

# 附录：部分源码浅析

# 1. 平行链阶段

- [run\\_collator@cumulus/test/parachain/src/service.rs](#)
- 86-103行，启动一个parachain的节点
- 116行，定义了一个在parachain上发布区块的闭包，后面会使用到
- 127行，再启动一个collator (polkadot节点+collator逻辑)
- 128行传入的builder，可以生成一个context，做为在collator逻辑里回调parachain的入口
- 后面的produce\_candidate就是通过该context进行回调的

```
77  /// Run a collator node with the given parachain `Configuration` and relaychain `Configuration`.
78  ///
79  /// This function blocks until done.
80  pub fn run_collator(
81      parachain_config: Configuration,
82      key: Arc<CollatorPair>,
83      polkadot_config: polkadot_collator::Configuration,
84      id: polkadot_primitives::parachain::Id,
85  ) -> sc_service::error::Result<impl AbstractService> {
86      let parachain_config = prepare_collator_config(parachain_config);
87
88      let (builder, inherent_data_providers) = new_full_start!(parachain_config);
89      inherent_data_providers
90          .register_provider(sp_timestamp::InherentDataProvider)
91          .unwrap();
92
93      let block_announce_validator = DelayedBlockAnnounceValidator::new();
94      let block_announce_validator_copy = block_announce_validator.clone();
95      let service = builder
96          .with_informant_prefix(format!("[{}]", Color::Yellow.bold().paint("Parachain")));
97          .with_finality_proof_provider(|client, backend| {
98              // GenesisAuthoritySetProvider is implemented for StorageAndProofProvider
99              let provider = client as Arc<dyn StorageAndProofProvider<_, _>>;
100             Ok(Arc::new(GrandpaFinalityProofProvider::new(backend, provider)) as _)
101         })?
102         .with_block_announce_validator(|_client| Box::new(block_announce_validator_copy));
103         .build_full()?;
104
105     let registry = service.prometheus_registry();
106
107     let proposer_factory = sc_basic_authorship::ProposerFactory::new(
108         service.client(),
109         service.transaction_pool(),
110         registry.as_ref(),
111     );
112
113     let block_import = service.client();
114     let client = service.client();
115     let network = service.network();
116     let announce_block = Arc::new(move |hash, data| network.announce_block(hash, data));
117     let builder = CollatorBuilder::new(
118         proposer_factory,
119         inherent_data_providers,
120         block_import,
121         id,
122         client,
123         announce_block,
124         block_announce_validator,
125     );
126
127     let polkadot_future = polkadot_collator::start_collator(
128         builder,
129         id,
130         key,
131         polkadot_config,
132         Some(format!("[{}]", Color::Blue.bold().paint("Relaychain"))),
133     )
134         .map(|_| ());
135     service.spawn_essential_task("polkadot", polkadot_future);
136
137     Ok(service)
138 }
```

# 1. 平行链阶段

- start collator@polkadot/collator/src/lib.rs
- 436行，启动polkadot节点，这里启动是不带验证功能的fullnode
- 446行，启动了collator服务
- 通过polkadot的spawn\_handle来启动
- 可见，collator服务是挂载在polkadot体系内部的服务

```
377     /// Async function that will run the collator node with the given `RelayChainContext` and `ParachainContext`  
378     /// built by the given `BuildParachainContext` and arguments to the underlying polkadot node.  
379     pub async fn start_collator<P>(  
380         build_parachain_context: P,  
381         para_id: ParaId,  
382         key: Arc<CollatorPair>,  
383         config: Configuration,  
384         informant_prefix: Option<String>,  
385     ) -> Result<(), polkadot_service::Error>  
386     where  
387         P: 'static + BuildParachainContext,  
388         P::ParachainContext: Send + 'static,  
389         <P::ParachainContext as ParachainContext>::ProduceCandidate: Send,  
390     {  
391         if matches!(config.role, Role::Light) {  
392             return Err(  
393                 polkadot_service::Error::Other("light nodes are unsupported as collator").into()  
394             .into());  
395         }  
396         if config.chain_spec.is_kusama() {...} else if config.chain_spec.is_westend() {  
397             let (service, client, handlers) = service::westend_new_full(  
398                 config,  
399                 Some(key.public(), para_id),  
400                 None,  
401                 false,  
402                 6000,  
403                 None,  
404                 informant_prefix,  
405             )?;  
406             let spawn_handle = service.spawn_task_handle();  
407             build_collator_service(  
408                 spawn_handle,  
409                 handlers,  
410                 client,  
411                 para_id,  
412                 key,  
413                 build_parachain_context  
414             )?.await;  
415         } else {  
416             let (service, client, handles) = service::polkadot_new_full(  
417                 config,  
418                 Some(key.public(), para_id),  
419                 None,  
420                 false,  
421                 6000,  
422                 None,  
423                 informant_prefix,  
424             )?;  
425             let spawn_handle = service.spawn_task_handle();  
426             build_collator_service(  
427                 spawn_handle,  
428                 handles,  
429                 client,  
430                 para_id,  
431                 key,  
432                 build_parachain_context,  
433             )?.await;  
434         }  
435     }  
436     Ok(())  
437 }
```

# 1. 平行链阶段

- build collator service@polkadot/collator/src/lib.rs
- 这是build\_collator\_service的第一部分
- 290行，创建了parachain\_context，该部分内容定义在cumulus里，稍后介绍

```
249 #[cfg(not(feature = "service-rewr"))]
250 fn build_collator_service<SP, P, C, R, Extrinsic>(
251     spawner: SP,
252     handles: FullNodeHandles,
253     client: Arc<C>,
254     para_id: ParaId,
255     key: Arc<CollatorPair>,
256     build_parachain_context: P,
257 ) -> Result<impl Future<Output = ()> + Send + 'static, polkadot_service::Error>
258     where
259         C: PolkadotClient<
260             service::Block,
261             service::TFullBackend<service::Block>,
262             R
263         > + 'static,
264         R: ConstructRuntimeApi<service::Block, C> + Sync + Send,
265         <R as ConstructRuntimeApi<service::Block, C>>::RuntimeApi:
266             sp_api::ApiExt<
267                 service::Block,
268                 StateBackend = <service::TFullBackend<service::Block> as service::Backend<service::Block>>::State,
269             >
270             + RuntimeApiCollection<
271                 Extrinsic,
272                 StateBackend = <service::TFullBackend<service::Block> as service::Backend<service::Block>>::State,
273             >
274             + Sync + Send,
275         P: BuildParachainContext,
276         P::ParachainContext: Send + 'static,
277         <P::ParachainContext as ParachainContext>::ProduceCandidate: Send,
278         Extrinsic: service::Codec + Send + Sync + 'static,
279         SP: Spawn + Clone + Send + Sync + 'static,
280     {
281         let polkadot_network = handles.polkadot_network
282             .ok_or_else(|| "Collator cannot run when Polkadot-specific networking has not been started")?;
283
284         // We don't require this here, but we need to make sure that the validation service is started.
285         // This service makes sure the collator is joining the correct gossip topics and receives the appropriate
286         // messages.
287         handles.validation_service_handle
288             .ok_or_else(|| "Collator cannot run when validation networking has not been started")?;
289
290         let parachain_context = match build_parachain_context.build(
291             client.clone(),
292             spawner,
293             polkadot_network.clone(),
294         ) {
295             Ok(ctx) => ctx,
296             Err(_) => {
297                 return Err("Could not build the parachain context!".into())
298             }
299         };

```

# 1. 平行链阶段

- build collator service@polkadot/collator/src/lib.rs
- 这是build\_collator\_service的第二部分
- 304行， notification\_stream是波卡链上import block的 stream
- 每次有block import后，会触发该stream的next()函数
- 333行，通过runtime api拿到validators list
- 335行，拿到当前parachain分配的validators
- 341行，通过函数collate去生成collation（区块）
- 350行，在reley chain网络上广播collation

```
301 let work = async move {
302     let mut notification_stream = client.import_notification_stream();
303
304     while let Some(notification) = notification_stream.next().await {
305         macro_rules! try_fr [...] {
306             $(
307                 $t:ident = $e:expr;
308             )*
309             => $body:tt = {
310                 let $t = $e;
311                 $body
312             };
313         }
314
315         let relay_parent = notification.hash;
316         let id = BlockId::hash(relay_parent);
317
318         let network = polkadot_network.clone();
319         let client = client.clone();
320         let key = key.clone();
321         let parachain_context = parachain_context.clone();
322
323         let work = future::lazy(move |_| {
324             let api = client.runtime_api();
325             let global_validation = try_fr!(api.global_validation_schedule(&id));
326             let local_validation = match try_fr!(api.local_validation_data(&id, para_id)) {
327                 Some(local_validation) => local_validation,
328                 None => return future::Either::Left(future::ok(())),
329             };
330             let downward_messages = try_fr!(api.downward_messages(&id, para_id));
331
332             let validators = try_fr!(api.validators(&id));
333
334             let targets = compute_targets(
335                 para_id,
336                 validators.as_slice(),
337                 try_fr!(api.duty_roster(&id)),
338             );
339
340             let collation_work = collate(
341                 relay_parent,
342                 para_id,
343                 global_validation,
344                 local_validation,
345                 downward_messages,
346                 parachain_context,
347                 key,
348             ).map_ok(move |collation| {
349                 network.distribute_collation(targets, collation)
350             });
351
352             future::Either::Right(collation_work)
353         });
354
355         let deadlined = future::select(
356             work.then(|f| f).boxed(),
357             futures_timer::Delay::new(COLLATION_TIMEOUT)
358         );
359
360         let silenced = deadlined.map(|either| {
361             either.map(|_| ...);
362         });
363
364         let future = silenced.map(drop);
365
366         tokio::spawn(future);
367
368     }
369
370     Ok(work)
371 }.boxed();
```

# 1. 平行链阶段

- [collate@polkadot/collator/src/lib.rs](#)
- 178行，通过回调para\_context的produce\_candidate，生成一个区块数据
- produce\_candidate定义在cumulus中
- 190行，使用collator的私钥对区块签名
- 196、205行，生成collation数据

```
164 pub async fn collate<P>(
165     relay_parent: Hash,
166     local_id: ParaId,
167     global_validation: GlobalValidationSchedule,
168     local_validation_data: LocalValidationData,
169     downward_messages: Vec<DownwardMessage>,
170     mut para_context: P,
171     key: Arc<CollatorPair>,
172 )
173     -> Result<parachain::Collation, Error>
174     where
175         P: ParachainContext,
176         P::ProduceCandidate: Send,
177     {
178         let (block_data, head_data) = para_context.produce_candidate(
179             relay_parent,
180             global_validation,
181             local_validation_data,
182             downward_messages,
183         ).map_err(Error::Collator).await?;
184
185         let pov_block = PoVBlock {
186             block_data,
187         };
188
189         let pov_block_hash: Hash = pov_block.hash();
190         let signature = key.sign(&parachain::collator_signature_payload(
191             &relay_parent,
192             parachain_index: &local_id,
193             &pov_block_hash,
194         ));
195
196         let info = parachain::CollationInfo {
197             parachain_index: local_id,
198             relay_parent,
199             collator: key.public(),
200             signature,
201             head_data,
202             pov_block_hash,
203         };
204
205         let collation = parachain::Collation {
206             info,
207             pov: pov_block,
208         };
209
210         Ok(collation)
211     }
```

# 1. 平行链阶段

- Collator@cumulus/collator/src/lib.rs
- Collator类型定义
- proposer\_factory, 是提议者实例（Proposer） , 可以propose一个区块
- inherent\_data\_providers, 提供inherent数据的实例, 包括babe、grandpa、timestamp等
- block\_import, 提供区块导入的逻辑
- 以上三个都是parachain上的对象实例, 下面的network是relay chain的对象实例
- collator\_network, relay chain的network对象

```
59     /// The implementation of the Cumulus `Collator`.  
60 ①↓ pub struct Collator<Block: BlockT, PF, BI> {  
61     proposer_factory: Arc<Mutex<PF>>,  
62     _phantom: PhantomData<Block>,  
63     inherent_data_providers: InherentDataProvider,  
64     collator_network: Arc<dyn CollatorNetwork>,  
65     block_import: Arc<Mutex<BI>>,  
66     wait_to_announce: Arc<Mutex<WaitToAnnounce<Block>>>,  
67 }
```

# 1. 平行链阶段

- CollatorBuilder.build@cumulus/collator/src/lib.rs
- 366行, . . . .
- 372行, . . . .
- 380行, . . . .
- 384行, 生成Collator实例并返回

```
348 ⚡ fn build<PClient, Spawner, Extrinsic>(
349     self,
350     polkadot_client: Arc<PClient>,
351     spawner: Spawner,
352     polkadot_network: impl CollatorNetwork + Clone + 'static,
353 ) -> Result<Self::ParachainContext, ()>
354 where
355     PClient: ProvideRuntimeApi<PBlock>
356         + BlockchainEvents<PBlock>
357         + HeaderBackend<PBlock>
358         + Send
359         + Sync
360         + 'static,
361     PClient::Api: RuntimeApiCollection<Extrinsic>,
362         <PClient::Api as ApiExt<PBlock>>::StateBackend: StateBackend<HashFor<PBlock>>,
363     Spawner: Spawn + Clone + Send + Sync + 'static,
364     Extrinsic: codec::Codec + Send + Sync + 'static,
365 {
366     self.delayed_block_announce_validator
367         .set(Box::new(JustifiedBlockAnnounceValidator::new(
368             polkadot_client.clone(),
369             self.para_id,
370         )));
371
372     let follow =
373         match cumulus_consensus::follow_polkadot(self.para_id, self.client, polkadot_client) {
374             Ok(follow) => follow,
375             Err(e) => {
376                 return Err(error!("Could not start following polkadot: {:?}", e));
377             }
378         };
379
380     spawner
381         .spawn_obj(Box::new(follow.map(|_| ()).into())
382             .map_err(|_| error!("Could not spawn parachain server!"))?);
383
384     Ok(Collator::new(
385         self.proposer_factory,
386         self.inherent_data_providers,
387         polkadot_network,
388         self.block_import,
389         Arc::new(spawner),
390         self.announce_block,
391     ))
392 }
393 }
```

# 1. 平行链阶段

- produce\_candidate@cumulus/collator/src/lib.rs
- 这是produce\_candidate的第一部分
- 186行， 获得最新区块头
- 194行， 基于最新区块头初始化提议者（Proposer）
- 199行， 等待初始化完成
- 208行， 获取inherent data， 包括babe、grandpa、timestamp、authorship等数据
- 215行， 调用proposer的propose方法来产生区块， 具体实现在substrate的sc-basic-authorship中，这里

```
173 fn produce_candidate(
174     &mut self,
175     relay_chain_parent: PHash,
176     global_validation: GlobalValidationSchedule,
177     local_validation: LocalValidationData,
178     downward_messages: Vec<DownwardMessage>,
179 ) -> Self::ProduceCandidate {
180     let factory = self.proposer_factory.clone();
181     let inherent_providers = self.inherent_data_providers.clone();
182     let block_import = self.block_import.clone();
183
184     trace!(target: "cumulus-collator", "Producing candidate");
185
186     let last_head = match HeadData::decode(&mut &local_validation.parent_head.0[..]) {
187         Ok(x) => x,
188         Err(e) => {
189             error!(target: "cumulus-collator", "Could not decode the head data: {:?}", e);
190             return Box::pin(future::ready(Err(InvalidHead)));
191         }
192     };
193
194     let proposer_future = factory.lock().init(&last_head.header);
195
196     let wait_to_announce = self.wait_to_announce.clone();
197
198     Box::pin(async move {
199         let proposer = proposer_future.await.map_err(|e| {
200             error!(
201                 target: "cumulus-collator",
202                 "Could not create proposer: {:?}",
203                 e,
204             );
205             InvalidHead
206         })?;
207
208         let inherent_data = Self::inherent_data(
209             inherent_providers,
210             global_validation,
211             local_validation,
212             downward_messages,
213         )?;
214
215         let Proposal {
216             block,
217             storage_changes,
218             proof,
219         } = proposer
220             .propose(
221                 inherent_data,
222                 Default::default(),
223                 //TODO: Fix this.
224                 Duration::from_secs(6),
225                 RecordProof::Yes,
226             )
227             .await
228             .map_err(|e| {
229                 error!(
230                     target: "cumulus-collator",
231                     "Proposing failed: {:?}",
232                     e,
233                 );
234                 InvalidHead
235             })?;
236     })
237 }
```

# 1. 平行链阶段

- produce\_candidate@cumulus/collator/src/lib.rs
- 这是produce\_candidate的第二部分
- 237行，产出了区块后，同时获得proof数据
- 248行，创建ParachainBlockData
- 259行，做block import的操作
- 272-278行，生成block candidate
- 280行，在parachain上广播该区块

```
237     let proof = proof.ok_or_else(|| {
238         error!(
239             target: "cumulus-collator",
240             "Proposer did not return the requested proof.",
241         );
242         InvalidHead
243     })?;
244
245     let (header, extrinsics) = block.deconstruct();
246
247     // Create the parachain block data for the validators.
248     let b = ParachainBlockData::<Block>::new(
249         header.clone(),
250         extrinsics,
251         proof.iter_nodes().collect(),
252     );
253
254     let mut block_import_params = BlockImportParams::new(BlockOrigin::Own, header);
255     block_import_params.body = Some(b.extrinsics().to_vec());
256     block_import_params.fork_choice = Some(ForkChoiceStrategy::LongestChain);
257     block_import_params.storage_changes = Some(storage_changes);
258
259     if let Err(err) = block_import
260         .lock()
261         .import_block(block_import_params, Default::default())
262     {
263         error!(
264             target: "cumulus-collator",
265             "Error importing build block (at {:?}): {:?}",
266             b.header().parent_hash(),
267             err,
268         );
269         return Err(InvalidHead);
270     }
271
272     let block_data = BlockData(b.encode());
273     let header = b.into_header();
274     let encoded_header = header.encode();
275     let hash = header.hash();
276     let head_data = HeadData::<Block> { header };
277
278     let candidate = (block_data, parachain::HeadData(head_data.encode()));
279
280     wait_to_announce
281         .lock()
282         .wait_to_announce(hash, relay_chain_parent, encoded_header);
283
284     trace!(target: "cumulus-collator", "Produced candidate: {:?}", candidate);
285
286     Ok(candidate)
287 }
288 }
```

# 1. 平行链阶段

- 以上，我们整理了：
- collator的启动流程，从cumulus到polkadot
- collator包括了三个服务：PC node、RC node、collator service
- 区块如何产生的，由collator key签名
- 区块广播到parachain，network.announce\_block，代码在[这里](#)
- 区块广播到relaychain，network.distribute\_collation，代码在[这里](#)和[这里](#)

## 2. 中继链阶段 - 验证服务

- 第二部分，我们来看看relay chain的验证服务部分
- 验证服务实例定义在这里：
- [ParachainValidationInstances@polkadot/validation/src/validation\\_service/mod.rs](#)
- client，是访问区块链服务的组件
- network，网络服务组件
- spawner：可以启动异步服务的handle
- live\_instances：一个map， relay chain hash关联到验证实例
- validation\_pool：验证hosts，用来提供实际验证的worker
- collation\_fetch：获取collation的组件

```
311     /// Constructs parachain-agreement instances.
312     pub(crate) struct ParachainValidationInstances<N: Network, P, SP, CF> {
313         // The client instance.
314         client: Arc<P>,
315         // The backing network handle.
316         network: N,
317         // handle to spawner
318         spawner: SP,
319         // Store for extrinsic data.
320         availability_store: AvailabilityStore,
321         // Live agreements. Maps relay chain parent hashes to attestation
322         // instances.
323         live_instances: HashMap<Hash, LiveInstance<N::TableRouter>>,
324         // The underlying validation pool of processes to use.
325         // Only `None` in tests.
326         validation_pool: Option<ValidationPool>,
327         // Used to fetch a collation.
328         collation_fetch: CF,
329     }
```

```
302     /// A live instance that is related to a relay chain validation round.
303     ///
304     /// It stores the `instance_handle` and the `_table_router`.
305     pub(crate) struct LiveInstance<TR> {
306         instance_handle: ValidationInstanceHandle,
307         // Make sure we keep the table router alive, to respond/receive consensus messages.
308         _table_router: TR,
309     }
310 }
```

## 2. 中继链阶段 - 验证服务

- [ParachainValidationInstances.get\\_or\\_instantiate@polkadot/validation/validation\\_service/mod.rs](#)
- 验证服务的初始化，第一部分
- 359行，如果有实例则返回，注意这里构建的实例，是和 parent\_hash相关联的，每一个共识轮次，都会做一次初始化
- 365行，从runtime api中得到validators数据
- 366行，拿到本地keystore中可以给validators签名的key
- 368行，从runtime api拿到validators的任务表，任务是：relay chain / Parachain(ID)
- 370行，通过validators、duty\_roster计算分组信息，以及 local\_duty
- 382、384行，从runtime拿到parachains信息
- 403行，验证runtime api的版本信息，并得到signing\_context

```
351     async fn get_or_instantiate(
352         &mut self,
353         parent_hash: Hash,
354         keystore: &KeyStorePtr,
355         max_block_data_size: Option<u64>,
356     ) -> Result<ValidationInstanceHandle, Error> {
357         use primitives::Pair;
358
359         if let Some(instance : &LiveInstance<<...>::TableRouter>) = self.live_instances.get(k: &parent_hash) {
360             return Ok(instance.instance_handle.clone());
361         }
362
363         let id : BlockId<Block> = BlockId::hash(parent_hash);
364
365         let validators = self.client.runtime_api().validators(&id)?;
366         let sign_with : Option<Arc<?>> = signing_key(validators: &validators[..], keystore);
367
368         let duty_roster = self.client.runtime_api().duty_roster(&id)?;
369
370         let (group_info : HashMap<Id, GroupInfo>, local_duty : Option<LocalDuty>) = crate::make_group_info(
371             duty_roster,
372             authorities: &validators,
373             local_id: sign_with.as_ref().map(f: |k : &Arc<?>| k.public()),
374         )?;
375
376         info!(
377             "Starting parachain attestation session on top of parent {:?}. Local parachain duty is {:?}",
378             parent_hash,
379             local_duty,
380         );
381
382         let active_parachains = self.client.runtime_api().active_parachains(&id)?;
383
384         debug!(target: "validation", "Active parachains: {:?}", active_parachains);
385
386         // If we are a validator, we need to store our index in this round in availability store.
387         // This will tell which erasure chunk we should store.
388         if let Some(ref local_duty : &LocalDuty) = local_duty {...}
389
400         let api : ApiRef<<...>::Api> = self.client.runtime_api();
401
402         let signing_context : SigningContext = if api.has_api_with::<dyn ParachainHost<Block, Error = ()>, _>(
403             at: &BlockId::hash(parent_hash),
404             pred: |version : u32| version >= 3,
405         )? {
406             api.signing_context(&id)?
407         } else {
408             trace!(
409                 target: "validation",
410                 "Expected runtime with ParachainHost version >= 3",
411             );
412             SigningContext {
413                 session_index: 0,
414                 parent_hash,
415             }
416         };
417     }
```

## 2. 中继链阶段 - 验证服务

- ParachainValidationInstances.get or instantiate@polkadot/validation/validation service/mod.rs
- 验证服务的初始化，第二部分
- 419行，构建SharedTable，当前共识round中，需要的各种数据，包括：验证者、分组、Key、context、availability\_store? ? ? ? ? validation\_pool
- 431行，构建当前共识轮次中的Router，router负责将callation数据传送给network层，并进行后续广播
- 442-457行，spawn出一个任务launch work，从collator获取callation数据，设置data到availability\_store，并将callation广播给其它validator
- 451行，提供collation fetch closure，在launch\_work会回调
- 464行，构建一个tracker，让调用者能够访问table的数据，跟踪到进度
- 469-473行，构建一个LiveInstance，和parent\_hash关联，并添加到map中

```
419 let table : Arc<SharedTable> = Arc::new( data: SharedTable::new(
420     validators: validators.clone(),
421     groups: group_info,
422     key: sign_with,
423     signing_context,
424     availability_store: self.availability_store.clone(),
425     max_block_data_size,
426     validation_pool: self.validation_pool.clone(),
427 );
428
429 // The router will join the consensus gossip network. This is important
430 // to receive messages sent for the current round.
431 let router = match self.network.build_table_router(
432     table: table.clone(),
433     authorities: &validators,
434 ).await {
435     Ok(res) => res,
436     Err(e) => {
437         warn!(target: "validation", "Failed to build router: {:?}", e);
438         return Err(Error::CouldNotBuildTableRouter(format!("{}: {:?}", e)));
439     }
440 };
441
442 if let Some((Chain::Parachain(id: Id), index: u32)) = local_duty.map(f: |d: LocalDuty| (d.
443     let nValidators = validators.len();
444     let availabilityStore: Store = self.availability_store.clone();
445     let client: Arc<P> = self.client.clone();
446     let collationFetch: CF = self.collation_fetch.clone();
447     let router = router.clone();
448
449     let res: Result<(), SpawnerError> = self.spawner.spawn(
450         future: launchWork(
451             collationFetch: move || collationFetch.collationFetch(parachain: id, relayParent:
452                 availabilityStore,
453                 router,
454                 nValidators,
455                 localId: index,
456             ),
457         ),
458
459         if let Err(e: SpawnerError) = res {
460             error!(target: "validation", "Failed to launch work: {:?}", e);
461         }
462     );
463
464     let tracker = ValidationInstanceHandle {
465         table,
466         started: Instant::now(),
467     };
468
469     let liveInstance = LiveInstance {
470         instanceHandle: tracker.clone(),
471         _tableRouter: router,
472     };
473     self.liveInstances.insert(k: parentHash, v: liveInstance);
474 }
```

## 2. 中继链阶段 - 验证服务

- 定义、实现、调用runtime api的方式
- decl runtime apis@polkadot/primitives/src/parachain.rs
- impl runtime apis@polkadot/runtime/polkadot/src/lib.rs

```
363     let id : BlockId<Block> = BlockId::hash(parent_hash);
364
365     let validators = self.client.runtime_api().validators(&id)?;
366     let sign_with : Option<Arc<?>> = signing_key(validators: &validators[..], keystore);
367
368     let duty_roster = self.client.runtime_api().duty_roster(&id)?;
```

```
669     sp_api::decl_runtime_apis! {
670         /// The API for querying the state of parachains on-chain.
671         #[api_version(3)]
672         pub trait ParachainHost {
673             /// Get the current validators.
674             fn validators() -> Vec<ValidatorId>;
675             /// Get the current duty roster.
676             fn duty_roster() -> DutyRoster;
677             /// Get the currently active parachains.
678             fn active_parachains() -> Vec<(Id, Option<(CollatorId, Retriable)>)>;
679             /// Get the global validation schedule that all parachains should
680             /// be validated under.
```

```
1007     impl parachain::ParachainHost<Block> for Runtime {
1008         fn validators() -> Vec<parachain::ValidatorId> {
1009             Parachains::authorities()
1010         }
1011         fn duty_roster() -> parachain::DutyRoster {
1012             Parachains::calculate_duty_roster().0
1013         }
1014         fn active_parachains() -> Vec<(parachain::Id, Option<(parachain::CollatorId, parachain::Retr
1015             Registrar::active_paras()
1016         }
1017         fn global_validation_schedule() -> parachain::GlobalValidationSchedule {
1018             Parachains::global_validation_schedule()
1019         }
```

## 2. 中继链阶段 - 验证服务

- [make\\_group\\_info@polkadot/validation/src/lib.rs](https://github.com/polkadot/validation/src/lib.rs#L163)
- 核心逻辑，180-195行，遍历roster、authorities，更新map，得到每个parachain分配的validators
- 同时，如果当前节点V是验证节点，那么计算出V的当前轮次的工作：LocalDuty
- 197-200行，计算每一个组需要的最小确认数，超过一半

```
163     /// Compute group info out of a duty roster and a local authority set.
164     pub fn make_group_info(
165         roster: DutyRoster,
166         authorities: &[ValidatorId],
167         local_id: Option<ValidatorId>,
168     ) -> Result<(HashMap<ParaId, GroupInfo>, Option<LocalDuty>), Error> {
169         if roster.validator_duty.len() != authorities.len() {
170             return Err(Error::InvalidDutyRosterLength {
171                 expected: authorities.len(),
172                 got: roster.validator_duty.len()
173             });
174         }
175
176         let mut local_validation = None;
177         let mut local_index : usize = 0;
178         let mut map : HashMap<Id, GroupInfo> = HashMap::new();
179
180         let duty_iter : Zip<Iter<?>, Iter<Chain>> = authorities.iter().zip(other: &roster.validator_duty);
181         for (i : usize, (authority : &?, v_duty : &Chain)) in duty_iter.enumerate() {
182             if Some(authority) == local_id.as_ref() {
183                 local_validation = Some(v_duty.clone());
184                 local_index = i;
185             }
186
187             match *v_duty {
188                 Chain::Relay => {}, // does nothing for now.
189                 Chain::Parachain(ref id : &Id) => {
190                     map.entry(key: id.clone()).or_insert_with(GroupInfo::default)
191                         .validity_guarantors
192                         .insert(value: authority.clone());
193                 }
194             }
195         }
196
197         for live_group : &mut GroupInfo in map.values_mut() {
198             let validity_len : usize = live_group.validity_guarantors.len();
199             live_group.needed_validity = validity_len / 2 + validity_len % 2;
200         }
201
202         let local_duty : Option<LocalDuty> = local_validation.map(f: |v : Chain| LocalDuty {
203             validation: v,
204             index: local_index as u32,
205         });
206
207         Ok((map, local_duty))
208     }
209 }
```

## 2. 中继链阶段 - 验证服务

- launch work@polkadot/validation/validation\_service/mod.rs
- 496行，通过collation\_fetch从collators拿到一个Collation，  
对collation的验证工作，也在这里做的
- 504-509行，解包输出的数据，得到承诺commitments、纠删  
码erasure\_chunks、验证数据available\_data
- 511行，将commitments添加到collation\_info得到receipt
- 512行，得到pov\_block数据，区块数据
- 514行，将收据、available\_data存入store，这里这个store是  
一个kv-db
- 525行，将纠删码数据存入store
- 533行，将collation发送给其它validators，在此之前，会对  
发送数据进行签名，并将其保存到shared\_table

```
485     } async fn launch_work<CFG, E>(
486         collation_fetch: impl FnOnce() -> CFG,
487         availability_store: AvailabilityStore,
488         router: impl TableRouter,
489         nValidators: usize,
490         local_id: ValidatorIndex,
491     ) where
492         E: std::fmt::Debug,
493         CFG: Future<Output = Result<(CollationInfo, FullOutput), E>> + Send,
494     {
495         // fetch a local collation from connected collators.
496         let (collation_info: CollationInfo, full_output: FullOutput) = match collation_fetch().await {
497             Ok(res: (CollationInfo, FullOutput)) => res,
498             Err(e: E) => {
499                 warn!(target: "validation", "Failed to collate candidate: {:?}", e);
500                 return;
501             }
502         };
503
504         let crate::pipeline::FullOutput {
505             commitments: CandidateCommitments,
506             erasure_chunks: Vec<ErasureChunk>,
507             available_data: AvailableData,
508             ..
509         } = full_output;
510
511         let receipt: AbridgedCandidateReceipt = collation_info.into_receipt(commitments);
512         let pov_block: PoVBlock = available_data.pov_block.clone();
513
514         if let Err(e: Error) = availability_store.make_available(
515             candidate_hash: receipt.hash(),
516             available_data,
517         ).await {
518             warn!(
519                 target: "validation",
520                 "Failed to make parachain block data available: {}",
521                 e,
522             );
523         }
524
525         if let Err(e: Error) = availability_store.clone().add_erasure_chunks(
526             candidate: receipt.clone(),
527             nValidators: nValidators as_,
528             chunks: erasure_chunks.clone(),
529         ).await {
530             warn!(target: "validation", "Failed to add erasure chunks: {}", e);
531         }
532
533         if let Err(e) = router.local_collation(
534             receipt,
535             pov_block,
536             chunks: (local_id, &erasure_chunks),
537         ).await {
538             warn!(target: "validation", "Failed to send local collation: {:?}", e);
539         }
540     }
```

## 2. 中继链阶段 - 验证服务

- 保存collation的数据结构
- [SlotEntries@polkadot/network/src/legacy/collator\\_pool.rs](#)
- SlotEntries枚举结构，通过定义了Blank、Pending、Awaiting三种值，以及当发生收到collation、添加接收人等事件时的状态流转，把collation和receiver连接了起来

```
68 #[derive(Debug)]
69 enum SlotEntries {
70     Blank,
71     // not queried yet
72     Pending(Vec<Collation>),
73     // waiting for next to arrive.
74     Awaiting(Vec<oneshot::Sender<Collation>>),
75 }
76
77 impl SlotEntries {
78     fn received_collation(&mut self, collation: Collation) {
79         *self = match std::mem::replace(self, SlotEntries::Blank) {
80             SlotEntries::Blank => SlotEntries::Pending(vec![collation]),
81             SlotEntries::Pending(mut cs) => {
82                 cs.push(collation);
83                 SlotEntries::Pending(cs)
84             }
85             SlotEntries::Awaiting(senders) => {
86                 for sender in senders {
87                     let _ = sender.send(collation.clone());
88                 }
89             }
90         };
91     }
92
93     fn await_with(&mut self, sender: oneshot::Sender<Collation>) {
94         *self = match std::mem::replace(self, SlotEntries::Blank) {
95             SlotEntries::Blank => SlotEntries::Awaiting(vec![sender]),
96             SlotEntries::Awaiting(mut senders) => {
97                 senders.push(sender);
98                 SlotEntries::Awaiting(senders)
99             }
100            SlotEntries::Pending(mut cs) => {
101                let next_collation = cs.pop().expect("empty variant is always `Blank`; qed");
102                let _ = sender.send(next_collation);
103
104                if cs.is_empty() {
105                    SlotEntries::Blank
106                } else {
107                    SlotEntries::Pending(cs)
108                }
109            }
110        };
111    }
112
113    fn into_vec(self) -> Vec<Collation> {
114        match self {
115            SlotEntries::Pending(cs) => cs.into_iter().collect(),
116            SlotEntries::Awaiting(senders) => senders.map(|s| s).collect(),
117            SlotEntries::Blank => Vec::new(),
118        }
119    }
120 }
```

## 2. 中继链阶段 - 验证服务

- 前文讲到launch\_work会调用router的local\_collation，该函数最终会调用到distribute validated collation
- 1312-1320行，构建签名的statement并p2p广播
- 1324-1335行，构建区块数据pov\_block并p2p广播
- 1338-1348行，构建纠删码chunk数据并p2p广播

```
1296 fn distribute_validated_collation(
1297     instance: &ConsensusNetworkingInstance,
1298     receipt: AbridgedCandidateReceipt,
1299     pov_block: PoVBlock,
1300     chunks: (ValidatorIndex, Vec<ErasureChunk>),
1301     gossip_handle: &impl GossipOps,
1302 ) {
1303     // produce a signed statement.
1304     let hash: Hash = receipt.hash();
1305     let validated: Validated = Validated::collated_local(
1306         receipt,
1307         collation: pov_block.clone(),
1308     );
1309
1310     // gossip the signed statement.
1311     {
1312         let statement: GossipStatement = crate::legacy::gossip::GossipStatement::new(
1313             relay_chain_leaf: instance.relay_parent,
1314             signed_statement: match instance.statement_table.import_validated(validated) {
1315                 None => return,
1316                 Some(s: SignedStatement) => s,
1317             }
1318         );
1319     }
1320     gossip_handle.gossip_message(instance.attestation_topic, message: statement.into());
1321 }
1322
1323 // gossip the PoV block.
1324 {
1325     let pov_block_message = crate::legacy::gossip::GossipPoVBlock {
1326         relay_chain_leaf: instance.relay_parent,
1327         candidate_hash: hash,
1328         pov_block,
1329     };
1330
1331     gossip_handle.gossip_message(
1332         topic: crate::legacy::gossip::pov_block_topic(parent_hash: instance.relay_parent),
1333         message: pov_block_message.into(),
1334     );
1335 }
1336
1337 // gossip erasure chunks.
1338 for chunk: ErasureChunk in chunks.1 {
1339     let message = crate::legacy::gossip::ErasureChunkMessage {
1340         chunk,
1341         candidate_hash: hash,
1342     };
1343
1344     gossip_handle.gossip_message(
1345         topic: crate::legacy::erasure_coding_topic(candidate_hash: &hash),
1346         message: message.into(),
1347     );
1348 }
1349 }
```

## 2. 中继链阶段 - 验证服务

- 在验证服务中，有使用collation\_fetch，这里我们来看看
- [collation\\_fetch@polkadot/validation/src/collation.rs](#)
- 函数主体是一个死循环loop，直到获取到正确的数据后才会返回
- 77行，通过collators.collate获得collation；这里最终也是使用前面讲述的SlotEntries机制来获取collation
- 79行，使用pipeline组件，对collation进行验证
- 91行，返回正确的数据

```
60     /// A future which resolves when a collation is available.
61     pub async fn collation_fetch<C: Collators, P>(
62         validation_pool: Option<crate::pipeline::ValidationPool>,
63         parachain: ParaId,
64         relay_parent: Hash,
65         collators: C,
66         client: Arc<P>,
67         max_block_data_size: Option<u64>,
68         nValidators: usize,
69     ) -> Result<(CollationInfo, crate::pipeline::FullOutput), C::Error>
70     where
71         P::Api: ParachainHost<Block, Error = sp_blockchain::Error>,
72         C: Collators + Unpin,
73         P: ProvideRuntimeApi<Block>,
74         <C as Collators>::Collation: Unpin,
75     {
76         loop {
77             let collation = collators.collate(parachain, relay_parent).await?;
78             let Collation { info, pov } = collation;
79             let res = crate::pipeline::full_output_validation_with_api(
80                 validation_pool.as_ref(),
81                 &*client,
82                 &info,
83                 &pov,
84                 &relay_parent,
85                 max_block_data_size,
86                 nValidators,
87             );
88
89             match res {
90                 Ok(full_output) => {
91                     return Ok((info, full_output))
92                 }
93                 Err(e) => {
94                     debug!("Failed to validate parachain due to API error: {}", e);
95
96                     // just continue if we got a bad collation or failed to validate
97                     collators.note_bad_collator(info.collator)
98                 }
99             }
100        }
101    }
```

## 2. 中继链阶段 - 验证服务

- 在pipeline组件中，核心入口是validate函数
- 217行，执行mode，如果有worker则使用spawn模式，否则是in-process模式
- 221行，调用wasm\_executor的validate\_candidate函数进行验证，传入的核心数据为：
  - parachain的STF的代码code
  - 参数：parent\_head, height, block\_data
  - 执行mode
- 验证结束后，227行，比对validator的验证结果，和collator传过来的结果，也就是区块头，是否一致
- 一切正常的话，234行，返回一个ValidatedCandidate数据

```
187 pub fn validate<'a>(
188     validation_pool: Option<&'_ ValidationPool>,
189     collation: &'a CollationInfo,
190     pov_block: &'a PoVBlock,
191     local_validation: &'a LocalValidationData,
192     global_validation: &'a GlobalValidationSchedule,
193     validation_code: &ValidationCode,
194 ) -> Result<ValidatedCandidate<'a>, Error> {
195     if collation.head_data.0.len() > global_validation.max_head_data_size as _ {
196         return Err(Error::HeadDataTooLarge(
197             collation.head_data.0.len(),
198             global_validation.max_head_data_size as _,
199         ));
200     }
201
202     let params = ValidationParams {
203         parent_head: local_validation.parent_head.clone(),
204         block_data: pov_block.block_data.clone(),
205         max_code_size: global_validation.max_code_size,
206         max_head_data_size: global_validation.max_head_data_size,
207         relay_chain_height: global_validation.block_number,
208         code_upgrade_allowed: local_validation.code_upgrade_allowed,
209     };
210
211     // TODO: remove when ext does not do this.
212     let fee_schedule = FeeSchedule {
213         base: 0,
214         per_byte: 0,
215     };
216
217     let execution_mode = validation_pool
218         .map(ExecutionMode::Remote)
219         .unwrap_or(ExecutionMode::Local);
220
221     match wasm_executor::validate_candidate(
222         &validation_code.0,
223         params,
224         execution_mode,
225     ) {
226         Ok(result) => {
227             if result.head_data == collation.head_data {
228                 let fees = validate_upward_messages(
229                     &result.upward_messages,
230                     fee_schedule,
231                     local_validation.balance,
232                 )?;
233
234             Ok(ValidatedCandidate {
235                 pov_block,
236                 global_validation,
237                 local_validation,
238                 upward_messages: result.upward_messages,
239                 fees,
240                 processed_downward_messages: result.processed_downward_messages,
241             })
242             } else {
243                 Err(Error::HeadDataMismatch)
244             }
245         }
246         Err(e) => Err(e.into()),
247     }
248 }
```

## 2. 中继链阶段 - 验证服务

- [validate\\_candidate@polkadot/parachain/src/wasm\\_executor/validation\\_host.rs](#)
- 验证实例池validate\_pool中有n (8) 个验证实例ValidationHost, 每个host负责验证一个parachain的block
- validate\_candidate是验证逻辑的入口
- 244行, 启动一个worker, 实际验证的工作, 是在worker中做的
- host与work之间, 是通过shared mem的方式传送数据的
- 245-267行, host将各种数据copy到shared mem中, 包括: 区块头、STF的code、params等
- 270行, 发送ready事件给worker
- 273行, 等待worker执行结束
- 291行, 返回最终的结果, 主要内容是新区块的区块头数据

```
234 pub fn validate_candidate(  
235     &mut self,  
236     validation_code: &[u8],  
237     params: ValidationParams,  
238     test_mode: bool,  
239 ) -> Result<ValidationResult, Error> {  
240     if validation_code.len() > MAX_CODE_MEM {  
241         return Err(Error::CodeTooLarge(validation_code.len()));  
242     }  
243     // First, check if need to spawn the child process  
244     self.start_worker(test_mode)?;  
245     let memory = self.memory.as_mut()  
246         .expect("memory is always `Some` after `start_worker` completes successfully");  
247     // Put data in shared mem  
248     let data: &mut [u8] = &mut **memory.wlock_as_slice(0)?;  
249     let (mut header_buf, rest) = data.split_at_mut(1024);  
250     let (code, rest) = rest.split_at_mut(MAX_CODE_MEM);  
251     let (code, _) = code.split_at_mut(validation_code.len());  
252     let (call_data, _) = rest.split_at_mut(MAX_RUNTIME_MEM);  
253     code[..validation_code.len()].copy_from_slice(validation_code);  
254     let encoded_params = params.encode();  
255     if encoded_params.len() ≥ MAX_RUNTIME_MEM {  
256         return Err(Error::ParamsTooLarge(MAX_RUNTIME_MEM));  
257     }  
258     call_data[..encoded_params.len()].copy_from_slice(&encoded_params);  
259     let header = ValidationHeader {  
260         code_size: validation_code.len() as u64,  
261         params_size: encoded_params.len() as u64,  
262     };  
263     header.encode_to(&mut header_buf);  
264     }  
265     debug!("{} Signaling candidate", self.id);  
266     memory.set(Event::CandidateReady as usize, EventState::Signaled)?;  
267     debug!("{} Waiting for results", self.id);  
268     match memory.wait(Event::ResultReady as usize, shared_memory::Timeout::Sec(EXECUTION_TIMEOUT))  
269     Err(e) => {...}  
270     Ok(_) => {}  
271     }  
272     debug!("{} Reading results", self.id);  
273     let data: &[u8] = &**memory.wlock_as_slice(0)?;  
274     let (header_buf, _) = data.split_at(1024);  
275     let mut header_buf: &[u8] = header_buf;  
276     let header = ValidationResultHeader::decode(&mut header_buf).unwrap();  
277     match header {  
278         ValidationResultHeader::Ok(result) => Ok(result),  
279         ValidationResultHeader::Error(message) => {  
280             debug!("{} Validation error: {}", self.id, message);  
281             Err(Error::External(message).into())  
282         }  
283     }  
284     }  
285 }
```

## 2. 中继链阶段 - 验证服务

- start\_worker@polkadot/parachain/src/wasm\_executor/validation\_host.rs
- 在前面一步validate\_candidate中，会调用start\_worker启动一个worker
- 211行，创建shared mem
- 212行，得到当前执行程序的path
- 215行，再次启动当前程序，单独的一个新进程，入参传入“WORKS\_ARGS”，同时传入shared mem的file path， /dev/shm
- 223行，等待run\_worker中的WorkerReady的事件
- 返回前，worker与memory都已经ready

```
190  impl ValidationHost {
191      fn create_memory() → Result<SharedMem, Error> {
192          let mem_size = MAX_RUNTIME_MEM + MAX_CODE_MEM + 1024;
193          let mem_config = SharedMemConf::default()
194              .set_size(mem_size)
195              .add_lock(shared_memory::LockType::Mutex, 0, mem_size)?
196              .add_event(shared_memory::EventType::Auto)? // Event::CandidateReady
197              .add_event(shared_memory::EventType::Auto)? // Event::ResultReady
198              .add_event(shared_memory::EventType::Auto)?; // Event::WorkerReady
199
200          Ok(mem_config.create()?)
201      }
202
203      fn start_worker(&mut self, test_mode: bool) → Result<(), Error> {
204          if let Some(ref mut worker) = self.worker {
205              // Check if still alive
206              if let Ok(None) = worker.try_wait() {
207                  // Still running
208                  return Ok(());
209              }
210
211              let memory = Self::create_memory()?;
212              let self_path = env::current_exe()?;
213              debug!("Starting worker at {:?}", self_path);
214              let mut args = if test_mode { WORKER_ARGS_TEST.to_vec() } else { WORKER_ARGS.to_vec() };
215              args.push(memory.get_os_path());
216              let worker = process::Command::new(self_path)
217                  .args(args)
218                  .stdin(process::Stdio::piped())
219                  .spawn()?;
220              self.id = worker.id();
221              self.worker = Some(worker);
222
223              memory.wait(
224                  Event::WorkerReady as usize,
225                  shared_memory::Timeout::Sec(EXECUTION_TIMEOUT_SEC as usize),
226              )?;
227              self.memory = Some(memory);
228          }
229      }
}
```

## 2. 中继链阶段 - 验证服务

- [run\\_worker@polkadot/parachain/src/wasm\\_executor/validation\\_host.rs](#)
- 86行, open shared mem
- 94-104行, 设置一个退出机制
- 106行, 发送WorkerReady事件
- 110行, 判断是否退出
- 115行, 等待CandidateReady事件
- 131-140行, 准备数据, code 和 call\_data
- 142行, 基于code执行call\_data
- 150-151行, 设置执行结果到shared mem
- 执行code的逻辑, 使用到了substrate中的WasmExecutor, 这里就不介绍了

```
85     pub fn run_worker(mem_id: &str) -> Result<(), String> {
86         let mut memory = match SharedMem::open(mem_id) {
87             Ok(memory) => memory,
88             Err(e) => {...}
89         };
90
91         let exit = Arc::new(atomic::AtomicBool::new(false));
92         // spawn parent monitor thread
93         let watch_exit = exit.clone();
94         std::thread::spawn(move || {
95             use std::io::Read;
96             let mut in_data = Vec::new();
97             // pipe terminates when parent process exits
98             std::io::stdin().read_to_end(&mut in_data).ok();
99             debug!("{} Parent process is dead. Exiting", process::id());
100            exit.store(true, atomic::Ordering::Relaxed);
101        });
102
103        memory.set(Event::WorkerReady as usize, EventState::Signaled)
104            .map_err(|e| format!("{} Error setting shared event: {:?}", process::id(), e))?;
105
106    loop {
107        if watch_exit.load(atomic::Ordering::Relaxed) {
108            break;
109        }
110
111        debug!("{} Waiting for candidate", process::id());
112        match memory.wait(Event::CandidateReady as usize, shared_memory::Timeout::Sec(3)) {...}
113
114        {
115            debug!("{} Processing candidate", process::id());
116            // we have candidate data
117            let mut slice = memory.wlock_as_slice(0)
118                .map_err(|e| format!("Error locking shared memory: {:?}", e))?;
119
120            let result = {
121                let data: &mut[u8] = &mut **slice;
122                let (header_buf, rest) = data.split_at_mut(1024);
123                let mut header_buf: &[u8] = header_buf;
124                let header = ValidationHeader::decode(&mut header_buf)
125                    .map_err(|_| format!("Error decoding validation request."))?;
126                debug!("{} Candidate header: {:?}", process::id(), header);
127                let (code, rest) = rest.split_at_mut(MAX_CODE_MEM);
128                let (code, _) = code.split_at_mut(header.code_size as usize);
129                let (call_data, _) = rest.split_at_mut(MAX_RUNTIME_MEM);
130                let (call_data, _) = call_data.split_at_mut(header.params_size as usize);
131
132                let result = validate_candidate_internal(code, call_data);
133                debug!("{} Candidate validated: {:?}", process::id(), result);
134
135                match result {
136                    Ok(r) => ValidationResultHeader::Ok(r),
137                    Err(e) => ValidationResultHeader::Error(e.to_string()),
138                };
139
140                let mut data: &mut[u8] = &mut **slice;
141                result.encode_to(&mut data);
142
143            }
144
145            debug!("{} Signaling result", process::id());
146            memory.set(Event::ResultReady as usize, EventState::Signaled)
147                .map_err(|e| format!("Error setting shared event: {:?}", e))?;
148        };
149
150    }
151
152 }
```

## 2. 中继链阶段 - 共识服务

- [@polkadot/network/src/protocol/mod.rs](#)
- 共识服务，主要由两个组件构成，一个是worker工作组件，另一个是ProtocolHandler
- Worker组件负责从网络、其它上层服务接受事件，并传送给各个具体的执行方，ProtocolHandler是执行方之一
- api: 通过api接入runtime获取共识需要的数据
- executor: 使用执行器执行一个后台任务
- gossip\_handle: 从p2p网络中获取/发送消息
- background\_to\_main\_sender、background\_receiver:

```
834 ⚡️ struct Worker<Api, Sp, Gossip> {  
835     protocol_handler: ProtocolHandler,  
836     api: Arc<Api>,  
837     executor: Sp,  
838     gossip_handle: Gossip,  
839     background_to_main_sender: mpsc::Sender<BackgroundToWorkerMsg>,  
840     background_receiver: mpsc::Receiver<BackgroundToWorkerMsg>,  
841     service_receiver: mpsc::Receiver<ServiceToWorkerMsg>,  
842     consensus_networking_receivers: FuturesUnordered<StreamFuture<ConsensusNetworkingReceiver>>,  
843 }
```

```
475 ⚡️ struct ProtocolHandler {  
476     service: Arc<dyn NetworkServiceOps>,  
477     peers: HashMap<PeerId, PeerData>,  
478     // reverse mapping from validator-ID to PeerID. Multiple peers can represent  
479     // the same validator because of sentry nodes.  
480     connected_validators: HashMap<ValidatorId, HashSet<PeerId>>,  
481     consensus_instances: HashMap<Hash, ConsensusNetworkingInstance>,  
482     collators: crate::legacy::collator_pool::CollatorPool,  
483     local_collations: crate::legacy::local_collations::LocalCollations<Collation>,  
484     config: Config,  
485     local_keys: RecentValidatorIds,  
486 }
```

```
391 ⚡️ struct ConsensusNetworkingInstance {  
392     statement_table: Arc<SharedTable>,  
393     relay_parent: Hash,  
394     attestation_topic: Hash,  
395     _drop_signal: exit_future::Signal,  
396 }
```

# AnV 6-2 Relay chain submission phase

- 上一节讲到，在relaychain上将collation广播出去
- [on\\_remote\\_collation@polkadot/network/src/protocol/mod.rs](#)
- 该函数做了很多检查
- 658行，这个是关键，调用了  
self.collators.on\_collation，把collation放到了  
collation pool中

```
475 struct ProtocolHandler {  
476     service: Arc<dyn NetworkServiceOps>,  
477     peers: HashMap<PeerId, PeerData>,  
478     // reverse mapping from validator-ID to PeerID. Multiple peers can represent  
479     // the same validator because of sentry nodes.  
480     connected_validators: HashMap<ValidatorId, HashSet<PeerId>>,  
481     consensus_instances: HashMap<Hash, ConsensusNetworkingInstance>,  
482     collators: crate::legacy::collator_pool::CollatorPool,  
483     local_collations: crate::legacy::local_collations::LocalCollations<Collation>,  
484     config: Config,  
485     local_keys: RecentValidatorIds,  
486 }
```

```
634  
635     fn on_remote_collation(&mut self, remote: PeerId, relay_parent: Hash, collation: Collation) {  
636         let peer : &mut PeerData = match self.peers.get_mut(k: &remote) {  
637             None => { self.service.report_peer(peer: remote, value: cost::UNKNOWN_PEER); return }  
638             Some(p : &mut PeerData) => p,  
639         };  
640  
641         let (collator_id, para_id : Id) = match peer.ready_and_collating_for() {  
642             None => {  
643                 self.service.report_peer(peer: remote, value: cost::UNEXPECTED_MESSAGE);  
644                 return  
645             }  
646             Some(x : (?, Id)) => x,  
647         };  
648  
649         let collation_para : Id = collation.info.parachain_index;  
650         let collated_acc = collation.info.collator.clone();  
651  
652         let structurally_valid : bool = para_id == collation_para && collator_id == collated_acc;  
653         if structurally_valid && collation.info.check_signature().is_ok() {  
654             debug!(target: "p_net", "Received collation for parachain {} from peer {}",  
655                 para_id, remote);  
656  
657             if self.collators.collator_id_to_peer_id(&collator_id) == Some(&remote) {  
658                 self.collators.on_collation(collator_id, relay_parent, collation);  
659                 self.service.report_peer(peer: remote, value: benefit::GOOD_COLLATION);  
660             }  
661         } else {  
662             self.service.report_peer(peer: remote, value: cost::INVALID_FORMAT);  
663         }  
664     }  
665 }
```

# AnV 6-2 Relay chain submission phase

- [CollationPool@polkadot/network/src/legacy/collation\\_pool.rs](#)
- CollationPool保存了collator的map
- 保存了某个parachain的collators的map
- 保存了接收到的collation数据

```
116 ① ↴ struct ParachainCollators {  
117     primary: CollatorId,  
118     backup: Vec<CollatorId>,  
119 }  
120  
121     /// Manages connected collators and role assignments from the perspective  
122     #[derive(Default)]  
123 ① ↴ pub struct CollatorPool {  
124     collators: HashMap<CollatorId, (ParaId, PeerId)>,  
125     parachain_collators: HashMap<ParaId, ParachainCollators>,  
126     collations: HashMap<(Hash, ParaId), CollationSlot>,  
127 }
```

```
50 ① ↴ struct CollationSlot {  
51     live_at: Instant,  
52     entries: SlotEntries,  
53 }  
54  
55     impl CollationSlot {...}  
67  
68     #[derive(Debug)]  
69 ① ↴ enum SlotEntries {  
70     Blank,  
71     // not queried yet  
72     Pending(Vec<Collation>),  
73     // waiting for next to arrive.  
74     Awaiting(Vec<oneshot::Sender<Collation>>),  
75 }
```