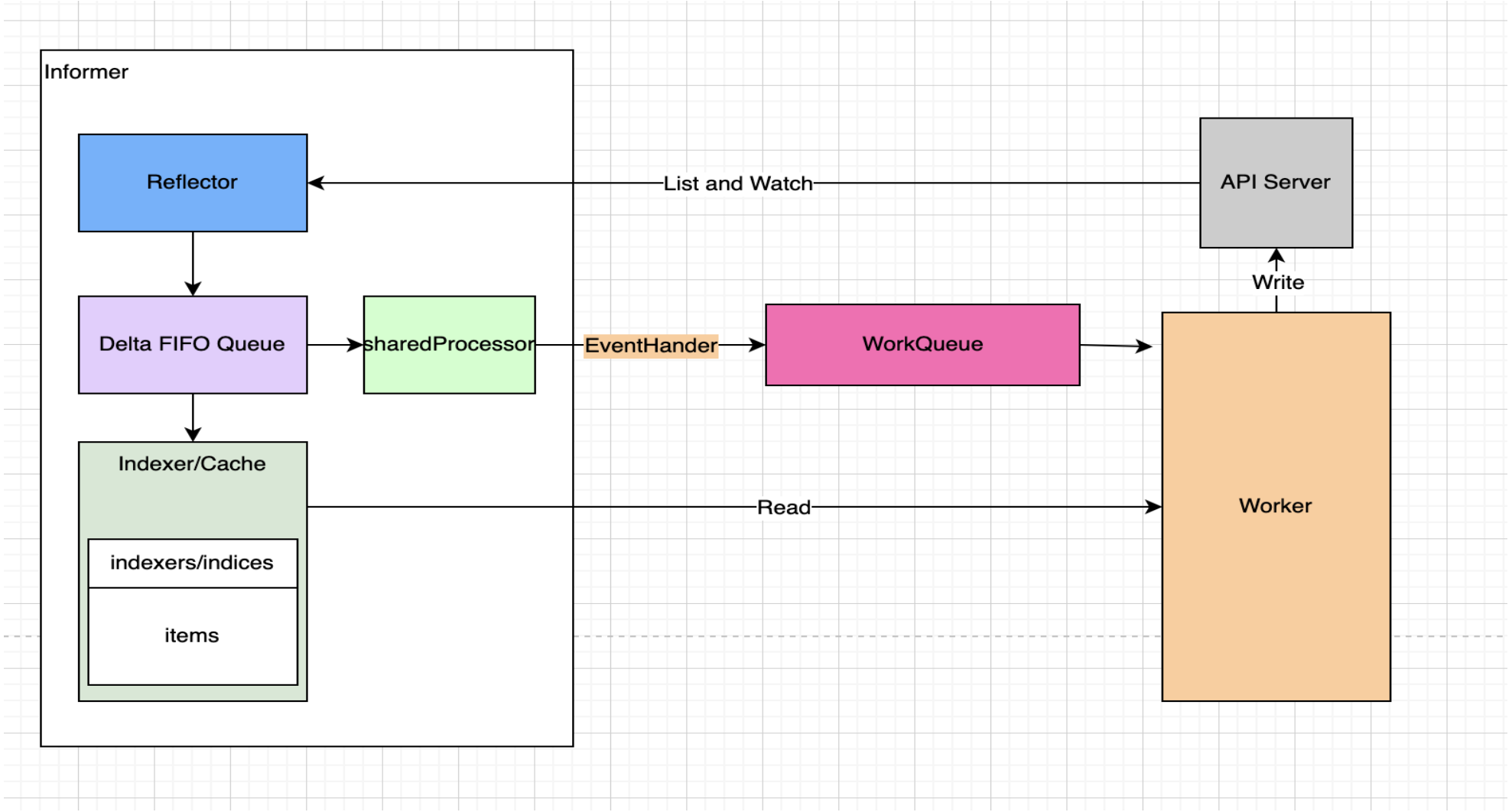
Reflector原理


Informer
Reflector
List and Watch
API Server
Write
Delta FIFO Queue
sharedProcessor
EventHander
WorkQueue
Indexer/Cache
Read
Worker
indexers/indices
items

Reflector的创建

```
func NewReflector(lw ListerWatcher, expectedType interface{}, store Store, resyncPeriod time.Duration) *Reflector {
    ...
}
```

参数说明:

- **lw**: interface，包含了interface Lister和Watcher。通过ListerWatcher获取初始化指定资源的列表和监听指定资源变化
- **expectedType**:指定资源类型
- **store**:指定存储，需要实现Store这个interface，下节课讲解
- **resyncPeriod**:同步周期，下节课讲解

List与Watch

保证可靠性、实时性和顺序性。

- List：指定类型资源对象的全量更新。并将其更新到缓存当中。

```
curl -iv http://127.0.0.1:8001/api/v1/namespaces/default/pods
```

- Watch：指定类型资源对象的增量更新。

```
curl -iv http://127.0.0.1:8001/api/v1/namespaces/default/pods\?watch\=true
```

ResourceVersion与Bookmarks

ResourceVersion

- 保证客户端数据一致性和顺序性
- 并发控制

Bookmarks

- 减少API Server负载
- 更新客户端保存的最近一次ResourceVersion

Reflector与RESTClient

```
&cache.ListWatch{
        ListFunc: func(options metav1.ListOptions) (runtime.Object, error) {
                if tweakListOptions != nil {
                        tweakListOptions(&options)
                }
                return client.CoreV1().Pods(namespace).List(context.TODO(), options)
        },
        WatchFunc: func(options metav1.ListOptions) (watch.Interface, error) {
                if tweakListOptions != nil {
                        tweakListOptions(&options)
                }
                return client.CoreV1().Pods(namespace).Watch(context.TODO(), options)
        },
},
```

谢谢